UNIVERSITY OF SOUTHAMPTON

Channel coding algorithms for Ultra-Reliable Low Latency Communication

by

Luping Xiang

A thesis submitted in partial fulfillment for the degree of Doctor of Philosophy

in the

Faculty of Engineering, Science and Mathematics

School of Electronics and Computer Science

December 2019

UNIVERSITY OF SOUTHAMPTON

ABSTRACT

FACULTY OF ENGINEERING, SCIENCE AND MATHEMATICS
SCHOOL OF ELECTRONICS AND COMPUTER SCIENCE

Doctor of Philosophy

by Luping Xiang

The Ultra-Reliable Low Latency Communication (URLLC) concept has been conceived for the emerging Fifth Generation (5G) systems, targeting a round-trip end-to-end latency of less than 1 ms in conjunction with ultra-high reliability. Therefore, this thesis proposes several novel channel coding schemes in order to meet the latency requirements of the URLLC mobile communication standard.

First, an Arbitrarily Parallel Turbo Decoder (APTD) is proposed to support an arbitrarily-high degree of parallel processing, facilitating significantly higher processing throughputs and substantially lower processing latencies than the State-of-the-art (SOTA) Long Term Evolution (LTE) turbo decoder. As in conventional turbo decoding algorithms, the proposed APTD decomposes each block of information bits into a sequence of windows, where the bits within different windows are processed simultaneously using forward and backward recursions in a serial manner. However, in contrast to conventional turbo decoding algorithms, the APTD does not require the different windows to be composed of an identical number of bits. This allows the use of an arbitrary number of windows and hence an arbitrary degree of parallelism, when decoding information bits of an arbitrary block length. Furthermore, conventional turbo decoding algorithms alternate between simultaneously processing the windows in the upper decoder and those in the lower decoder. By contrast, the APTD processes the odd-indexed windows in the upper decoder at the same time as the even-indexed windows in the lower decoder and alternates between this and the reversed arrangement, hence further improving the decoding throughput and latency. Furthermore, the APTD achieves a reduced hardware resource requirement by calculating the extrinsic information based only on the outputs of the forward recursions, rather than being based on both the forward and backward recursions of conventional turbo decoding algorithms. We demonstrate that the proposed APTD achieves superior latency, throughput and computational efficiency compared to the SOTA LTE turbo decoder at all block lengths, but particularly at the short block lengths that are typically used in URLLC approaches.

For example, at a block length of $N = 504$ bits, the proposed APTD achieves an Block Error Rate (BLER) of 10^{-5} at the same E_b/N_0 as $I = 8$ iterations of a conventional turbo decoder, but with a computational efficiency that is 6 times higher than that of the SOTA turbo decoder, while achieving a latency and throughput that are 0.7 and 1.4 times those of the SOTA decoder, respectively.

Additionally, the URLLC service requires an order of magnitude improvements in all layers of the wireless communication stack. This is a particular challenge for the physical layer, where typically a processing time of the order of microseconds is required for the computationally intensive demodulation and error correction processing, among other operations. Conventionally, the reception of signals, the demodulation processing and the error correction processing are performed consecutively at the receiver. However, this approach is associated with processing times on the order of hundreds of microseconds, preventing URLLC. Therefore, this paper proposes a novel processing architecture, which is capable of performing reception, Orthogonal Frequency-Division Multiplexing (OFDM) demodulation and turbo decoding concurrently, rather than consecutively, hence significantly reducing the processing time. In order to achieve concurrent operation, the OFDM demodulation is performed using a novel cumulative Fast Fourier Transform (FFT), which produces successively more reliable estimates of all transmitted symbols in each successive clock cycle. At the same time, a Fully-Parallel Turbo Decoder (FPTD) is used to recover successively more reliable estimates of all bits in each successive clock cycle.

Then, a detailed tutorial on the Cyclic Redundancy Check (CRC)-aided Logarithmic Successive Cancellation Stack (Log-SCS) algorithm conceived for polar codes is provided, followed by a pair of refinements for improving the error correction performance. We also apply these algorithms for the ultra-reliable decoding of polar codes, which has relevance for the control channels of the URLLC version of the 3rd Generation Partnership Project (3GPP) New Radio (NR). In contrast to the bit probabilities of all previous work on SCS polar decoding, the Log-SCS algorithm operates on the basis of Logarithmic-Likelihood Ratios (LLRs), which facilitates low-complexity fixed point implementation and reduced storage requirements. Furthermore, we extend the computation to consider frozen bits in stack decoding when determining the most likely sequence of information bits, which improves the error correction performance despite reducing the decoding complexity. During the exploitation of the CRC codes, for improving the error correction performance, we propose a novel technique which limits the number of CRC checks performed, in order to maintain a consistent error detection performance. Additionally, a pair of techniques for further improving the performance of the Log-SCS polar decoder are proposed and we demonstrate that the proposed $S = 128$ Improved Log-SCS decoder achieves a similar error correction capability as a Logarithmic Successive Cancellation List (Log-SCL) decoder having a list size of $L = 128$ across the full range of block lengths supported by the 3GPP NR Physical Uplink Control Channel (PUCCH). This is achieved

without increasing its memory requirement, while dramatically reducing its complexity, which becomes up to seven times lower than that of a $L = 8$ Log-SCL decoder.

Following the Improved Log-SCS algorithm, a novel fast Log-SCS polar decoder is proposed, which employs several techniques that is previously considered by the fast SCL decoder. This Log-SCS polar decoder is capable of attaining a decoding latency that is lower than that of the SOTA fast SCL polar decoders without the loss of error correction performance. First, a 32-bit fixed point Log-SCS polar decoder is achieved in this paper, which is capable of maintaining the same BLER as that of the floating-point Log-SCS polar decoder, allowing the software implementation on x86 processors. In addition, the simplified path-metric computation of the rate-0, rate-1 and repetition subgraphs is applied in the proposed fast Log-SCS decoder which reduces the decoding complexity by 50% on average. In addition, the software implementation of the fast Log-SCS polar decoder is achieved on the x86 processors that support Single Instruction Multiple Data (SIMD) instructions with 512-bit Advanced Vector Extensions (AVX-512) for the first time, satisfying the low-latency requirements of Software-Defined Radio (SDR) systems. By implementing the 32-bit fast Log-SCS polar decoder into the x86 processors in conjunction with AVX-512 SIMD instructions, a maximum parallelization degree of 16 may be attained, and an 80% latency reduction may be achieved.

𝔇edicated to my beloved family and friends . . .

Contents

List of Publications

1. **L. Xiang**, M. Brejza, R. Maunder, B. Al-Hashimi and L. Hanzo, "Arbitrarily Parallel Turbo Decoding for Ultra-Reliable Low Latency Communication in 3GPP LTE," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 4, pp. 826-838, 2019

2. **L. Xiang**, Z. Egilmez, R. Maunder and L. Hanzo, "CRC-Aided Logarithmic Stack Decoding of Polar Codes for Ultra Reliable Low Latency Communication in 3GPP New Radio," *IEEE ACCESS*, vol. 7, pp. 28559-28573, 2019

3. **L. Xiang**, R. Maunder and L. Hanzo, "Concurrent OFDM Demodulation and Turbo Decoding for Ultra Reliable Low Latency Communication," *IEEE Transactions on Vehicular Technology*, accepted, 2019

4. **L. Xiang**, R. Maunder and L. Hanzo, "Fast Log-SCS polar decoder and software implementation," in preparation, 2019

5. Z. Babar, Z. Egilmez, **L. Xiang**, D. Chandra, R. Maunder, S. Ng and L. Hanzo, "Polar Codes and Their Quantum-Domain Counterparts," *IEEE Communications Surveys & Tutorials*, accepted, 2019

6. A. Li, **L. Xiang**, T. Chen, R. Maunder and L. Hanzo, "VLSI Implementation of Fully Parallel LTE Turbo Decoders," *IEEE ACCESS*, vol. 4, pp. 323-346, 2016

7. Z. Egilmez, **L. Xiang**, R. Maunder and L. Hanzo, "The Development, Operation and Performance of the 5G Polar Codes," *IEEE Communications Surveys & Tutorials*, accepted, 2019

DECLARATION OF AUTHORSHIP

I, **Luping Xiang**, declare that the thesis entitled **Channel coding algorithms for Ultra-Reliable Low Latency Communication** and the work presented in the thesis are both my own, and have been generated by me as the result of my own original research. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University;
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated;
- Where I have consulted the published work of others, this is always clearly attributed;
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work;
- I have acknowledged all main sources of help;
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly What was done by others and what I have contributed myself;
- Parts of this work have been published in the provided list of publications

Signed: .....

Date: .....

Acknowledgements

I would like to first express my sincere gratitude to my supervisor Prof. Robert G. Maunder for his professional supervision, his generous support and great patience with me throughout my research. At the earlier stage of my PhD research, he introduced me the field of channel coding and helped me to discover my potential at research with constructive discussions and encouragements. I cannot complete this thesis without him.

I would also like to thank my supervisor Prof. Lajos Hanzo for his guidance throughout the past four years. His passion towards research, diligence and positive altitudes towards life inspired me a lot and would constantly encourage me in the future.

Many thanks my colleagues and all the staffs of the Next Generation Wireless Research Group for their useful discussions and comments throughout my research. Special thanks to Prof. Lie-Liang Yang, Dr. Soon X. Ng, Mr. Shuai Shao, Mr. Yanqing Zhang, Mr. Xiaoyu Zhang Miss. Zeynep Egilmez and Miss Yusha Liu, for their technical support and collaborative work.

I would like to express my warmest gratitude to my parents for the lifelong understanding and support. Finally, I would like to greatly thank my girlfriend, Miss. Yusha Liu, for her insightful advice, encouragement and support.

List of Symbols

General notation

- The superscript $*$ represents complex conjugate.
- The superscript ^{H} represents complex conjugate transpose operation
- The operator A_i represents the ith entry of $\mathbf{A}$.

Special symbols

Chapter 2

N	Block length
P	The number of processors
p	The number of active processors
I	The number of iterations
$\mathbf{b}_1^{\mathrm{u}}$	Message block
$\mathbf{b}_2^{\mathrm{u}}$	Upper *parity block*
$\mathbf{b}_3^{\mathrm{u}}$	Upper *systematic block*
$\mathbf{b}_1^{\mathrm{l}}$	Interleaved message block
$\mathbf{b}_2^{\mathrm{l}}$	Lower *parity block*
R	Coding rate
M	The number of states of in the transition diagram
K	The number of transitions per state
S	Encoder state
$\bar{\mathbf{b}}_2^{\mathrm{a}}$	Parity *a priori* LLRs
$\bar{\mathbf{b}}_3^{\mathrm{a}}$	Systematic *a priori* LLRs
$\bar{\mathbf{b}}_1^{\mathrm{e}}$	Extrinsic LLRs
$\bar{\mathbf{b}}_1^{\mathrm{a}}$	*A priori* LLRs

$\bar{\boldsymbol{\alpha}}$	Forward state metrics
$\bar{\boldsymbol{\beta}}$	Backward state metrics
L	The number of algorithmic units in a processor
$\bar{\gamma}$	*a priori* transition metrics
f	Scaling factor

Chapter 3

K	The number of message bits
T	The number of encoded bits
N	The number of transmit symbols
$\mathbf{X}$	Encoded signal
$\mathbf{x}$	OFDM signal
M	Constellation size
$\mathcal{S}$	Constellation set
L	The number of cyclic prefix symbols
$\mathbf{h}$	Channel Impulse Response
$\mathbf{n}$	Additive White Gaussian Noise
H_k	Single tap channel gain of the QAM symbol X_k
C	The number of clock cycles
$\mathbf{Y}$	Received signal in the frequency domain
$\mathbf{y}$	Received signal in the time domain
γ	Singal-to-noise ratio
σ^2	Varaince of AWGN noise

Chapter 4 & chapter 5

L	List size
S	Stack size
A	Information block length
A'	Segmented information block length
K	The number of information bits and CRC bits
N	Information block length
G	Encoded block length
$\mathbf{u}$	Information bits
$\mathbf{x}$	Encoded bits
$\hat{\mathbf{u}}$	Estimated information bits
ϕ_i	Path metric at the ith bit
$\tilde{x}_i$	LLR for the ith bit

$\mathbf{y}$	Received LLRs
$\mathcal{S}$	Stack with a single entry ℓ_{S_T}
P	CRC length
R	The maximum times that each of the N bits in the code tree may be visited

Chapter 1

Introduction

Channel coding has been used for correcting transmission errors over diverse channels incurred by noise, fading or interference. The venture into channel coding started with Shannon's groundbreaking paper in 1948 [1], where he quantifies the channel capacity over the Additive White Gaussian Noise (AWGN) channel. Since then, different capacity-approaching channel coding schemes have been proposed for wireless communication systems. More specifically, a channel encoder is employed in the transmitter to convert information bits into a longer bit sequence by concatenating redundant bits. In this way, the redundant bits protect the information bits in the transmission over different channels. At the receiver, a channel decoder recovers the original information bits.

In the Fourth Generation (4G) communications, turbo codes [2–4] and convolutional codes [5] have been adopted in the 3rd Generation Partnership Project (3GPP) Long Term Evolution (LTE) standard for mobile communication [6], whereas the 3GPP's group has standardised the Low-Density Parity-Check (LDPC) code [7] and polar code [8] as the channel coding schemes for the data channel and control channel, respectively, in the Fifth Generation (5G) New Radio (NR).

Table 1.1: Channel coding schemes employed in 4G LTE and 5G NR communication systems.

	LTE	NR
Control Channel	Convolutional	Polar
Data Channel	Turbo	LDPC

The three main 5G NR applications, namely the Ultra-Reliable Low Latency Communication (URLLC), enhanced Mobile Broadband (eMBB) communication, and the massive Machine-Type Communications (mMTC), aim for providing significantly improved user experience for cellular communications. The particular requirements of these three scenarios are highlighted in Figure 1.1. For example, eMBB communication aims for

achieving a maixmum throughput of 20 Gbps, which is 20 times higher compared to that of 4G LTE. In addition, URLLC targets a Block Error Rate (BLER) below 10^{-5} with an end-to-end latency within 1 ms. This is in contrast to the 10^{-4} BLER within 20 ms achieved by 4G LTE standard. This ultra-low latency requirement may hardly be satisfied employing the State-of-the-art (SOTA) transmission and receiving schemes [9]. A smooth progress towards 5G URLLC services will be based on the LTE URLLC mode [10] with several enhancements, with a particular focus on latency and error correction capability. Under this development, turbo codes require further exploitation to satisfy the latency and reliability requirement of LTE URLLC service. In addition, the turbo codes adopt the advantage of lower decoding complexity and better error-correction performance compared to the LDPC codes at low coding rates that are motivated in URLLC for the sake of ensuring ultra-low latency and ultra-high reliability [11, 12].

In the case of 5G NR, both LDPC and polar codes are employed for protecting the channel. However, in contrast to LDPC codes that have been excessively investigated, polar codes are far less mature. More specifically, while a better BLER performance has been demonstrated for polar codes in case of short block lengths, the throughput bottleneck of polar decoding has prevented its application in the 5G NR data channel.

Against this background, this thesis aims to design low-latency decoding algorithms that satisfies the requirements of the URLLC and joint decoding and demodulation schemes that could further reduce the decoding latency. To build upon this further, this chapter briefly overviews the challenges of our work and outlines the contributions of this thesis.

1.1 Ultra-Reliable Low Latency Communication

URLLC, as one of the three typical scenarios of the 5G communications, targets a 99.999% transmission reliability below 10^{-5} with an end-to-end latency within 1 ms. As shown in Figure 1.1, the eMBB and mMTC modes aim for maximizing the data rate at a moderate reliability of 10^{-3}, and for supporting massive connection in grant-free access, respectively. By contrast, the URLLC service focuses on ultra-reliable communications, where designing an outstanding channel coding scheme becomes one of the key targets in the forthcoming years. This motivates our research of designing practical channel coding schemes that have potential of satisfying the URLLC requirements.

In order to achieve a smooth transition, the initial roll-out of 5G will be based on the 3GPP's non-stand alone NR, which is built upon a foundation offered by 3GPP LTE [6]. Therefore, the 3GPP is currently standardising a URLLC mode for the LTE standard for mobile communication as part of the 5G development [10, 13–15]. Several enhancements have been introduced by 3GPP LTE URLLC, aiming for meeting the 1 ms latency and 10^{-5} reliability requirements [13]. To be more specific, LTE Release 15 has introduced the shortened Transmission Time Interval (sTTI) technique, which imposes a 7-fold

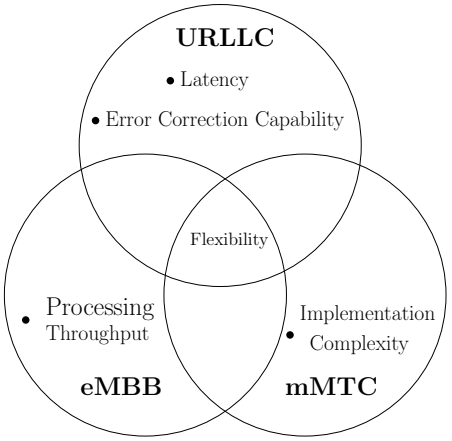


FIGURE 1.1: The requirements for different 5G target scenarios.

reduction upon the signal processing time available at the User Equipment (UE) and evolved Node B (eNB) base station, namely a reduction from 3 ms in Release 14, to 0.43 ms in Release 15 [16, 17]. Within this 0.43 ms, several signal processing tasks must be completed, including receive buffering, Fast Fourier Transform (FFT), channel decoding, Inverse Fast Fourier Transform (IFFT) and transmit power control [14, 15]. In order for an eNB to support the processing of multiple users' transmissions within this processing time without employing a unique, user-specific set of signal processing hardware per user, each of these tasks must be completed in much less than 0.43 ms. This significant reduction in the decoding latency requirement motivates the design of new low-latency channel decoding techniques.

1.2 Turbo codes for LTE URLLC

Turbo codes have been widely applied in the 4G LTE communication systems. In order to achieve the latency requirement of LTE URLLC, it will be necessary to complete the turbo decoder processing in tens of microseconds [18], motivating the replacement of the Logarithmic Bahl-Cocke-Jelinek-Raviv (Log-BCJR) algorithm in the associated turbo decoding [2–4, 19], which imposes strict data dependencies. More specifically, a turbo decoder comprises an upper-branch and a lower-branch convolutional decoder, which may operate alternately using the Log-BCJR to generate extrinsic information exchanged via an interleaver. However, the data dependencies of the Log-BCJR algorithm require the processing to be performed one step at a time, using forward and backward recursions alternately along the entire length of the interleaved information frame. This serial nature of the Log-BCJR algorithm imposes a bottleneck upon the throughput and results in processing latencies of hundreds of micro seconds for the turbo decoding process [19], unless parallel processing techniques are employed.

For the application of turbo codes in the 4G LTE system, several refinements of the Log-BCJR turbo decoder have been previously proposed for improving both the throughput and latency, such as the Radix-4 transform of [20, 21] and non-sliding window technique of [22]. To be more specific, the Radix-4 transform allows two steps of the Log-BCJR algorithm's forward and backward recursions to be completed at a time, therefore doubling the throughput and halving the latency, albeit at the cost of significantly increasing the hardware resource requirements. Meanwhile, the non-sliding window based technique of [23] decomposes each information frame of N bits into P number of windows, which can be processed simultaneously using separate parallel processors. Furthermore, the forward and backward recursions of the Log-BCJR can be completed simultaneously, rather than one after the other, hence further doubling the throughput and halving the latency. Using these techniques, a turbo decoder having 64 parallel processors has been proposed in [23].

However, the Log-BCJR turbo decoder is only able to fully exploit this parallelism for the longest frame lengths and disables up to 56 of the parallel processors at the short frame lengths that are common in URLLC applications, leading to poor hardware efficiency. Recently, a Fully-Parallel Turbo Decoder (FPTD) has been proposed for significantly increasing the grade of parallelism in the LTE turbo decoder, hence reducing the latency [24]. The employment of odd-even interleavers [25, 26] in LTE allows the FPTD to alternate between processing all odd-indexed bits in the upper decoder at the same time as all even-indexed bits in the lower decoder, and vice-versa, hence completing each decoding iteration in two clock cycles. However, the hardware requirement of the FPTD is dictated by the longest supported frame length. Since none of these hardware improvements can be exploited for shorter frame lengths, the hardware efficiency and flexibility of the FPTD remains limited. Therefore, a superior decoding algorithm which is able to simultaneously satisfy the latency and the reliability requirement, and, in the meantime, achieve higher hardware efficiency is required for URLLC. This challenge is addressed in this thesis.

In addition to improving the channel coding operation of LTE, exploiting diversity is another way of improving the reliability. The current LTE physical layer achieves a high throughput and reliability by employing the classic Orthogonal Frequency-Division Multiplexing (OFDM) technique [27, 28] for mitigating echoes in the wireless channel, as well as a turbo code for correcting any remaining transmission errors [2, 3, 29, 30]. However, these techniques impose a high signal processing complexity upon the physical layer, particularly in the receiver, which fails to meet the $\ll 0.43$ ms latency requirement of the URLLC scenario. Therefore, this thesis proposes a concurrent receiving, processing and decoding scheme, which may achieve a threefold reduction in the associated latency.

1.3 Polar codes for 5G NR URLLC

While turbo and LDPC codes have been exploited for decades and have been applied in different wireless communications scenarios, polar codes [8, 31] have been developed only within the past decade and have already demonstrated superior error correction performance at short block lengths compared to turbo codes. While polar coding has already been selected by 3GPP for protecting the eMBB control channels in 5G NR mobile communications [32], it still has promising application to the URLLC service, where the short block lengths dominate the transmission patterns.

Polar decoding algorithms of the polar codes fall into two categories, which either operate on the basis of Successive Cancellation (SC) [8, 33] or Belief Propagation (BP) [34–36]. Originally, the low-complexity SC [8, 33] decoding algorithm was originally proposed for polar codes, and was simplified in [37] to further reduce the complexity and the latency. However, more sophisticated decoding algorithms [38–45] are required for achieving near capacity performance in practical wireless channels. Rather than considering only the locally most-likely value for each successive bit, as in SC decoding, the Successive Cancellation List (SCL) decoder [39–41] uses a breadth-first search for identifying the L number of locally most-likely bit values, allowing it to more frequently spot the globally most-likely values. By contrast, the Successive Cancellation Stack (SCS) decoder [42] uses a depth-first approach to directly search for the globally most-likely values, although its ability to achieve this depends on the number of decoding candidates S that can be stored within the memory available for the stack. Note that while all L decoding candidates will have the same length in the SCL algorithm at high channel Signal to Noise Ratios (SNRs), many of the S candidates in the SCS algorithm tend to have much shorter lengths, hence reducing the complexity of the SCS algorithm below that of the SCL algorithm, approaching that of the SC algorithm.

In order to adapt to hardware implementations, the authors of [46] proposed a Logarithmic Successive Cancellation List (Log-SCL) decoder that operates on the basis of Logarithmic-Likelihood Ratios (LLRs), rather than bit probabilities as in the SCL decoder, which enables low-complexity fixed point implementation and reduced storage requirements, owing to the low dynamic range of LLRs [47–50]. In addition, [51] proposed a Logarithmic Successive Cancellation Stack (Log-SCS) decoder, which uses LLRs to bring the same advantages to the SCS decoder. However, when attaining similar BLER performance to that of SCL decoder, the Log-SCS decoder suffers from requiring massive stack size, and hence a massive amount of memory. Therefore, novel techniques are required to reduce the stack size while maintaining the BLER performance. In addition, the latency bottleneck is hard to break when employing the SOTA polar decoders, owing to the strict data dependencies of successive cancellation. Therefore, novel operations and decoding algorithms are required for meeting the latency requirement of the URLLC service.

TABLE 1.2: Contributions of each chapter to URLLC requirements

	Error correction performance	**Latency**	**Complexity**	**Flexibility**
Chapter 2	✓	✓	✓	✓
Chapter 3		✓		✓
Chapter 4	✓		✓	
Chapter 5	✓	✓	✓	✓

1.4 Contributions and thesis structure

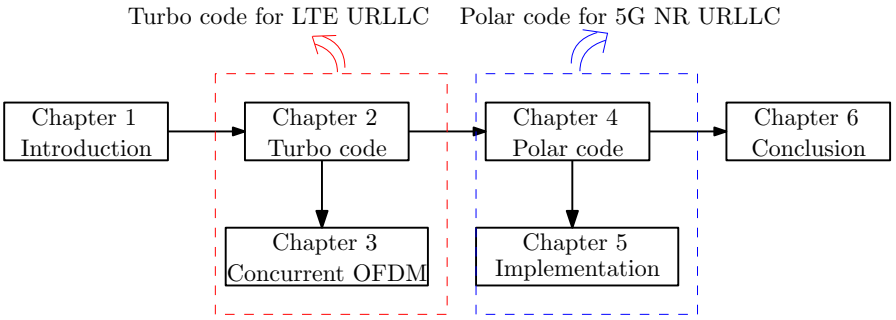


FIGURE 1.2: Thesis structure.

In this section, the thesis structure is outlined along with the novel contributions in each chapter. Figure 1.2 summarises the topics that are covered in this thesis, whereas Table 1.2 highlights the major contribution of our design towards the realisation of URLLC services. To start with, Chapter 2 reviews several LTE turbo decoder implementations and proposes a novel turbo decoding algorithm, referred to as Arbitrarily Parallel Turbo Decoder (APTD), which aims for achieving both high processing throughput and low processing latencies, while maintaining a high grade of flexibility. The novel contributions of the APTD are as follows.

- First, the APTD algorithm allows the parallel operation of an arbitrary number P of processors, which is not limited to an integer factor of the frame length N, nor to N itself. The first of two versions of the proposed APTD decomposes the frame into the highest possible number of windows that can support equal window lengths, where some processors are disabled as in the non-sliding window based and FPTD algorithms, respectively. For many frame lengths, this facilitates a grade of parallelism in excess of 64, which is the highest achieved previously in the literature owing to the requirement for P having to be a common divisor of many supported frame lengths N. In this first version of the APTD, the processors operate on the basis of windows of equal length, benefitting from the contention-free property of the LTE quadratic permutation polynomial interleaver. However, in a second version of the APTD, all processors are activated for all but the shortest

of frame lengths. Here, the different processors operate on windows of slightly different lengths. This breaks the contention-free property of the LTE quadratic permutation polynomial interleaver [52, 53], but we show that contention can be avoided by rescheduling the interleaver operation independently from the generation of extrinsic information, hence achieving even higher throughput.

- Furthermore, rather than operating the upper and lower decoder alternately like in conventional turbo decoding algorithms, the arbitrarily parallel turbo decoding algorithm employs odd-even operation, in a similar fashion to the FPTD of [24]. More specifically, the APTD alternates between processing the odd-indexed windows of the upper decoder at the same time as the even-indexed windows of the lower decoder, and vice versa. We will show that for short frame lengths, this odd-even operation gives superior BLER performance compared to the upper-lower operation.

- As a further contribution, we conceive and compare two techniques for providing systematic information to the APTD, namely the interleaved, and the non-interleaved systematic approaches. In the interleaved systematic approach, the systematic information is entered into the upper decoder, but additionally it is also interleaved and entered to the lower decoder. By contrast, in the non-interleaved systematic approach, the systematic information is only entered directly into the upper decoder, but it is also added into the upper decoder's extrinsic information, which is then interleaved and entered into the lower decoder. The benefit of the latter is that this reduces the number of interleavers that are required at the cost of a slightly degraded BLER performance.

- Finally, while conventional turbo decoders compute their extrinsic information in both the forward and backward recursions of the Log-BCJR algorithm in the proposed design, the extrinsic information is calculated only in the forward recursions of the APTD. We will show that this further reduces the proposed APTD's decoding complexity and its interleaving complexity, albeit at the cost of requiring slightly more decoding iterations to maintain the same BLER performance.

Chapter 3 also targets the latency requirements of URLLC but extends the turbo decoding work of Chapter 2 by also exploiting the frequency diversity. More specifically, the turbo decoding algorithm is jointly designed with an OFDM demodulation scheme, so that the physical layer latency can be significantly further reduced. In Chapter 3, a new architecture, in which the physical layer receiver components are operated concurrently, rather than consecutively, is proposed to potentially facilitate sub-microsecond physical layer latencies in the case of low-latency capital market trading. The novel contributions of Chapter 3 are as follows.

- We propose a novel cumulative FFT technique, which is processed incrementally and concurrently with the FPTD of [24], throughout the process of receiving a single OFDM symbol. Since the information carried by each turbo encoded bit is spread throughout the duration of the OFDM symbol, the proposed concurrent FFT can obtain some information about each bit as soon as the reception of the OFDM symbol begins, allowing turbo decoding to start immediately. As more and more of the OFDM symbol is received with passing time, the cumulative FFT can obtain more and more information about the turbo encoded bits, which can be fed into the concurrent turbo decoding process.
- We show that if the turbo decoder can complete a sufficient number of iterations within the duration of the OFDM symbol, then it can achieve the same error correction performance as if the turbo decoding process had only began after the reception of the OFDM symbol had been completed.
- In the case of the URLLC communications [6, 54, 55], the proposed approach can reduce the associated latency from 210 μs to 70 μs, which is far less than the 100 μs latency target of the Tactile Internet [56]. This leaves 30 μs for propagation and for the remaining, lower-complexity physical layer components, including channel estimation, Multiple-Input and Multiple-Output (MIMO) detection and transmitter processing.

Later in Chapter 4, we expand our scope to a less mature but promising channel code, the polar code, and aim for improving the error correction performance for satisfying the URLLC requirements highlighted in Figure 1.1. In this chapter, a tutorial of the uplink polar encoding and decoding process is detailed under the 5G NR standard and several enhancements are proposed to further increase the error correction performance of Log-SCS polar decoder of [51]. The novel contributions of Chapter 4 are as follows.

- We apply the Log-SCS algorithm to the polar code of the Physical Uplink Control Channel (PUCCH) and Physical Uplink Shared Channel (PUSCH) of the recently standardised URLLC version of 3GPP NR [57]. We provide a tutorial on how the Log-SCS algorithm may be obtained by transforming the computations of the SCS algorithm into the logarithmic domain, such that it can operate on the basis of LLRs.
- We provide a tutorial on how the Log-SCS algorithm improves on the SCS by considering the frozen bits, when determining the most likely sequence of information bits, which improves the error correction performance and reduces the decoding complexity.
- During the exploitation of the Cyclic Redundancy Check (CRC) codes to improve the error correction performance, we propose a novel technique for limiting the

number of CRC checks performed in stack decoding, in order to maintain a consistent error detection performance. In this application, we demonstrate that the application of the Log-SCS decoder to the 3GPP NR polar code of PUCCH and PUSCH offers as much as 0.5 dB improved error correction performance, compared to the previous SCS algorithm using the same stack size of $S = 8$.

- We propose a pair of novel techniques for further improving the performance of the Log-SCS polar decoder. We show that across the full range of block lengths supported by NR PUCCH and PUSCH, the proposed $S = 128$ Improved Log-SCS decoder achieves a similar error correction capability as the $L = 128$ Log-SCL decoder. This is achieved despite dramatically reducing its complexity by up to seven times compared to a $L = 8$ Log-SCL decoder and without increasing its memory requirement. Owing to this, the proposed Improved Log-SCS decoder offers practical ultra-reliable error correction within as little as 0.5 dB of the channel capacity bound. Hence, it is particularly well-suited to the URLLC mode of 3GPP NR.

Chapter 5 proposes and characterises a the software implementation of the Improved Log-SCS decoder of Chapter 4, with a focus on the latency improvement and complexity reduction. The novel contributions of Chapter 5 are as follows.

- We propose a novel CRC Aided (CA) fast Log-SCS decoder that employs several techniques that were previously considered by the fast Simplified Successive Cancellation List (SSCL) decoder [58], attaining a decoding latency level that is lower than the SOTA fast SCL polar decoders of [58].
- Each LLR that is input to the decoder is quantised using 32 bits, enabling the fixed-point implementation. In addition, a simplified path-metric computation of the rate-0, rate-1 and repetition subgraphs is applied in the proposed fast Log-SCS decoder which reduces the decoding complexity by 50% on average.
- The software implementation on x86 CPUs with Single Instruction Multiple Data (SIMD) instructions is demonstrated for the first time. In contrast to [59] which exploits the inter-block parallelism of the polar codes, the fast Log-SCS decoder considers only the intra-block parallelism. This guarantees the low-latency implementation in the Software-Defined Radio (SDR) systems. By implementing the 32-bit fast Log-SCS polar decoder into the x86 processors with 512-bit Advanced Vector Extensions-512 (AVX-512) SIMD instructions, a maximum parallelism degree of 16 may be achieved, and a 50% improvement in latency may be attained.

Finally, Chapter 6 summarises the main conclusions of the thesis and suggests potential research interest in channel coding schemes for URLLC.

Chapter 2

Arbitrarily Parallel Turbo Decoder

2.1 Introduction

As introduced in Section 1.1, the Fifth Generation (5G) Ultra-Reliable Low Latency Communication (URLLC) service targets a Block Error Rate (BLER) below 10^{-5} with an end-to-end latency within 1 ms. The initial roll-out of 5G will be based on the 3rd Generation Partnership Project (3GPP) non-standalone New Radio (NR), which is built upon a foundation offered by 3GPP Long Term Evolution (LTE) [6]. Therefore, the 3GPP is currently standardising a URLLC mode as part of the 5G development [10, 13–15]. Several enhancements have been proposed in LTE Release 15, aiming for meeting the latency and reliability requirements [13]. To be more specific, LTE Release 15 has introduced the shortened Transmission Time Interval (sTTI) technique, which requires a seven-fold signal processing time reduction at the User Equipment (UE) and evolved Node B (eNB) basestation, namely a reduction from 3 ms in Release 14, to 0.43 ms in Release 15 [16, 17, 57]. Within this 0.43 ms, several signal processing tasks must be completed, including receive buffering, Fast Fourier Transform (FFT), channel decoding, Inverse Fast Fourier Transform (IFFT) and transmit power control [14, 15]. In order for an eNB to support the processing of multiple users' transmissions within this processing time without employing a unique, user-specific set of signal processing hardware per user, each of these tasks must be completed in much less than 0.43 ms.

Owing to their near-capacity error-correction performance, turbo codes [2–4] have been widely adopted in the LTE standard for mobile communication [6]. Therefore, in order to achieve the LTE URLLC mode, the implementation of turbo codes requires much less latency in the decoding process than that of the existing decoding approaches in the literature. It is reasonable to assume that a 7-fold reduction from the 52 μs latency of the commercial implementations of 3GPP Release 14 [60] is required, giving a specification of 7.4 μs for 3GPP Release 15. This significant reduction in the turbo decoding latency requirement motivates the design of new low-latency turbo decoding techniques.

The latency of a turbo decoder is dictated by the data dependencies imposed by the Logarithmic Bahl-Cocke-Jelinek-Raviv (Log-BCJR) algorithm [19], which is typically employed for the iterative decoding of turbo codes [2–4]. More specifically, a turbo decoder comprises an upper-branch and a lower-branch convolutional decoder, which may operate alternately using the Log-BCJR to generate extrinsic information exchanged via an interleaver. However, data dependencies of the Log-BCJR algorithm require the processing to be performed one step at a time, using forward and backward recursions alternately along the entire length of the interleaved information block. This serial nature of the Log-BCJR algorithm imposes a bottleneck upon the throughput and results in processing latencies of hundreds of micro seconds of the turbo decoding process [19], unless parallel processing techniques are employed.

Several sophisticated approaches have been proposed for improving both the throughput and latency of the Log-BCJR turbo decoder. For example, the forward and backward recursions of the Log-BCJR can be completed simultaneously [22], rather than one after the other, hence doubling the throughput and halving the latency. Furthermore, the Radix-4 transform of [20, 21] allows two steps of the Log-BCJR algorithm's forward and backward recursions to be completed at a time, therefore further doubling the throughput and halving the latency, albeit at the cost of significantly increasing the hardware resource requirements. Meanwhile, the non-sliding window based technique of [23] decomposes each information block of N bits into P number of windows, which can be processed simultaneously using separate parallel processors. However, the combination of these techniques requires as many as $4P$ extrinsic values to be interleaved at a time. The interleaver has to simultaneously read and write these extrinsic values from and to memories associated with the parallel processors, causing contentions when attempting to make more than one access to a particular memory at any instant. This contention problem is solved in the LTE standards by the employment of the quadratic permutation polynomial interleavers of [52, 53], which inherently avoid contention provided that the P number of windows in each of the upper and lower decoder is an integer factor of the information block length N, hence ensuring that each window is composed of an equal number of N/P bits. Motivated by this, conventional implementations of the LTE turbo decoder [60] typically employ a parallelism of $P = 8$, since this is the Greatest Common Divisor (GCD) of the LTE turbo code's 188 supported block lengths N, which are in the range 40 to 6144 bits. It is using the combination of techniques described above, that the commercial Field-Programmable Gate Array (FPGA) implementation of the LTE turbo decoder of [60] achieves processing latencies of up to 52 μs.

In order to achieve the 7-fold improvement to turbo decoding latency required for URLLC, the parallelism may be increased to $P = 64$, as in the State-of-the-art (SOTA) turbo decoder of [23]. However, this decoder is only able to fully exploit this parallelism for the longest block lengths and disables up to 56 of the parallel processors at the short block lengths that are common in URLLC applications, leading to poor hardware efficiency.

Recently, a Fully-Parallel Turbo Decoder (FPTD) has been proposed for significantly increasing the grade of parallelism in the LTE turbo decoder to $p = N$, hence reducing the latency [24]. The employment of odd-even interleavers [25, 26] in LTE allows the FPTD to alternate between processing all odd-indexed bits in the upper decoder at the same time as all even-indexed bits in the lower decoder, and vice-versa, hence completing each decoding iteration in two clock cycles. However, the hardware requirement of the FPTD is dictated by the longest supported block length, leading to poor hardware efficiency at short block lengths.

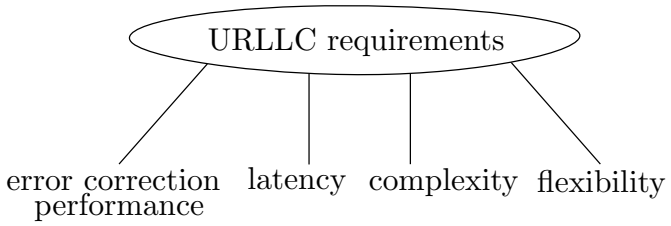


FIGURE 2.1: The contribution of Chapter 2.

The low hardware efficiency of SOTA non-sliding window based techniques [23] and the FPTD [24] motivates the novel Arbitrarily Parallel Turbo Decoder (APTD) concept we propose in this chapter for achieving both high processing throughputs and low processing latencies, while maintaining a high grade of hardware efficiency flexibility across all block lengths, as highlighted in Figure 2.1. Our APTD owns the following properties.

- First, our APTD algorithm allows the parallel operation of an arbitrary number P of processors, which is not limited to an integer factor of N, nor to N itself. The first of two versions of the proposed APTD decomposes the block into the highest possible number of windows that can support equal window lengths, where some processors are disabled as in the non-sliding window based and FPTD algorithms, respectively, when processing a block length of N. For many block lengths, this facilitates a grade of parallelism in excess of 64, which is the highest achieved previously in the literature owing to the requirement for P having to be a common divisor of many supported block lengths N. In this first version of the APTD, the processors operate on the basis of windows of equal length, benefitting from the contention-free property of the LTE quadratic permutation polynomial interleaver. However, in a second version of the APTD, all processors are activated for all but the shortest of block lengths. Here, the different processors operate on windows of slightly different lengths for block lengths N. This breaks the contention-free property of the LTE quadratic permutation polynomial interleaver [52, 53], but we show that contention can be avoided by rescheduling the interleaver operation independently from the generation of extrinsic information, hence achieving even higher throughputs.

- Furthermore, rather than operating the upper and lower decoder alternately like in conventional turbo decoding algorithms, the arbitrarily parallel turbo decoding algorithm employs odd-even operation, in a similar fashion to the FPTD. More specifically, the APTD alternates between processing the odd-indexed windows of the upper decoder at the same time as the even-indexed windows of the lower decoder, and vice versa. We will show that for short block lengths, this odd-even operation gives superior BLER performance compared to the upper-lower operation.

- As a further contribution, we conceive and compare two techniques for providing systematic information to the APTD, namely the interleaved, and the non-interleaved systematic approaches. In the interleaved systematic approach, the systematic information is entered into the upper decoder, but additionally it is also interleaved and entered to the lower decoder. By contrast, in the non-interleaved systematic approach, the systematic information is only entered directly into the upper decoder, but it is also added into the upper decoder's extrinsic information, which is then interleaved and entered into the lower decoder. The benefit of the latter is that this reduces the number of interleavers that are required at the cost of a slightly degraded BLER performance.

- Finally, while conventional turbo decoders compute their extrinsic information in both the forward and backward recursions of the Log-BCJR algorithm in the proposed design, the extrinsic information is calculated only in the forward recursions of the APTD. We will show that this further reduces the proposed APTD's decoding complexity and its interleaving complexity, albeit at the cost of requiring slightly more decoding iterations to maintain the same BLER performance.

When combining the proposed techniques described above, the proposed APTD achieves superior latency, throughput and computational efficiency than the SOTA LTE turbo decoder at all block lengths, but particularly at the short block lengths that are typically used in URLLC approaches. For example, at a block length of $N = 504$ bits, the proposed APTD achieves an BLER of 10^{-5} at the same E_b/N_0 as $I = 8$ iterations of a conventional turbo decoder, but with a computational efficiency that is 6 times higher than that of the SOTA turbo decoder, while achieving a latency and throughput that are 0.7 and 1.4 times those of the SOTA decoder, respectively. Note however that this is achieved at the cost of increasing the computational complexity by 2.3 times compared to the SOTA decoder of $N = 504$.

The proposed APTD algorithm offers particular benefits at short block lengths, which is also the particular focus of the URLLC service. This is because the conventional approach to parallel processing is limited by the greatest common divisor of the Quadratic Permutation Polynomial (QPP) or Almost Regular Permutation (ARP) interleaver lengths, which tends to be low for short block lengths. For example, the information block lengths

supported by the LTE turbo code in the range of 40 to 512 bits are all multiples of 8 bits, limiting the degree of parallel processing to 8 in conventional implementations. By contrast, the proposed APTD can usefully apply significantly higher degrees of parallel processing, leading to significant improvements in throughput and latency.

Furthermore, in [61], a 6144-bit FPTD was found to occupy a chip area of 109 mm^2, when using Taiwan Semiconductor Manufacturing Company (TSMC) 65 nm process technology. However, this design has only limited flexibility to support shorter block lengths and it may be expected that a significantly greater chip area would be required to support all 188 block lengths of the LTE turbo code. Hence, the chip area requirements of the FPTD may be deemed excessive for LTE applications. In contrast to this, the proposed APTD allows the degree of parallelism to be arbitrarily flexible between fully-serial and fully-parallel turbo decoding. In this way, the parallelism of the proposed APTD may be carefully selected in order to meet the strict latency and throughput requirements of URLLC LTE, but with minimum chip area.

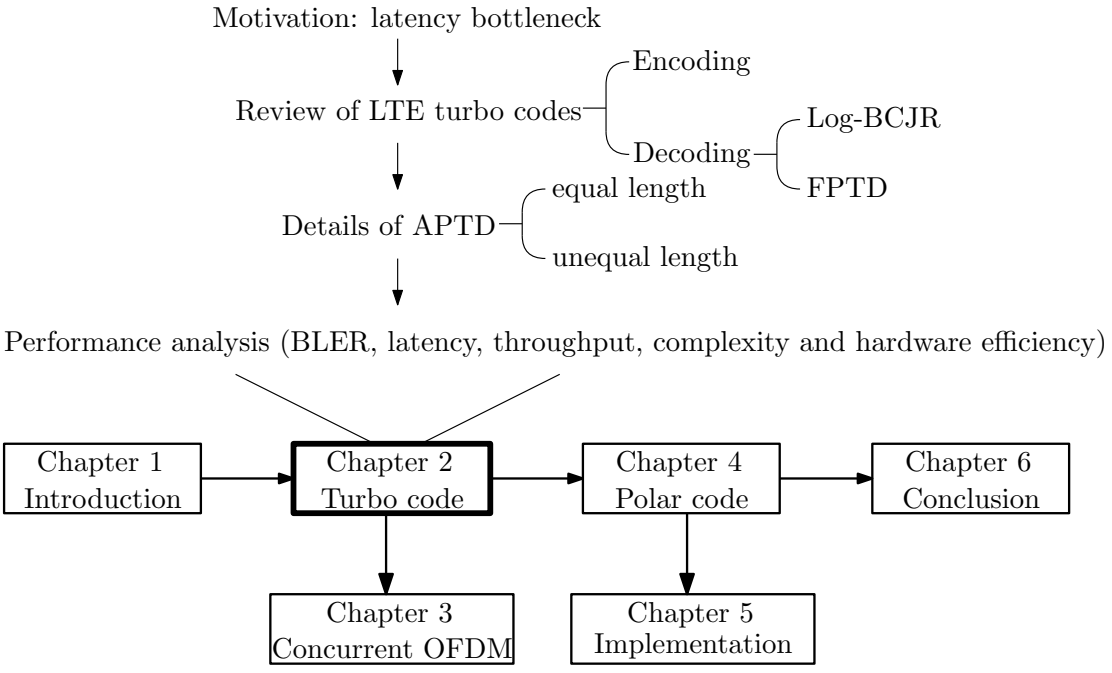


FIGURE 2.2: The development of Chapter 2.

As shown in Figure 2.2, this chapter is structured as follows. Section 2.2 gives a brief overview of LTE turbo decoding, including both the SOTA turbo decoder, also known as a non-sliding window decoder [23], and of the FPTD. Following this, both versions of the APTD are proposed in Section 2.3. Section 2.4 characterises the BLER performance associated with the techniques employed by the APTD, both individually and in combination. Then the SOTA LTE turbo decoder, the FPTD and the proposed APTD are compared in terms of their latency, throughput, complexity and hardware resource

requirement in Section 2.5. Finally, our conclusions and future work ideas are discussed in Section 2.6.

2.2 LTE turbo codes overview

This section provides an overview of LTE turbo encoding and decoding process, as well as defining the notation employed in the following sections. Sections 2.2.1 and 2.2.2.1 describe the encoding and decoding process of the SOTA LTE turbo codes, while the FTPD of [24] is highlighted in Section 2.2.2.2.

2.2.1 Turbo Encoder

In an LTE transmitter, each block of information bits has one of 188 legitimate lengths N in the range of 40 to 6144 bits, which are turbo encoded before being modulated and transmitted over the wireless channel. To be more specific, a message block $\mathbf{b}_1^{\mathrm{u}} = [b_{1,k}^{\mathrm{u}}]_{k=1}^{N}$ comprising N bits, where $b_{1,k}$ is a binary value, $b_{1,k}^{u} \in \{0, 1\}$ is first encoded by the upper convolutional encoder in Figure 2.3, generating two N-bit encoded blocks, referred to as the upper *parity block* $\mathbf{b}_2^{\mathrm{u}} = [b_{2,k}^{\mathrm{u}}]_{k=1}^{N}$ and an upper *systematic block* $\mathbf{b}_3^{\mathrm{u}} = [b_{3,k}^{\mathrm{u}}]_{k=1}^{N}$, respectively. Furthermore, the message block $\mathbf{b}_1^{\mathrm{u}}$ is interleaved by an odd-even interleaver, obtaining the N-bit interleaved message block $\mathbf{b}_1^{\mathrm{l}} = [b_{1,k}^{\mathrm{l}}]_{k=1}^{N}$, as shown in Figure 2.3. The interleaved message block $\mathbf{b}_1^{\mathrm{l}}$ is then forwarded to a lower convolutional encoder, which generates another N-bit encoded block, referred to as the lower *parity block* $\mathbf{b}_2^{\mathrm{l}} = [b_{2,k}^{\mathrm{l}}]_{k=2}^{N}$. Note that the superscripts 'u' and 'l' imply the upper and lower convolutional encoders, respectively. However, these two superscripts will be omitted in the following discussions when the corresponding analysis applies equally to both upper and lower encoders. It is notable that the N bits of the message block $\mathbf{b}_1^{\mathrm{u}}$ are encoded into three encoded blocks through the turbo encoder, so the total number of encoded bits is $3N$, resulting in a coding rate of $R = 1/3$.

Both the upper and lower convolutional encoders operate in the same manner, based on a state transition diagram. We illustrate the operations of the pair of convolutional encoders using the example of the LTE state transition diagram, which has $M = 8$ states, and $K = 2$ transitions per state, as shown in Figure 2.4. At the start of the encoding process, the convolutional encoder begins with the initial state $S_0 = 0$ and transits successively into the subsequent states $S_k \in \{0, 1, 2, ..., M-1\}$ on the basis of the successive message bits $b_{1,k}$. Since the message bits are binary values, $b_{1,k} \in \{0, 1\}$, there are $K = 2$ possible values for the current state S_k, which can be reached from the selected previous state S_{k-1}. In the case of the LTE transition diagram of Figure 2.4, the previous state $S_{k-1} = 0$ indicates the possible subsequent state S_k is either 0 or 4. We use the notation $c(0, 0) = 1$ and $c(0, 4) = 1$ in this example, where $c(S_{k-1}, S_k) = 1$

means that the subsequent state S_k can be transited from the previous state S_{k-1}, while $c(S_{k-1}, S_k) = 0$ indicates an illegitimate transition pair. For $K = 2$ possible values, the value of S_k is chosen such that $b_1(S_{k-1}, S_k) = b_{1,k}$. Take the example in Figure 2.4, where $b_{1,k} = 1$ corresponds to a transition from $S_{k-1} = 0$ to $S_k = 4$. Similarly, the binary bits selected for the corresponding bits in parity block $\mathbf{b}_2$ and systematic block $\mathbf{b}_3$, based on $b_{2,k} = b_2(S_{k-1}, S_k)$ and $b_{3,k} = b_3(S_{k-1}, S_k)$, respectively. For LTE turbo encoder, $S_{k-1} = 0$ and $S_k = 0$ gives $b_{2,k} = 0$ and $b_{3,k} = 0$ whereas $S_{k-1} = 0$ and $S_k = 4$ gives $b_{2,k} = 1$ and $b_{3,k} = 1$.

Following turbo encoding, the encoded blocks are modulated into a wireless channel and transmitted to the receiver.

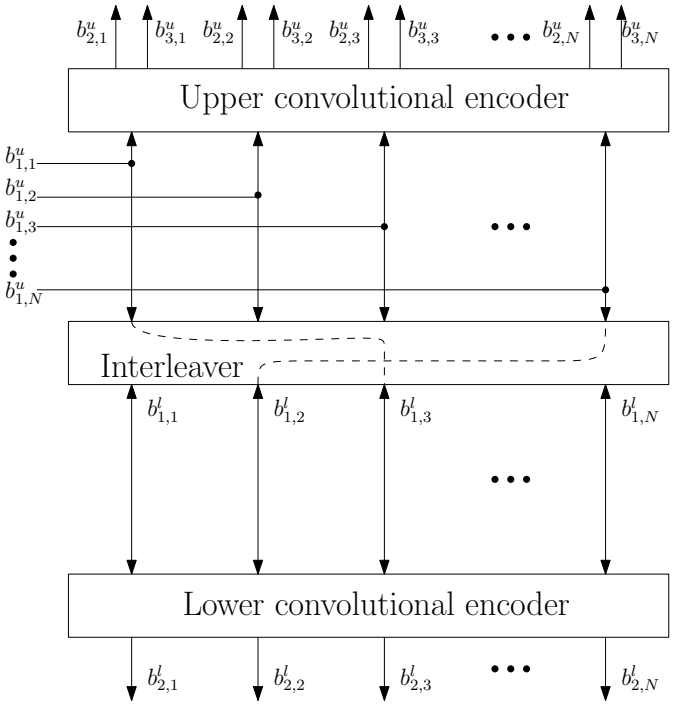


FIGURE 2.3: Schematic of a Turbo Encoder

2.2.2 Turbo Decoders

2.2.2.1 State-of-the-art LTE turbo decoder

After demodulation in the receiver, soft information pertaining for the turbo encoded bits is provided to the turbo decoder in the form of Logarithmic-Likelihood Ratios (LLRs). Here, we define the LLR $\bar{b}$ pertaining to a bit $b \in \{0, 1\}$ as

$$\bar{b} = \ln \frac{\Pr(b = 1|\mathbf{y})}{\Pr(b = 0|\mathbf{y})}, \tag{2.1}$$

where $\mathbf{y}$ is the received signal vector.

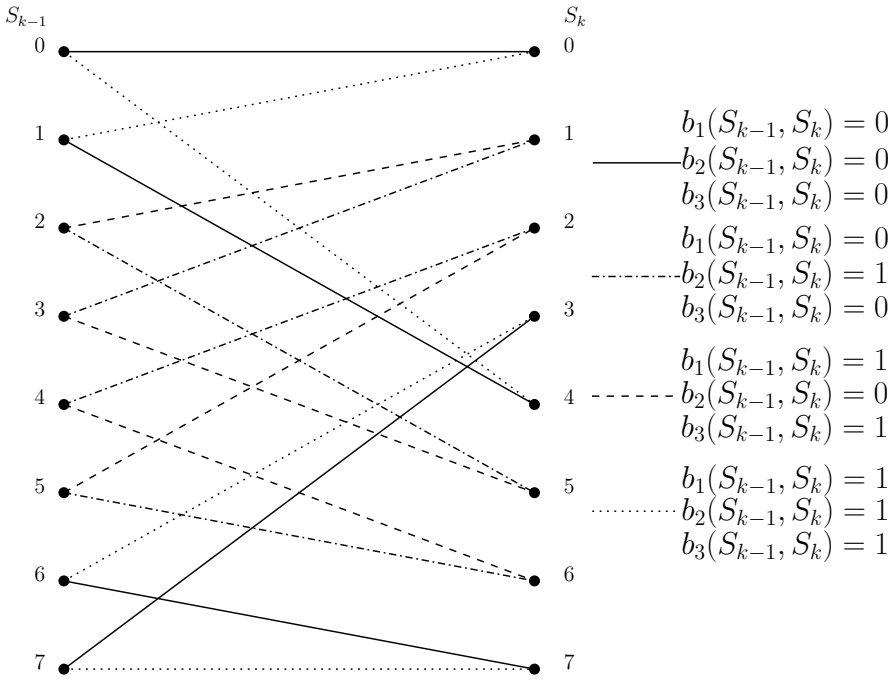


FIGURE 2.4: State transition diagram of the LTE turbo code

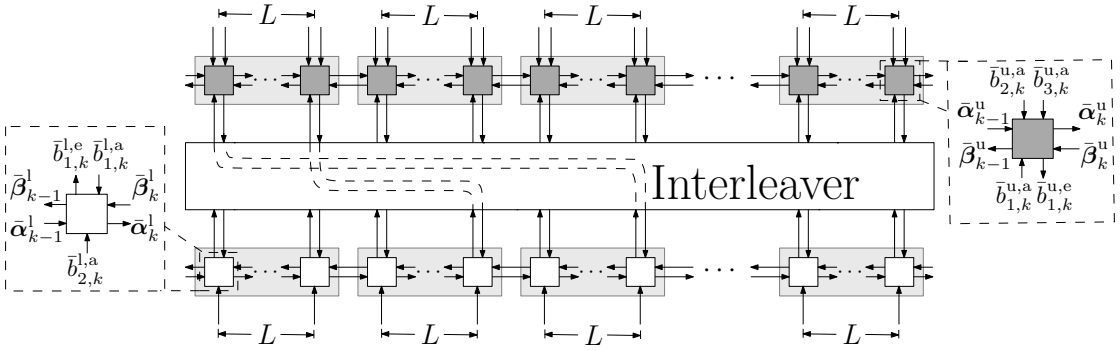


FIGURE 2.5: Schematic of the State-of-the-art (SOTA) turbo decoder.

As depicted in Figure 2.5, the upper and lower decoder each comprises a row of N algorithmic units. Each units in the upper decoder is associated with the corresponding one of the parity *a priori* LLRs $[\bar{b}^{\mathrm{u,a}}_{2,k}]^N_{k=1}$ and the corresponding one of the systematic *a priori* LLRs $[\bar{b}^{\mathrm{u,a}}_{3,k}]^N_{k=1}$, which are provided by the demodulator. Likewise, the demodulator provides each unit in the lower decoder with the corresponding one of the parity *a priori* LLRs $[\bar{b}^{\mathrm{l,a}}_{2,k}]^N_{k=1}$. Note that the demodulator also provides LLRs pertaining to twelve termination bits [62], although we do not discuss these further for the sake of simplicity.

The upper and lower decoders operate alternately, where each algorithmic unit generates the corresponding one of the extrinsic LLRs $[\bar{b}^{\mathrm{u,e}}_{1,k}]^N_{k=1}$ or $[\bar{b}^{\mathrm{l,e}}_{1,k}]^N_{k=1}$, which are interleaved to provide *a priori* LLRs $[\bar{b}^{\mathrm{u,a}}_{1,k}]^N_{k=1}$ or $[\bar{b}^{\mathrm{l,a}}_{1,k}]^N_{k=1}$ for the next operation of the other decoder. In addition to the LLRs described above, each processor is provided with a vector of forward state metrics $\bar{\boldsymbol{\alpha}}^{\mathrm{u}}_{k-1}$ or $\bar{\boldsymbol{\alpha}}^{\mathrm{l}}_{k-1}$, expressed in (2.2) as well as a vector of backward

state metrics $\bar{\boldsymbol{\beta}}_k^{\mathrm{u}}$ or $\bar{\boldsymbol{\beta}}_k^{\mathrm{l}}$, expressed in (2.3), which are used for initiating forward and backward processing recursions, respectively.

$$\bar{\alpha}_k(S_k) = \max_{\{S_{k-1}|c(S_{k-1},S_k)=1\}}{}^{*} \left[\bar{\gamma}_k(S_{k-1}, S_k) + \bar{\alpha}_{k-1}(S_{k-1})\right], \tag{2.2}$$

$$\bar{\beta}_{k-1}(S_{k-1}) = \max_{\{S_k|c(S_{k-1},S_k)=1\}}{}^{*} \left[\bar{\gamma}_k(S_{k-1}, S_k) + \bar{\beta}_k(S_k)\right]. \tag{2.3}$$

Here, the $\max^*$ operation may be approximated by the Jacobian approximation expressed as

$$\begin{aligned} \max{}^{*}(a,b) &= \max(a,b) + \log[1+\exp^{-|a+b|}] && (2.4)\\ &\approx \max(a,b), && (2.5) \end{aligned}$$

and the $\bar{\gamma}$ values in (2.2) and (2.3) are *a priori* transition metrics that can be obtained by (2.6).

$$\bar{\gamma}_k(S_{k-1}, S_k) = \sum_{j=1}^{L} \left[b_j(S_{k-1}, S_k) \cdot \bar{b}_{j,k}^{\mathrm{a}}\right]. \tag{2.6}$$

All algorithmic units in Figure 2.5 operate in an identical manner. The SOTA turbo decoder performs this processing by activating P parallel processors, where each alternates between processing the corresponding window of $L = N/P$ algorithmic units in the upper and lower decoder, as shown in Figure 2.5. For example, the first processor performs the processing for the first L algorithmic units of the upper decoder, then performs the processing for the first L algorithmic units of the lower decoder, before repeating the process in each successive decoding iteration. Here, we define the time required for one of the algorithmic units as one clock cycle. Although the SOTA LTE turbo decoder comprises $p = 64$ parallel processors in total, different numbers P of these p processors are activated for different block lengths N, according to [23].

$$P = \begin{cases} 8, & N \in [40, 48, 56, ..., 504]; \\ 16, & N \in [512, 528, 544, ..., 1008]; \\ 32, & N \in [1024, 1056, 1088, ..., 2016]; \\ 64, & N \in [2048, 2112, 2176, ..., 6144]. \end{cases} \tag{2.7}$$

Note that some of the $p = 64$ parallel processors are unused when shorter block lengths are decoded, in order to ensure that all windows have an equal length L. This is necessary for exploiting the contention-free property of the LTE interleaver [6,52]. More specifically, if all windows have an equal length, their processing can then be scheduled so that none of the parallel processor is provided with more than one *a priori* LLR at a time by the LTE interleaver. This allows the processors to have simple input/output interfaces.

The scheduling of these recursions over several successive clock cycles is depicted in Figure 2.6 for the non-sliding window technique of [23]. During the forward recursion, each successive algorithmic unit in a left-to-right ordering is processed in each successive clock cycle. Simultaneously, the backward recursion processes the successive algorithmic units in a right-to-left ordering in the successive clock cycles. Each successive algorithmic unit passes a vector of state metrics $\bar{\boldsymbol{\alpha}}_k^{\mathrm{u}}$, $\bar{\boldsymbol{\alpha}}_k^{\mathrm{l}}$, $\bar{\boldsymbol{\beta}}_{k-1}^{\mathrm{u}}$ or $\bar{\boldsymbol{\beta}}_{k-1}^{\mathrm{l}}$ values to the next algorithmic unit in the forward or backward recursion, respectively. The requirement to exchange these state metrics between successive algorithmic units imposes data dependencies, which requires the forward and backward recursions. Furthermore, once the two recursions have crossed over, as shown in Figure 2.6, the algorithmic units generate the extrinsic LLRs $\bar{b}_{j,k}^{\mathrm{e}}$ mentioned above, based on both the forward and the backward state metrics, as expressed in (2.8).

$$\bar{b}_{j,k}^{\mathrm{e}} = \left[\max_{\{(S_{k-1},S_k)|b_j(S_{k-1},S_k)=1\}}^{*} \left[\bar{\delta}_k(S_{k-1},S_k)\right] \right] - \left[\max_{\{(S_{k-1},S_k)|b_j(S_{k-1},S_k)=0\}}^{*} \left[\bar{\delta}_k(S_{k-1},S_k)\right] \right] - \bar{b}_{j,k}^{\mathrm{a}}, \tag{2.8}$$

where

$$\bar{\delta}_k(S_{k-1},S_k) = \bar{\gamma}_k(S_{k-1},S_k) + \bar{\alpha}_{k-1}(S_{k-1}) + \bar{\beta}_k(S_k). \tag{2.9}$$

Each iteration comprises a total of $2N/P$ clock cycles, with N/P clock cycles used by each of the upper and lower decoders. Here, $I = 8$ iterations may be required in order to achieve iterative decoding convergence towards the best possible BLER, corresponding to hundreds or thousands of clock cycles.

The Radix-4 [20, 21] further improves both the throughput and latency of the SOTA LTE turbo decoder, by merging the trellis stages within each pair of adjacent algorithmic units and processing them at the same time, as shown in Figure 2.7 (b). Therefore, in contrast to the Radix-2 operation depicted in Figure 2.7 (a), where *two* extrinsic LLRs are calculated simultaneously once the forward and backward recursions have crossed over, the non-sliding window technique calculates *four* LLRs at a time once the forward and backward recursions have crossed over, when we employ Radix-4 technique. In this way, the Radix-4 technique further doubles the throughput and halves the latency, although at the cost of significantly increasing the hardware resource requirement.

As shown in Figure 2.8 (a), the extrinsic LLRs $\bar{b}_{1,k}^{\mathrm{u,e}}$ and $\bar{b}_{1,k}^{\mathrm{l,e}}$ may be scaled in order to improve the BLER when employing the Max-Log-MAP algorithm [61,63]. Here, typically a scaling factor of $f = 0.75$ is selected, owing to its ease of implementation when using fixed point numbers to represent the LLRs. Figure 2.8 (a) also shows that the upper decoder can benefit from the systematic LLRs $\bar{b}_{3,k}^{\mathrm{u,a}}$ by adding them into the *a priori*

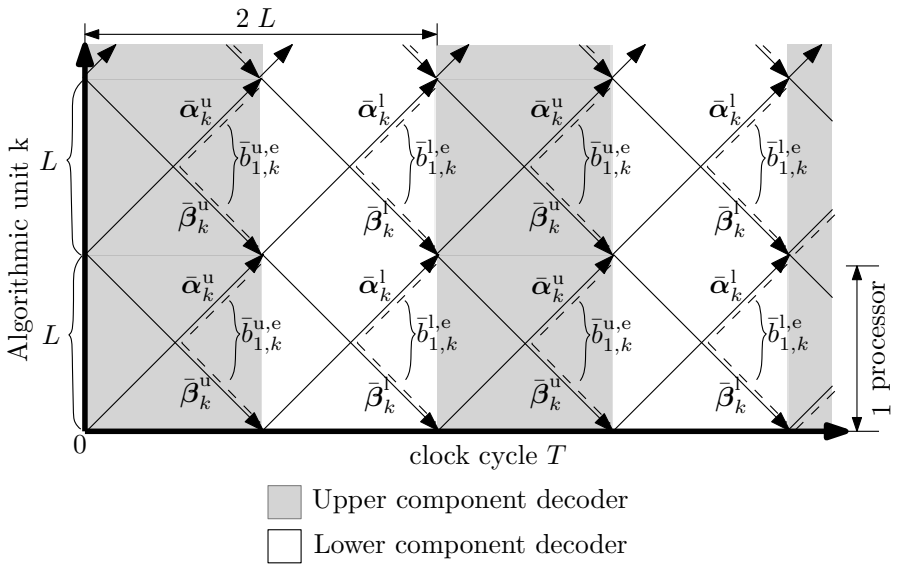


FIGURE 2.6: Schedule of the forward and backward recursions for the non-sliding window technique, which uses equal window lengths, upper-lower processing Radix-4 operation, and calculates four extrinsic LLRs at a time once the forward and backward recursions have crossed over.

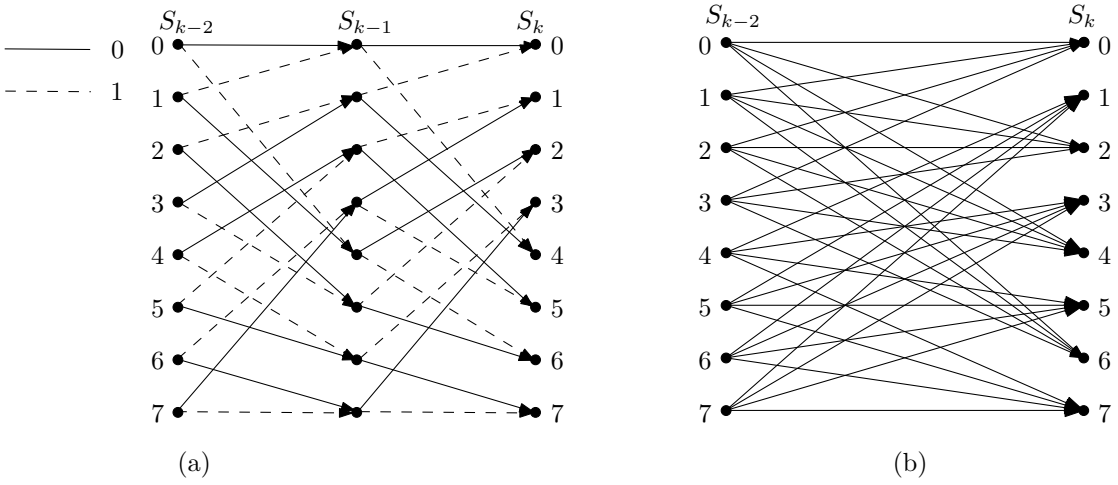


FIGURE 2.7: Trellis diagram of (a) Radix-2 computation, and (b) Radix-4 computation.

LLRs $\bar{b}^{\mathrm{u,a}}_{1,k}$. A similar approach can be used for ensuring that the lower decoder can benefit from these systematic LLRs. However, since the upper decoder's operations are processed before the lower decoder in the SOTA turbo decoder, the systematic LLRs $\bar{b}^{\mathrm{u,a}}_{3,k}$ can be delivered to the lower decoder by adding them into the upper decoder's extrinsic LLRs $\bar{b}^{\mathrm{u,e}}_{1,k}$, which are interleaved to become the lower decoder's *a priori* LLRs $\bar{b}^{\mathrm{l,a}}_{1,k}$. This avoids the hardware requirement to employ a separate interleaver to interleave the systematic LLRs $\bar{b}^{\mathrm{u,e}}_{3,k}$, so that they can be added to the *a priori* LLRs of the lower decoder $\bar{b}^{\mathrm{l,a}}_{1,k}$, as shown in Figure 2.8 (b).

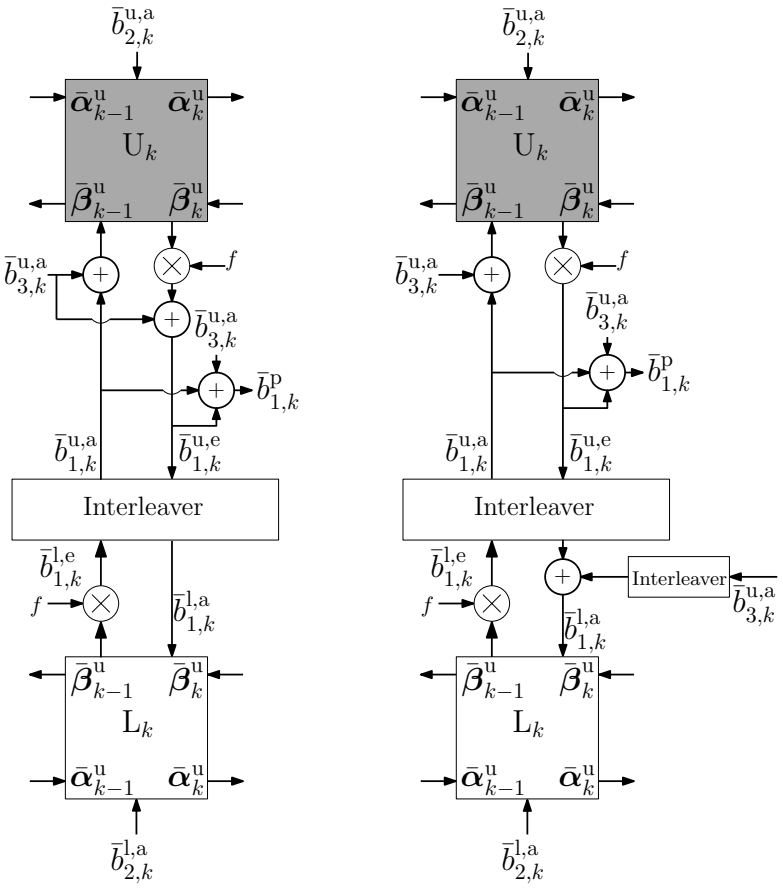


FIGURE 2.8: (a) Non-interleaved systematic LLRs. (b) Interleaved systematic LLRs.

2.2.2.2 Fully-parallel turbo decoder

In the SOTA LTE turbo decoder of [23], the data dependencies of the forward and backward recursions require the turbo-encoded bits to be processed serially, spread over numerous consecutive clock cycles. As a result, hundreds or thousands of clock cycles are required for completing the iterative decoding process, hence limiting the achievable processing throughput and latency. In order to address this problem, a FPTD algorithm was proposed in [24], which operates based on (2.10) to (2.14) and dramatically increases the grade of parallelism in the decoding process. The full parallelism is achieved by dispensing both with the recursions of the Log-BCJR algorithm and with the associated data dependencies, allowing a much higher number of parallel processors to be used. Indeed, the FPTD algorithm is capable of processing all LLRs corresponding to both the upper and lower decoders at the same time, as detailed below [24].

$$\gamma_k(s_{k-1}, s_k) = b_1(s_{k-1}, s_k) \cdot b^a_{1,k} + b_2(s_{k-1}, s_k) \cdot b^a_{2,k} + b_3(s_{k-1}, s_k) \cdot b^a_{3,k} \tag{2.10}$$

$$\alpha_k(s_k) = \max_{\{\text{all}s_{k-1}\}}^{*} \left[\gamma_k^t(s_{k-1}, s_k) + \alpha_{k-1}(s_{k-1})\right] \tag{2.11}$$

$$\beta_{k-1}(s_{k-1}) = \max_{\{\text{all}s_{k-1}\}}^{*} \left[\gamma_k^t(s_{k-1}, s_k) + \beta_k(s_k)\right] \tag{2.12}$$

$$\begin{aligned} b_{1,k}^e = &\left[\max_{\{(s',s)|b_1(s',s)=1\}}^{*} \left[\gamma_k(s', s) + \alpha_{k-1}(s') + \beta_k(s)\right]\right] - \\ &\left[\max_{\{(s',s)|b_1(s',s)=0\}}^{*} \left[\gamma_k(s', s) + \alpha_{k-1}(s') + \beta_k(s)\right]\right] - \\ &\left[b_{1,k}^a + b_{3,k}^a\right] \end{aligned} \tag{2.13}$$

$$\hat{b}_{1,k} = (b_{1,k}^a + b_{1,k}^e + b_{3,k}^a). \tag{2.14}$$

While a dedicated processor could be employed for each of the $2N$ algorithmic units, our previous work demonstrated that the same processing throughput and latency can be achieved using just N parallel processors. More specifically, in contrast to the alternated operation of the upper and lower decoders in the non-sliding window technique of the SOTA turbo decoder, the FPTD exploits the odd-even property of the LTE interleaver to enable an odd-even processing schedule for the algorithmic units. To be more specific, the LTE interleaver of all 188 supported block lengths only connects the algorithmic units of the upper decoder having an odd index k to algorithmic units in the lower decoder that also have an odd index k. Likewise, even-indexed algorithmic units in the upper decoder are only connected to algorithmic units with even indices in the lower decoder. As a result, the algorithmic units can be grouped into two sets, where no two units in the same set have connections to each other. To be more explicit, the first set consists of all the odd-indexed units in the upper decoder, along with all even-indexed units in the lower decoder. Likewise, the second set is composed of the remaining units, namely of the even-indexed units in the upper decoder, together with the odd-indexed units in the lower decoder. Accordingly, the exchange of extrinsic LLRs and state metrics among the whole set of $2N$ algorithmic units in the iterative decoding process can now be considered as an iterative exchange process between the two sets. This allows the processing of the $2N$ algorithmic units to be mapped onto N processors, which alternate between the processing of the two sets. In this odd-even FPTD operation, each iteration requires only two clock cycles, but more iterations are needed than by the conventional Log-BCJR decoders in order to achieve the same BLER. However, the total number of clock cycles required in order to achieve iterative decoding convergence decreases from the hundreds or thousands in the SOTA LTE turbo decoder to just tens of clock cycles.

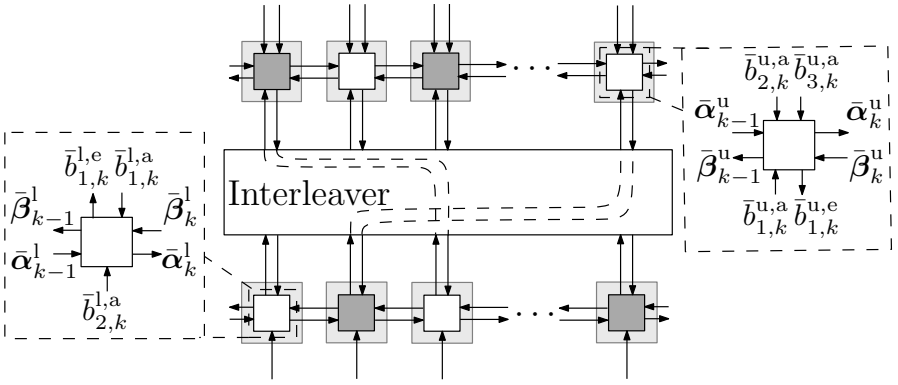


FIGURE 2.9: Schematic of the Fully-Parallel Turbo Decoder (FPTD).

2.3 Arbitrarily Parallel Turbo Decoder

As discussed in Section 2.2, both the SOTA turbo decoding algorithm and the FPTD algorithm have their own restrictions. To be more specific, a large number of processors remain idle when decoding short block lengths N in the SOTA turbo decoder. Meanwhile, a block having a length of N must be decoded using N numbers of FPTD processors, which limits flexibility. Motivated by this, we propose a novel APTD algorithm in this section, which can employ an arbitrary number p of processors and exploit them flexibly across all of the LTE interleaver lengths. Two versions of the APTD will be proposed separately in Section 2.3.1 and 2.3.2, respectively. The first version of the APTD decomposes the block into the highest number of windows that can support equal window lengths, exploiting the contention-free property of the LTE interleaver, as it will be detailed in Section 2.3.1. However, some idle processors still remain for some block lengths in this version. By contrast, the second version of Section 2.3.2 does not rely on the contention-free property of the LTE interleaver, allowing different windows to adopt different lengths. This enables all processors to be exploited for all but the shortest block lengths, which is achieved by decomposing the block into the same number of windows as there are processors.

2.3.1 APTD employing equal window lengths

The first version of the APTD employs an arbitrary number p of processors in total, but activates only a subset P of these depending on the block length N. When decoding blocks of length N, the number of activated processors P is given by the *greatest integer factor (gif)* of N that is also smaller than the number of processors p, which can be expressed as $P = gif(N, p)$. Figure 2.10 (a) illustrates the relationship between the number of activated processors P and the block lengths N when employing a total number of $p = 64$ or $p = 128$ processors, as compared to the SOTA turbo decoders and FPTD discussed above. Each processor alternates between processing a window from the

upper decoder and the corresponding window from the lower decoder. Likewise, Figure 2.10 (b) shows the relationship between the window length $L = N/P = N/gif(N,p)$ and the block length N, for $p = 64$ and $p = 128$. Note that for shorter block lengths $N \leq p$, the APTD operation becomes identical to that of the FPTD, where each window effectively has a length L of one bit.

Like the FPTD, the APTD relies on odd-even operation, as depicted in Figure 2.11. However, rather than alternating the operation of odd- and even-indexed algorithmic units, the odd-even operation is performed at the window level in the APTD algorithm. To be more specific, the odd-indexed windows in the upper decoder and the even-indexed windows in the lower decoder operate at the same time. This alternates with operating the even-indexed windows in the upper decoder at the same time as the odd-indexed windows in the lower decoder. A total number of $2L$ clock cycles have to be completed in one iteration. The forward state metrics $\bar{\boldsymbol{\alpha}}_k^{\mathrm{u}}$, $\bar{\boldsymbol{\alpha}}_k^{\mathrm{l}}$ and the backward state metrics $\bar{\boldsymbol{\beta}}_{k-1}^{\mathrm{u}}$, $\bar{\boldsymbol{\beta}}_{k-1}^{\mathrm{l}}$ are calculated according to the forward and backward recursions that are synchronised among all windows processed at the same time. Note that the odd-even operation ensures that the end of the recursions of the odd-indexed windows seamlessly leads to the beginning of the adjacent even-indexed windows, and vice versa. This allows information to more quickly propagate along the upper and lower decoder than in the upper-lower operation. Radix-4 operation [20, 21] is also employed in the proposed APTD, which further doubles the decoding throughput. However, since the Radix-4 operation processes a pair of trellis stages at a time, a special solution is required when the window length L is not an even integer. More specifically, the final algorithmic unit in each window may be processed using Radix-2 operation.

The APTD calculates two extrinsic LLRs at a time alongside the forward recursion, when using Radix-4 operation. During the first half of each forward recursion, the $\bar{\boldsymbol{\beta}}$ values calculated during the previous iteration are recalled and are used for calculating the extrinsic LLRs. Once the forward and backward recursions have crossed over in the second half of the recursions, the $\bar{\boldsymbol{\beta}}$ values calculated during the first half of the backward recursion are used. This is in contrast to the SOTA LTE turbo decoder, which calculates four LLRs at a time, during the second half of the forward and backward recursions once they have crossed over. Our approach requires only two extrinsic LLR calculators and two interleavers that are used all the time, rather than four LLR calculators and four interleavers that are only used in the second half of the recursions, which is less efficient.

Note that a particular extrinsic LLR in an odd-indexed window of one component decoder may become an *a priori* LLR for an even-indexed window of the other component decoder or vice versa in the proposed FPTD. Since these windows are processed at the same time in the FPTD, some interesting interactions may arise. In some cases, the extrinsic LLR may be generated and interleaved before it is used as an *a priori* LLR in the same iteration, depending on the position of these LLRs in the windows. This is advantageous compared to the classic upper-lower operation, which must always wait until the next

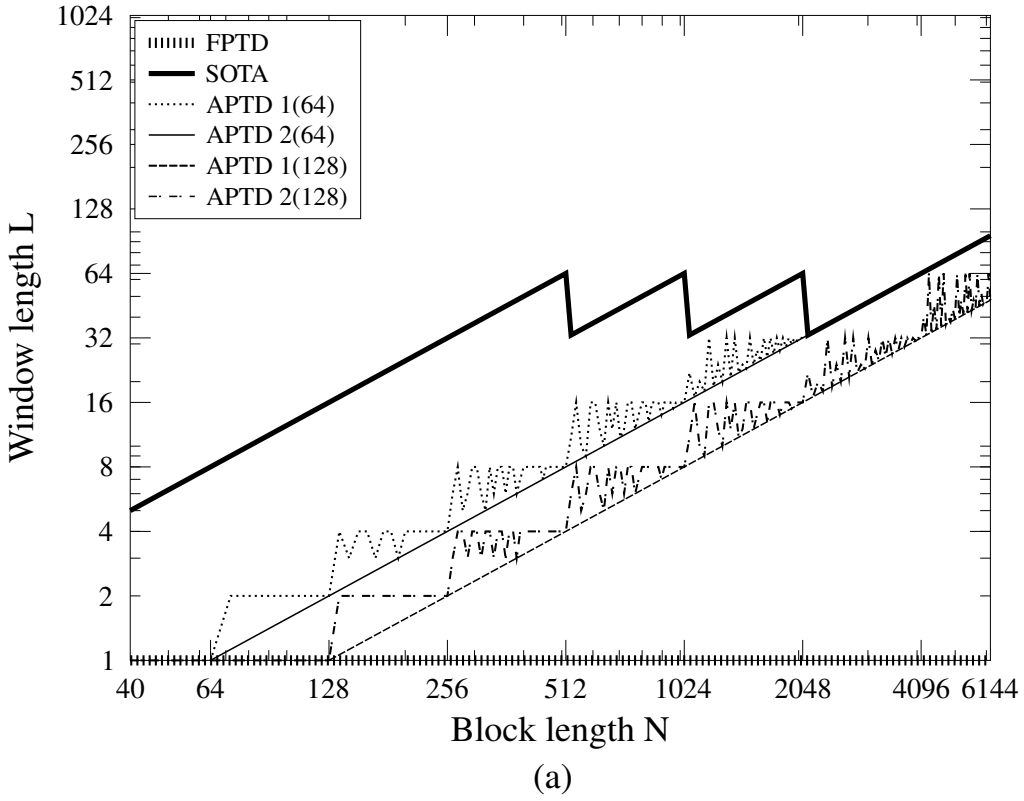


FIGURE 2.10: (a) The relationship between window length L and block length N, for various turbo decoders; (b) The relationship between the number of activated processors P and the block length N for various turbo decoders. Note that in the SOTA turbo decoder, FPTD and the first version of the APTD algorithms, all windows have the same length, since the number of activated processors P is chosen as an integer factor of N in these schemes. By contrast, some windows have a length $\lceil \frac{N}{P} \rceil$ and others have the length $\lfloor \frac{N}{P} \rfloor$ in the second version of APTD algorithm. In this case, the average window lengths is plotted.

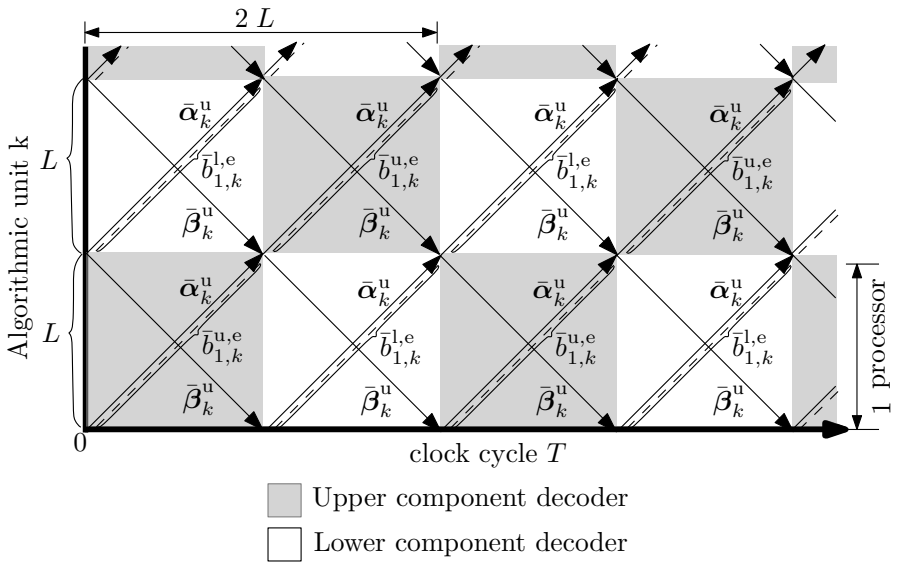


FIGURE 2.11: Schedule of the forward and backward recursions for the first version of the APTD, which uses equal windows lengths, Radix-4 operation, odd-even processing and calculates two extrinsic LLRs at a time alongside the forward recursion.

half iteration before an interleaved LLR can be exploited. Other times, however, the extrinsic LLR may be delivered too late to be used in the same iteration, but can be saved for use in the next iteration. This is a disadvantage compared to the upper-lower operation, since the delay before the interleaved LLR can be exploited is extracted in this case. Overall, the advantage of sometimes being able to exploit an interleaved LLR immediately and the disadvantage of sometimes having to wait until the next iteration may be expected to cancel out. However, the advantage of quicker $\bar{\boldsymbol{\alpha}}$ and $\bar{\boldsymbol{\beta}}$ propagation can be exploited in the APTD versions, as we will demonstrate for short blocks in Section 2.4.

In addition to the non-interleaved systematic approach discussed in Section 2.2.2.1, we also consider an interleaved systematic approach, as shown in Figure 2.8 (b). As described in Section 2.2.2.1, the non-interleaved systematic approach has the advantage of avoiding the requirement for a separate interleaver for the systematic LLRs $\bar{b}^{\mathrm{u,a}}_{3,k}$. In the upper-lower schedule, there is no disadvantage compared to the non-interleaved systematic selection, since the lower decoder is not activated until after the systematic LLRs $\bar{b}^{\mathrm{u,a}}_{3,k}$ are combined with the extrinsic LLRs $\bar{b}^{\mathrm{u,e}}_{1,k}$ and passed through the interleaver by the upper decoder. However, in the case of our odd-even schedule, some units in the lower decoder are activated in the first half of the first iteration, before the systematic LLRs can be passed through the interleaver by the upper decoder. This motivates the consideration of the interleaved systematic approach, which improves the BLER by providing systematic LLRs for these units in the lower decoder at the start of the iterative decoding process, albeit at the cost of requiring an additional interleaver.

2.3.2 APTD employing unequal window lengths

In a second version of the APTD algorithm, block lengths N that are not divisible by the number of processors p are handled in a different way. Rather than activating only a subset P of the processors, so that each process has an equal window length, this second version activates all processors for block lengths $N \geq p$, where some process windows having a slightly shorter length, while others process windows have a slightly longer window. More specifically, the first $[P - \mathrm{mod}(N, P)]$ windows have a length of $L = \lfloor N/P \rfloor$ trellis stages, while the remaining $\mathrm{mod}(N, P)$ windows have a length of $L + 1 = \lceil N/P \rceil$, as shown in Figure 2.12. Indeed, for block lengths N lower than the number of processors p, this version of the APTD also operates identically to the FPTD, as discussed in Section 2.3.1.

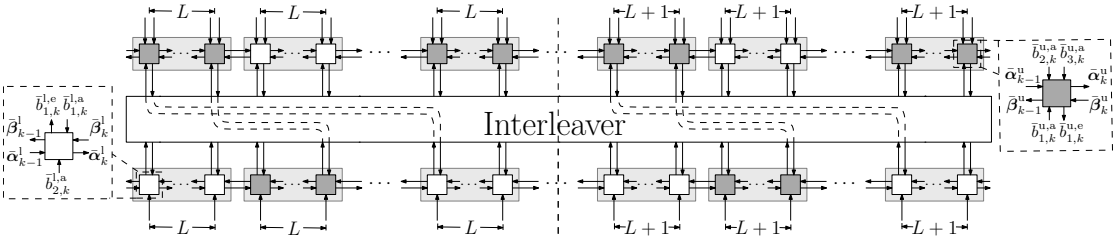


FIGURE 2.12: Schematic of the second version of the proposed Arbitrarily Parallel Turbo Decoder (APTD) with unequal window lengths.

Figure 2.13 illustrates the scheduling of the forward and backward recursions in the second version of the APTD. Each processor alternates between processing the corresponding window in the upper decoder and the corresponding window in the lower decoder. As in the first version of the APTD, the extrinsic LLRs are calculated alongside the forward recursions. Note that the windows having length L are grouped together and are scheduled independently of the windows having lengths $L + 1$, because they require different numbers of clock cycles to complete each iteration. As a result, the two groups of windows become de-synchronised as the decoding process proceeds. More specifically, the forward recursion of the last processor for the shorter windows does not seamlessly lead into the forward recursion of the first processor for the longer windows, since a one-clock-cycle delay exists between the two groups. Likewise, the backward recursion does not seamlessly flow across the boundary between the processors of the shorter windows and those of the longer windows. Instead, the boundary state metrics may be written into memory when generated by a processor on one side of the boundary, and read from that memory later upon initialising a recursion on the other side of the boundary.

Since not all window lengths are identical in the second version of the APTD algorithm, interleaving the extrinsic LLRs $\bar{b}^{\mathrm{u,e}}_{1,k}$, $\bar{b}^{\mathrm{l,e}}_{1,k}$ according to the schedule at which they are generated does not benefit from the LTE interleaver's contention free property. More specifically, some processors would be provided with more than one *a priori* LLR in some

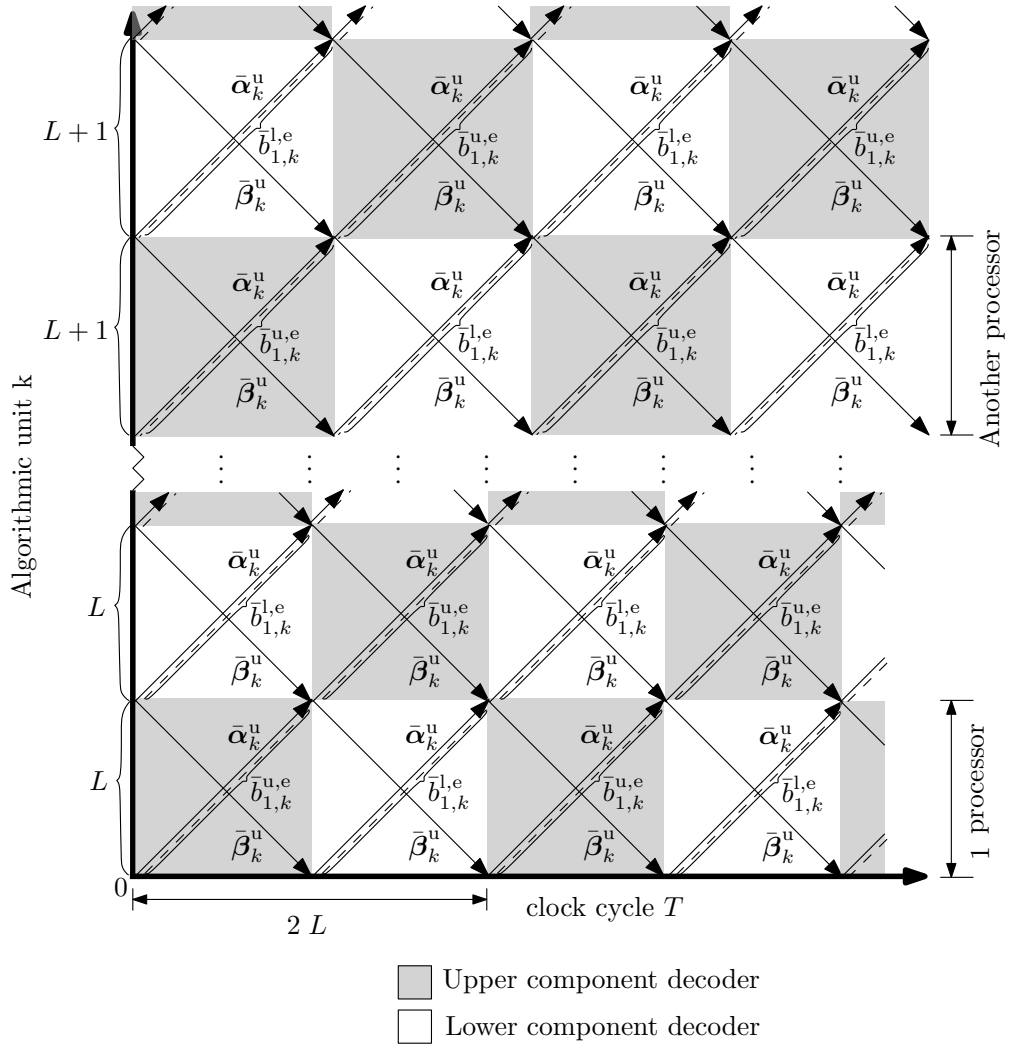


FIGURE 2.13: Schedule of the forward and backward recursions for the second version of the APTD, which uses unequal window lengths, odd-even processing and calculates two extrinsic LLRs at a time alongside the forward recursion.

clock cycles by the LTE interleaver, hence requiring a complex input/output interface. However, we note that when interleaving the LLRs according to this schedule, many of the resultant *a priori* LLRs $\bar{b}_{1,k}^{\mathrm{u,a}}$, $\bar{b}_{1,k}^{\mathrm{l,a}}$ do not get used by the other decoder for a number of clock cycles later, when they are reached by the forward or backward recursions. This illustrates that the interleaving of some extrinsic LLRs can be delayed by a number of clock cycles without having any effect on the operation of the turbo decoder. Note however that the operation is adversely affected if the interleaving of an extrinsic LLR is delayed beyond the time when the resultant *a priori* LLRs are first used by the other decoder, requiring it to instead use an out-of-date LLR from a previous iteration. Motivated by this, our future work will address the contention problem in the second version of the APTD algorithm by designing schedules for the interleaving of the extrinsic LLRs, so that only one extrinsic LLR is delivered to each processor at a time, enabling the employment of a simple input/output interface. In the meantime, Section 2.4 will

investigate the effect of delaying the interleaving of the extrinsic LLRs in the second version of the APTD.

2.4 Performance analysis

In this section, we characterise the error correction performance associated with the different techniques of Section 2.3 individually, as well as in combination with the techniques employed by the two versions of the APTD, which include odd-even operation, non-interleaved systematic LLRs and the calculation of the extrinsic values alongside the forward recursion.

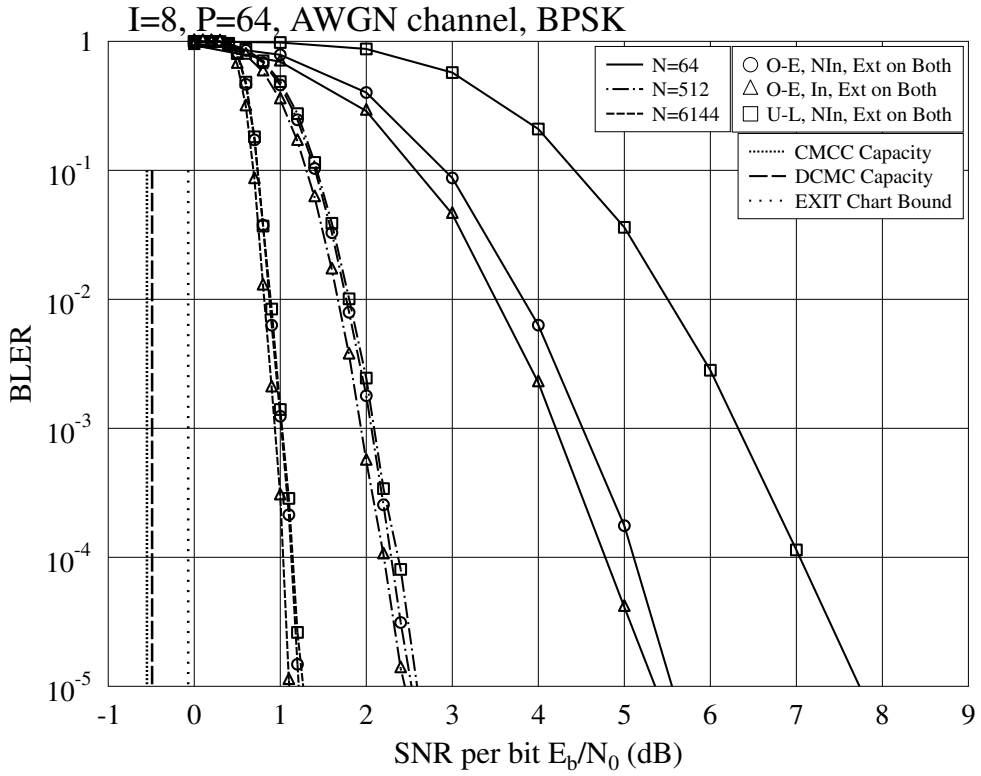


FIGURE 2.14: BLER performance of the APTD algorithm for the $N = 64, 512$ and 6144-bit LTE turbo code, employing $P = 64$ processors, $I = 8$ iterations, Radix-4 operation, interleaved (In) or non-interleaved (NIn) systematic, upper-lower (U-L) or odd-even (O-E) operation, where the extrinsic LLRs are calculated four at a time once the forward and backward recursions have crossed over (Ext on Both).

The error correction performance achieved is characterised in Figures 2.14 to 2.17, which present a range of capacity bounds that provide references for the attainable performance of turbo codes. More specifically, given a code rate of $R = 1/3$ and QPSK modulation, upon communicating over an AWGN channel, the Continuous-input Continuous-output Memoryless Channel (CCMC) capacity of $E_b/N_0 = -0.49$ dB is obtained according

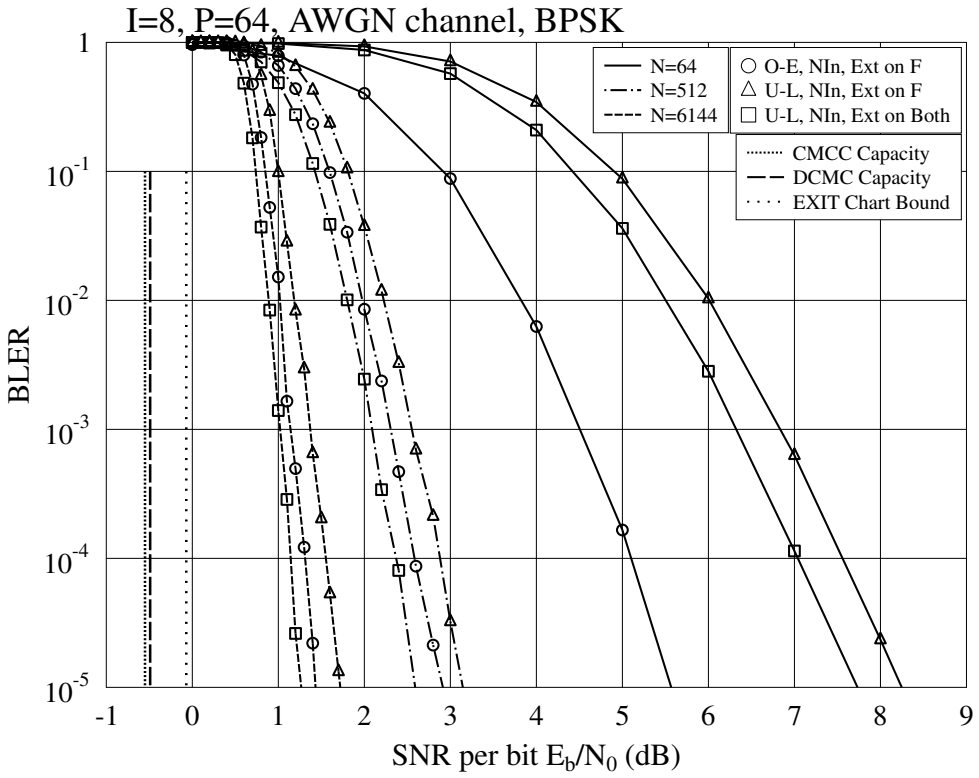


FIGURE 2.15: BLER performance of the APTD algorithm for the $N = 64, 512$ and 6144-bit LTE turbo code, employing $P = 64$ processors, $I = 8$ iterations, Radix-4 operation, *NIn*, U-L or O-E operation, where the extrinsic LLRs are either obtained two at a time alongside the forward recursions (Ext on F), or four at a time once the forward and backward recursions have crossed over (Ext on Both).

to [64]. The corresponding modulation-specific Discrete input Continuous-output Memoryless Channel (DCMC) capacity of $E_b/N_0 = -0.51$ dB was derived in [65]. Additionally, the EXtrinsic Information Transfer (EXIT) chart capacity band of -0.07 dB derived in [66] is the lowest E_b/N_0 value for which an open tunnel can be created between the two mutual information curves of the pair of convolutional decoders in the EXIT chart. We note that in all cases considered, the various turbo decoders do not present any error floors above an BLER of 10^{-5}, indicating that the quality of service requirements of URLLC LTE are indeed met.

Figure 2.14 compares the BLER performance of the APTD for the LTE turbo code employing $P = 64$ processors, $I = 8$ iterations, Radix-4 operation, odd-even (O-E) and the conventional upper-lower (U-L) operation, in cases of short, medium and long block lengths of $N = 64, 512$ and 6144 bits, where the non-interleaved systematic LLRs (*NIn*) are employed and the extrinsic LLRs are calculated four at a time, once the forward and backward recursions have crossed over (Ext on Both). Note that the combination of U-L, *NIn* and Ext on Both is as employed in the SOTA approach. For the short block length of $N = 64$ bits, Figure 2.14 shows that a 2dB gain is achieved at a BLER of 10^{-5}

by employing O-E, rather than U-L. However, this gain decreases as the window length $L = N/P$ is increased. This is because the advantage offered by having the recursions of one window seamlessly leading into those of its neighboring window becomes less significant, when the windows are longer. Furthermore, Figure 2.14 shows that *NIn* degrades the BLER performance by 0.2 dB associated with O-E compared to *In* in the case of short block lengths N, but no degradation is observed for longer block lengths. This represents a small price to pay for the benefit of eliminating the requirement for an additional interleaver. Note that *In* and *NIn* attain an identical BLER performance for U-L, as discussed in Section 2.3.

Recall from Section 2.3, that the proposed APTD algorithm calculates the extrinsic LLRs two at a time alongside the forward recursions (Ext on F), hence resulting in a significant reduction in hardware resources, compared to schemes that calculate the extrinsic LLRs four at a time, once the forward and backward recursions have crossed over (Ext on Both). However, this hardware reduction is achieved at the cost of a slight BLER degradation for medium and long block lengths, as shown in Figure 2.15. Note that the BLER degradation associated with Ext on F is higher when employing U-L, demonstrating a further benefit of O-E.

Based on Figures 2.14 and 2.15, we may conclude that the combination of O-E, *Nln* and Ext on F results in an attractive APTD algorithm, since it achieves significantly reduced complexity at the cost of only slight BLER performance degradation, compared to the SOTA LTE turbo decoder. In the following discussions, we will employ this combination for investigating the BLER performance of the two versions of the APTD algorithms.

In order to better demonstrate the difference in operation between the two versions of the proposed APTD algorithm, we employ $p = 56$ processors in the following discussions, rather than $p = 64$ as in our discussions above. Since 56 is not divisible by the block lengths of $N = 64, 512$ or 6144, this choice will result in different schedules between the two versions of the APTD. More specifically, the first version will activate $P = 32, 32$ and 48 of the $p = 56$ processors, when $N = 64, 512$ and 6144, respectively, in order to ensure that each of the P windows has the same length L. By contrast, the second version will activate all $p = 56$ processors for all block lengths N by employing windows having different lengths. Figure 2.16 shows that the second version achieves the best BLER performance among the four different turbo decoders at the same number of clock cycles T, especially for the short block lengths. We can see that while the conventional and SOTA decoder suffer from bad reliability, which may result in poor quality of service (QoS), the second version is capable of achieving the URLLC BLER requirement of 10^{-5}. Note that when a block length of $N = 6144$ is employed, the SOTA turbo decoder slightly outperforms both versions of the APTD. However, this gain is negligible compared to the gap between the BLER performance and the capacity bounds.

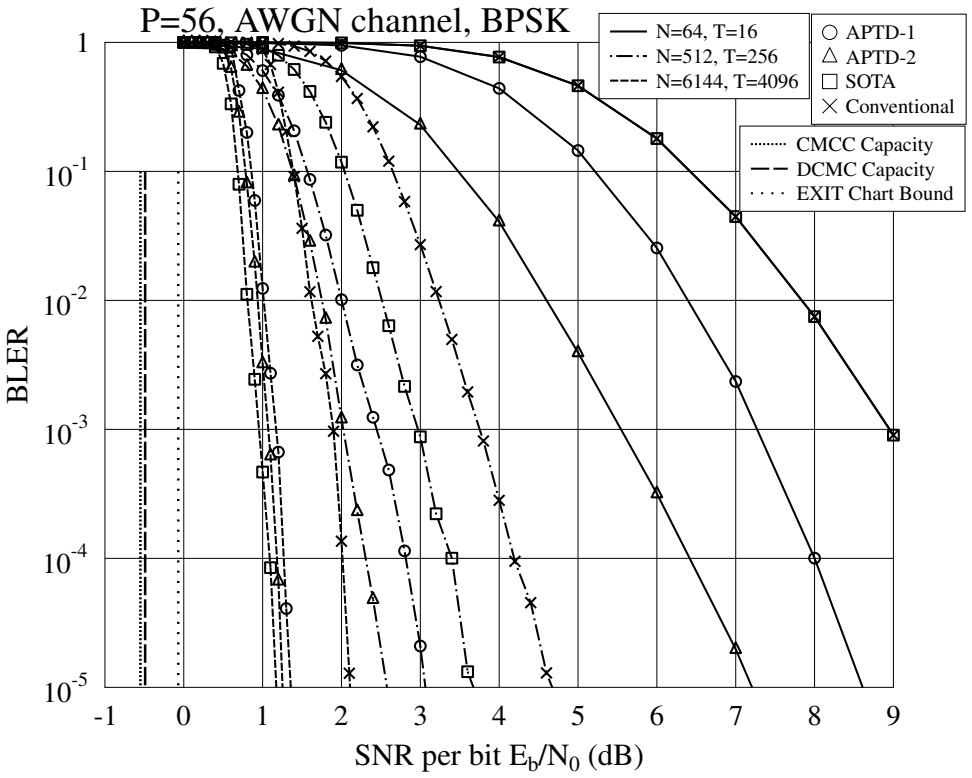


FIGURE 2.16: BLER performance of the conventional and SOTA LTE turbo decoder and both versions of the proposed APTD for the $N = 64,512$ and 6144 bits LTE turbo code that employs $T = 16,256$ and 4096 clock cycles, respectively. The proposed APTD employs $P = 56$ processors, Radix-4 operation, *NIn* systematic approach, O-E operation, and the extrinsic values are obtained from forward recursions only (Ext on F).

In order to facilitate the practical implementation of the second version of the APTD, it is necessary to solve the contention problem, which occurs when more than one processor attempts to pass LLRs through the interleaver to the same processor at the same time. A measure that may be taken to avoid contention in the second version of the APTD is to reschedule the operation of the interleaver, by delaying the interleaving of some LLRs that would otherwise cause contention. This approach is motivated since many *a priori* LLRs are not used in O-E until a number of clock cycles after they are generated. However, in some cases, the contention may only be eliminated by delaying the interleaving of some LLRs until after they would have otherwise been used. In this case, the LLRs generated in the most recent previous iteration may be used instead, albeit at the cost of degraded BLER. In our investigations, we found that 1/3 of the LLRs must be rescheduled, in order to avoid contention in the second version of the APTD. However, in order to investigate the worst-case BLER degradation that may be imposed, we quantify the impact of delaying the interleaving of *all* LLRs, rather than just that of the subset which causes contention. Figure 2.17 characterises the BLER when ω clock cycles of delay is

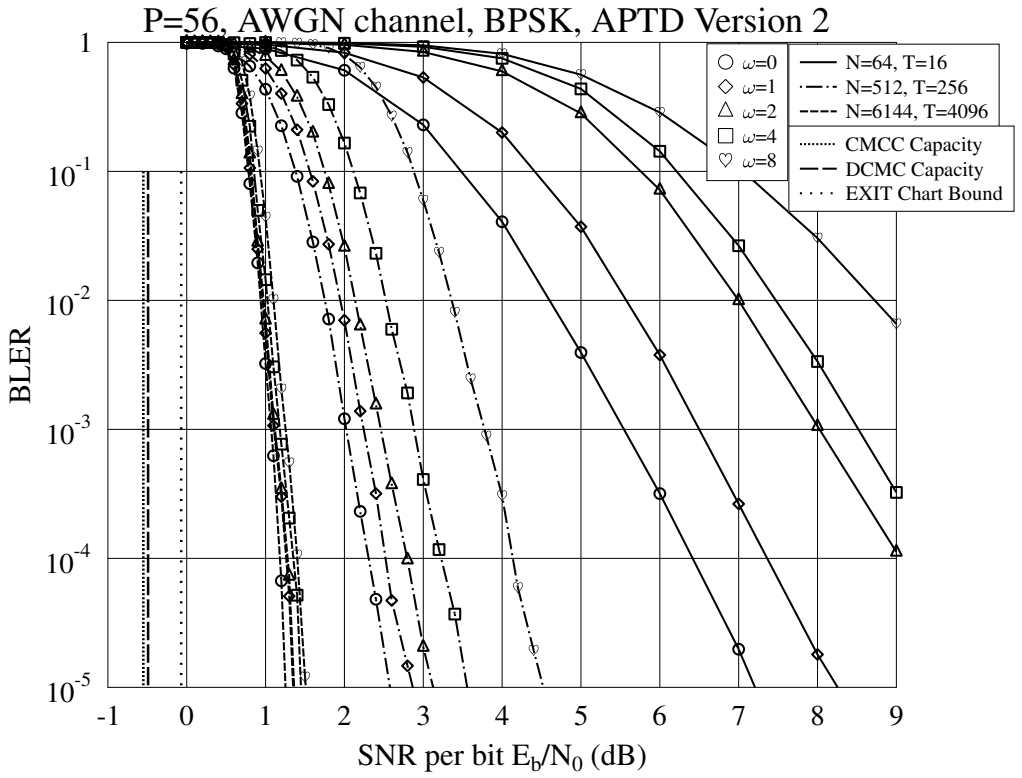


FIGURE 2.17: BLER performance for second version of the APTD algorithm for $N = 64, 512, 6144$, with $T = 16, 256$ and 4096 clock cycles, respectively, activating $P = 56$ processors when different delays are imposed. Here, *NIn* systematic and O-E operation are employed and the extrinsic values are obtained from forward recursions only (Ext on F).

imposed on the interleaving of the LLRs. By comparing Figures 2.16 and 2.17, it may be seen that the second version of the APTD offers superior BLER over the first version, even if 1 or 2 clock cycles of delay are applied to all LLRs. Since contention can be eliminated by delaying only a subset of the LLRs, we may conclude that the second version is superior.

2.5 Complexity analysis

This section compares the proposed APTD employing $p = 64$ processors to the conventional LTE turbo decoder of [60] employing $p = 8$ processors, to the SOTA LTE turbo decoder of [23] employing $p = 64$ processors, and to the FPTD of [24] employing $p = 6144$ processors, both in terms of the number of clock cycles required for achieving a low BLER and in terms of the complexity of each processor. These are used for characterising the latency, throughput and hardware resource requirements that may be expected by these turbo decoder algorithms.

TABLE 2.1: Complexity analysis of different turbo decoders

	Conventional $N \in [40, 6144]$	SOTA $N \in [40, 6144]$	FPTD $N \in [40, 6144]$	APTD $N \in [40, 6144]$	
				Version 1	Version 2
Window length (L)	$N/8$	N/P[1]	2	$\frac{N}{gif(N,p)}$	$\lfloor N/P \rfloor$ or $\lceil N/P \rceil$
Overall latency (T)	T_{C}	T_{S}	T_{F}	T_{A1}	T_{A2}
Overall throughput ($1/T$)	$\frac{1}{T_{\mathrm{C}}}$	$\frac{1}{T_{\mathrm{S}}}$	$\frac{1}{T_{\mathrm{F}}}$	$\frac{1}{T_{\mathrm{A1}}}$	$\frac{1}{T_{\mathrm{A2}}}$
Complexity per processor per clock cycle (C)	320	320	155	320	
Interleaved LLRs by each PE at a time	4	4	1	2	
Overall complexity (TCP)	$2560T_{\mathrm{C}}$	$320T_{\mathrm{S}}P$	$155T_{\mathrm{F}}N$	$320T_{\mathrm{A1}}P$	$320T_{\mathrm{A2}}P$
Computational efficiency ($1/TCp$)	$\frac{1}{320T_{\mathrm{C}}p}$	$\frac{1}{320T_{\mathrm{S}}p}$	$\frac{1}{155T_{\mathrm{F}}p}$	$\frac{1}{320T_{\mathrm{A1}}p}$	$\frac{1}{320T_{\mathrm{A2}}p}$

Table 2.1 summarises our comparisons of the proposed APTD with the conventional LTE turbo decoder, with the FPTD, and with the SOTA LTE turbo decoder. In contrast to the Radix-2 operation of the FPTD, the conventional turbo decoder, the SOTA LTE turbo decoder and the APTD use Radix-4 processing [20, 21], allowing two algorithmic units to be processed per clock cycle. For example, in the case of a block length of $N = 504$ bits, the conventional and the SOTA LTE turbo decoder decomposes the block into $P = 8$ windows and processes them using 8 processors. In the case of the SOTA decoder, its other 56 processors remain disabled. Here, each window in the conventional or the SOTA LTE turbo decoder comprises $L = 63$ bits, requiring 63 clock cycles to complete the processing of the whole block length, when using Radix-4 processing. By contrast, when $N = 504$, the first and the second version of our APTD activate $P = 63$ and 64 processors, respectively. In both schemes, 8 clock cycles are used for processing the entire 504 bits once, which is an eight-fold delay reduction. Indeed, the second version of the APTD activates all processors for all block lengths that satisfy $N \geq P$, hence avoiding wasted hardware, as discussed in Section 2.3.2. Since the FPTD processes the entire block simultaneously, alternating between the odd- and even-indexed algorithmic units, its window length L is consistently 2, regardless of the block length N.

The overall latency may be compared in terms of the number of clock cycles required for each turbo decoder to perform sufficient iterations to achieve a BLER of 10^{-5} at the same E_b/N_0 as the conventional turbo decoder using $I = 8$ iterations. Meanwhile, the overall throughput is proportional to the reciprocal of the decoding latency. Figures 2.18 and 2.19 characterise the latency and the throughput as functions of block length $N \in [40, 6144]$ for both versions of the proposed APTD and compare them to those of the conventional, FPTD and the SOTA turbo decoders. In the case of $N = 504$ bits, a total of $T_C = T_S = 1008$ clock cycles are required for the conventional and SOTA LTE turbo decoder to complete $I = 8$ iterations. By contrast, the first and second version of the APTD require 320 and 296 clock cycles, respectively. The APTD achieves this latency improvement of at least 2.23 times by activating all of its 64 processors, while the conventional and SOTA decoder activate only 8 processors at $N = 504$. In a hardware implementation operating at the same clock frequency of 250 MHz used in the commercial LTE turbo decoder implementation of [60], the corresponding processing latency of the APTD would become around 3.6 μs, achieving a 14-times improvement compared to the 52 μs demonstrated in [60]. Hence, the proposed APTD is capable of meeting the 7.4 μs latency requirement of LTE URLLC, even in the case of the longest $N = 6144$-bit blocks. Meanwhile, at the longest block length of $N = 6144$ bits a total of 12288 clock cycles are required for the conventional turbo decoder to complete $I = 8$ iterations, while the proposed APTD requires only 890 clock cycles to achieve the same BLER performance.

[1] The number of activated processors P for different block lengths N of the SOTA LTE turbo decoder is given in (2.7).

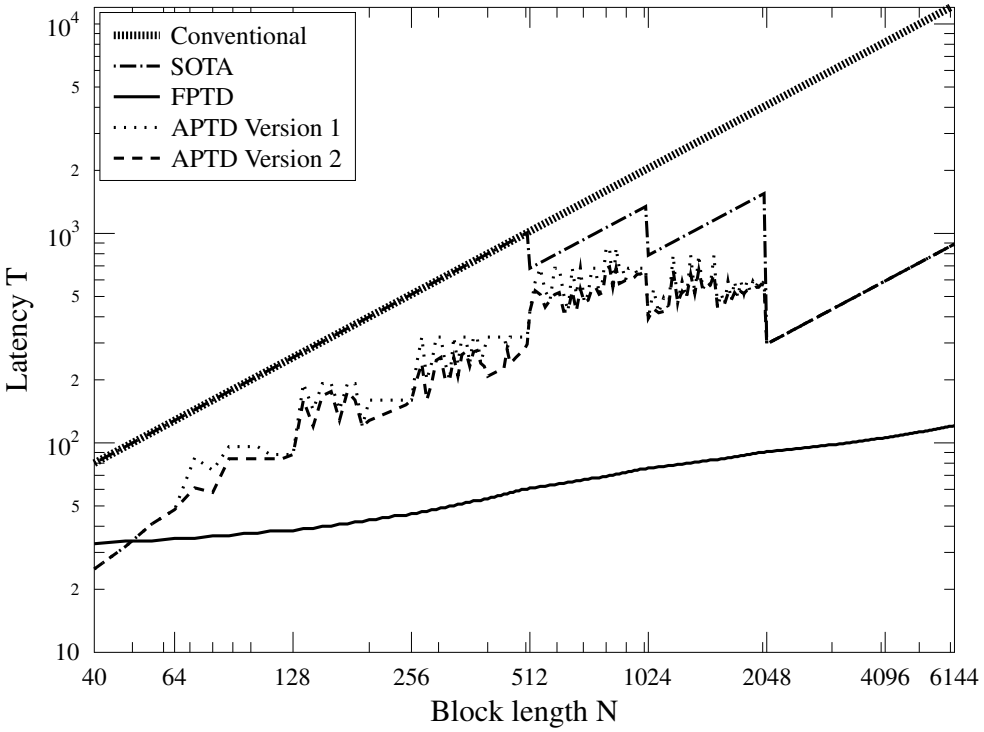


FIGURE 2.18: The number of clock cycles required for all block lengths $N \in [40, 6144]$ to achieve a BLER of 10^{-5} at the same E_b/N_0 as the conventional turbo decoder using $I = 8$ iterations in different turbo decoders.

The computational complexity may also be used for characterizing both the energy consumption and the hardware resource requirement of a practical hardware implementation. In [24], the complexity C per processor per clock cycle imposed by the SOTA LTE turbo decoder and by the FPTD was given as 320 and 155 Add Compare Select (ACS) operations, respectively. For the conventional LTE turbo decoder and both versions of the APTD algorithm, the number of ACS operations performed per processor per clock cycle is also $C = 320$, as in the SOTA turbo decoder, since they also employ Radix-4 processing.

As discussed in Section 2.2, the maximum number of LLRs that must be interleaved at a time by each PE varies among the decoding algorithms compared in Table 2.1. Since the proposed APTD algorithm obtains extrinsic LLRs alongside the forward recursions, its PEs generate two extrinsic LLRs for interleaving in each clock cycle when using Radix-4 processing. Similarly, the PEs of the FPTD algorithm process windows comprising only a single trellis stage, and so they generate only one extrinsic LLR for interleaving at a time. By contrast, the conventional and the SOTA LTE turbo decoder generates four extrinsic LLRs at a time once the forward and backward recursions have crossed over, when using Radix-4 processing. The requirement for the conventional and the

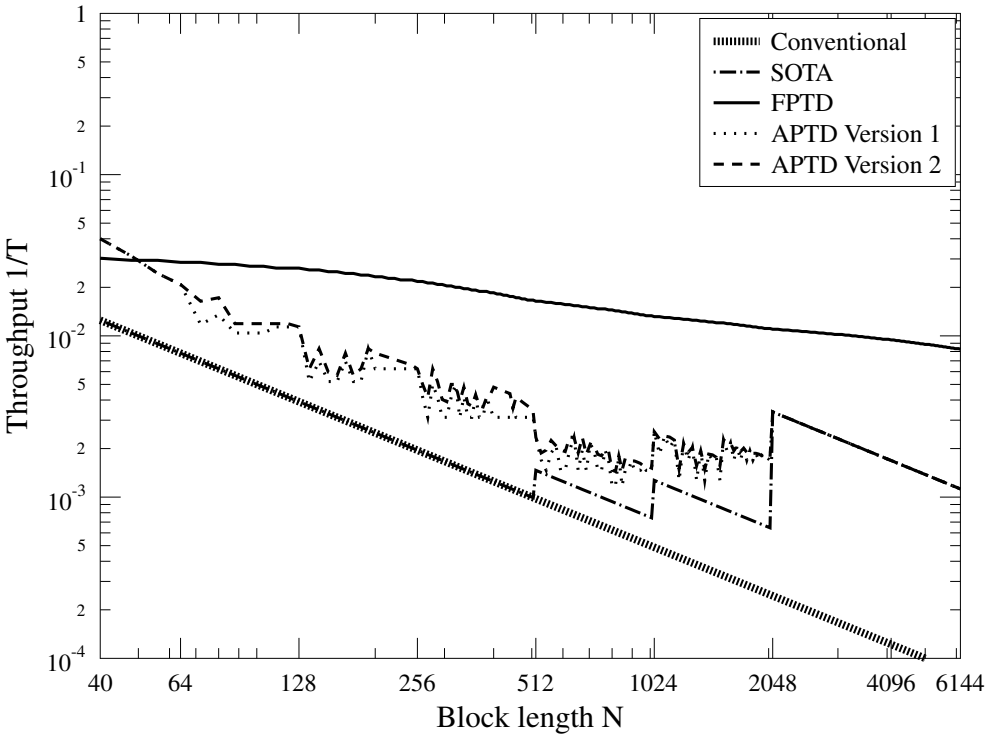


FIGURE 2.19: The reciprocal of the number of clock cycles required for all block lengths $N \in [40, 6144]$ to achieve a BLER of 10^{-5} at the same E_b/N_0 as the conventional turbo decoder using $I = 8$ iterations in different turbo decoders in different turbo decoders.

SOTA turbo decoder to interleave four extrinsic LLRs at a time represents a significant hardware overhead, causing the interleaver to occupy 15.3% of the chip area in the turbo decoder implementation [67].

Combining the above-mentioned design considerations, the energy consumption of a turbo decoder implementation may be characterised by its overall computational complexity, as shown in Table 2.1. The overall complexity is defined as the product of the number T of clock cycles required to achieve a BLER of 10^{-5} at the same E_b/N_0 as the conventional LTE turbo decoder using $I = 8$ iterations, the computational complexity C per processor per clock cycle and the number of activated processors P. Figure 2.20 shows the overall complexity associated with all block lengths $N \in [40, 6144]$ for the various turbo decoders considered in this section. In the example of $N = 504$ bits, the overall complexity of the second version of the proposed APTD employing $P = 64$ PEs is 2.3 times that of the conventional and the SOTA LTE turbo decoders, which employ 8 processors. However, its complexity is 1.3 times higher than that of the FPTD in this case. Note that for block lengths in excess of $N = 2048$, the number of activated processors and the overall latency are the same in the SOTA LTE turbo decoder and both versions of the APTD, so the overall computational complexity remains identical.

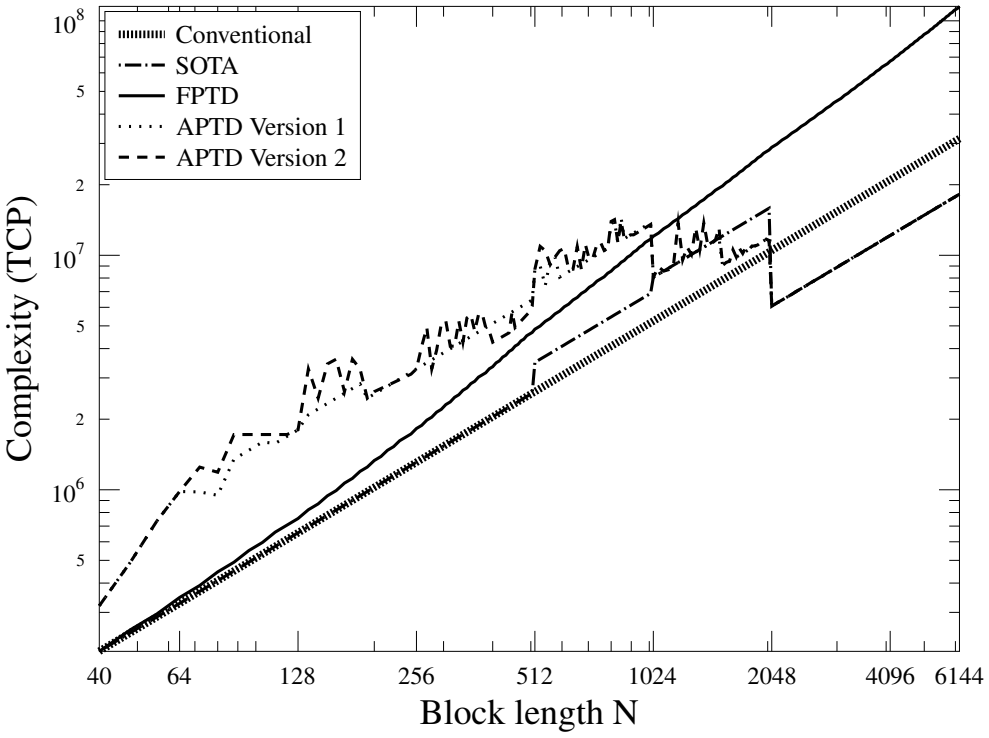


FIGURE 2.20: The overall complexity in different turbo decoders as a function of the block length N.

In addition to the computational complexity TCP, we also compare the decoders in terms of their hardware efficiency $1/(TCp)$. Note that while the computational complexity depends on the number P of activated processors, the hardware efficiency considers the total number p of processors employed in the decoder, as summarised in Table 2.1. This is because all processors occupy a certain chip-area, regardless whether they are activated or not. Figure 2.21 characterizes the hardware efficiency of the three turbo decoders discussed in this chapter, as functions of the block length $N \in [40, 6144]$. In the example of $N = 504$ bits, the second version of the APTD employing $p = 64$ processors achieves a 7-times efficiency improvement compared to the SOTA turbo decoder employing $p = 64$ processors. Note that for block lengths in excess of $N = 2048$, the computational efficiency is similar in both versions of the APTD and the SOTA turbo decoder. In the case of the FPTD, it is assumed that $p = 6144$ processors are employed and that $P = N$ processors are activated when decoding blocks of length N. Owing to this, the hardware efficiency of the FPTD is poor, especially when the block length N is small compared to p. In the case of $N = 504$ bits, the second version of the APTD employing $P = 64$ processors achieves a 22-times efficiency improvement compared to the FPTD decoder.

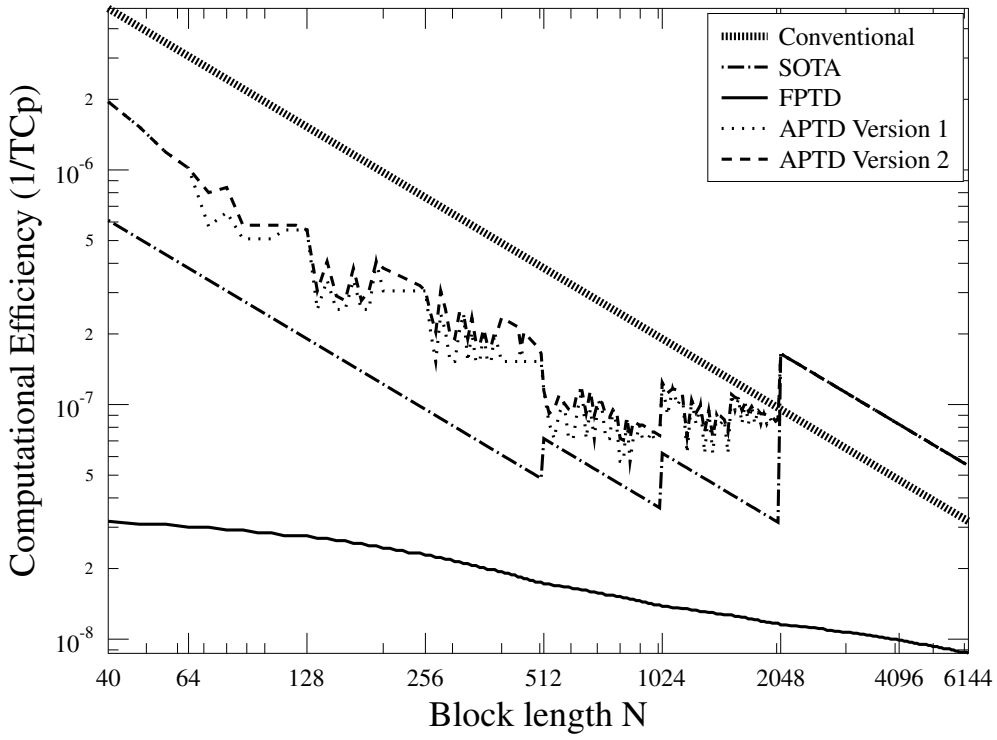


FIGURE 2.21: The overall computational efficiency in different turbo decoders as a function of the block length N.

2.6 Conclusions

TABLE 2.2: Latenct and complexity analysis of different turbo decoders in the case of $N = 504$.

	Conventional	SOTA	FPTD	APTD
Overall latency (T)	1008	1008	60	704
Overall complexity (TCP) ($\times 10^{-6}$)	2.6	2.6	4.7	6.1

This chapter proposed a novel APTD algorithm, which facilitates an arbitrarily high degree of turbo decoding parallelism for the first time, enabling significantly improved throughput, latency, and computational efficiency in comparison to the SOTA turbo decoder while meeting the requirements of LTE URLLC. More specifically, conventional commercial implementations of the LTE turbo decoder have latencies of up to 52 μs, which are not able to meet the 7.4 μs requirements of LTE URLLC. By contrast, the proposed APTD can achieve the same error correction performance as the conventional decoder down to BLERs of 10^{-5}, but with latencies of no more than 3.6 μs, meeting the requirements of LTE URLLC. Furthermore, the APTD achieves a significant reduction

in complexity in long block lengths, compared to FPTD. In particular, none of the processors in our proposed algorithm remain idle for any block length $N \geq p$, leading to better BLER performance than the SOTA turbo decoder in cases of short block lengths. For instance, when $p = 56$ processors and 16 clock cycles are employed for decoding a block length $N = 64$, our APTD achieves a coding gain of 3.5 dB compared to the SOTA LTE turbo decoder at a BLER level of 10^{-3}, as shown in Figure 2.16, whereas only slight improvements can be observed when decoding a block length of $N = 6144$ employing the same number of $p = 56$ processors with 4096 clock cycles. We have proposed an odd-even processing of windows in the upper and lower decoder, which achieves better BLER performance for short block lengths, compared to conventional upper-lower processing. Furthermore, we reduce the interleaving complexity by generating extrinsic LLRs alongside the forward recursion, at the cost of slightly degraded BLER performance. Like the FPTD, the proposed APTD is capable of achieving the same error correction performance as a conventional LTE turbo decoder, at all block lengths. However, our APTD achieves this using significantly fewer decoding iterations and hence a lower complexity at long block lengths. As shown in Figures 2.18 to 2.21, the proposed APTD achieves superior latency, throughput and computational efficiency than the SOTA LTE turbo decoder at all block lengths, but particularly at the short block lengths that are typically used in URLLC approaches. For example, at a block length of $N = 504$ bits, the proposed APTD achieves an BLER of 10^{-5} at the same E_b/N_0 as $I = 8$ iterations of a conventional turbo decoder, but with a computational efficiency that is 6 times higher than that of the SOTA turbo decoder, while achieving a latency and throughput that are 0.7 and 1.4 times those of the SOTA decoder, respectively, as shown in Table 2.2. Note however that this is achieved at the cost of increasing the computational complexity by 2.3 times compared to the SOTA decoder of $N = 504$.

Chapter 3

Concurrent OFDM Demodulation and Turbo Decoding Architecture

3.1 Introduction

As we discussed in Chapter 1, a significant reliability and latency improvement is required for the Ultra-Reliable Low Latency Communication (URLLC) service in the upcoming Fifth Generation (5G) New Radio (NR). However, achieving a significantly increased throughput and reliability as well as a significant reduction in latency represents a substantial challenge [68–70], having no simple solutions. However once realised, this URLLC paradigm will allow humans or machines to communicate with remote mobile devices and control them seamlessly, without suffering from the lag that prevents accurate control using wireless communication systems [71]. This will enable a wide variety of new applications in remote surgery, automated driving, and virtual reality, having significant economic and societal impact [72–76]. However, the end-to-end latency of a wireless communication system is fundamentally limited by its physical layer [56, 77], which performs demodulation and error correction, among other tasks. Therefore, our main concern in this chapter lies in the low-latency receiver design, as illustrated in Figure 3.1.

Different future applications of the URLLC mode will impose different demands on the physical layer latency. For example, machine-automated low-latency capital market trading relies on multi-hop wireless communication links, where financial institutions use algorithms running on their own computers for automatically buying and selling stocks, whenever they momentarily have different values on stock exchanges in different cities.

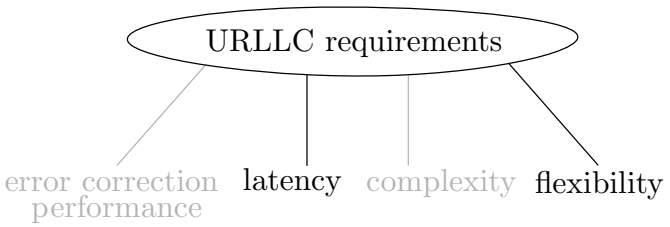


FIGURE 3.1: The contribution of Chapter 3.

In this application, each relay has in these links to have a sub-microsecond physical layer latency [78], not including the propagation delay associated with each hop. By contrast, the so-called Tactile Internet [56, 79] will allow humans to seamlessly control remote devices, provided that a physical layer latency of below 100 μs can be achieved.

However, State-of-the-art (SOTA) wireless communication systems have physical layer latencies that are significantly higher than these targets. For example, the world's fastest low-latency capital market trading links impose a physical layer latency of around 5 μs per hop, not including propagation latency. Meanwhile, SOTA implementations of the Long Term Evolution (LTE) cellular telephony standard have a physical layer latency, which significantly exceeds the 100 μs target of the Tactile Internet [56, 80]. To elaborate, the LTE physical layer achieves a high throughput and reliability by employing the Orthogonal Frequency-Division Multiplexing (OFDM) technique [27, 28] for mitigating echoes in the wireless channel, as well as a turbo code for correcting any remaining transmission errors [2, 3, 29, 30]. However, these techniques impose a high signal processing complexity upon the physical layer, particularly in the receiver. As shown in Figure 3.2(a), the processing of the receiver's Fast Fourier Transform (FFT) [81, 82] cannot begin until the whole message block has been received, since each of its outputs is a function of the whole received block. Owing to this, the FFT produces all of its outputs simultaneously, preventing the turbo decoding process from beginning until after the FFT has been completed. In practical LTE deployments, the transmission latency incurred while receiving, the processing latency incurred while performing the FFT and the processing latency incurred while performing turbo decoding are each around 70 μs [29, 56], allowing pipelining as shown in Figure 3.2(a). The sum of these latencies is 210 μs, which already exceeds the above-mentioned 100 μs target, even without considering the latency associated with propagation, channel estimation, Multiple-Input and Multiple-Output (MIMO) detection and transmitter processing.

This motivates our new architecture, in which the physical layer receiver components are operated concurrently, rather than consecutively. This approach is exemplified by Figure 3.2(b), in which the reception, FFT processing and turbo decoding of each block is performed concurrently, potentially facilitating sub-microsecond physical layer latencies in the case of low-latency communications in 5G. In the case of the URLLC communications [6, 54, 55], this approach can reduce the associated latency from 210 μs to 70 μs, which is within the above-mentioned 100 μs latency target of the Tactile Internet.

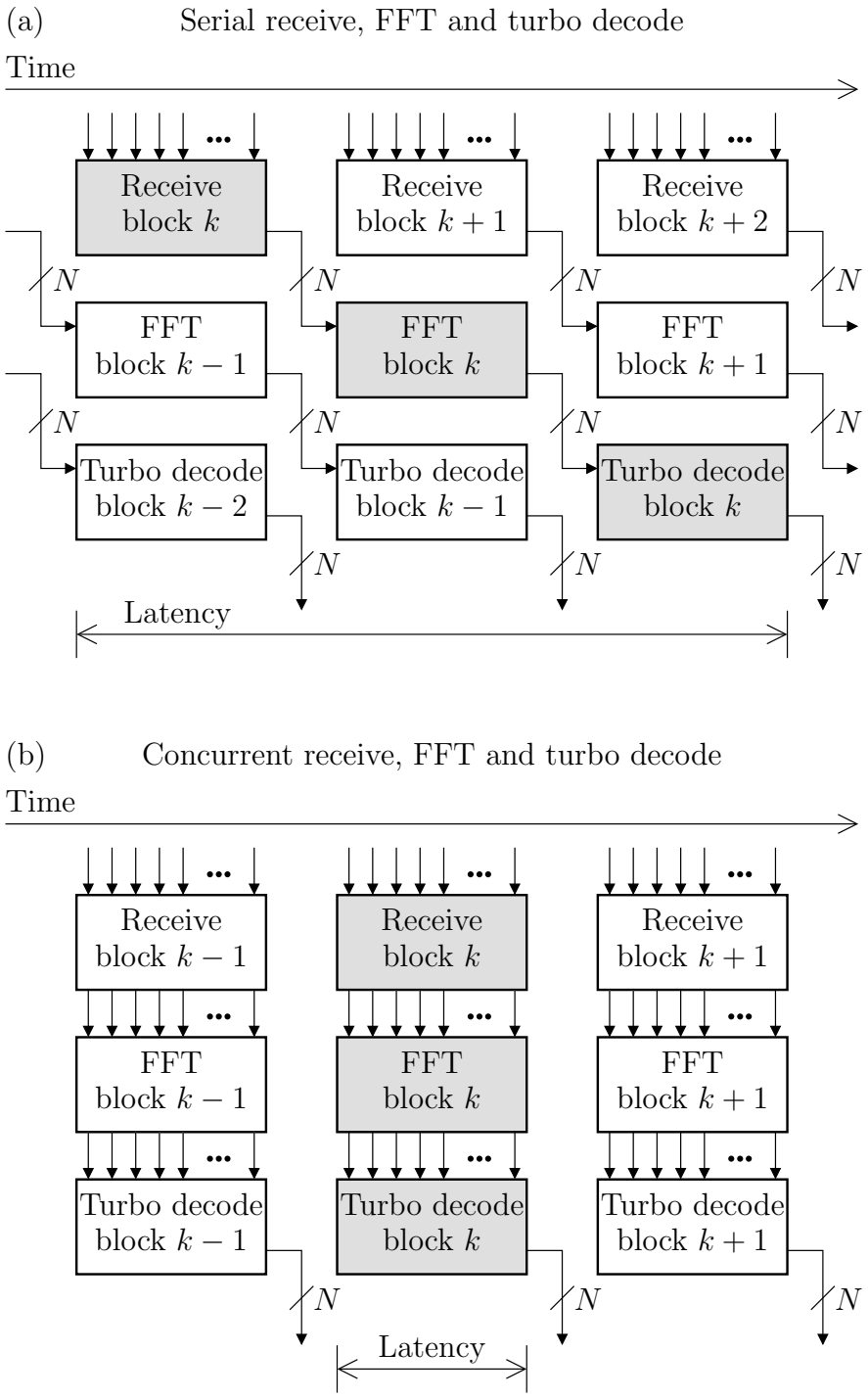


FIGURE 3.2: Timing diagram for (a) the conventional approach and (b) the proposed ultra-low-latency approach.

This leaves 30 μs for propagation and for the remaining, lower-complexity physical layer components, including channel estimation, MIMO detection and transmitter processing. Indeed, it may be expected that the proposed technique can be extended to perform some of these operations concurrently with those of Figure 3.2(b), within the same 70 μs. In addition, the turbo codes adopt the advantage of lower decoding complexity and better error-correction performance compared to the Low-Density Parity-Check (LDPC) codes at low coding rates that are motivated in mission-critical vehicular communications for the sake of ensuring low Bit Error Ratio (BER) and ultra-high reliability [11, 12]. Our new contributions are as follows.

- we propose a novel cumulative FFT, which is processed incrementally and concurrently with the Fully-Parallel Turbo Decoder (FPTD) that is reviewed in Section 2.2.2.2, throughout the process of receiving a single OFDM symbol. Since the information carried by each turbo encoded bit is spread throughout the duration of the OFDM symbol, the proposed concurrent FFT can obtain some information about each bit as soon as the reception of the OFDM symbol begins, allowing turbo decoding to start immediately. As more and more of the OFDM symbol is received with passing time, the cumulative FFT can obtain more and more information about the turbo encoded bits, which can be fed into the concurrent turbo decoding process.
- We show that if the turbo decoder can complete a sufficient number of iterations within the duration of the OFDM symbol, then it can achieve the same error correction performance as if the turbo decoding process had only began after the reception of the OFDM symbol had been completed.

As shown in Figure 3.3 this chapter is structured as follows. Section 3.2 provides a brief overview of the FFT technique that is employed to implement OFDM. Following this, the proposed concurrent OFDM demodulation and turbo decoding architecture is proposed and detailed in Section 3.3. The validation of this architecture is presented in Section 3.4, while its error correction performance and extensions to manage the trade-offs between latency, reliability and complexity are presented in Section 3.5. Finally, we offer our conclusions in Section 3.7.

3.2 Fast Fourier Transform

In OFDM, a bit stream is decomposed into several parallel bit streams, each of which has much a proportionately reduced bit rate and is modulated onto a different subcarrier. In this way, rather than using a serial Time Domain (TD) bit stream, OFDM uses many low-rate parallel Frequency Domain (FD) bit streams, which are less prone to dispersion.

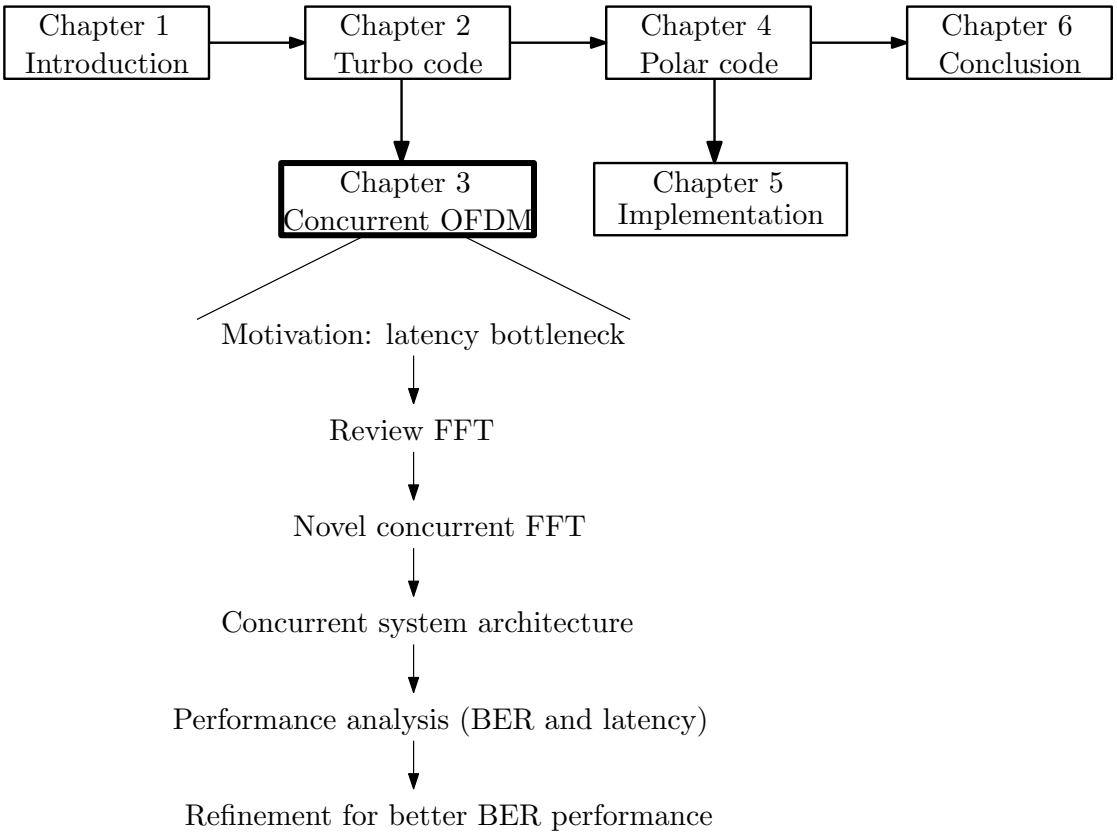


FIGURE 3.3: The development of Chapter 3.

Typically, OFDM schemes are implemented using Discrete Fourier Transform (DFT) techniques [83–85]. To be more specific, the Inverse Discrete Fourier Transform (IDFT) is performed in the transmitter to generate a single TD OFDM symbol to represent the set of the FD bit streams, each of which typically carries a Quadrature Amplitude Modulation (QAM) symbol. Meanwhile, the corresponding DFT is performed at the receiver to recover the QAM symbols carried by the sub-carriers of the OFDM symbol. In the receiver, a N-point DFT can be defined as

$$Y_z = \sum_{n=0}^{N-1} y_n W_N^{nz}, \quad 0 \leqslant z \leqslant N-1, \tag{3.1}$$

where $W_N = \exp(-j2\pi/N)$. Here, y_n is the nth sample of the received TD OFDM symbol, where the set of N samples are received consecutively, spread over time. Meanwhile, Y_z is the zth FD subcarrier's QAM symbol, where each of the N FD QAM symbols is dependent on all N TD samples of the TD OFDM symbol.

Note that in practice, the demodulator's DFT is typically implemented using the FFT, which has a significantly reduced complexity if N is high [81, 82]. If N is a power of 2, a Radix-2 FFT is achieved by recursively partitioning Y_z into odd- and even-indexed terms,

across $v = \log_2(N)$ stages. The odd and even terms in Y_z can be expressed respectively as

$$Y_{2z} = \sum_{n=0}^{N/2-1} \left(y_n + y_{n+N/2}\right) W_{N/2}^{nz}, \quad 0 \leqslant z \leqslant \frac{N}{2} - 1, \tag{3.2}$$

$$Y_{2z+1} = \sum_{n=0}^{N/2-1} \left[\left(y_n - y_{n+N/2}\right) W_N^n\right] W_{N/2}^{nz}, \quad 0 \leqslant z \leqslant \frac{N}{2} - 1. \tag{3.3}$$

Here, each of (3.2) and (3.3) can be considered to be a DFT in its own right. This allows each of (3.2) and (3.3) to be further decomposed into two DFTs, comprising the odd and even elements, respectively. This may be repeated recursively, until the DFT has been fully decomposed into an individual radix-2 structure, completing the FFT.

The block diagram for the example of a $N = 16$-point radix-2 FFT with $v = 4$ computation stages is depicted in Figure 3.4. The inputs on the left-hand edge of Figure 3.4 represent TD samples. These inputs are first interleaved into a bit reversed ordering, separating the odd terms and even terms. The TD samples are passed through $v = 4$ stages of radix-2 butterflies, each of which performs a radix-2 FFT calculation, as shown in Figure 3.5. Following the final stage of radix-2 butterflies, the $N = 16$ FD samples are obtained.

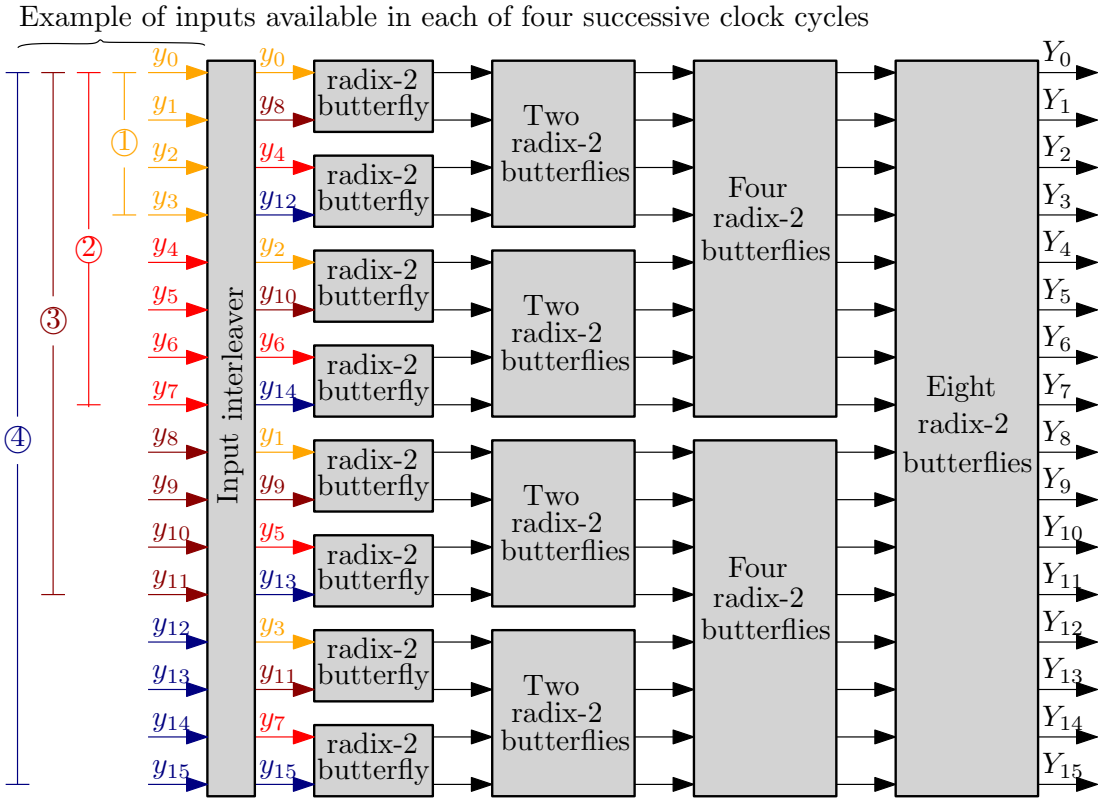


FIGURE 3.4: The FFT block diagram.

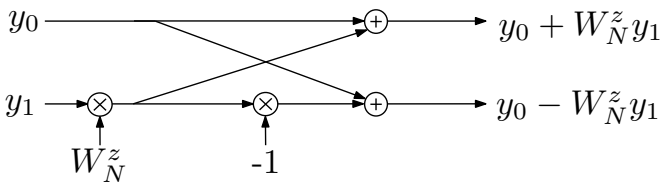


FIGURE 3.5: The radix-2 FFT calculation.

In the conventional approach, reception and demodulation are performed serially, where the FFT operation does not begin until after all the transmitted TD samples are received. However, in our concurrent approach, the demodulation process is started promtly after a small number of samples have been received. Successively more TD samples are received in each of a series of successive clock cycles. In a naive approach, the full-length N-sample FFT may be replaced in each clock cycle, using all TD samples received so far and assuming zero values for all TD samples not yet received, as shown in Figure 3.4. In order to significantly reduce the complexity of repeating the N-point FFT in each clock cycle, we propose an efficient cumulative FFT in Section 3.3.2. This eliminates all redundant calculations associated with zero-valued samples and reuses calculations from one clock cycle to the next. More specifically, an incremental part of the FFT is calculated in each clock cycle and these are accumulated across the series of clock cycles.

3.3 Proposed turbo-coded OFDM scheme

The proposed concurrent turbo-coded OFDM scheme is discussed in this section. Section 3.3.1 introduces our notation and details the proposed scheme's transmitter, which is the same as in a conventional turbo-coded OFDM scheme. Following this, the proposed concurrent detection, FFT and turbo decoding approach of the proposed receiver is detailed in Section 3.3.2.

3.3.1 Transmitter

In the turbo-coded OFDM transmitter of Figure 3.6, the K number of message bits $\mathbf{b}_1^{\mathrm{u}} = [b_{1,k}]_{k=0}^{K-1}$ are encoded by an LTE turbo encoder. To be more specific, the vector of message bits $\mathbf{b}_1^{\mathrm{u}}$ is interleaved to obtain the vector of interleaved message bits $\mathbf{b}_1^{\mathrm{l}}$. These two vectors are encoded by two identical Convolutional Encoders (CEs), referred to as the upper and lower encoders, respectively. The resultant parity bit vectors $\mathbf{b}_2^{\mathrm{u}}$ and $\mathbf{b}_2^{\mathrm{l}}$ are interleaved with the systematic bit vector $\mathbf{b}_3^{\mathrm{u}}$, which is a replica of the message bit vector $\mathbf{b}_1^{\mathrm{u}}$. The resultant bit vector is punctured or repeated depending on the code length and then output as the turbo-encoded bit vector $\mathbf{b}_4 = [b_{3,k}]_{t=0}^{T-1}$, which comprises T bits. The T turbo-encoded bits are then converted into $N = T/\log_2 M$ symbols

$\mathbf{X} = [X_n]_{n=0}^{N-1}$, using an M-ary Quadrature Amplitude Modulation (MQAM) mapper. Here, the MQAM symbols are selected from a set of M complex constellation points $\mathcal{S} = \{s_0, s_1, \cdots s_{M-1}\}$, which satisfy $\sum_{i=0}^{M-1} |s_i|^2 / M = 1$, where each QAM symbol carries $b = \log_2 M$ bits. Following this, a Serial to Parallel Converter (SPC) is used to convert the series of N symbols $\mathbf{X} = [X_n]_{n=0}^{N-1}$ into the input of the Inverse Fast Fourier Transform (IFFT), which obtains a corresponding set of N complex TD samples of the OFDM symbol $\mathbf{x} = [x_n]_{n=0}^{N-1}$. Next, the TD OFDM symbol $\mathbf{x}$ is concatenated with L samples provided by the corresponding Cyclic Prefix (CP) $[x_n]_{n=N}^{N+L-1}$, in order to avoid the inter-symbol interference associated with dispersive channels [27]. Finally, the resultant turbo-encoded, OFDM-modulated symbol is passed through a Parallel to Serial Converter (PSC) and transmitted using a Digital to Analogue Converter (DAC) and a Radio Frequency (RF) front end.

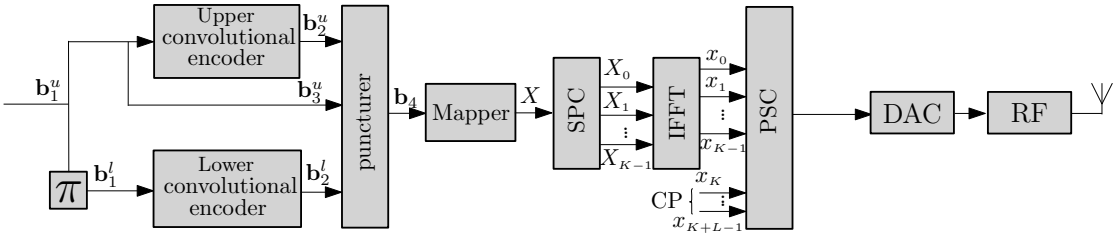


FIGURE 3.6: Transceiver schematic of a turbo-coded OFDM communication system.

3.3.2 Receiver

The proposed receiver schematic is depicted in Figure 3.7, where the signal is received using a RF front end, an Analogue to Digital Converter (ADC), a CP remover, and a SPC. These components have naturally low latencies compared to the rest of the schematic, which comprises a novel cumulative FFT, a bank of N novel QAM demappers and the FPTD [24]. The operations of the cumulative FFT, the N QAM demappers and the FPTD are spread over a total of C clock cycles. Figure 3.7 illustrates a 'toy' example, in which $C = 4$ clock cycles are used to recover $K = 8$ bits from $N = 16$ samples of the 4QAM-modulated OFDM symbol.

After the removal of the CP, the received samples of the OFDM symbol can be expressed as

$$\mathbf{y} = \mathbf{h} * \mathbf{x} + \mathbf{n}, \tag{3.4}$$

where $\mathbf{h} = [h_n]_{n=0}^{N-1}$ is the Channel Impulse Response (CIR) and $\mathbf{n} = [n_n]_{n=0}^{N-1}$ is the Additive White Gaussian Noise (AWGN), which has a zero mean and a variance of $\sigma^2 = 1/(2\gamma)$, where γ denotes the Signal to Noise Ratio (SNR). The corresponding FD

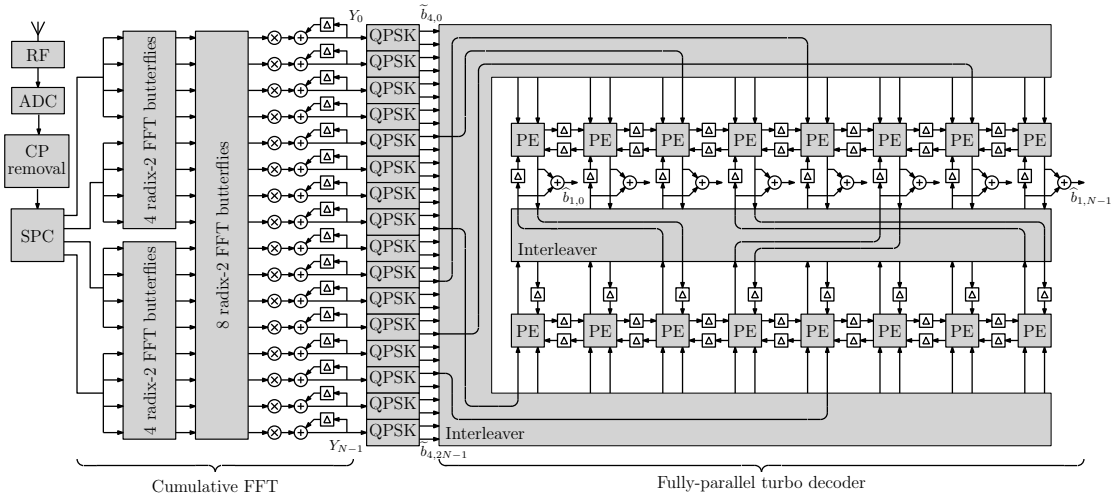


FIGURE 3.7: A toy example of the proposed ultra-low-latency architecture for the concurrent receive, FFT and turbo decode approach of Figure 3.2(b).

signal obtained after the FFT operation can be expressed as

$$\begin{aligned}\mathbf{Y} =& \boldsymbol{\mathcal{F}}\{\mathbf{h} * \mathbf{x} + \mathbf{n}\} \\ =& \mathbf{HX} + \mathbf{N}.\end{aligned} \tag{3.5}$$

Here, for the kth element in $\mathbf{Y}$, we have $Y_k = H_k X_k + N_k$, where H_k is the single tap channel gain of the corresponding QAM symbol X_k. However, instead of waiting to receive all N samples of the OFDM symbol before performing the FFT operation, the novel approach of Figure 3.7 performs the cumulative FFT during each clock cycle, while the TD samples are still being received. The N/C samples of the OFDM-modulated symbol received in each clock cycle comprises the fraction $1/C$ of the total number of samples N. Within the same clock cycle, these N/C TD samples are immediately forwarded to the cumulative FFT, which updates its output symbols $\mathbf{Y}$ by incorporating these samples. More specifically, the cumulative FFT effectively calculates an FFT of all samples received so far, while assuming zero values for the remaining samples in the OFDM symbol that have not yet been received. The operation of the cumulative FFT may be understood by decomposing (3.1) in terms of the TD samples received in each clock cycle, as follows.

$$\begin{aligned}Y_z &= \sum_{n=0}^{N-1} y_n e^{-j2\pi nz/N} \\ &= \sum_{c=0}^{C-1} \sum_{m=0}^{N/C-1} y_{\frac{N}{C}c+m} e^{-j2\pi z(\frac{N}{C}c+m)/N} \\ &= \sum_{c=0}^{C-1} e^{-j2\pi zc/C} \sum_{m=0}^{N/C-1} y_{\frac{N}{C}c+m} e^{-j2\pi zm/N}\end{aligned} \tag{3.6}$$

where $y_{\frac{N}{C}c+m}$ is the mth TD sample received in the cth clock cycle. This decomposition reveals that the operation of the FFT over all the N received symbols is equivalent to performing an N/C-point FFT over each set of N/C TD samples received in each clock cycle and performing a weighted sum of the results across the C clock cycles.

Figure 3.5 shows that if either of the two inputs of a radix-2 butterfly adopts a value of zero, then its two applied outputs adopt the value of the other input, but with different phase shifts. This observation is exploited in the proposed cumulative FFT, where the N/C TD samples received in each clock cycle are evenly distributed by the input interleaver to provide only one non-zero value among each set of C adjacent nodes. Therefore, the output of the first $\log_2 C$ stages would simply be the replicas of the non-zero values, but with different phase shifts applied. By exploiting this, only $\log_2(N/C)$ layers of radix-2 butterflies are required in order to perform the N-point FFT operation, where multipliers $\otimes$ are employed to shift the phase $e^{-j2\pi zc/N}$ of the N resultant symbols. Following this, adders $\oplus$ and registers $\boxed{\triangle}$ are used to accumulate the results obtained in each successive clock cycle. The proposed cumulative FFT behaves as an SPC, with each successive clock cycle providing N QAM-modulated symbols of progressively higher quality, containing a diminishing level of Inter-Carrier Interference (ICI).

Each of the N QAM demappers of Figure 3.7 processes the corresponding $M = 4$QAM-modulated symbol using a novel technique, which approximately models the ICI as additional Gaussian-distributed noise. More specifically, the detection of symbols modulated using a set $\mathcal{S}$ of M constellation points begins by equalising the symbol that it is provided with in each clock cycle, converting it into soft LLRs. The *a priori* probability of the ith bit $b_{4,i}$ in symbol $X_k \in \mathcal{S}$ being 0, given the kth received symbol Y_k, is

$$P(\tilde{b}_{4,i} = 0|Y_k) = \sum_{X_k \in \mathcal{S}|X_{k,i}=0} P(X_k|Y_k) \tag{3.7}$$

According to Bayes' rule, we have

$$P(X_k|Y_k) = \frac{P(X_k)P(Y_k|X_k)}{P(Y_k)} \tag{3.8}$$

For the case of the single tap channel characterised in (3.5), we have

$$\begin{aligned} P(Y_k|X_k) &= \frac{1}{\sqrt{2\pi}\sigma_r} \exp\left(-\frac{1}{2\sigma_r^2}\left\|Y_k - H_k X_k\right\|^2\right) \\ &\propto \exp\left(-\frac{1}{2\sigma_r^2}\left\|Y_k - H_k X_k\right\|^2\right), \end{aligned} \tag{3.9}$$

where $\sigma_r^2 = \frac{N+(N-D)\gamma\sum_{k=0}^{N-1}\|H_k\|^2}{2D\gamma}$, as shown in Appendix A. Here, D denotes the number of QAM symbols received in each clock cycle. The LLR of the ith bit in symbol X_k can now be expressed as

$$\begin{aligned}\mathrm{LLR}(\tilde{b}_{4,i}) &= \log\frac{P(\tilde{b}_{4,i}=0|Y_k)}{P(\tilde{b}_{4,i}=1|Y_k)} \\ &= \log\frac{\sum_{X_{k,0}\in\mathcal{S}_0}\frac{1}{M\sigma_r}\exp(-\frac{1}{2\sigma_r^2}\|Y_k - H_kX_{k,0}\|)}{\sum_{X_{k,1}\in\mathcal{S}_1}\frac{1}{M\sigma_r}\exp(-\frac{1}{2\sigma_r^2}\|Y_k - H_kX_{k,1}\|)},\end{aligned} \tag{3.10}$$

where the symbol set $\mathcal{S}_0$ comprises all constellation points in $\mathcal{S}$ that imply 0 values for the ith bit $\tilde{b}_{4,i}$, while $\mathcal{S}_1$ comprises the constellation points that imply a value of 1 for the ith bit $\tilde{b}_{4,i}$. The max* operator [19] may then be employed to simplify the calculation of (3.10), according to

$$\begin{aligned}\max^*(a,b) &= \log(e^a + e^b) \\ &= \max(a,b) + \log(e^a + e^b) \\ &\approx \max(a,b).\end{aligned} \tag{3.11}$$

Therefore, the LLR of $b_{4,i}$ can be expressed as

$$\mathrm{LLR}(\tilde{b}_{4,i}) = \max_{X_{k,0}\in\mathcal{S}_0}{}^*\left(-\frac{1}{2\sigma_r^2}\|Y_k - H_kX_{k,0}\|\right) - \max_{X_{k,1}\in\mathcal{S}_1}{}^*\left(-\frac{1}{2\sigma_r^2}\|Y_k - H_kX_{k,1}\|\right). \tag{3.12}$$

Then, the resultant set of $N\log_2 M$ LLRs is distributed among the inputs of the FPTD by its interleaver of Figure 3.7. As described in [24], the FPTD comprises two rows of N concurrently-operated Processing Elements (PEs), allowing it to process all $N\log_2 M$ LLRs provided in each clock cycle, in contrast to a conventional turbo decoder. Each PE uses registers $\boxed{\triangle}$ to iteratively exchange LLRs with its neighbouring PEs in the same row, as well as with a corresponding PE in the other row, via a second interleaver. The quality of the iteratively exchanged LLRs improves in each clock cycle, until final LLR decisions are obtained for the N bits, with K being the number of message bits, using additions $\oplus$ in the final clock cycle.

3.4 Validation

In this section, we validate the correctness of our cumulative FFT and ICI-aware soft QAM demapper by confirming that the resultant LLRs satisfy the consistency condition

given in [66]. More specifically, given 2 real values random variables X and Y, the Mutual Information (MI) between X and Y is defined as

$$I(X;Y) = \int\int f(x,y)\log\frac{f(x)}{f(x)}\mathrm{d}x\mathrm{d}y, \tag{3.13}$$

with $I(X;Y) = H(Y) - H(Y|X)$, where

$$H(Y|X) = \int\int f(x,y)\log\frac{1}{f(y|x)}\mathrm{d}x\mathrm{d}y, \tag{3.14}$$

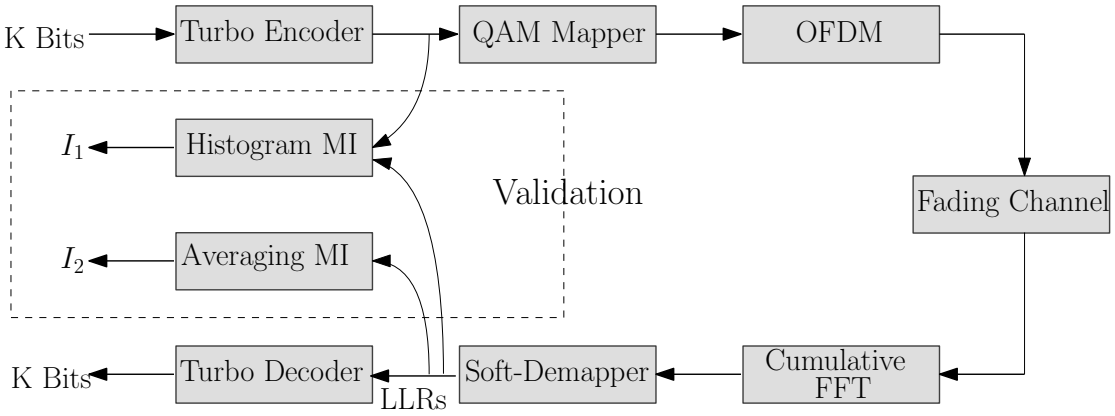


FIGURE 3.8: Validation using the histogram and averaging methods of MI calculation.

Two methods for measuring the MI of LLRs are proposed in [66], referred to as the averaging method and the histogram-based method. While the histogram method computes the MI of LLRs by comparing them to the correct bit values, the averaging method does not consider the correct bit values. Instead, it assumes that the LLRs satisfy the consistency condition and that the MI can be correctly computed based on the magnitudes of the LLRs alone. If a vector of LLRs satisfies the consistency condition, then the averaging method will measure the same MI value as the histogram method, which does not assume consistency. Figure 3.8 illustrates the employment of the averaging and histogram methods of calculating the MI to validate our proposed approach. The results of the comparisons are shown in Figures 3.9 and 3.10, where 4QAM and 16QAM are employed, respectively. As shown in Figures 3.9 and 3.10, both methods of measuring the MI give similar results across a variety of different E_b/N_0 values, and as successively more symbols D are received. This confirms that the LLRs satisfy the consistency condition and validates the accuracy of the proposed approach. As expected, higher E_b/N_0 values and more received symbols D result in higher LLR reliability, whereas higher-order QAM decreases the LLR reliability, at a given E_b/N_0 value.

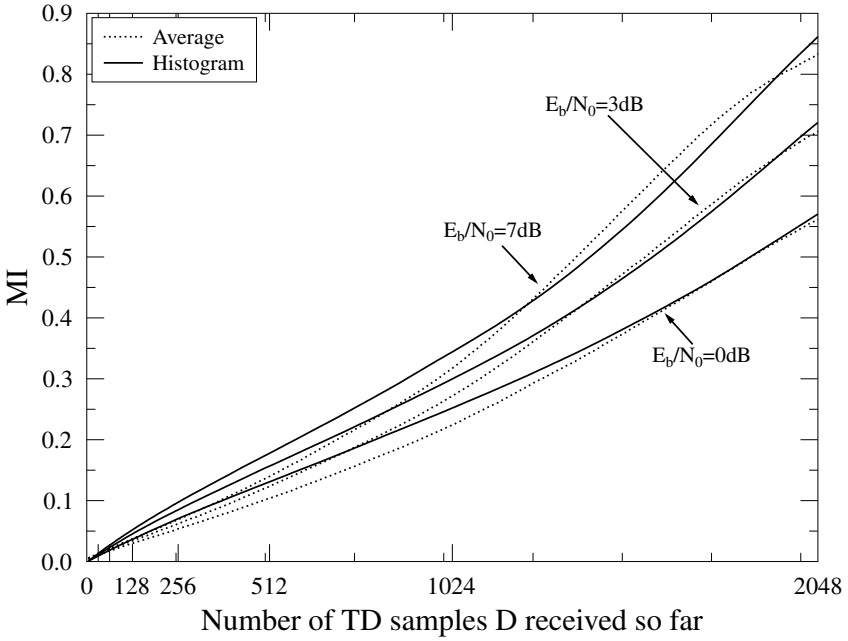


FIGURE 3.9: MI calculated by histogram and averaging methods, when employing a punctured LTE turbo code, Gray-coded 4QAM, OFDM and a quasi-static Extended Typical Urban model (ETU) Rayleigh fading channel, where $K = 1376$ and $N = 2048$.

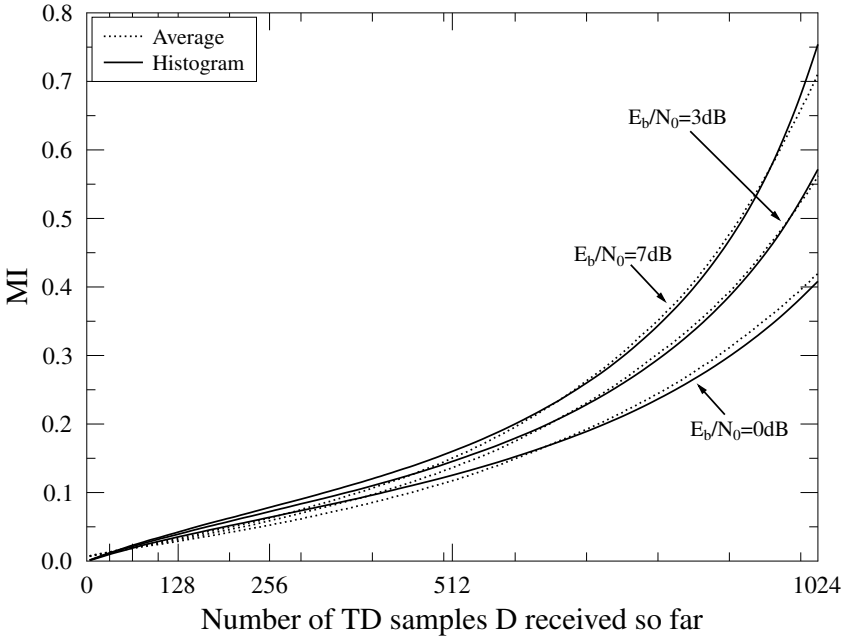


FIGURE 3.10: MI calculated by histogram and averaging methods, when employing a punctured LTE turbo code, Gray-coded 16QAM, OFDM and a quasi-static ETU Rayleigh fading channel, where $K = 1376$ and $N = 1024$.

3.5 Performance analysis

In this section, we present and benchmark the performance of the proposed concurrent OFDM demodulation and turbo decoding architecture. Figures 3.11 and 3.12 characterise the performance of the proposed scheme and that of a benchmarker for the case of a punctured LTE turbo code, Gray-coded QAM, OFDM and a quasi-static Extended Typical Urban model (ETU) Rayleigh fading channel [86], where $K = 1376$, $N = 2048$ for 4QAM, as shown in Figure 3.11 and $N = 1024$ for 16QAM, as shown in Figure 3.12. Here, the concurrent receive, FFT and turbo decode approach of Figure 3.2(b) employs the architecture proposed in Figure 3.7 and $C \in \{128, 256, 512, 1024\}$ clock cycles. This is compared to a benchmarker employing the serial receive, FFT and turbo decode approach of Figure 3.2(a), when employing a conventional turbo decoder. As the number of clock cycles C is increased, the BER performance of the proposed scheme can be seen to converge to that of the benchmarker, proving the concept of the concurrent receive, FFT and turbo decode approach. A similar result may be observed when employing the higher-order QAM, where higher bandwidth efficiency is obtained at the cost of degraded BER performance. However, for the 4QAM modulation scheme, the proposed architecture requires up to $C = 1024$ clock cycles in order to closely match the performance of the benchmarker, while even more clock cycles are required for the 16QAM scheme to approach the benchmarker. A significant reduction in processing energy consumption could be achieved upon employing only $C = 128$ clock cycles, although this is associated with a performance loss of up to 2.5 dB, compared to the benchmarker.

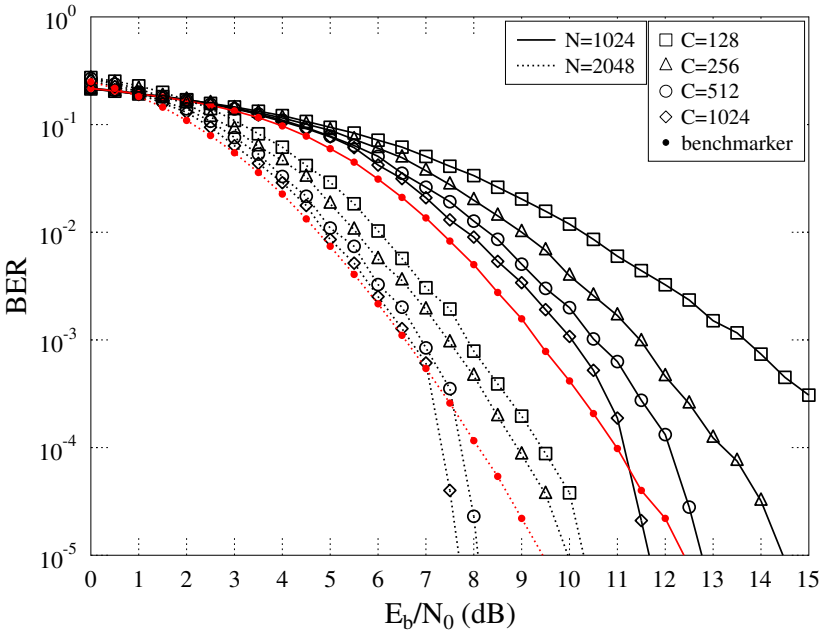


FIGURE 3.11: BER of the proposed concurrent architecture, when using $C \in \{128, 256, 512, 1024\}$ and employing a punctured LTE turbo code, Gray-coded QAM, OFDM and a quasi-static ETU Rayleigh fading channel, where $K = 1376$, $N = 1024, 2048$.

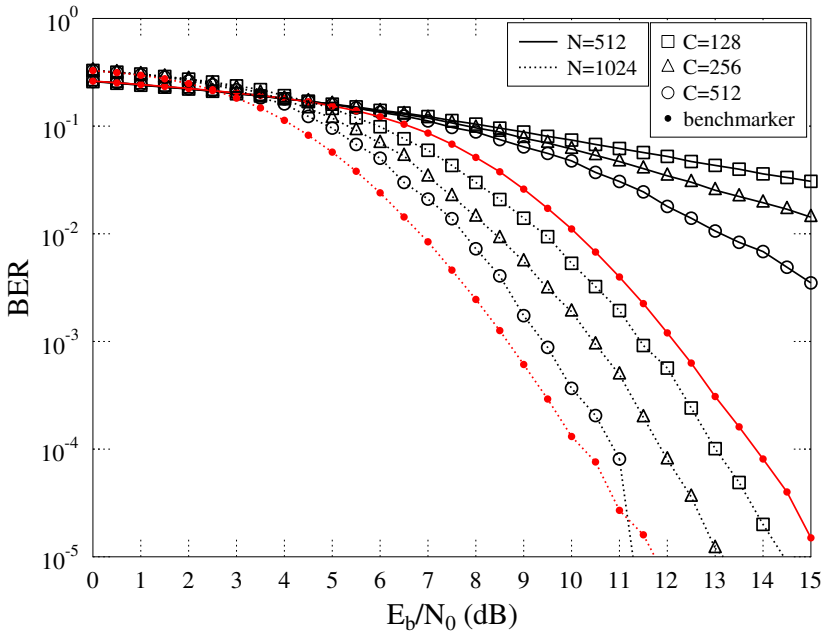


FIGURE 3.12: BER of the proposed concurrent architecture, when using $C \in \{128, 256, 512\}$ and employing a punctured LTE turbo code, Gray-coded 16QAM, OFDM and a quasi-static ETU Rayleigh fading channel, where $K = 1376$, $N = 512, 1024$.

3.6 System refinements

In order to mitigate this performance loss, the architecture of Figure 3.7 may be further refined. Therefore, in the following subsections, we propose a pair of refinements for improving the BER performance of the proposed architecture, referred to as the staggered receive approach and scaled approach, respectively.

3.6.1 Staggered receive approach

In our first refinement, referred to as the staggered receive, FFT and turbo decode approach, the operation of the FPTD may be staggered relative to that of the cumulative FFT of Figure 3.7. More specifically, the operation of the FPTD maybe delayed until $S \in [0, N]$ symbols have been received, facilitating a gradated trade-off between latency and processing energy consumption. Figures 3.13 and 3.14 show that upon adopting this approach, $C = 128$ clock cycles and a stagger of $S = 3N/8$ symbols is sufficient for closely matching the BER performance of the benchmarker, when employing both 4QAM and 16QAM.

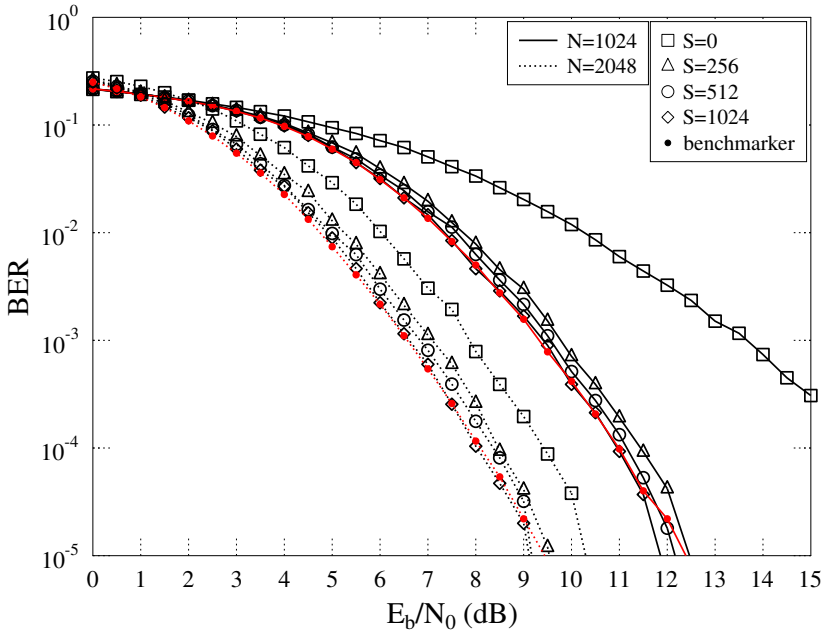


FIGURE 3.13: BER of the staggered architecture, when using $C = 128$ and $S \in \{0, 256, 512, 1024\}$, employing a punctured LTE turbo code, 4QAM, OFDM and a quasi-static ETU Rayleigh fading channel, where $K = 1376$, $N = 1024, 2048$.

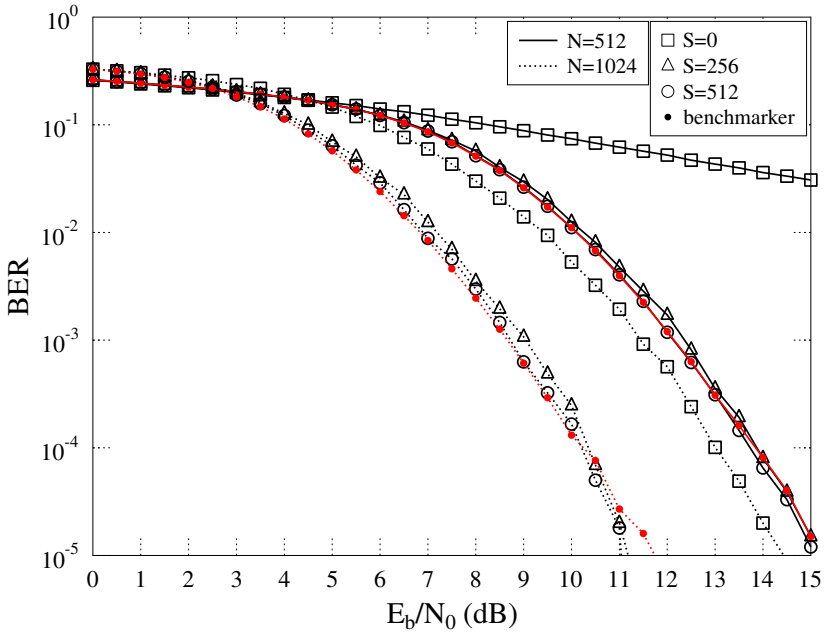


FIGURE 3.14: BER of the staggered architecture, when using $C = 128$ and $S \in \{0, 256, 512\}$, employing a punctured LTE turbo code, 16QAM, OFDM and a quasi-static ETU Rayleigh fading channel, where $K = 1376$, $N = 512, 1024$.

3.6.2 Scaled approach

In a second refinement, referred to as the scaled concurrent receive FFT and turbo decode approach, the BER performance loss can also be mitigated by reducing the weighting of the ICI-dominated LLRs provided by the soft demappers during the early clock cycles. This can be achieved by applying a gradually increasing scaling factor approached to the LLRs in successive clock cycles according to an exponential function $y = \exp x$. When the scaled approach is applied together with the staggered reception, Figure 3.15 shows that decreasing the weighting of LLRs provided in early cycles improves the overall BER performance by 0.2 dB at a BER level of 10^{-5}.

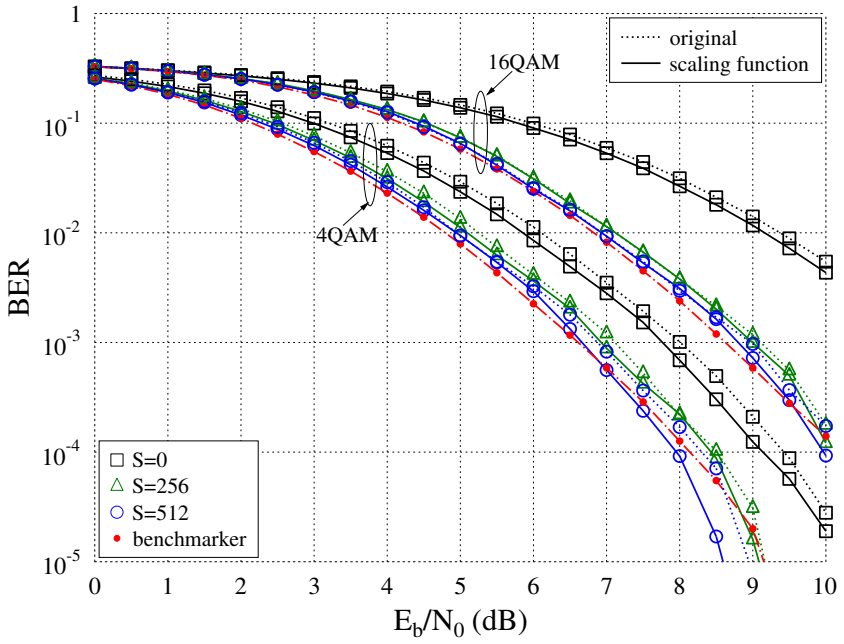


FIGURE 3.15: BER of the proposed architecture with an exponential scaling factor function, when using $S \in \{0, 256, 512\}$ and employing a punctured LTE turbo code, Gray-coded QAM, OFDM and a quasi-static ETU Rayleigh fading channel, where $K = 1376$, $N = 2048$ for 4QAM and $N = 1024$ for 16QAM.

By designing the proposed architecture of Figure 3.7 to implement the staggered and/or scaled receive, FFT and turbo decode approach using $C = 128$ and a clock frequency of at least 176 MHz, sub-microsecond physical layer latencies will be facilitated for applications such as low-latency capital market trading or robotic cars. In applications such at LTE, where a transmission latency of 70 μs is imposed, the operation of the cumulative FFT and the FPTD can be spread over this duration, in order to achieve a significantly improved hardware resource efficiency or processing energy consumption.

3.7 Conclusions

TABLE 3.1: Comparison of different receiving approaches in terms of BER performance and latency.

	BER performance	Latency
Serial approach benchmarker	Good	210 μs
Concurrent approach without scaling	Up to 2.5 dB loss	70 μs
Concurrent approach with scaling	Approaching benchmarker	70 μs

In this chapter, we proposed a concurrent OFDM demodulation and turbo decoding architecture for significantly reducing the associated processing latency. Rather than completing the reception, FFT and turbo decoding operations one after another, these three processes are performed concurrently in the proposed approach. In this way, the latency requirement of URLLC for the next generation communication can be satisfied, enabling the URLLC applications in different fields such as remote surgery, automated driving, and virtual reality, as summarised in Table 3.1. In order to allow the trade off between latency and complexity to be adjusted and to improve the associated BER performance, we proposed staggered and scaled refinements to the proposed approach.

Chapter 4

CRC-aided Logarithmic Stack Decoding of Polar Codes

4.1 Introduction

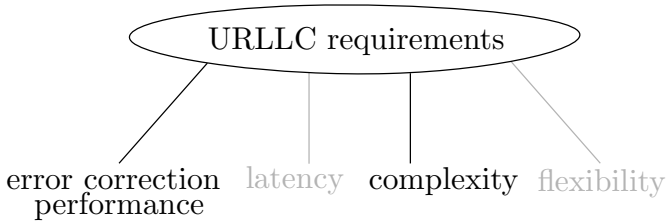


FIGURE 4.1: The contribution of Chapter 4.

While the turbo codes of Chapters 2 and 3 and Low-Density Parity-Check (LDPC) codes require iterative decoding techniques to achieve near-capacity communication, Arikan's polar transform allows the capacity to be achieved using simple encoder and decoder operations, having $O(N \log N)$ complexity, where N denotes the length of a polar code. The power of polar codes is a benefit of the polarization phenomenon: in contrast to conventional communication channels that have capacities somewhere between 0 or 1, Arikan's polar transform synthesises channels that have capacities converging to either 0 or 1, where coding becomes trivial. With the aid of the Cyclic Redundancy Check (CRC) codes and advanced decoding algorithms, polar codes can attain superior error-correction performance than the more complex turbo and LDPC codes, particularly at short block lengths N. Owing to their attractive error correction performance at

short block lengths, polar codes [8, 31] have been selected for protecting the control channels of the 3rd Generation Partnership Project (3GPP) New Radio (NR) candidate for Fifth Generation (5G) mobile communications [32]. However, compared to turbo and LDPC codes, polar codes are far less mature and a de-facto standard approach to their implementation has not yet emerged. In particular, the application of polar codes to wireless fading channels at ultra high throughputs and with strong error-correction remains unsolved, which has prevented the application of polar codes in the 5G data channels. More specifically, the construction of a polar code involves the identification of channel reliability values associated with each bit to be encoded. This identification can be effectively performed given a code length and a specific Signal to Noise Ratio (SNR). However, within the 5G framework, various code lengths, rates and channel conditions are foreseen, and having a different reliability vector for each parameter combination is infeasible. This makes it difficult to implement flexible, high performance polar decoders.

The existing decoding algorithms of the polar codes fall into two categories, which operate either on the basis of Successive Cancellation (SC) [8, 33], or Belief Propagation (BP) [34–36,87]. Originally, the low-complexity SC [8,33] decoding algorithm was proposed for polar codes, and was simplified in [37] to further reduce the complexity and the latency. However, more sophisticated decoding algorithms [39,42,44,46,47,51,88–97] are required for achieving near capacity performance in practical wireless channels, as summarised in Figure 4.2. Rather than considering only the locally most-likely value for each successive bit, as in SC decoding, the Successive Cancellation List (SCL) decoder [39–41] uses a breadth-first search for identifying the L number of locally most-likely bit values, allowing it to more frequently spot the globally most-likely values. By contrast, the Successive Cancellation Stack (SCS) decoder [42,92] uses a depth-first approach to directly search for the globally most-likely values, although its ability to achieve this depends on the number of decoding candidates S that can be stored within the memory available for the stack. Note that while all L decoding candidates will grow to the same full length during the processing of the SCL algorithm at high channel Signal to Noise Ratios (SNRs), many of the S candidates in the SCS algorithm typically only reach much shorter lengths, hence reducing the complexity of the SCS algorithm below that of the SCL algorithm, approaching that of the SC algorithm. Typical benchmarkers of the SC-based decoders are listed in in Table 4.1.

The error correction performance of polar codes can be further improved by concatenating them with CRC codes [40, 88, 93, 96, 98–104], as in 3GPP NR. During SCL or SCS decoding, the specific decoding candidates that do not satisfy the CRC can be discarded, even if this would otherwise appear to offer the globally most likely bit values. As an additional benefit, the CRC can enable error detection, although this ability is degraded by using the CRC to aid error correction in the manner described above. Furthermore, the distributed CRC technique allows early termination of the SCL or SCS decoding, further reducing the decoding complexity [91, 92, 105].

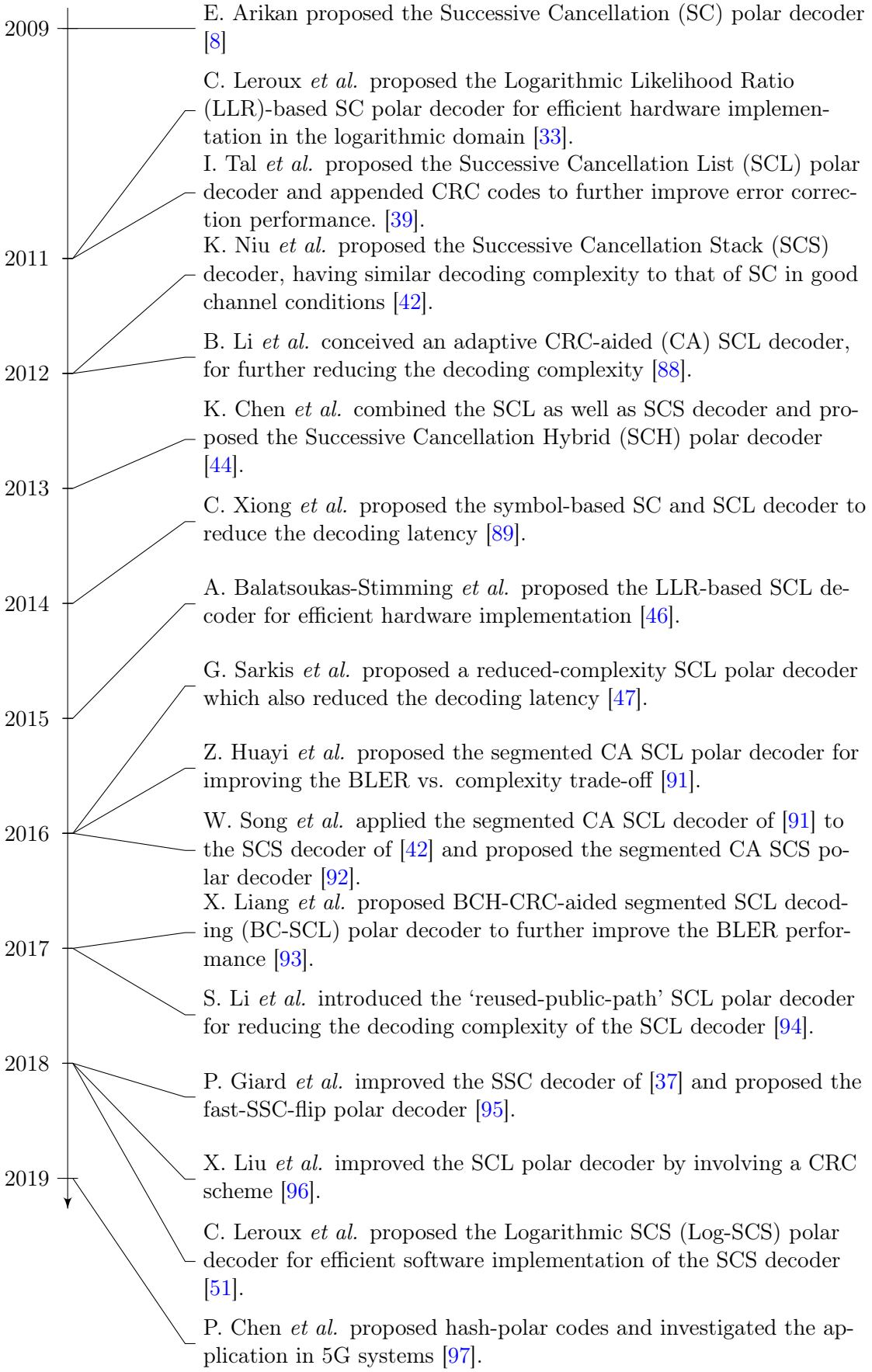


FIGURE 4.2: Timeline of the development of SC-based polar decoders.

TABLE 4.1: Summary of the error correction capability, computational complexity and memory requirement of various polar decoding algorithms. Specific measurements are provided for the example of $A = 84, G = 272$, where the error correction capability is measured by the E_s/N_0 gap to the finite block-length capacity bound at BLER of 10^{-3}, while the computational complexity is measured by the number of f, g and ϕ operations. In terms of the memory requirement, 1 byte is assumed for the storage of each bit probability and LLR.

	Novelty	Error correction capability	Computational complexity	Memory requirement
SC [8]	Originally proposed for polar decoding	poor (gap: 2.46 dB)	low (2048 operations)	low (0.56 KB)
SCL [39] (L=8)	Breadth-first search for L candidates	good (gap: 0.86 dB)	high (6.34 times higher than the SC)	medium (4.91 times higher than the SC)
SCS [42] (S=1024)	Depth-first search for S candidates	good (gap: 0.85 dB)	medium (3.17 times higher than the SC)	high (571.88 times higher than the SC)
Log-SCL [46] (L=128)	Extend the SCL to the logarithmic domain	good (gap: 0.70 dB)	high (174995 operations)	medium (40.25 KB)
Log-SCS [51] (S=1024)	Extend the SCS to the logarithmic domain	good (gap: 0.71 dB)	medium (21.31 times lower than the Log-SCL)	high (6.97 times higher than the Log-SCL)
Improved Log-SCS (S=128)	**Two techniques to further reduce the memory requirement and complexity**	good (gap: 0.71 dB)	low (41.67 times lower than Log-SCL)	medium (same as the Log-SCL)

However, the conventional SCL [39] and SCS [42] decoders operate on the basis of bit probabilities, which have a high dynamic range and suffer from poor numerical stability, even in double precision floating point implementations. In order to address this problem, the authors of [46] proposed a Logarithmic Successive Cancellation List (Log-SCL) decoder that operates on the basis of Logarithmic-Likelihood Ratios (LLRs), which enables low-complexity fixed point implementation and reduced storage requirements, owing to the low dynamic range of LLRs [47–50]. In addition, [51] proposed a Logarithmic Successive Cancellation Stack (Log-SCS) decoder, which uses LLRs to bring the same advantages to the SCS decoder.

Motivated by this, we apply the Log-SCS algorithm to the polar code of the Physical Uplink Control Channel (PUCCH) and Physical Uplink Shared Channel (PUSCH) of the recently standardised Ultra-Reliable Low Latency Communication (URLLC) version of 3GPP NR [57]. Similar to [51], we provide a tutorial on how the Log-SCS algorithm may be obtained by transforming the computations of the SCS algorithm into the logarithmic domain, such that it can operate on the basis of LLRs. Furthermore, we provide a tutorial on how the Log-SCS algorithm improves on the SCS by considering the frozen bits, when determining the most likely sequence of information bits, which improves the error correction performance and reduces the decoding complexity. In addition, during the exploitation of the CRC codes to improve the error correction performance, we propose a novel technique for limiting the number of CRC checks performed in stack decoding, in order to maintain a consistent error detection performance. In this application, we demonstrate that the application of the Log-SCS decoder to the 3GPP NR polar code of PUCCH and PUSCH offers as much as 0.5 dB improved error correction performance, compared to the previous SCS algorithm using the same stack size of $S = 8$.

Additionally, we propose a pair of novel techniques for further improving the performance of the Log-SCS polar decoder. We show that across the full range of block lengths supported by NR PUCCH and PUSCH, the proposed $S = 128$ Improved Log-SCS decoder achieves a similar error correction capability as the $L = 128$ Log-SCL decoder. This is achieved despite dramatically reducing its complexity by up to seven times compared to a $L = 8$ Log-SCL decoder and without increasing its memory requirement. Owing to this, the proposed Improved Log-SCS decoder offers practical ultra-reliable error correction within as little as 0.5 dB of the channel capacity bound. Hence, it is particularly well-suited to the Ultra-Reliable Low Latency Communication (URLLC) mode of 3GPP NR.

As shown in Figure 4.3, this chapter is structured as follows. Section 4.2 provides a tutorial on the construction and the Log-SCS decoding of the 3GPP NR polar code for PUCCH and PUSCH. Following this, the performance versus complexity and memory requirements of the CRC-aided Log-SCS decoder are compared to those of several benchmarkers in Section 4.3. Then, our two novel improvements of the CRC-aided Log-SCS decoder are proposed in Section 4.4, where the error correction performance versus

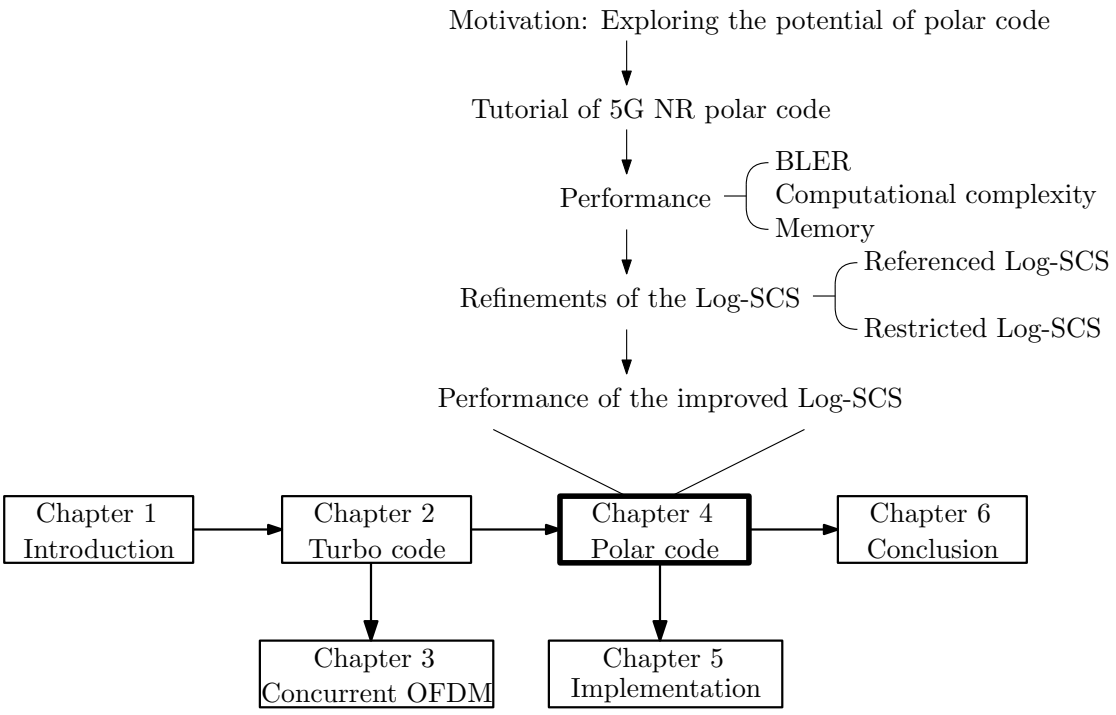


FIGURE 4.3: The development of Chapter 4.

complexity and memory resource requirement of the Improved Log-SCS decoder are also compared to those of the SCS and log-SCL decoders. Finally, we offer our conclusions and suggest avenues for future work in Section 4.5.

4.2 Overview of the 3GPP NR Uplink Polar Codes

This section provides an overview of the 3GPP NR uplink polar codes [57], with a particular focus on the CRC-aided code used in the PUCCH and PUSCH channels, when the information block length is in the range of $A \in [20, 1706]$ bits[1]. Figure 4.4 illustrates the components of the 3GPP NR uplink polar encoder and decoder, each of which is discussed in the following subsections. More specifically, Sections 4.2.1 to 4.2.6 detail the code block segmentation and concatenation, CRC generation, appending and CRC check, frozen bit insertion and removal, polar encoder and decoder core operation, rate matching and dematching, as well as channel interleaving and deinterleaving, respectively.

[1]Note that a Parity Check (PC) code is concatenated with the polar code in the case of PUCCH and PUSCH with information block lengths in the range $A \in [12, 19]$ bits [57]. We will apply our proposed Log-SCS to this case in future work.

(a) Polar encoder in transmitting user equipment

(b) Polar decoder in receiving base station

FIGURE 4.4: Schematic of the 3GPP NR uplink polar encoder and decoder.

4.2.1 Code block segmentation and concatenation

If the information block length A and the encoded block length G satisfy $A \geq 1013$ OR ($A \geq 360$ AND $G \geq 1088$), then the information block input to the polar encoder of Figure 4.4 is first decomposed into two segments, as shown in Figure 4.5, for halving the decoding latency. Here, the first segment comprises the first $\lfloor A/2 \rfloor$ bits of the information block, but prepended with an additional zero-valued padding bit if A is odd, giving a total of $A' = \lceil A/2 \rceil$ bits. Meanwhile, the second segment is comprised of the remaining $A' = \lceil A/2 \rceil$ bits from the information block. The two information block segments are appended with CRC bits, inserted with frozen bits, encoded, rate matched and interleaved independently as discussed in the following sections. This results in a pair of encoded block segments, each comprising $E = \lfloor G/2 \rfloor$ bits, which are concatenated together and input to the modulator. If G adopts an odd value, then an additional zero-valued padding bit is appended to the block, to give a total of G bits. If $A \geq 1013$ OR ($A \geq 360$ AND $G \geq 1088$) is not satisfied, then the information block is processed as a single segment, as shown in Figure 4.5, with $A' = A$ and $E = G$. The code block segmentation strictly limits the core block length to be 1024 bits but facilitates support for the largest required information block length of $A = 1076$. In this way, the decoding complexity that increases with $N \log(N)$ is further reduced. Note that the maximum value of G supported in the NR standard is $G = 8192$ for the one-segment scenario and $G = 16385$ for the two-segment scenario. The corresponding inverse operations are performed in the polar decoder of Figure 4.4, where the padding bits are discarded.

4.2.2 CRC generation and appending

The polar code used in the NR uplink control channels for information blocks having lengths A in the range $[20, 1706]$ is protected by an 11-bit CRC code[2], having the generator polynomial $D^{11} + D^{10} + D^9 + D^5 + 1$. A set of 11 CRC bits are generated as a function of the A' information bits in each information block segment and are appended to give a sequence of $K = A' + 11$ bits [57], as shown in Figure 4.4.

Whenever the decoding core that will be detailed in Section 4.2.4 obtains a decoding candidate comprising a full set of N bits, then a CRC check may be performed in order to detect errors within the decoding candidate. If the CRC check succeeds, then the decoding candidate is output and the algorithm is terminated. However, if the CRC check fails, then a counter j that has an initial value of zero is incremented and the decoding candidate is eliminated from the candidate sets. In this chapter, we propose a novel technique of terminating the decoding algorithm if the counter reaches a value of $j = C = 8$, upon which a decoding failure is declared as will be detailed in Section 4.2.4.

[2]Note that a 6-bit CRC having the polynomial $D^6 + D^5 + 1$ is used to aid the PC polar code used for $A \in [12, 19]$ in 3GPP NR PUCCH and PUSCH.

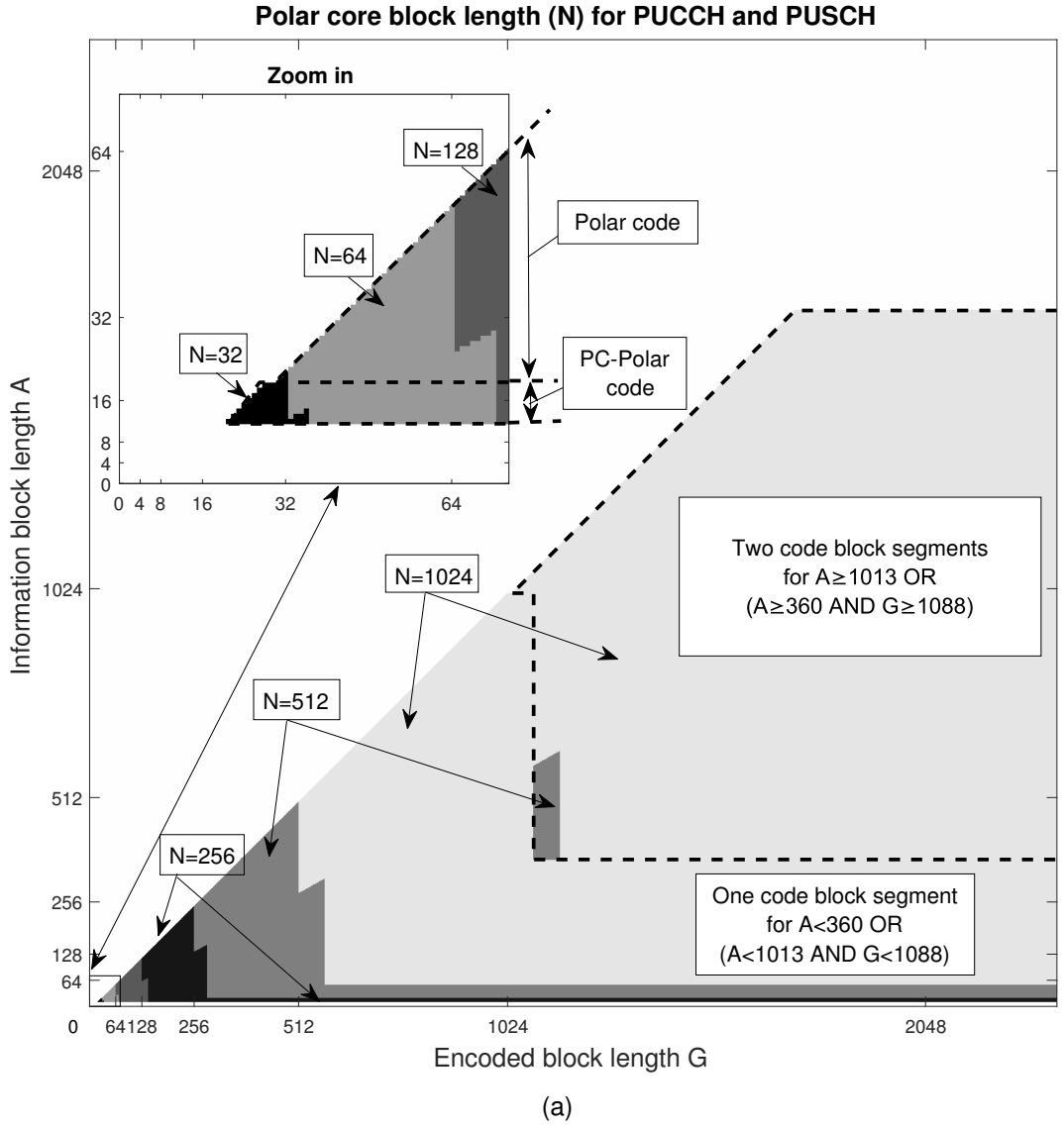


FIGURE 4.5: Polar code block segmentation and core block length N for different combinations of information and encoded block lengths in PUCCH and PUSCH of 3GPP NR.

Note that this is in contrast to the CRC-aided SCS algorithm of [98], which continues considering successive CRCs indefinitely, until a pass is found, hence resulting in a high prevalence of undetected block errors.

4.2.3 Frozen bit insertion and removal

Before invoking the polar encoder core, zero-valued frozen bits are inserted into the sequence of K information and CRC bits, in order to obtain a longer bit vector $\mathbf{u}$ comprising N bits, as shown in Figure 4.4. Here, N is a power of 2 that is higher than (or in

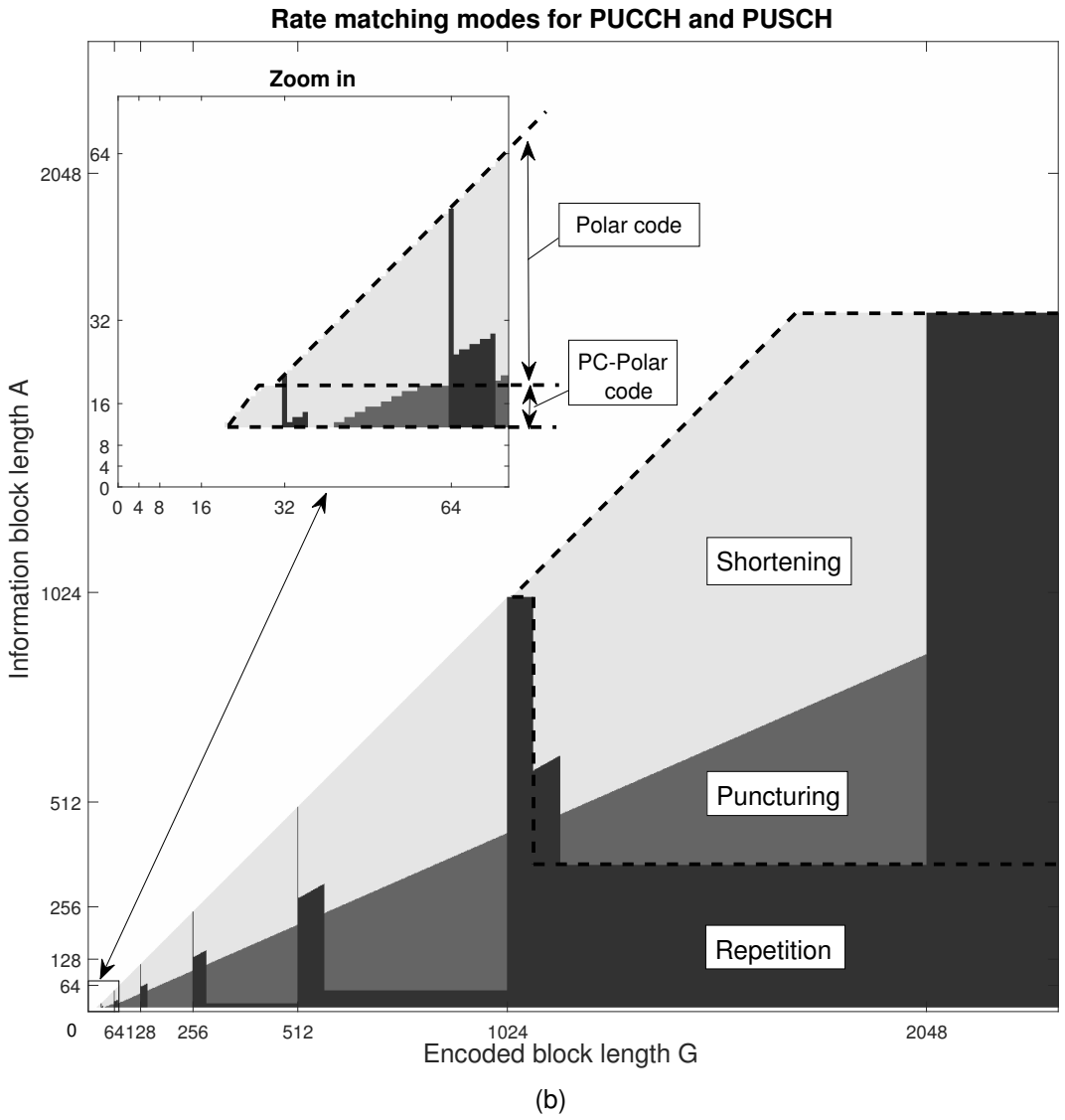


FIGURE 4.6: Rate matching modes for different combinations of information and encoded block lengths in PUCCH and PUSCH of 3GPP NR.

principle, equal to) K, which may adopt values up to $N_{\max} = 1024$ in the NR PUCCH and PUSCH, as shown in Figure 4.6. Each of the N bits can be thought of as a polarised channel. Depending on the operation of the rate matching discussed in Section 4.2.5 and detailed in [57], the K most reliable channels are used to transmit the set of K information and CRC bits, while the frozen bits are mapped to the remaining $(N - K)$ least reliable channels [57]. In the receiver, the knowledge that the frozen bits have values of zero is exploited for aiding polar decoding, as will be detailed in Section 4.2.4. Following polar decoding, the information bits are extracted from among the frozen bits, as shown in Figure 4.4.

4.2.4 Polar encoding and decoding core

As shown in Figure 4.4, the vector $\mathbf{u}$ of $N = 2^n$ bits may be polar encoded into a vector $\mathbf{x}$ of N encoded bits, according to the modulo-2 matrix multiplication

$$\mathbf{x} = \mathbf{u}\mathbf{F}_2^{\otimes n}. \tag{4.1}$$

Here, $\mathbf{F}_2^{\otimes n}$ is the generator matrix, where the superscript $\otimes n$ represents the n^{th} Kronecker power of the kernel matrix $\mathbf{F}_2$, which is given by

$$\mathbf{F}_2 = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}.$$

This operation may be represented graphically by the polar code graph of Figure 4.7, where the core information block is input on the left-hand edge of the graph and the successive stages of eXclusive-OR (XOR) operations produce the core encoded block on its right-hand edge.

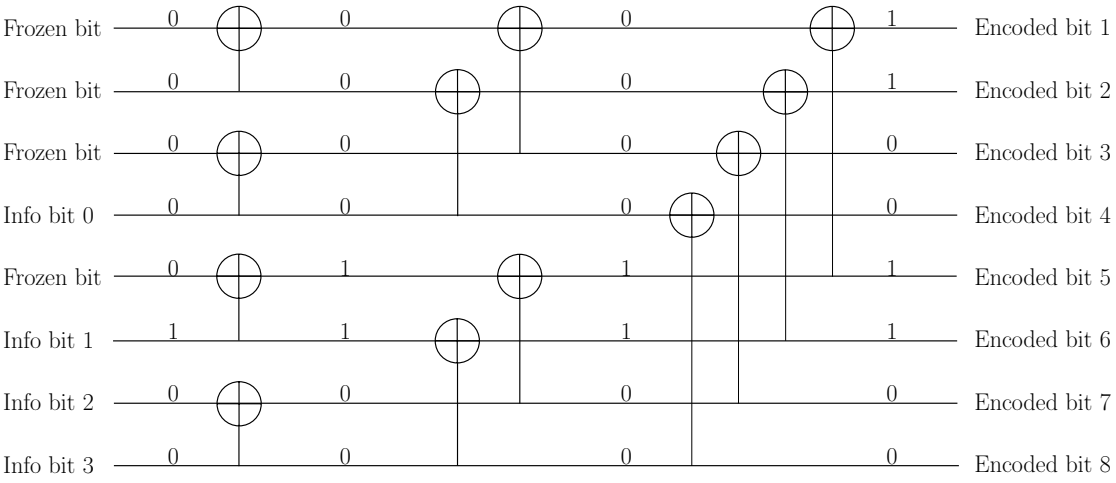


FIGURE 4.7: The XOR operations performed in the polar encoder core, for an example where $K = 4$ information bits [0100] are converted into $N = 8$ core encoded bits [11001100].

The corresponding decoding process of NR uplink polar codes may be completed by employing various algorithms including Log-SCL, SCS and Log-SCS. The operation of the Log-SCL decoding algorithm in the logarithmic domain may be applied to SCS decoding, by employing several modifications. The CRC-aided Log-SCS decoder is summarised in Algorithm 1 and compared to the SCS and Log-SCL algorithms in Figures 4.8 to 4.10. The Log-SCS decoder of [51] benefits from an improved error correction capability, despite its reduced complexity and other practical advantages compared to the CRC-aided SCS decoders of [42, 98]. These benefits are achieved in three ways. Firstly,

the CRC-aided Log-SCS decoder of [51] operates on the basis of LLRs rather than the bit probabilities of the CRC-aided SCS decoders of [42, 98]. These LLRs have improved numerical stability and reduced dynamic range compared to bit probabilities, hence facilitating fixed point implementation and other practical benefits [47–49]. Furthermore, in contrast to the SCS algorithm, the Log-SCS algorithm considers not only information and CRC bits, but also frozen bits, when determining the most likely sequence of information bits, as shown in Figure 4.8. As we shall demonstrate in Section 4.3, this decreases the decoding complexity and improves the error correction performance. As an additional refinement, this section proposes the novel technique of limiting the number of CRC checks performed during the exploitation of the CRC codes to improve the error correction performance, in order to maintain a consistent error detection performance, as detailed below.

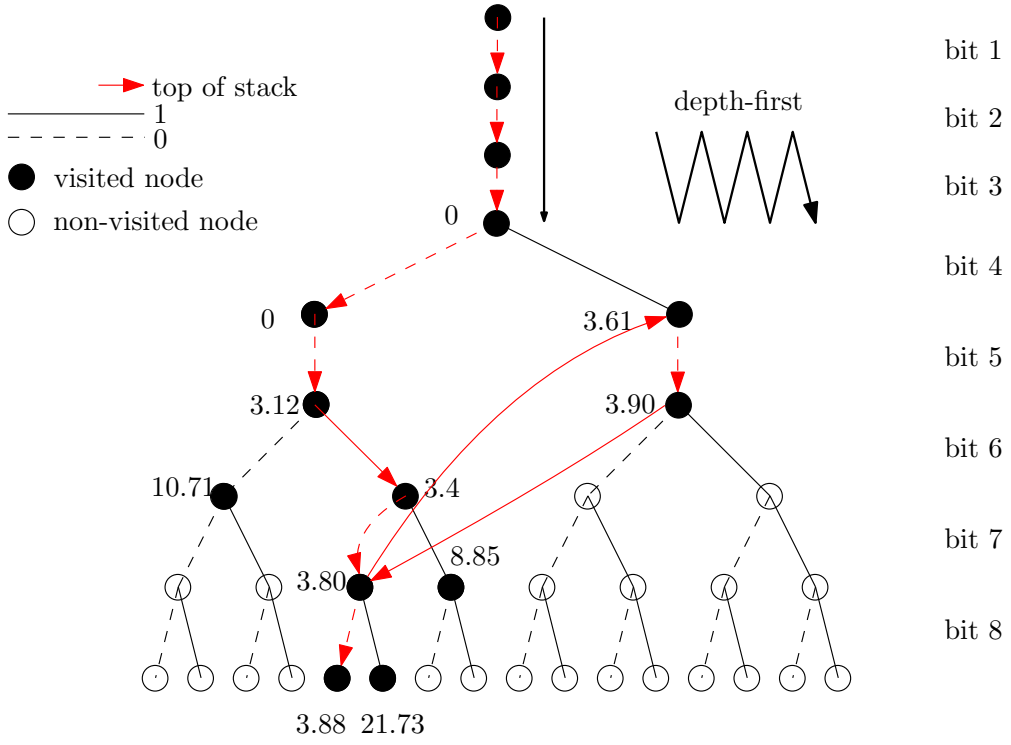


FIGURE 4.8: Code tree in $S = 2$ Log-SCS polar decoder when decoding the example $K = 4$ information bits from the $N = 8$ core encoded bits of Figure 4.7, using the exact f and ϕ functions of (4.2a) and (4.5a).

Like the Log-SCL decoder, the Log-SCS decoder operates on the basis of LLRs, where $\text{LLR} = \ln \frac{\Pr(\text{bit}=0)}{\Pr(\text{bit}=1)}$. These LLRs are combined in order to compute Path Metrics (PMs) that quantify the likelihood associated with a particular corresponding candidate sequence $[\hat{u}_i]_{i=1}^{N}$ of the N core information bits. Note that all frozen bits will adopt a value of 0 in any decoding candidate, but the information and CRC bits can adopt values of either 0 or 1. In both the Log-SCL and the Log-SCS decoders, the N bits in a decoding candidate are considered in sequential order, one at a time, with each successive bit

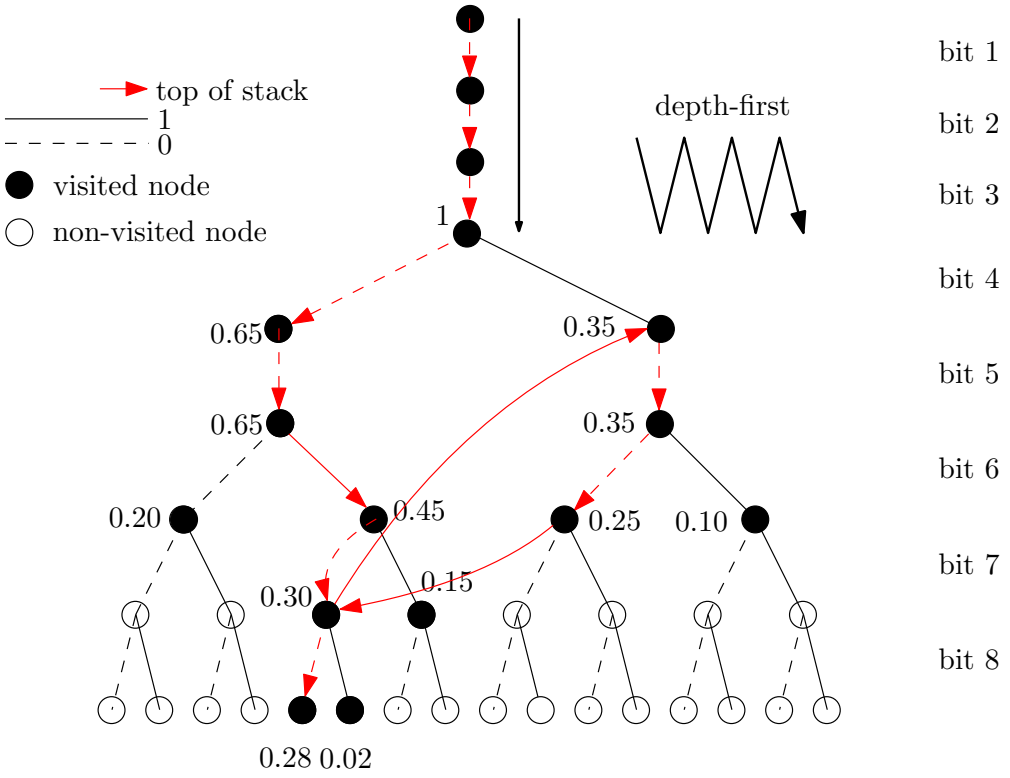


FIGURE 4.9: Code tree in $S = 2$ SCS polar decoder when decoding the example $K = 4$ information bits from the $N = 8$ core encoded bits of Figure 4.7, using the exact f and ϕ functions of (4.2a) and (4.5a).

updating the corresponding PM. During this process, the LLRs input at the right-hand edge of the polar code-graph of Figure 4.7 are combined by the XORs of the graph [106], as shown in Figure 4.11. More specifically, there are three types of computations that can be performed by a particular XOR in the graph, depending on the availability of LLRs provided on the connections at its right-hand side, as well as upon the availability of bits provided on the connections at its left-hand side [8].

The first occasion when an XOR can contribute to the processing of a decoding candidate is when an LLR has been provided for both of the connections at its right-hand side, referred to as $\tilde{x}_a$ and $\tilde{x}_b$, respectively, as illustrated in Figure 4.11(a). This enables the XOR to compute a new LLR $\tilde{x}_c$ for the first of the two connections at its left-hand side,

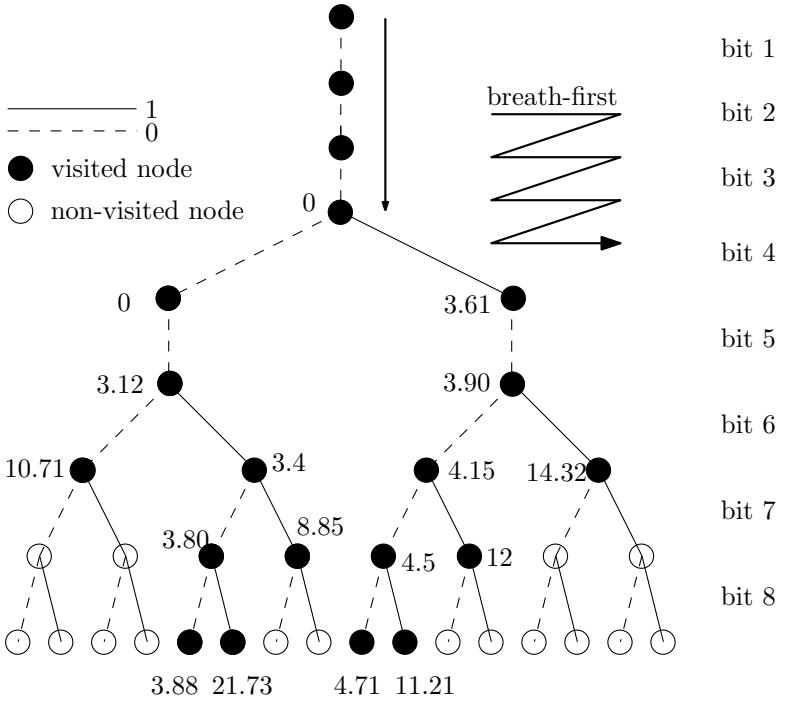


FIGURE 4.10: Code tree in $L = 2$ Log-SCL polar decoder, when decoding the example $K = 4$ information bits from the $N = 8$ core encoded bits of Figure 4.7, using the exact f and ϕ functions of (4.2a) and (4.5a).

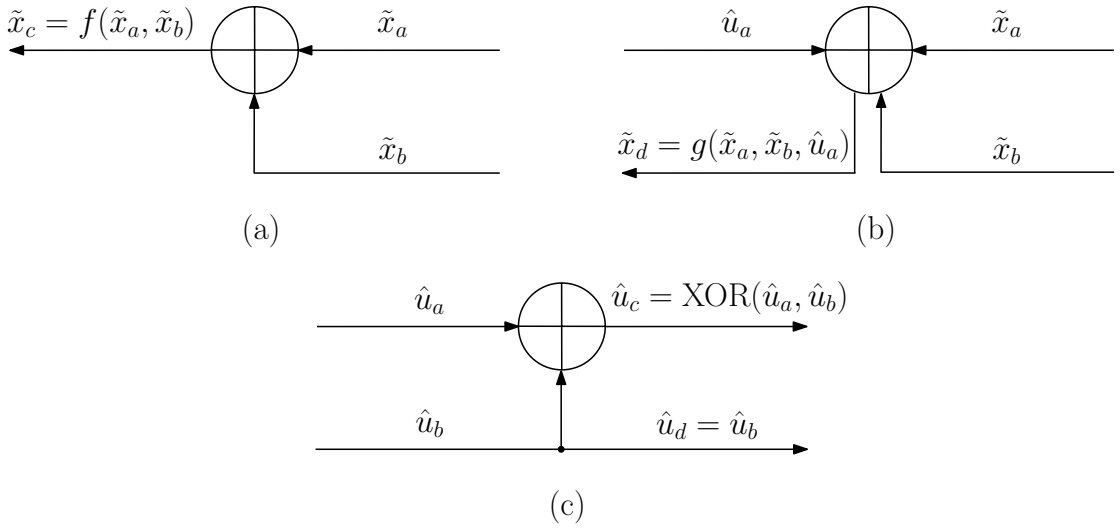


FIGURE 4.11: The three computations that can be performed for an XOR in the polar code graph: (a) the f function, (b) the g function and (c) partial sum calculation.

according to the f function [106], which can be expressed as

$$\begin{aligned}\tilde{x}_c &= f(\tilde{x}_a, \tilde{x}_b) \\ &= 2\tanh^{-1}\left(\tanh\left(\frac{\tilde{x}_a}{2}\right)\tanh\left(\frac{\tilde{x}_b}{2}\right)\right) \qquad (4.2a) \\ &= \tilde{f}(\tilde{x}_a, \tilde{x}_b) + \log(1+\exp(-|\tilde{x}_a + \tilde{x}_b|)) \\ &\quad - \log(1+\exp(-|\tilde{x}_a - \tilde{x}_b|)) \qquad (4.2b) \\ &\approx \tilde{f}(\tilde{x}_a, \tilde{x}_b), \qquad (4.2c)\end{aligned}$$

where we have $\tilde{f}(\tilde{x}_a, \tilde{x}_b) \triangleq \text{sign}(\tilde{x}_a)\,\text{sign}(\tilde{x}_b)\min(|\tilde{x}_a|, |\tilde{x}_b|)$, and where (4.2b) is a numerically stable calculation of (4.2a), which may be further simplified to (4.2c).

Later in the decoding process, the hard bit decision $\hat{u}_a$ will be provided to the first of the two connections on the left-hand side of the XOR, as shown in Figure 4.11(b). Note that for the left-most XORs in the polar code graph, the corresponding hard bit decision will be provided by the corresponding bit of the decoding candidate under consideration. The hard bit decision $\hat{u}_a$ may be combined with the LLRs $\tilde{x}_a$ and $\tilde{x}_b$ in order to compute a new LLR $\tilde{x}_d$ for the second connection on the left-hand side, according to the g function of [106]

$$\begin{aligned}\tilde{x}_d &= g(\tilde{x}_a, \tilde{x}_b, \hat{u}_a) \\ &= (-1)^{\hat{u}_a}\tilde{x}_a + \tilde{x}_b. \qquad (4.3)\end{aligned}$$

Later still, a bit $\hat{u}_b$ will be provided on the second of the connections on the left-hand side of the XOR, as shown in Figure 4.11(c). Again, this is provided by the corresponding bit of the decoding candidate, in the case of the left-most XORs. Together with $\hat{u}_a$, we can perform the partial sum computation [106] of the bits $\hat{u}_c$ and $\hat{u}_d$ for the first and second connections on the right-hand side of the XOR, where we have

$$\begin{aligned}\hat{u}_c &= \text{XOR}(\hat{u}_a, \hat{u}_b) \\ \hat{u}_d &= \hat{u}_b. \qquad (4.4)\end{aligned}$$

By performing the three types of XOR computations in a prescribed SC schedule [46], an LLR may be obtained for each of the N connections on the left-hand edge of the polar code graph, one at a time in sequential order from top to bottom. When the i^{th} LLR in this sequence $[\tilde{x}_i]_{i=1}^{N}$ is obtained, a PM may be updated for the decoding candidate

according to [46]

$$\phi_i = \phi_{i-1} + \ln\left(1 + \exp\left[-\left(1 - 2\hat{u}_i\tilde{x}_i\right)\right]\right) \tag{4.5a}$$

$$\approx \begin{cases} \phi_{i-1}, & \text{if } \hat{u}_i = \frac{1}{2}\left[1 - \text{sign}(\tilde{x}_i)\right]; \\ \phi_{i-1} + |\tilde{x}_i|, & \text{otherwise,} \end{cases} \tag{4.5b}$$

where $\phi_0 = 0$. Here, the PM quantifies the likelihood of the decoding candidate sequence of bits up to the index i, where having lower PMs implies higher likelihoods. Note that frozen bits contribute to the PM of both the Log-SCL and the Log-SCS algorithm, particularly when the corresponding LLR is negative. This is in contrast to the SCS algorithm of [42], where frozen bits do not impact the metrics used for selecting decoding candidates.

However, in contrast to the breadth-first approach of the Log-SCL algorithm, the Log-SCS algorithm adopts a depth-first approach. More specifically, the Log-SCL algorithm constructs a list of L candidate decoded bit sequences that are built one bit at a time in parallel, as shown in Figure 4.10. By contrast, the Log-SCS algorithm exemplified in Figure 4.8 considers only the most likely decoded bit sequence at a time, building it up one bit at a time, until it comprises N bits, or until its likelihood drops below that of another decoded bit sequence candidate, whereupon the focus switches to that candidate. More specifically, the Log-SCS decoder uses a stack to keep track of up to S decoding candidates at a time, where S is referred to as the stack size.

As shown in Algorithm 1, the Log-SCS begins with only a single decoding candidate in the stack, which initially has undefined values for all N bits. At each step of the Log-SCS algorithm, the decoding candidate at the top of the stack is selected and the value of its next undefined bit is considered. If this is an information or CRC bit, then the decoding candidate is updated to adopt a specific binary value for this bit and a replica is created that adopts the other binary value for this bit. The PMs for these two decoding candidates are updated and the replica is inserted into the stack. The decoding candidates are sorted in the order of their PM, with the top element in the stack having the lowest PM and the bottom element having the highest. If the insertion of the replica into the stack has resulted in exceeding the stack size S, then the bottom element in the stack is eliminated. By contrast, if the next undefined bit in the top decoding candidate is a frozen bit, then the bit in the decoding candidate is set to zero and no replica is made. The PM of the decoding candidate is updated and the stack is sorted in the order of increasing PM.

In contrast to the SCS algorithm shown in Figure 4.9, Figure 4.8 illustrates the advantage of updating the PMs with consideration of the frozen bits in the Log-SCS algorithm. In particular, when the decoding reaches the 7th bit, the top of the stack shifts from the decoding candidate with PM of 3.80 in Figure 4.8 or with likelihood of 0.30 in Figure 4.9,

Algorithm 1 CRC-aided Log-SCS decoder

Input:

the received vector of N LLRs $\mathbf{y}$;

Output:

the decoded vector of N frozen, information and CRC bits $\hat{\mathbf{u}}$ when decoding is successful, or a NULL output when decoding is unsuccessful;

Initialization:

Initialise stack $\mathcal{S}$ with a single entry ℓ_{S_T};
Initialise corresponding PM $\phi[\ell_{S_T}] = 0$;
Initialise corresponding bit index $i[\ell_{S_T}] = 1$;
Initialise current stack size $s = 1$;
Initialise failed CRC counter $j = 0$;
Initialise the maximum number of CRC checks $C = 8$;

while $j < C$ and $s > 0$ **do**
  $i \leftarrow i[\ell_{S_T}]$;
  if $i \leq N$ **then**
    Use (4.2), (4.3) and (4.4) to compute the corresponding LLR $\tilde{x}_i[\ell_{S_T}]$ as a function of $\mathbf{y}$;
    if $i \notin \mathcal{A}$ **then**
      i is the index of a frozen bit;
      Set the corresponding bit $\hat{u}_i[\ell_{S_T}] = 0$;
      Use (4.5) to update the corresponding PM $\phi[\ell_{S_T}]$ as a function of $\tilde{x}_i[\ell_{S_T}]$ and $\hat{u}_i[\ell_{S_T}]$;
      Increment $i[\ell_{S_T}]$;
    else
      i is the index of a information or CRC bit;
      Create a replica of ℓ_{S_T}, referred to as ℓ_{S_B};
      Set the corresponding bits $\hat{u}_i[\ell_{S_T}] = 0$ and $\hat{u}_i[\ell_{S_B}] = 1$;
      Use (4.5) to update the corresponding PMs $\phi[\ell_{S_T}]$ and $\phi[\ell_{S_B}]$ as functions of $\tilde{x}_i[\ell_{S_T}]$, $\hat{u}_i[\ell_{S_T}]$ and $\hat{u}_i[\ell_{S_B}]$;
      Increment $i[\ell_{S_T}]$ and $i[\ell_{S_B}]$;
      Insert ℓ_{S_B} into the stack;
      Increment the stack size s;
      if $s > S$ **then**
        Remove the entry in the stack having the worst PM;
      end if
    end if
    Set ℓ_{S_T} to point to the entry in the stack having the best PM;
  else
    if CRC_Check($\hat{\mathbf{u}}[\ell_{S_T}]$) **then**
      return the decoded vector of N frozen, information and CRC bits $\hat{\mathbf{u}}[\ell_{S_T}]$;
    else
      Remove ℓ_{S_T} from the stack;
      Set ℓ_{S_T} to point to the entry in the stack having the best PM;
      Increment the failed CRC counter j;
    end if
  end if
end while
return NULL

to the decoding candidate with PM 3.61 of the 4^{th} bit in Figure 4.8 or with likelihood of 0.35 in Figure 4.9. When processing the subsequent frozen 5^{th} bit, the Log-SCS algorithm in Figure 4.8 increases the PM to 3.90, but the SCS algorithm in Figure 4.9 simply copies the likelihood 0.35 of the previous bit. In Figure 4.8, the resultant PM of 3.90 is compared with the previous decoding candidate's PM of 3.80 and the Log-SCS returns to continue the decoding of its 8^{th} bit. By contrast, the SCS algorithm requires the additional consideration of the 6^{th} bit in the second decoding candidate, before it returns to complete the decoding of the first decoding candidate. In this way, the Log-SCS algorithm completes the decoding with a lower complexity than the SCS algorithm.

4.2.5 Rate matching and dematching

As shown in Figure 4.4, rate matching is applied after the polar encoder core, in order to adjust the length of the encoded block from N to E bits, where E does not have to be a power of 2. Here, $E \geq K$ must be satisfied, although E may be higher or lower than N. Here, rate matching for $E \geq N$ is achieved using repetition, while $E < N$ is achieved using puncturing or shortening. To be more specific, repetition repeats some of the N bits in the bit vector $\mathbf{x}$ [57], while shortening and puncturing remove some of the N bits in $\mathbf{x}$. Note that shortening removes some specific bits that are guaranteed to have values of 0, while puncturing removes some bits that may have either 0 or 1 values [57]. The selection between these three different rate matching modes in the NR uplink polar code depends on the combination of the information block length A and the encoded block length G [57], as shown in Figure 4.6.

In contrast to the bits used in the polar encoder, a logorithmic polar decoder operates on the basis of LLRs. In this case, the input to the rate dematching of the polar decoder shown in Figure 4.4 will be a vector of E LLRs. In the case of repetition, rate dematching is achieved by summing the LLRs corresponding to the repetition of each of the N bits in the vector $\mathbf{x}$, in order to obtain a corresponding vector of N LLRs. In the case of shortening, infinite-valued LLRs are inserted among the E LLRs in the positions of the shortened bits, in order to obtain the vector of N LLRs. By contrast, zero-valued LLRs are inserted in the positions of punctured bits, in order to represent the uncertainty over whether they have values of 0 or 1.

4.2.6 Channel interleaving and deinterleaving

Following the rate matching of Figure 4.4, the order of the E bits in the encoded block is rearranged by a channel interleaver, according to a prescribed triangular interleaving pattern of [57]. The corresponding deinterleaving operation is performed upon the E

LLRs before they are subjected to the rate dematching in the polar decoder of Figure 4.4.

4.3 Performance, complexity and memory analysis

In the following subsections, the error correction and error detection performance, the computational complexity, and the memory requirement of the CRC-aided Log-SCS polar decoder of [51] for the 3GPP NR are characterised and compared to those of the SCS and Log-SCL algorithms of [42] and [46], in the context of PUCCH transmission. We investigate polar decoding employing QPSK modulation over an AWGN channel, since this is the channel model that was used throughout all of the 3GPP standardisation process and allows the direct comparison of our work to the large catalogue of results produced by 3GPP. Additionally, channel capacity bounds of the finite block length are available for AWGN channel allowing comparison with the theoretical limit. Some of our simulation results also consider uncorrelated Rayleigh fading channels, relying on the idealized simplifying assumption of perfect channel estimation, providing results for transmission over a more practical fading communication channel.

4.3.1 Error correction and error detection performance

Figures 4.12 to 4.14 characterise the BLER performance of the polar decoders considered, when using Quaternary Phase Shift Keying (QPSK) modulation to convey blocks of various lengths over Additive White Gaussian Noise (AWGN) or Rayleigh channels. More specifically, an information block length of $A = 84$ bits is combined with encoded block lengths of $G = 136$, $G = 204$ and $G = 272$ bits, which respectively correspond to rate matching using shortening, puncturing and repetition, as shown in Figure 4.6. Here, the approximate f and ϕ computations of (4.2c) and (4.5b) are employed.

Observe in Figures 4.12 to 4.14 that superior BLER performance is achieved when higher encoded block lengths G are employed in both AWGN and flat Rayleigh fading channels, which is the result of two conflicting trends. Explicitly the coding-performance improves upon reducing the coding rate, which outweighs the effect of encountering more blocks having bit errors when extending the block length. As shown in the figures, the Log-SCS decoder achieves up to 0.5 dB gain compared to the SCS decoder having the same stack sizes. However, the Log-SCL polar decoder achieves the best BLER performance when the list size L of the Log-SCL decoder and the stack size S of the SCS and the Log-SCS decoders are both equal to 8. But as the stack size S is increased towards $S = 2048$, the BLER performance of the Log-SCS decoder becomes better than that of the $L = 8$ Log-SCL decoder, approaching the capacity bound within 1dB provided by the $\mathcal{O}(n^{-2})$ meta-converse Polyanskyi-Poor-Verdù (PPV) upper bound of [107], as shown in Figures 4.12 (a), 4.13 (a), and 4.14 (a). In contrast to the vertical Shannon capacity bound that

is associated with the SNR where an infinitesimally low BLER may be achieved under the assumption of infinite block length, the PPV upper bound of [107] takes account of the finite block-length, in order to bound the achievable BLER as a function of SNR. The particular version of the PPV bound used here assumes QPSK modulation over an AWGN channel. Indeed, like the SCS decoder, the BLER performance of the Log-SCS decoder converges towards optimal maximum likelihood decoding, as the stack size S is increased towards infinity. However, an infinite stack size S leads to an infinite memory requirement, which is impractical for hardware implementations, as it will be detailed in Section 4.3.3. Note that enhancements to the Log-SCS decoder will be proposed in Section 4.4, which address this performance discrepancy relative to the Log-SCL decoder.

Since the 3GPP PUCCH polar code employs a CRC having a limited length of $P = 11$ bits, there is a non-zero probability of the CRC failing to detect the erroneous bits in some blocks. The prevalence of this problem may be characterised by the False Alarm Rate (FAR), which can be defined as the fraction of blocks comprising random Gaussian distributed LLRs that nonetheless pass the CRC [108]. Our simulations have revealed that the FAR of the Log-SCS polar decoder remains near constant at around $2^{-(P-3)}$, which is consistent with that characterised in [108] for the Log-SCL decoder. Note that this FAR of $2^{-(P-3)}$ is maintained regardless of the stack size S, owing to our proposed approach of terminating the decoding when 8 failing CRCs have been encountered. Hence, this mechanism may be considered to use $\log_2(8) = 3$ CRC bits to aid error correction, while using the remaining $(P-3)$ CRC bits to perform error detection. Without this mechanism, the stack would continue offering new decoding candidates until eventually one with a passing CRC is found. However, the further down the stack this decoding candidate is found, the higher the probability of it containing erroneous bits. Hence, without the proposed approach for terminating the decoding, large stack sizes would lead to high FARs.

4.3.2 Computational complexity

In this section, we compare the computational complexity of the Log-SCS polar decoder to that of the Log-SCL decoder. Compared to the computation of the f, g and ϕ functions of (4.2), (4.3) and (4.5a), the partial sum calculation of (4.4) is of much lower computational complexity, which may be neglected in the complexity analysis of the algorithms considered. Therefore, Figures 4.15 and 4.16 show the total number of f, g and ϕ computations performed by the Log-SCS decoder, as well as by the Log-SCL decoder when employing an information block length of $A = 84$ and different encoded block lengths of $G = 136, 204$ or 272. As shown in Figures 4.15 and 4.16, the complexity of the Log-SCL decoder is independent of the channel SNR E_s/N_0, but the complexity of the Log-SCS decoder reduces in the SNR regions, where a low BLER is achieved. In these regions, the Log-SCS decoder has a complexity that is only one fifth of the $L = 8$ Log-SCL decoder's complexity regardless of stack size S. Note that the Log-SCS algorithm

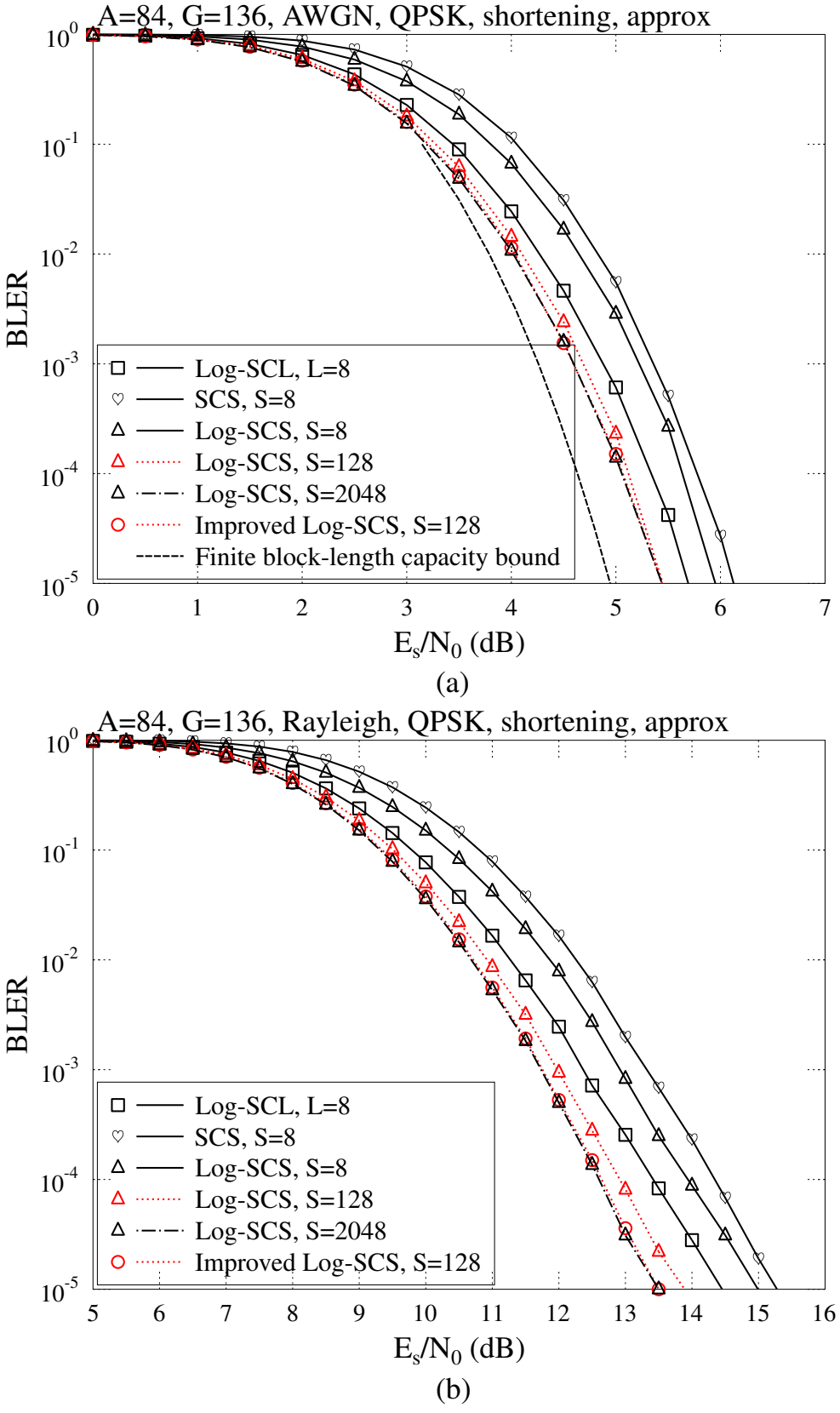


FIGURE 4.12: BLER performance of the polar decoders considered for $A = 84, G = 136$ polar codes with various stack sizes S, for the case where QPSK modulation is used for communication over (a) an AWGN channel; (b) an uncorrelated Rayleigh fading channel.

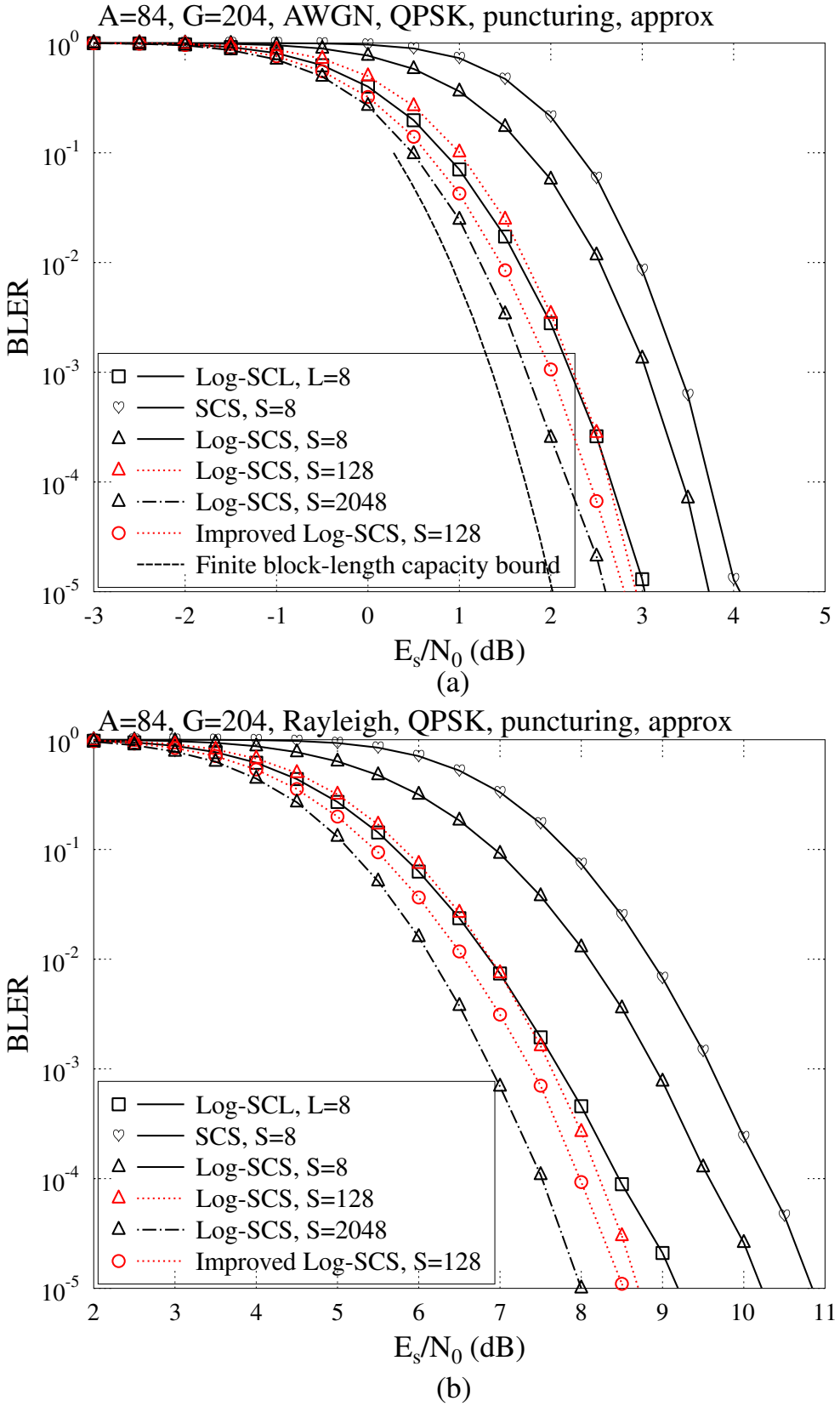


FIGURE 4.13: BLER performance of the polar decoders considered for $A = 84, G = 204$ polar codes with various stack sizes S, for the case where QPSK modulation is used for communication over (a) an AWGN channel; (b) an uncorrelated Rayleigh fading channel.

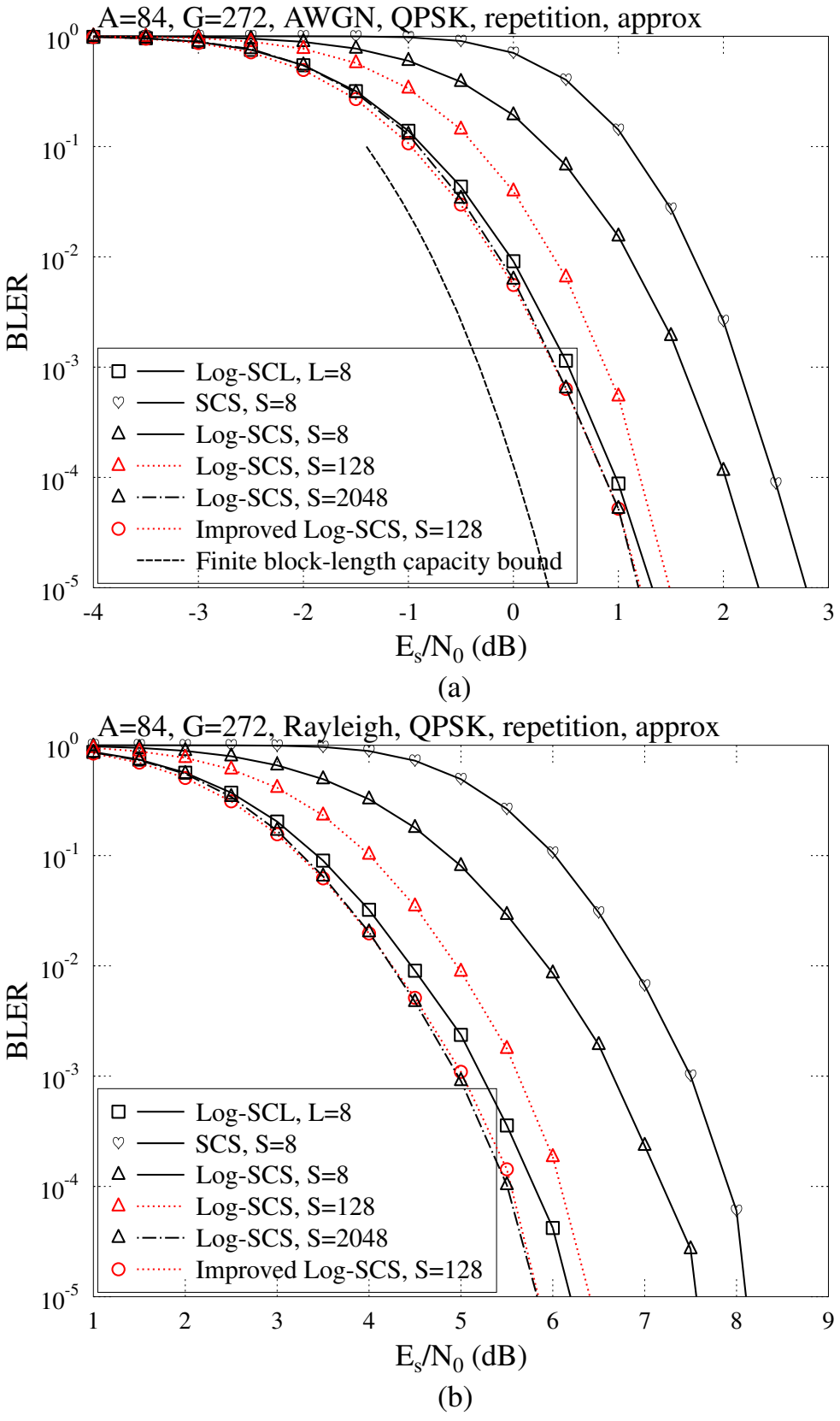


FIGURE 4.14: BLER performance of the polar decoders considered for $A = 84, G = 272$ polar codes with various stack sizes S, for the case where QPSK modulation is used for communication over (a) an AWGN channel; (b) an uncorrelated Rayleigh fading channel.

has a similar complexity to a decoder that uses the Log-SC algorithm in a first decoding attempt and, only if that is unsuccessful, activates the Log-SCL algorithm having $L = 8$ in a second attempt. However, the Log-SCS algorithm has a lower processing latency, since it makes only a single decoding attempt and offers superior BLER when employing a sufficiently high stack size S, as shown in Section 4.3.1.

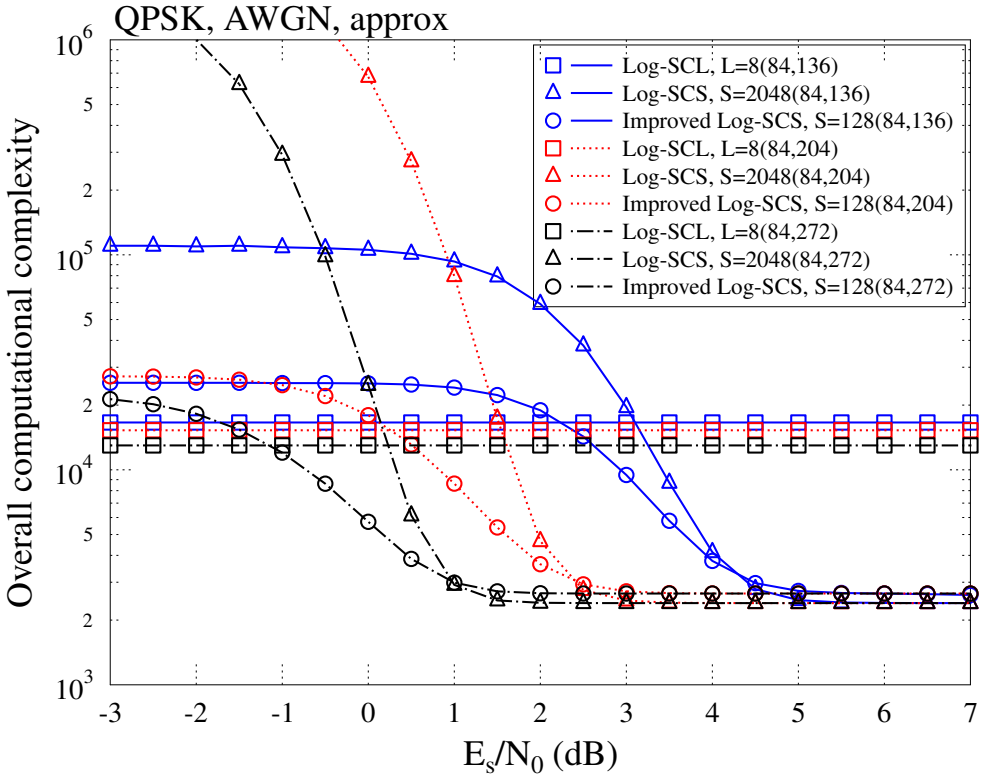


FIGURE 4.15: The number of f, g and ϕ functions performed by the polar decoders considered for various combinations of information block length A, encoded block length G and stack size S, for the case where QPSK modulation is used for communication over an AWGN channel.

4.3.3 Memory requirement

In addition to the computational complexity, the hardware resource requirements of a practical polar decoder also depend on the maximum amount of memory required by the decoding algorithm, which comprises two parts. More specifically, memory is required for storing the partial sum bits processed by the g and XOR calculations of (4.3) and (4.4), as well as the LLRs processed by the f, g and ϕ calculations of (4.2), (4.3) and (4.5a).

For the Log-SCL decoder, memory is required for storing the $N_{\max}L$ candidate decoded bits that are generated during the decoding process, where the list size L quantifies the number of candidates and $N_{\max} = 1024$ is the maximum core block size used in 3GPP

TABLE 4.2: Memory requirements for the polar decoders considered.

	Log-SCL	**Log-SCS**	**Improved Log-SCS**
LLR memory X (bits)	$(N_{\max} + LN_{\max}\sum_{i=1}^{\log_2 N_{\max}} 2^{-i})b_{\text{LLR}}$	$(N_{\max} + SN_{\max}\sum_{i=1}^{\log_2 N_{\max}} 2^{-i})b_{\text{LLR}}$	
Bit memory Y (bits)	$2N_{\max}L$	$2N_{\max}S$	
Overall memory $X + Y$ (bits)	$\approx (N_{\max} + LN_{\max})b_{\text{LLR}} + 2N_{\max}L$	$\approx (N_{\max} + SN_{\max})b_{\text{LLR}} + 2N_{\max}S$	
$N_{\max} = 1024$, $b_{\text{LLR}} = 8$, $X + Y$	$(L = 8)$11KB	$(L = 128)$161KB	$(S = 128)$161KB

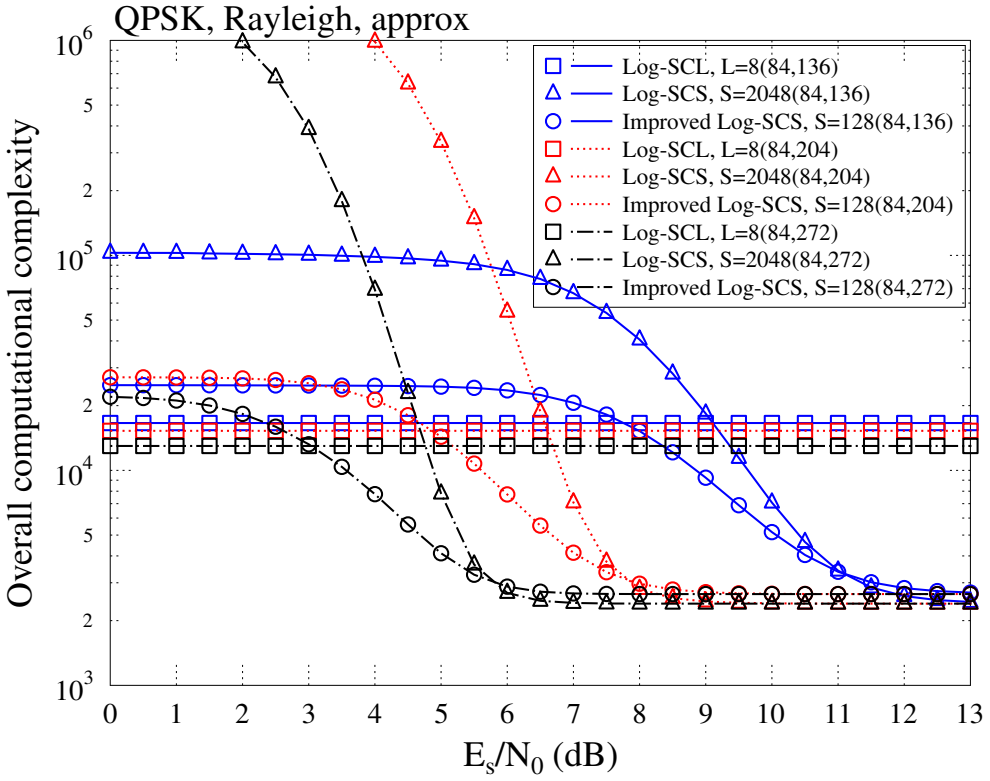


FIGURE 4.16: The number of f, g and ϕ functions performed by the polar decoders considered for various combinations of information block length A, encoded block length G and stack size S, for the case where QPSK modulation is used for communication over a flat uncorrelated Rayleigh fading channel.

NR PUCCH and PUSCH. Furthermore, during the decoding process, the partial sum bits generated after each XOR operation must also be stored in memories. However, once we obtain the output bits of the XOR operations, the input bits are no longer needed. Therefore, memory is only required for $N_{\max}L$ partial sum bits during the Log-SCL decoding, comprising $N_{\max}$ bits for each of the L decoding candidates. So, the combination of the candidate decoded bits and the partial sum bits requires memory for a total of $2N_{\max}L$ bits is required by Log-SCL decoding, as shown in Table 4.2. By contrast, in Log-SCS decoding, the S decoding candidates may comprise different numbers of bits, as the decoding process proceeds. However, in the worst case, all S candidates will all comprise $N_{\max}$ bits. Hence, memory is required for $2N_{\max}S$ bits, including both the decoding candidate bits and the partial sum bits, as shown in Table 4.2.

The Log-SCL decoder requires $N_{\max}b_{\text{LLR}}$ bits to store the $N_{\max}$ LLRs input to the decoder, where b_{LLR} is the number of bits used for storing each LLR. The first decoding stage of the Log-SCL decoder requires storage for a further $LN_{\max}b_{\text{LLR}}/2$ bits, in order to represent the L LLR candidates output by each of the $N_{\max}/2$ f operations performed by this stage. Later on, this memory can be reused to store the L LLR candidates output

by each of the $N_{\max}/2$ g operations performed by this stage. In this way, memory can be reused between the f and g operations performed by each stage in the polar code graph. Each successive stage requires half as much memory as the previous, requiring storage for a total of $(N_{\max} + LN_{\max}\sum_{i=1}^{\log_2 N_{\max}} 2^{-i})b_{\text{LLR}}$ bits for LLRs generated during the decoding process, as shown in Table 4.2. Similar logic can be applied when considering the worst case of the Log-SCS decoder, where at most $(N_{\max}+SN_{\max}\sum_{i=1}^{\log_2 N_{\max}} 2^{-i})b_{\text{LLR}}$ bits of storage is required for the LLRs, as shown in Table 4.2. Note that when $N_{\max}$ is sufficiently large, we have $\sum_{i=1}^{\log_2 N_{\max}} 2^{-i} \approx 1$.

Table 4.2 shows the example of the memory requirement for the considered polar decoders having $b_{\text{LLR}} = 8$ and $N_{\max} = 1024$, as in 3GPP NR PUCCH and PUSCH. Here, we can see that the $S = 128$ Log-SCS decoder and the $L = 128$ Log-SCL decoder require more than ten times as much memory as the $L = 8$ Log-SCL decoder.

4.4 Improvements of the CRC-aided Log-SCS polar decoder

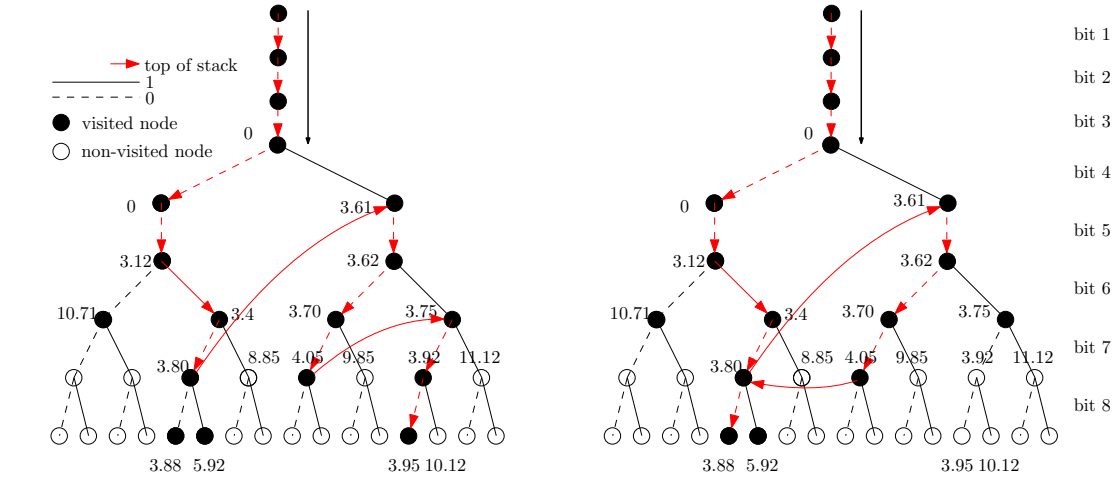
While the Log-SCS polar decoder achieves superior BLER versus complexity performance compared to the $L = 8$ Log-SCL decoder, its requirement for a stack size of up to $S = 2048$ imposes a significantly higher memory requirement. Motivated by this, we propose two improvements to the Log-SCS decoder in Section 4.4.1 and 4.4.2, allowing a smaller stack size S to be used, while still maintaining reliable low-complexity decoding. Following this, the error correction performance versus complexity obtained when combining these two techniques is characterised and compared to the benchmarkers in Section 4.4.3.

4.4.1 Referenced Log-SCS polar decoder

Consider the example shown in Figure 4.17(a), where the first, second, third and fifth of the $N = 8$ bits are frozen. Here, the Log-SCS polar decoder fails to recover the correct bit sequence, which is associated with a PM of 3.88. As shown in Figure 4.17(a), this is because the path to the correct bit sequence is abandoned in the 10^{th} step of the algorithm, when its PM reaches 3.80, in favor of other paths having PMs of 3.70 and 3.75, but ultimately leading to incorrect bit sequences.

In order to address this situation, the Log-SCS algorithm may be further improved by preventing the longest path encountered so far from being removed from the stack. An example of this improvement is shown in Figure 4.17(b), where the Referenced Log-SCS decoder manages to decode and recover the original bit sequence by always retaining the longest path in the stack. In the example of Figure 4.17(b), the path having the metric 3.80 becomes the longest path encountered so far, and so it remains in the stack at the 10th step of the algorithm instead of being removed. This avoids the elimination

of the correct path and achieves the correct decoding result. In this way, the Referenced Log-SCS decoder of Figure 4.17(a) can achieve a superior performance compared to the Log-SCS decoder, particularly in the case of a small stack size S.



(a) Log-SCS

PMs inside the stack	Step t											
1···3	4	5	6	7	8	9	10	11	12	13	14
0	0	3.12	3.4	3.80	3.61	3.62	3.70	4.05	3.75	3.92	3.95
	3.61	3.61	3.61	3.61	3.80	3.80	3.75	3.75	4.05	4.05	4.05

(b) Referenced Log-SCS

PMs inside the stack	Step t										
1···3	4	5	6	7	8	9	10	11	12	13
0	0	3.12	3.4	3.80	3.61	3.62	3.70	4.05	3.80	3.88
	3.61	3.61	3.61	3.61	3.80	3.80	3.80	3.80		5.92

FIGURE 4.17: An example of code trees in (a) the original Log-SCS polar decoder and (b) Referenced Log-SCS polar decoder, where the stack size S is 2. Here, an $N = 8$-bit polar code is used, where the first, second, third and fifth of $N = 8$ bits are frozen. Here the exact of f and ϕ calculations of (4.2b) and (4.5a) are employed.

4.4.2 Restricted Log-SCS polar decoder

Another technique for maintaining the error correction performance despite having a reduced stack size S is to set a limit R for the maximum number of times that each of the N bits in the code tree may be visited during the process of traversing through the code tree. When this limitation is reached for any particular bit, the stack is pruned of all paths that have not yet reached this bit. In this way, more space can be released in the stack for storing paths that have longer path lengths, hence reducing the total memory required. Note that a larger R results in an improved error correction performance, but requires a higher stack size S, hence imposing a performance versus complexity trade-off. In order to achieve a BLER similar to that of the $L = 128$ Log-SCL decoder while maintaining a reduced decoding complexity, we recommend $R = 32$ associated with $S = 128$.

4.4.3 Performance of the Improved CRC-aided Log-SCS polar decoder

The two refinements proposed in Section 4.4.1 and 4.4.2 may be combined for creating the Improved Log-SCS polar decoder. This section characterises the performance versus complexity of the Improved Log-SCS polar decoder, as well as of the Log-SCL polar decoder, across the full range of block lengths supported by the 3GPP NR PUCCH and PUSCH polar code.

Figure 4.18 shows the SNR E_s/N_0 required by the polar decoders considered to obtain a BLER of 10^{-3}, across the full range of block lengths supported by the 3GPP NR PUCCH and PUSCH polar code, when using QPSK for communication over an AWGN channel. While Figure 4.18(a) employs the approximate f and ϕ functions of (4.2c) and (4.5b), Figure 4.18(b) is obtained using the exact expressions of the f and ϕ functions in (4.2b) and (4.5a). Here, the stack size of the Improved Log-SCS decoder is fixed to $S = 128$, whereas the list sizes of both $L = 8$ and 128 are considered for Log-SCL decoding. Additionally, the corresponding capacity bounds are provided by the $\mathcal{O}(n^{-2})$ meta-converse Polyanskyi-Poor-Verdù (PPV) upper bound of [107]. Compared to the approximate computations performed by the decoders of Figure 4.18(a), the exact decoders of Figure 4.18(b) require a lower E_s/N_0 for achieving a BLER of 10^{-3}, as may be expected. Both Figure 4.18(a) and (b) show that the $S = 128$-based Improved Log-SCS polar decoder consistently achieves a BLER of 10^{-3} at a similar E_s/N_0 as the $L = 128$ Log-SCL decoder. This is further illustrated in Figure 4.12 to 4.14, where the $S = 128$-based Improved Log-SCS polar decoder can be seen to offer a similar BLER to the $S = 1024$ Log-SCS decoder.

Figure 4.19 characterises the Computational Complexity Reduction (CCR) of the proposed $S = 128$-based Improved Log-SCS polar decoder across the full range of block lengths supported by the 3GPP NR PUCCH polar code. To be more specific, the CCR is defined as the ratio of the overall complexity of the $L = 8$ or $L = 128$-based Log-SCL decoder to that of the $S = 128$-based Improved Log-SCS decoder, at a BLER of 10^{-3}. Observe from Figure 4.19, that our proposed $S = 128$-based Improved Log-SCS decoder imposes a lower complexity than the $L = 8$-based Log-SCL decoder across nearly all combinations of block length. The complexity of the proposed $S = 128$-based Improved Log-SCS decoder is up to 7 times lower than that of the $L = 8$ Log-SCL decoder and up to 100 times lower than that of the $L = 128$ Log-SCL decoder, while having the same memory requirement in the latter case.

4.5 Conclusions

In this chapter, we have provided a detailed tutorial on the application of the Log-SCS algorithm to the 3GPP NR uplink polar code. We have demonstrated that the Log-SCS algorithm improves upon the BLER of the state-of-the-art Log-SCL polar decoder, while

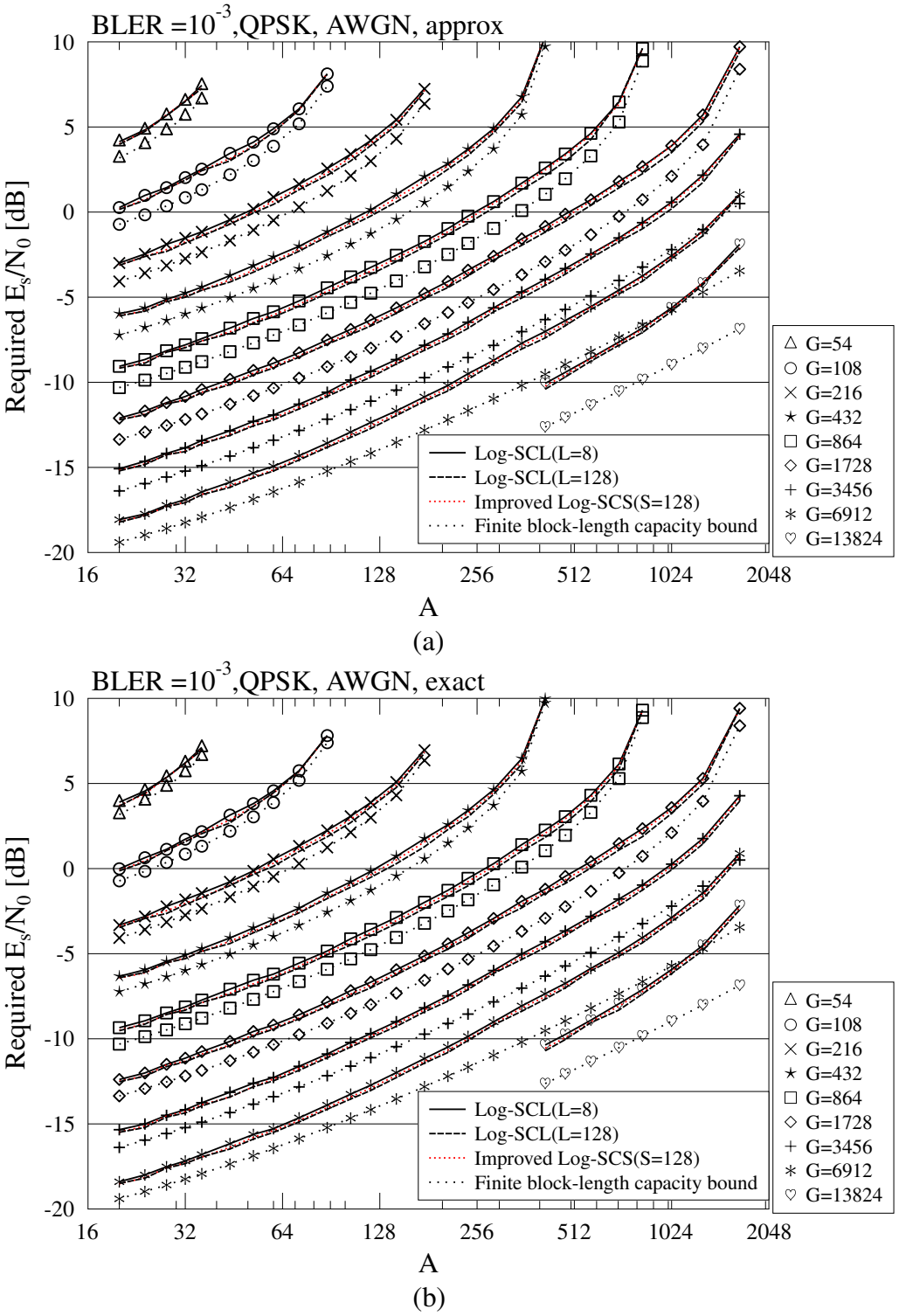


FIGURE 4.18: The SNR required to achieve a BLER of 10^{-3} for different combinations of information block length A and encoded block length G of the 3GPP NR PUCCH polar code, using QPSK modulation for communication over an AWGN channel, when employing (a) the approximate f and ϕ functions of (4.2c) and (4.5b); and (b) the exact f and ϕ functions of (4.2b) and (4.5a). Here, the list size L for Log-SCL decoding is 8 or 128, whereas the stack size S for Log-SCS and Improved Log-SCS decoder is 128.

TABLE 4.3: Comparison of different polar decoders.

	SC	Log-SCL ($L = 8$)	Log-SCL ($L = 128$)	Log-SCS ($S = 1024$)	Improved Log-SCS ($S = 128$)
Distance to capacity	2.46 dB	0.86 dB	0.70 dB	0.71 dB	0.71 dB
Computational complexity	2048	$1.5 * 10^4$	$1.7 * 10^5$	$7.8 * 10^3$	$4.1 * 10^3$
Memory requirement	0.56 KB	3.3 KB	40.3 KB	320.8 KB	320.8 KB

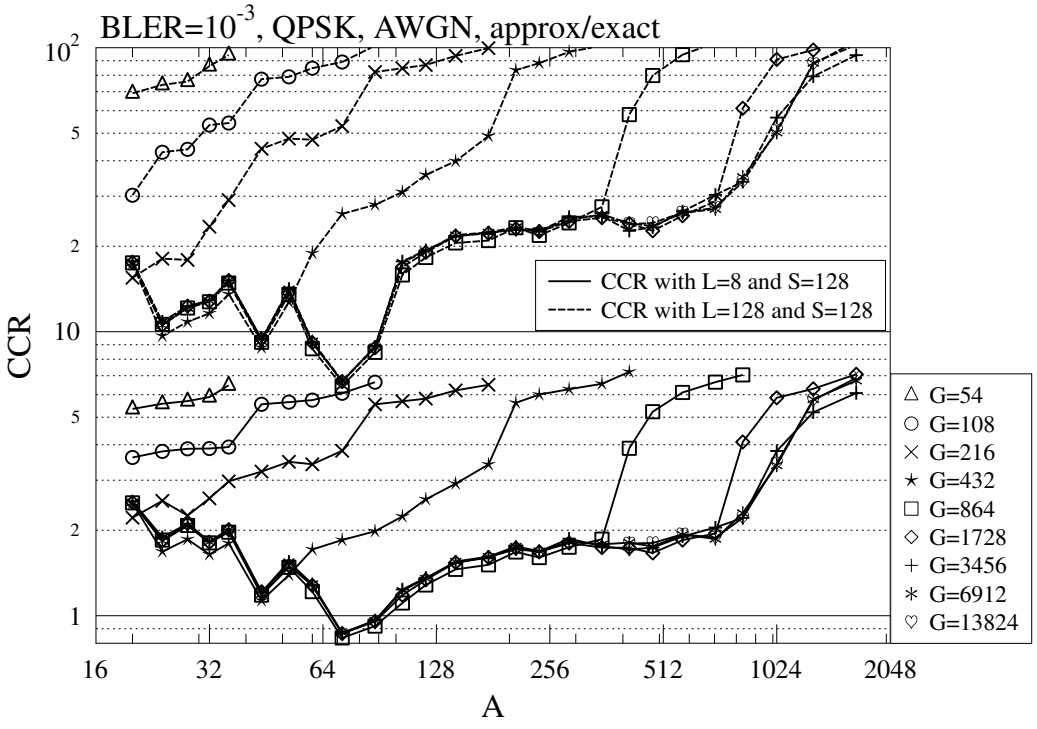


FIGURE 4.19: The computational complexity Reduction (CCR) for different combinations of information block length A and encoded block length G of the 3GPP NR PUCCH and PUSCH polar code to achieve a BLER of 10^{-3}, when employing QPSK for communication over an AWGN channel. Here, the list size L for Log-SCL decoding is 8 or 128, whereas the stack size S for the Improved Log-SCS decoder is 128.

reducing the complexity, as shown in Table 4.3. We have analysed the performance versus complexity and memory requirements in the context of PUCCH and PUSCH transmissions. During the exploitation of the CRC codes to improve the error correction performance, we have introduced the novel technique of limiting the number of CRC checks performed, in order to maintain a consistent error detection performance. Additionally, we have proposed two techniques for further improving the error correction performance of the Log-SCS polar decoder. We have compared the Improved Log-SCS decoder to the Log-SCL and SCS benchmarkers in terms of its BLER versus computational complexity and memory requirement. We have shown that the proposed $S = 128$-based Improved Log-SCS decoder achieves a similar error correction capability to the $L = 128$-based Log-SCL decoder, without increasing its memory requirement and while imposing only a complexity that is up to seven times lower than that of an $L = 8$-based Log-SCL decoder.

Our future work will focus on the application of the Improved Log-SCS decoder to the Physical Downlink Control Channel (PDCCH) transmissions in 3GPP NR, where a distributed CRC is used, as well as to the short block lengths of uplink PUCCH and PUSCH transmissions, where a PC code is concatenated with the polar code. Furthermore, our

future work will consider the software and hardware implementation of the Improved Log-SCS polar decoder.

Chapter 5

Fast Log-SCS Polar Decoder and its Software Implementation

5.1 Introduction

As discussed in Chapter 4, as the selected channel coding scheme for protecting the control channels of 3rd Generation Partnership Project (3GPP) New Radio (NR) [57, 109], polar codes have been shown to be capacity-achieving error correction codes with the Successive Cancellation (SC) decoding algorithm in the case of infinite code block lengths and binary memoryless channels [8]. We have proposed an Improved Log-SCS in Chapter 4, which improves the Block Error Rate (BLER) performance by 0.5 dB, compared to the State-of-the-art (SOTA) polar decoder. However, the serial nature of the SC decoding algorithm imposes data dependencies, resulting in a high decoding latency and low throughput. Therefore, several techniques have been proposed to solve this disadvantage relative to other channel codes, such as Low-Density Parity-Check (LDPC) and turbo codes [37, 50, 110]. For example, the authors of [37] proposed a Simplified Successive Cancellation (SSC) polar decoder that considers the group decoding of successive frozen or information bits, referred to as rate-0 and rate-1 nodes. Following this, the decoding of repetition nodes and Serial to Parallel Converter (SPC) nodes was proposed in [50], further simplifying the decoding operations and improving the throughput and latency. This was demonstrated in [111] for the implementation of the SC decoder in a Software-Defined Radio (SDR) system employing x86 processors with Single Instruction Multiple Data (SIMD) parallel processing.

However, the error correction performance of the SC polar decoder is limited for finite block lengths and in wireless channel. Therefore, more complicated decoding algorithms have been proposed to further improve the Block Error Rate (BLER) performance in those applications [39]. In contrast to the SC decoding which only selects each successive bit value having the highest likelihood, the Successive Cancellation List (SCL)

decoding algorithm of [39, 40] considers a list of L possible bit values at a time enhancing the decoding reliability. In the Cyclic Redundancy Check (CRC)-aided (CA) SCL decoder, any decoding candidates that do not satisfy the CRC are discarded, even if they would otherwise appear to offer the globally most-likely bit values. The practical implementation of the SCL typically represents the likelihoods of candidate bit values into Logarithmic Likelihood Ratios (LLRs), allowing efficient and numerically stable implementation of the SCL decoder [46]. Further improvements that reduce the decoding latency and increase the throughput of the SCL decoder are proposed in [47,58,112,113] and are referred to as the fast simplified SCL (SSCL) decoder. The real time software implementation of the SCL decoder are demonstrated in [47,59]. In addition, appending the CRC codes, the BLER performance may be enhanced as well to improve the decoding reliability.

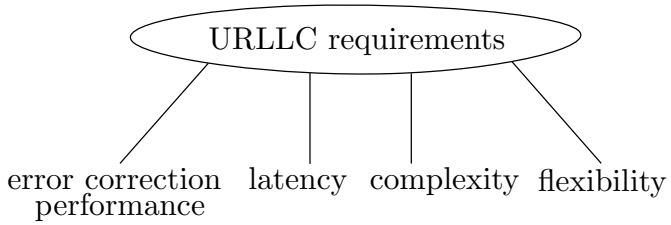


FIGURE 5.1: The contribution of Chapter 5.

In the meantime, the Successive Cancellation Stack (SCS) decoding of [42] uses a depth-first search, rather than the breath-first search of the SCL decoder. The SCS algorithm uses a stack of up to S decoding candidates in order to perform a directed search for the bit values having the globally highest likelihoods. The SCS algorithm achieves a lower decoding complexity than the SCL, which approaches to that of SC when used in channels having high Signal-to-Noise Ratios (SNRs). However, to the best our knowledge, there exists no literature considering the the real time software implementation of the Log-SCS decoder. Therefore, designing a fast Log-SCS decoder to improve the decoding latency, reduce the complexity, as well as maintaining the error correction performance, becomes the target of this chapter, as highlighted in Figure 5.1

- The proposed CA Fast Log-SCS decoder exploits the unique property of Log-SCS decoding and modifies several techniques that is previously considered only by the Fast-SSCL decoder [47,58,112,113] to apply to the proposed Fast Log-SCS decoder. More specifically, the simplified path-metric computation of the rate-0, rate-1 and repetition nodes for Fast-SSCL decoder is applied to the corresponding sub-graphs of the proposed Fast Log-SCS decoder, which reduces the decoding complexity by 50% on average, compared to the Log-SCS polar decoder.
- We also propose a real-time software implementation that is capable of attaining a decoding latency that is only 21% of that of the state-of-the-art Fast SCL polar

decoder implementation, by employing the recursive template metaprogramming [114].

- Each LLR that is input to the proposed decoder is quantised using a 32-bit fixed point number representation, enabling the fixed-point implementation. In addition, the proposed software implementation exploits the 512-bit AVX SIMD operations of the Intel x86 processor architecture to achieve software parallel processing of the SCS decoding for the first time. More specifically, in contrast to [59] which exploits parallelism between the L decoding candidates of the SCL polar decoder, the proposed Fast Log-SCS decoder can only consider parallelism within the processing of only one decoding candidate, since the SCS algorithm considers decoding candidates in series. In this way, a parallelism degree of 16 may be achieved, and a 21% latency may be attained, compared to that presented in [47].

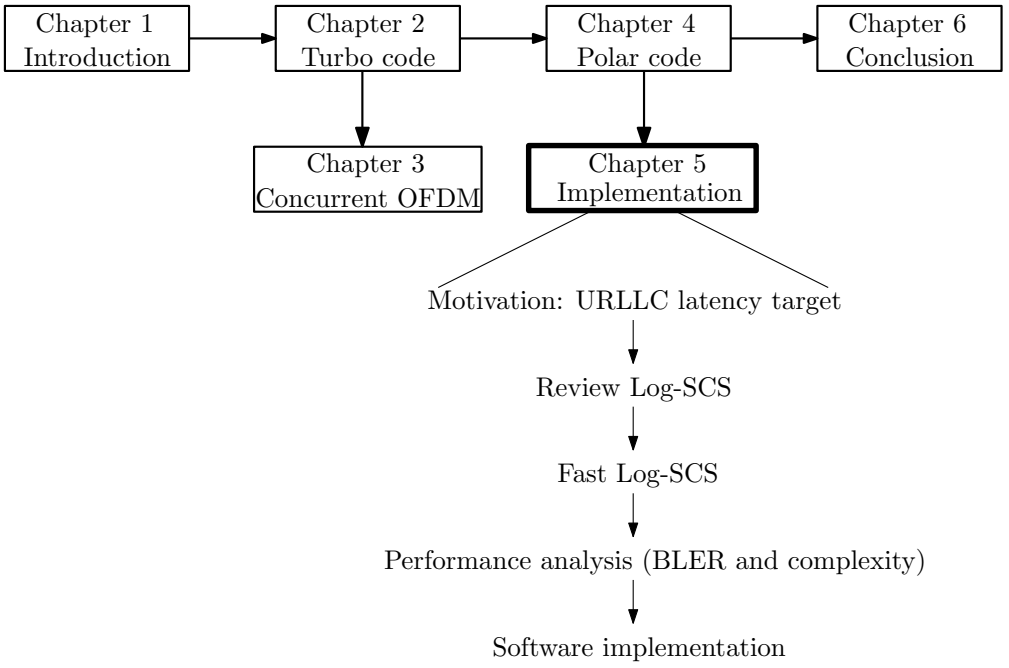


FIGURE 5.2: The development of Chapter 5.

The structure of this chapter is summarised in Figure 5.2. To be more specific, Section 5.2 reviews polar encoding and the Log-SCS decoding process. Following this, a fast Log-SCS polar decoder is proposed in Section 5.3 and is implemented and characterised for x86 processors with SIMD instructions in Section 5.4. Finally, the main conclusions of our work and direction for future research are summarised in Section 5.5.

5.2 Review of Logarithmic successive cancellation stack decoding

The encoding and decoding process for polar codes has been detailed in Chapter 4.2, in the context of the 3GPP NR uplink. Therefore, we only use an example to review the polar encoding Log-SCS decoding.

Figure 5.3 exemplifies this encoding operation, where the core information block $\mathbf{u} = [00000010]$ is input on the lefthand edge of the graph and the successive stages of XOR operations produce the core encoded block $\mathbf{x}$ on its righthand edge. The encoded bits $\mathbf{x} = [10101010]$ are output from the encoder for subsequent operations, which typically includes rate matching, as in the 3GPP NR uplink polar code [115]. After rate matching, a polar-coded block comprising G bits is transmitted over the wireless channel, with the code rate R being defined as $R = A/G$.

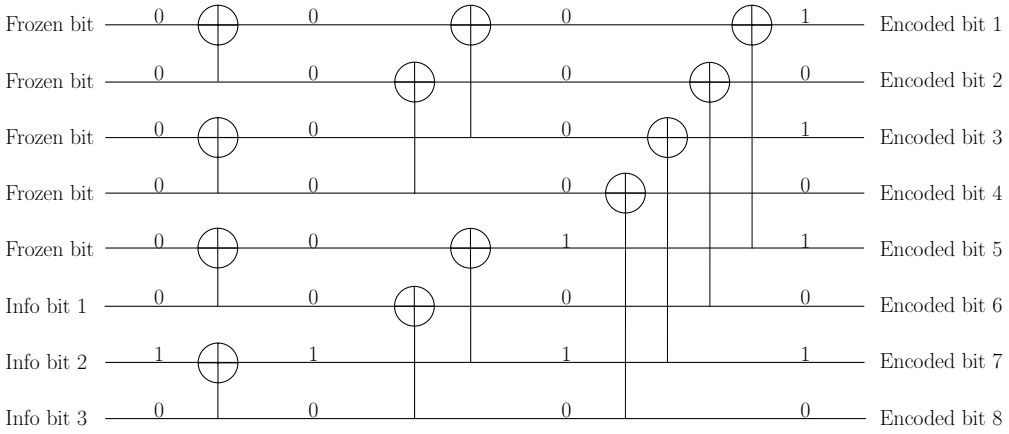
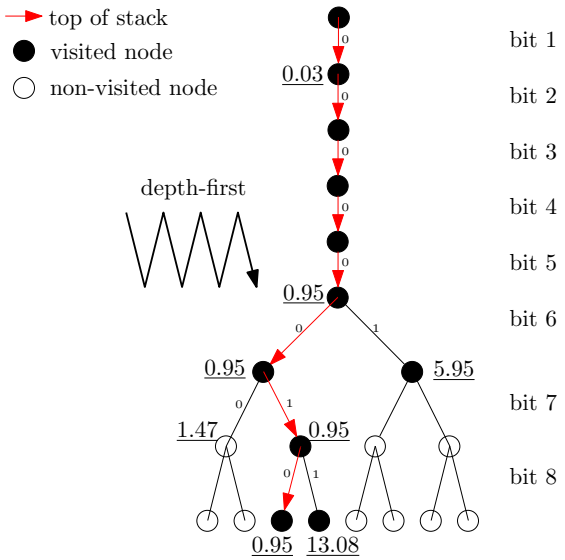


FIGURE 5.3: The XOR operations performed in the polar encoder, for an example where the $K = 3$ information or CRC bits [010] are mixed with $N - K = 5$ frozen bits to give the $N = 8$ input bits $\mathbf{u} = [00000010]$ are then polar encoded into $N = 8$ encoded bits $\mathbf{x} = [10101010]$.

The corresponding Log-SCS decoding is exemplified in Figure 5.4, following the principles detailed in 4.2.4.

The error correction and detection performance of a polar code may be enhanced by appending the information bits with a CRC code, before polar encoding them. For example, a $P = 11$-bit CRC code with the generator polynomial $X^{11}+X^{10}+X^9+X^5+1$ is appended to each block of A information bits, before polar encoding in the NR PUCCH for cases where $A \in [20, 1706]$. In a CA Log-SCS decoder, a CRC check may be performed whenever a decoding candidate having defined values for all N bits is obtained, in order to detect any erroneous bits in the candidate block. If the decoding candidate passes the CRC check, then the decoding process is terminated immediately and successful decoding is reported. On the other hand, if the CRC check fails, a counter is incremented and the decoding candidate is eliminated from the stack. If the counter reaches a predefined limit, then the decoding process is abandoned and a decoding failure is declared. In this



PMs inside the stack	Step t					
	1	$2\cdots4$	5	6	7	8
	0.03	0.03	0.95	0.95	0.95	0.95
	—	—	—	—	—	—
				5.95	1.47	1.47

FIGURE 5.4: Code trees in Log-SCS, when decoding the example $K = 3$ information or CRC bits from the $N = 8$ core encoded bits of Figure 5.3, using the exact f and ϕ functions of (4.2a) and (4.5a).

paper, we set the maximum number of CRC check attempts to be $C = 8$, in order to maintain a consistent false alarm rate. More specifically, this approach effectively uses $\log_2(8) = 3$ of the $P = 11$ CRC bits to improve the error correction of the polar decoder, while the remaining $P = 3$ CRC bits are used for error detection. Note that this is the balance that was assumed when the length of the CRC was selected during the 3GPP standardisation process [57].

5.3 Fast Log-SCS decoder

This section proposes a Fast Log-SCS polar decoder and a software implementation that offers an improved decoding latency compared to the state-of-the-art Fast SSCL polar decoder. The proposed Fast Log-SCS decoder employs the fixed-point implementation detailed in Section 5.3.1, as well as special computation for rate-0, rate-1 and repetition sub-graphs, as detailed in Sections 5.3.2 to 5.3.4 respectively. Note that we refer to the rate-0, rate-1 and repetition nodes of [37] as sub-graphs in this paper, in order to avoid confusion with the nodes of the tree exemplified in Figure 5.6, which has a different structure to the tree used in [37]. Following the descriptions of our Fast Log-SCS decoder, we characterize its BLER performance and computational complexity in Section 5.3.5 and 5.3.6 respectively.

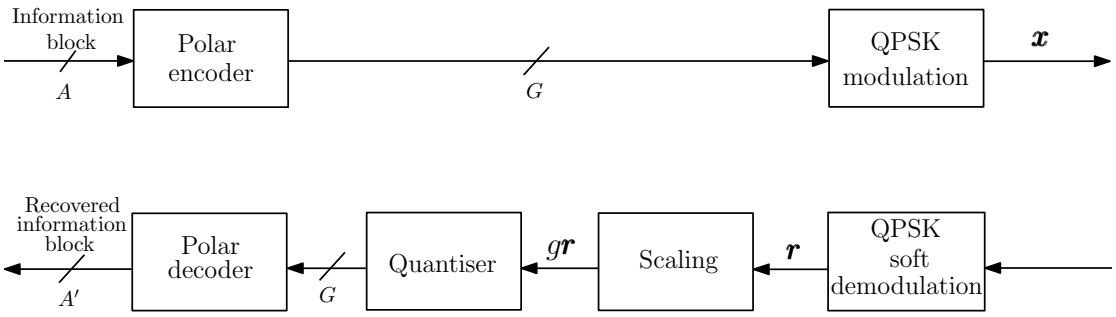


FIGURE 5.5: Schematic of polar encoding and fixed-point decoding.

5.3.1 Fixed-point implementation

As shown in Figure 5.5, the proposed software implementation of the Fast Log-SCS decoder operates on the basis of fixed point arithmetic, where scaling and quantisation are employed to preserve the error correction performance. Here, scaling is applied to the channel LLRs $\boldsymbol{r}$ provided after the demodulation of the received signals, in order to match their range to that of the quantisation applied afterwards, which maps the continuous LLR amplitudes into discrete ranges. To elaborate further, while higher LLR magnitudes conventionally result in superior BLER performance, fixed-point overflow may occur if the LLR magnitudes are too high. Hence, scaling can be applied to reduce the overflow, but at the cost of increasing underflow, which reduces the resolution of the critical LLRs having low magnitudes. Therefore, an optimal scaling factor g balances the trade-off between fixed-point overflow and underflow, in order to optimise the BLER performance. In the performance simulations of Section 5.3.5, the value of the scaling factor g will be selected as a function of the channel SNRs according to the technique of [116].

Furthermore, the conversion from floating-point to the fixed-point arithmetic requires the selection of values for three parameters, namely the number of bits used to represent the channel LLRs, internal LLRs, and PMs. The channel LLRs are output by the quantizer shown in Figure 5.5, while the internal LLRs are generated during the Log-SCS decoding process, as the output of the f and g functions. In the proposed fixed-point software implementation, the channel LLRs are quantised using 24 bits whereas the internal LLRs and PMs are quantised using 32 bits. Here, a lower channel LLR width is employed to reduce the occurrence of overflow during the additions performed by the f functions and the PM computations. Further mitigation of overflow is achieved by considering that the SCS algorithm does not depend on the absolute values of the PMs, but rather it depends on the difference between PMs, when determining which path is most likely. Instead of storing the absolute value of each PM, the occurrence of overflow is reduced by storing only the difference between the PM and the minimum comparable PM.

The simulations of Section 5.3.5 will show that the fixed-point operation described here has similar error correction performance to that of a floating-point implementation.

5.3.2 Rate-0 sub-graph computation

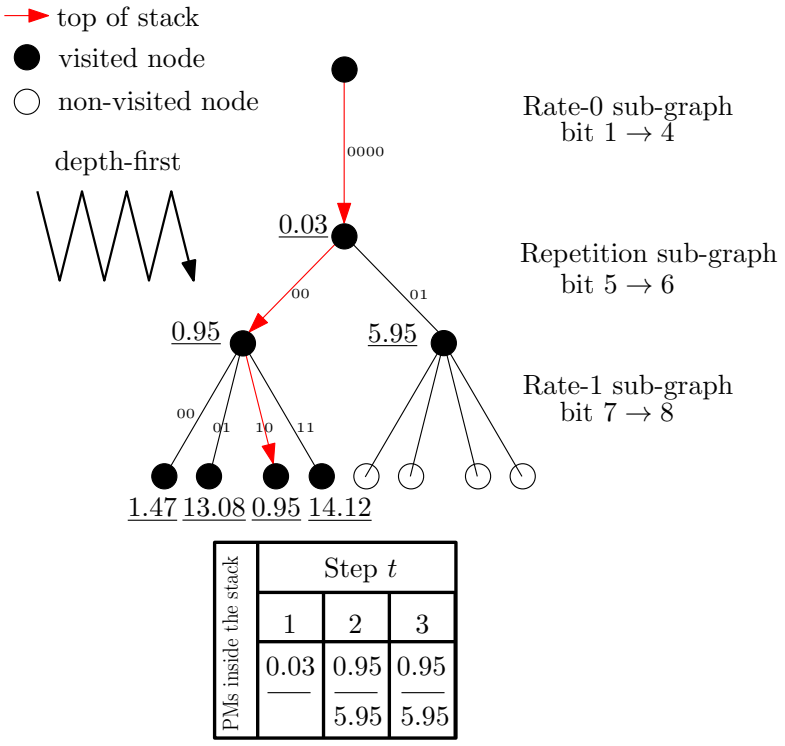


FIGURE 5.6: Code trees in Log-SCS, when decoding the example $A = 3$ information bits from the $N = 8$ core encoded bits of Figure 5.3, using the exact f and ϕ functions of (4.2a) and (4.5a).

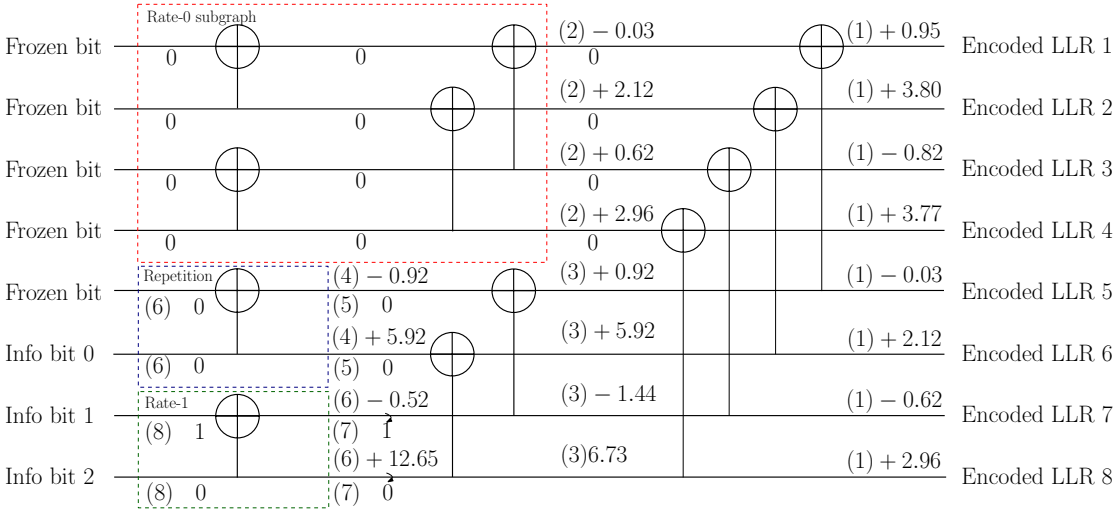


FIGURE 5.7: The XOR operations performed in the polar encoder core, for an example where $A = 3$ information bits [010] are converted into $N = 8$ encoded bits with corresponding LLRs.

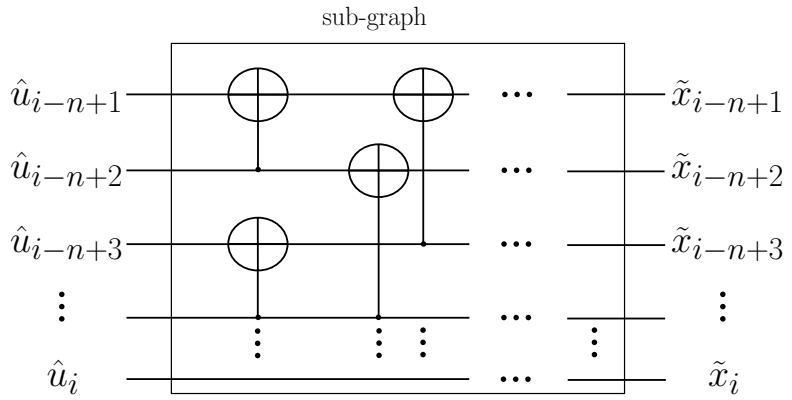


FIGURE 5.8: The decoding process of the rate-0, rate-1, or repetition sub-graph.

A rate-0 sub-graph in the XOR graph is one which contains only frozen bits, as exemplified in Figures 5.6 and 5.7. Since the decoder has *a priori* knowledge that all frozen bits have the value 0, and hence all encoded bits produced by the subgraph also have a value of 0, the decoding of a rate-0 sub-graph may be carried out immediately, without propagating LLRs into the sub-graph. Using the notation of Figure 5.8, the decoded bits $[\hat{u}_j]_{j=i-n+1}^{i}$ of a rate-0 sub-graph in any valid decoding candidate are zeros and the corresponding PM $\phi_i[s_1]$ may be updated as [113]

$$\phi_i[s_1] = \phi_{i-n} + \sum_{j=i-n+1}^{i} |\min\{0, \tilde{x}_j\}| \,. \tag{5.1}$$

Note that this calculation can be completed without any computation of any f or g functions within the subgraph.

5.3.3 Rate-1 sub-graph computation

In contrast to the rate-0 sub-graph, a rate-1 sub-graph contains only information bits, as exemplified in Figures 5.6 and 5.7. For a sub-graph comprising n information bits, 2^n decoding candidates can be produced considering each possible combination of the n bits. However, this leads to the exponential expansion of the candidate set, and it is impractical to consider all candidates when n is not trivially small. Therefore, instead of considering all the 2^n decoding candidates, we adopt the most likely hard decision values for the $(n-2)$ most reliable bits and consider the four possible decoding candidates that are obtained by varying the values of the the two least-reliable bits [37]. For example, when the decoder encounters a subgraph, comprising $n = 4$ consecutive information bits, $\hat{u}_{i-3}, \hat{u}_{i-2}, \hat{u}_{i-1}, \hat{u}_i$, hard decisions are made for the two bits corresponding to the two LLRs from the set $\{\tilde{x}_{i-3}, \tilde{x}_{i-2}, \tilde{x}_{i-1}, \tilde{x}_i\}$ having the highest magnitudes, according to the signs of these two LLRs. Following this, the LLRs $\tilde{x}_{\min 1}$ and $\tilde{x}_{\min 2}$ of the two least-reliable bits in the subgraph, where $\tilde{x}_{\min 1} \leq \tilde{x}_{\min 2}$, are used to compute PMs for the four possible decoding candidates, and then generate four PMs, $\phi_i[s_1], \phi_i[s_2], \phi_i[s_3]$ and

$\phi_i[s_4]$, according to the following rules,

$$\phi_i[s_1] = \phi_{i-4}, \tag{5.2a}$$
$$\phi_i[s_2] = \phi_{i-4} + |\tilde{x}_{\min 1}|, \tag{5.2b}$$
$$\phi_i[s_3] = \phi_{i-4} + |\tilde{x}_{\min 2}|, \tag{5.2c}$$
$$\phi_i[s_4] = \phi_{i-4} + |\tilde{x}_{\min 1}| + |\tilde{x}_{\min 2}|. \tag{5.2d}$$

In this case, only three additional decoding candidates are generated and stored in the stack, rather than the 15 that would be required if all possible candidates were computed.

5.3.4 Repetition sub-graph computation

When only the last bit u_i in a group of $n = 2^v, v \geq 1$ bits is an information bit, with the rest being frozen bits, then the sub-graph is referred to as repetition sub-graph, as exemplified in Figures 5.6 and 5.7 The corresponding n encoded bits have only two possible hard decisions, either n zeros or n ones, depending on whether the decoding candidate uses $\hat{u}_i = 0$ or $\hat{u}_i = 1$ respectively. In order to consider both options, we store both possible decoding candidates in the stack and update the PM according to the following rules [113],

$$\phi_i[s_1] = \phi_{i-n} + \sum_{j=i-n+1}^{i} |\min\{0, \tilde{x}_j\}|, \tag{5.3a}$$
$$\phi_i[s_2] = \phi_{i-n} + \sum_{j=i-n+1}^{i} |\max\{0, \tilde{x}_j\}|, \tag{5.3b}$$

where the updated $\phi_i[s_1]$ stores the PM when $\hat{u}_i = 0$ and $\phi_i[s_2]$ stores the PM when $\hat{u}_i = 1$. In this way, these PM calculations are performed without the computation of f or g functions, as that in the rate-0 or rate-1 sub-graph.

5.3.5 Error correction performance

When a scaling factor g optimised for the particular channel SNR E_s/N_0 and 32-bit fixed-point quantization is applied to the channel LLRs, BLER performance of the Log-SCS decoder remains the same as that of the floating-point Log-SCS decoder [116]. This may be observed for the case of $G = 1024$ polar codes of various coding rates R ranging from 1/8 to 1/2, as shown in Figure 5.9.

Furthermore, Figure 5.10 demonstrates the SNR required for the $S = 128$ 32-bit fixed-point Log-SCS decoder of [115] and the proposed $S = 128$ fast fixed-point Log-SCS decoder to achieve a BLER of 10^{-3} for different combinations of information block length A and encoded block length G of the 3GPP NR PUCCH polar code. We can see that when further combining the techniques discussed in Sections 5.3.1 to 5.3.4 together, the

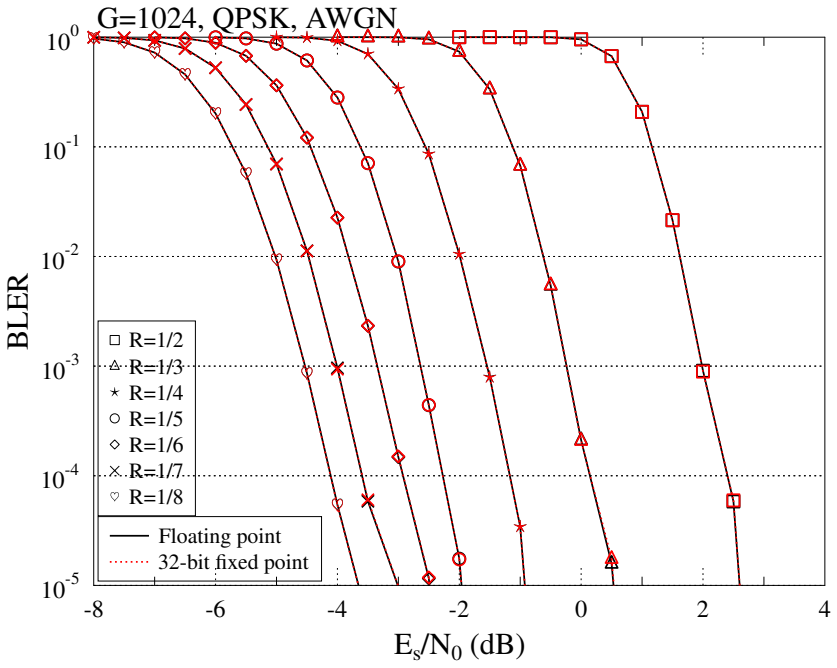


FIGURE 5.9: BLER performance of the fixed-point and floating-point implementations of the Log-SCS polar decoder for $G = 1024$ polar codes of different coding rates R over AWGN channel, where QPSK modulation is employed.

BLER performance remains similar to that of the 32-bit fixed-point Log-SCS decoder as shown in Figure 5.10, while the decoding latency reduces significantly as will be discussed in Section 5.4.2.

5.3.6 Computational complexity

The overall decoding complexity of the proposed fast Log-SCS decoder may be quantified by the number of f, g and ϕ functions performed in the decoder at the channel SNR where a BLER of 10^{-3} is achieved. Figure 5.11 compares the overall complexity of the $S = 128$ 32-bit fixed-point Log-SCS decoder and the proposed $S = 128$ fast 32-bit fixed-point Log-SCS decoder for $G = 1024$-bit polar codes with various information block lengths A, where QPSK modulation is used for communication over an AWGN channel. Here, it may be observed that exploiting the simplified rate-0, rate-1 and repetition sub-graph computation of the proposed fast Log-SCS polar decoder yields a 50% reduction in the complexity.

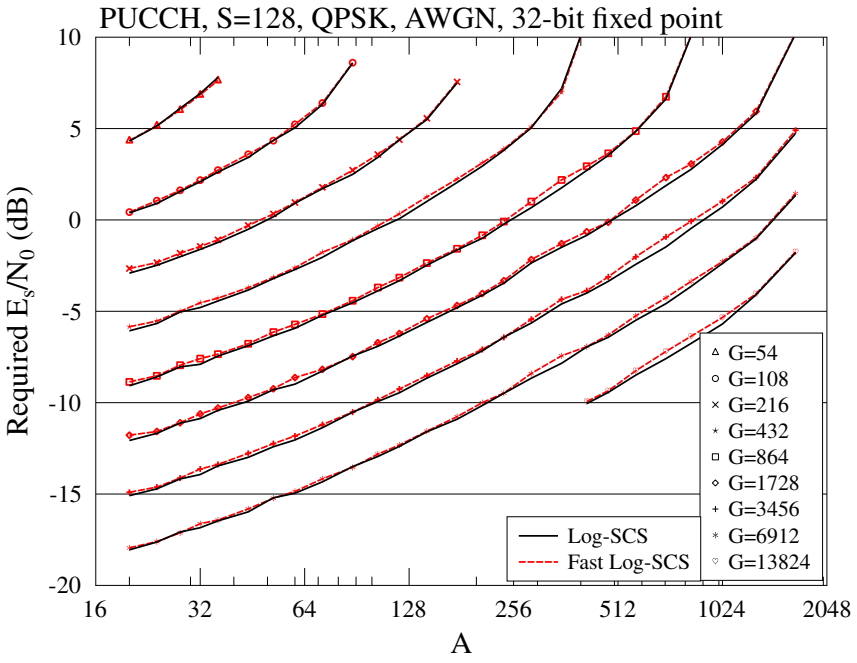


FIGURE 5.10: The SNR required for the $S = 128$ 32-bit fixed-point Log-SCS decoder of [115] and the proposed $S = 128$ fast fixed-point Log-SCS decoder to achieve a BLER of 10^{-3} for different combinations of information block length A and encoded block length G of the 3GPP NR PUCCH polar code, where QPSK modulation is used for communication over an AWGN channel.

5.4 SIMD implementation of the proposed Fast Log-SCS decoder

In this section, we introduce an implementation of the proposed fixed-point Fast Log-SCS decoder using SIMD instructions on an x86 CPU that supports AVX-512. The implementation is detailed in Section 5.4.1 and characterised in Section 5.4.2.

5.4.1 f and g functions computation

A number of techniques are adopted in order to optimise the throughput and latency of the proposed implementation of the Fast Log-SCS decoder. First, we employ recursive template metaprogramming [114], as shown in Algorithms 1 and 2, with the special case of the code length $Nsc = 1$ shown in Algorithm 3. This technique enables a recursive programming style that is easy to maintain, while also enabling the computer to optimize and pre-process as much of the processing as possible at compile time. To be more specific, in the process of metaprogramming, polar decoders having each power-of-2 block length N up to the maximum used in 3GPP NR of 1024 are instantiated recursively during the compilation process. The recursive structure of the polar codes

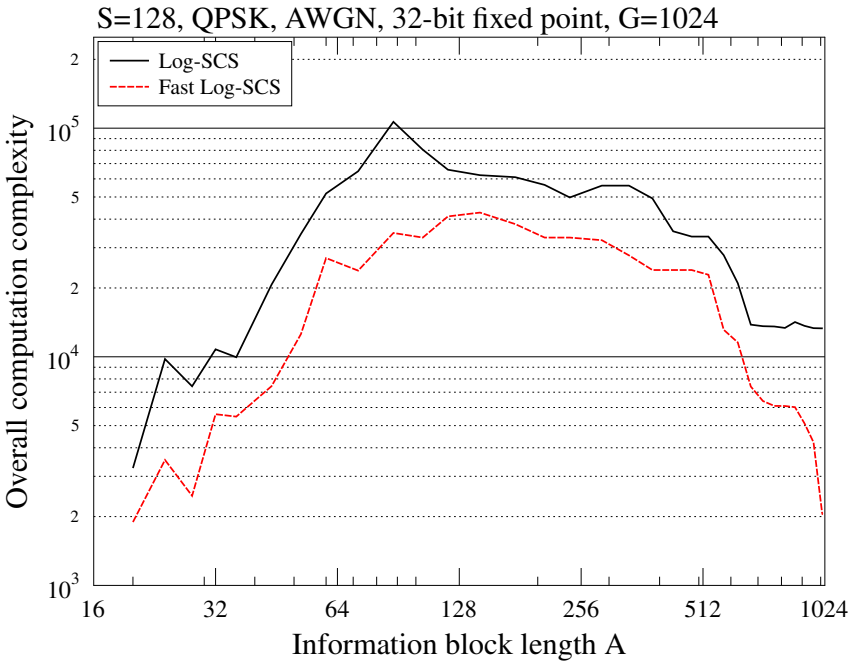


FIGURE 5.11: Overall computational complexity of the $S = 128$ 32-bit fixed-point Log-SCS decoder and the proposed $S = 128$ fast 32-bit fixed-point Log-SCS decoder for $G = 1024$-bit polar codes with various information block lengths A, where QPSK modulation is used for communication over an AWGN channel.

Algorithm 2 Main function

Iteration: stack_index=0; bit_index;

Input: main()

1: PolarDecoder<1024> polar_decoder; {Initialize an $S = 128$ stack decoder with $N = 1024$}
2: polar_decoder[128]=0; {Initialize the length of decoding candidates in the stack decoder}
{3-6: Complete the decoding when the length of the top element in the stack reaches N}
3: **while** decod_bit[stack_index]<N **do**
4: bit_index=0;
5: polar_decoder.decode(llrs+stack_index*N, bits+stack_index*N, PM, decod_bit);
6: **end while**

Output:

allows the recursive computations of the f and g functions to be unrolled at compile time, producing highly optimised compiled code.

The AVX-512 implementation of f and g functions are detailed in the C++ code of Algorithms 4 and 5, respectively. Here, **__m512i** registers have 512 bits and the LLRs are quantised using 32 bits as discussed in Section 5.3.1, resulting in a maximum parallelism degree of $P_1 = 16$. Note that in this paper, the parallelism of the SIMD instructions is achieved by performing parallel computations for up to 16 bits within the same block,

Algorithm 3 Template meta-programming for the fast Log-SCS polar decoder

Iteration:

```
1: template <int Nsc>;      {Class for a polar decoder having a length of Nsc}
2: class PolarDecoder{
     {A polar decoder of length Nsc recursively contains a polar decoder of length
   Nsc/2}
3:   PolarDecoder<Nsc/2> next;
4:   Public:
5:   void decode(int *llrs, int *bits, int *decod_bit,int *PM) {
6:     if is_rate0_subgraph(decod_bit, Nsc) then
7:       decode_rate0(llrs, bits, decod_bit, PM);
8:     else if is_rate1_subgraph(decod_bit, Nsc) then
9:       decode_rate1(llrs, bits, decod_bit, PM);
10:    else if is_rep_subgraph(decod_bit,Nsc) then
11:      decode_rep(llrs, bits, decod_bit, PM);
12:    else
13:      if decod_bit[stack_index]==bit_index then
14:        this → f_function(llrs);
15:      end if
         {Use the Nsc/2-length polar decoder to decode the top half}
16:      next.decode(llrs, bits, decod_bit,PM);
17:      if decod_bit[stack_index]==bit_index then
18:        this → g_function(llrs+Nsc/2, bits+Nsc/2);
19:      end if
         {Use the Nsc/2-length polar decoder to decode the bottom half}
20:      next.decode(llrs+Nsc/2, bits+Nsc/2, decod_bit, PM);
21:    end if
22:  }
23: }
```

Algorithm 4 Template meta-programming special case when Nsc=1

Iteration:

```
1: class PolarDecoder<1>{
2:   void decode(int *llrs, int *bits, int *decod_bit,int *PM) {
3:     if decod_bit[stack_index]==bit_index then
4:       PM_calculate(PM,llrs,bits);  {Calculate the PM of the top element in the
   stack}
5:       decod_bit[stack_index]++;  {Extend the length of top candidate by 1}
6:       expand_stack(llrs,bits,decod_bit,PM);  {Expand the current stack size}
7:       stack_index=sort_stack(PM);  {Allow stack_index to point to the top ele-
   ment in the stack}
8:     end if
9:   bit_index++;{Indicate calculations required for decoding the next bit}
10:  }
11: }
```

Algorithm 5 f function calculation using AVX-512 instructions

Iteration: static const Is32vec16 const_0 = _mm512_set1_epi32(0);
{constant-zero AVX512 vector}
Input: f_function(**int** *llrs)

```
1: int i=0;
   {Use different mask if the parallelism is less 16}
2: if Nsc>=16 then
3:   while i<Nsc/2 do
4:     Is32vec16 x_m = _mm512_load_epi32(llrs+i);  {load llrs of x_m to the AVX512 vector}
5:     Is32vec16 x_n = _mm512_load_epi32(llrs+i + Nsc/2);  {load llrs of x_n to AVX512 vector}
6:     Is32vec16 min_a_b = simd_min(_mm512_abs_epi32(x_m), _mm512_abs_epi32(x_n));
7:     Is32vec16 sign_x_m1 = _mm512_mask_set(const_0, _mm512_cmpgt_epi32_mask(x_m, const_0), 1);
8:     Is32vec16 sign_x_m2 = _mm512_mask_set(const_0, _mm512_cmpgt_epi32_mask(x_m, const_0), -1);
9:     Is32vec16 sign_x_m = sign_x_m1 | sign_x_m2;
10:    Is32vec16 sign_x_n1 = _mm512_mask_set(const_0, _mm512_cmpgt_epi32_mask(x_n, const_0), 1);
11:    Is32vec16 sign_x_n2 = _mm512_mask_set(const_0, _mm512_cmpgt_epi32_mask(x_n, const_0), -1);
12:    Is32vec16 sign_x_n = sign_x_n1 | sign_x_n2;
13:    Is32vec16 result = sign_x_m*sign_x_n* min_a_b;
14:    _mm512_storeu_epi32(llrs+i, result);  {store the AVX-512 vector to the corresponding llrs memory address}
15:    i+=16;
16:  end while
17: end if
```

Output:

Algorithm 6 g function calculation using AVX-512 instructions

Iteration: static const Is32vec16 const_1 = _mm512_set1_epi32(1);
static const Is32vec16 const_2 = _mm512_set1_epi32(2) ;
Input: g_function(**int** *llrs, **int** *bits)

```
1: int i=0;
   {Use different mask if the parallelism is less 16}
2: if Nsc>=16 then
3:   while i<Nsc/2 do
4:     Is32vec16 x_m = _mm512_load_epi32(llrs+i-Nsc/2);
5:     Is32vec16 x_n = _mm512_load_epi32(llrs+i);
6:     Is32vec16 u_m = _mm512_load_epi32(bits+i-Nsc/2);
7:     Is32vec16 temp = const_1-u_m*const_2;
8:     Is32vec16 result = temp*x_m+x_n;
9:     _mm512_storeu_epi32(llrs+i, result);
10:    i+=16;
11:  end while
12: end if
```

Output:

rather than performing parallel computations for bits from 16 separate blocks, as discussed for the SCL decoder of [59]. In this way, our approach provides a benefit even when 16 blocks having the same information and encoded block lengths are not available for decoding at the same time. Besides, the approach to parallelism of [59] cannot be applied to SCS decoding, since it is unlikely that the decoding of the 16 blocks would stay synchronised, throughout the dynamic SCS decoding process.

5.4.2 Latency, Throughput and Memory requirement

We have compiled our AVX-512-enabled C++ code using the Intel C++ Compiler with the -Ofast optimisation, and characterised it on an Intel(R) Xeon(R) Gold 6138 CPU operating at $f_1 = 2.00$ GHz. The average over 100 runs of the latency D time required to perform the $S = 128$ Fast Log-SCS decoding of each block is characterised as a function of the number of information bits A and resultant encoded bits G in each block.

Figure 5.12 characterises the latency D of the Fast Log-SCS polar decoder and AVX-512 Fast Log-SCS polar decoder for encoded block lengths of $G = 512$, 1024 and 2048 with various information block lengths A for the case where the channel SNR is such that a BLER of 10^{-3} is achieved. Note that in contrast to the Fast SSCL decoder, whose latency is independent of channel SNR, the latency of the Fast Log-SCS polar decoder decoder decreases as the SNR increases. Note that the encoded block lengths of $G = 512$ and 1024 are obtained using the 3GPP NR PUCCH polar codes having core block lengths of $N = 512$ and 1024, respectively. By contrast, density evolution with the gaussian approximation (DE-GA) algorithm is employed to parameterize an $N = 2048-$bit polar code, which is used to generate the encoded block length of $G = 2048$, since the 3GPP NR PUCCH polar codes only supports a maximum core block length of $N = 1024$ bits. Figure 5.12 also includes benchmark results for

Furthermore, Table 5.1 compares the latency, throughput and memory requirement of different polar decoders when achieving a BLER of 10^{-3} for block lengths of $A = 1723$ and $G = 2048$. More specifically, the proposed CA Fast Log-SCS decoder is compared with the AVX2 Fast SCL decoders of [112], [47] and [113], which were characterised using clock frequencies of $f_2 = 3.90$ and $f_3 = f_4 = 3.40$ GHz, respectively. These implementations use LLR widths of 8 or 32 bits and since AVX2 has register widths of 256 bits, a parallelism of $P_2 = 32$ and $P_3 = P_4 = 8$ is achieved. Note that rather than using the 32-bit floating point LLRs of the other implementations, the Fast SSCL implementation of [113] uses 8-bit fixed point LLRs, same as our proposed CA Fast Log-SCS decoder. For the sake of enabling fair comparison, we define the normalised latency

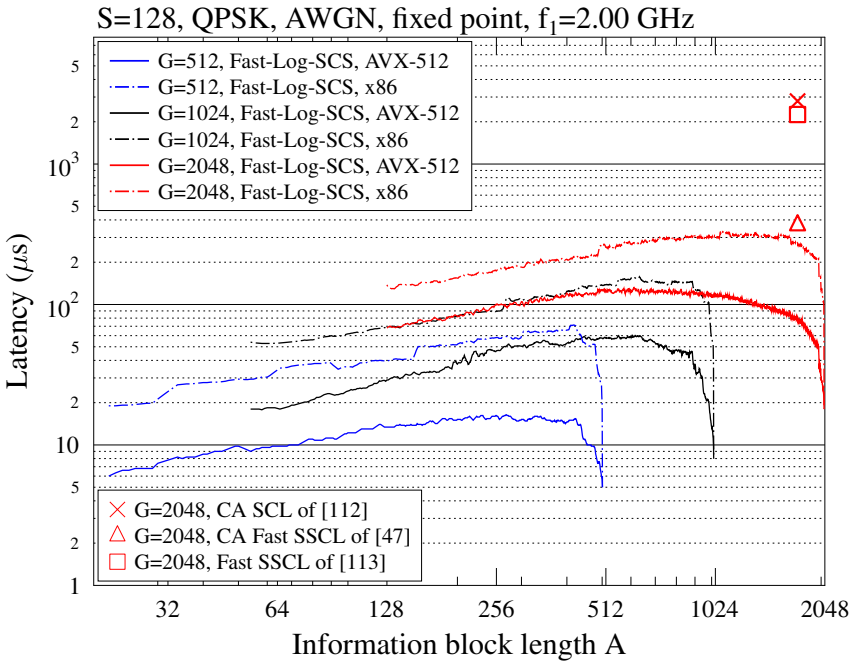


FIGURE 5.12: Latency D of the proposed $S = 128$ AVX-512 fixed-point Fast Log-SCS decoder operating at $f_1 = 2.00$ GHz for $G = 512, 1024$ and 2048 bits with various information block lengths A, as well as the normalised latency $\bar{D}$ of the works in [47, 112, 113] for $G = 2048$ and $A = 1723$ bits, where QPSK modulation is employed for communication over an AWGN channel in all cases.

$\bar{D}$ as

$$\bar{D} = \frac{P_i \times f_i \times D}{P_1 \times f_1}, \quad i = 2, 3, 4, \tag{5.4}$$

where $P_1 = 16$ and $f_1 = 2.00$ GHz are the parallelism and the clock frequency of the proposed CA Fast Log-SCS decoder and where P_i and f_i are the parallelism and the clock frequency of the decoder under consideration. As shown in Table 5.1 and plotted in Figure 5.12, our proposed AVX-512 Fast Log-SCS decoder has a normalised decoding latency $\bar{D}$ that is 21% that of the best performing benchmarker, namely, the CA Fast SSCL decoder of [47]. The throughput T of a software polar decoder is proportional to the reciprocal of its latency, according to $T = A/D$, whereas the normalised throughput $\bar{T}$ can be defined as $\bar{T} = A/\bar{D}$. Table 5.1 shows that the normalised throughput of our proposed AVX-512 CA Fast Log-SCS decoder is more than 6 times higher than that of the Fast SSCL decoder of [47]. The performance of the proposed Fast Log-SCS decoder performance on a 32-bit x86 CPU is also characterised in Table 5.1, which demonstrates that AVX-512 offers a 4.66 times improvement to throughput, in the case of CA Fast Log-SCS polar decoding. Table 5.1 also quantifies the maximum amount of memory required for storing the partial sum bits and the LLRs generated by the f, g and ϕ functions in the various implementations considered. These comparisons reveal that the

TABLE 5.1: Latency, throughput and memory requirement comparisons of different polar decoders when achieving a BLER of 10^{-3} for block lengths of $A = 1723$ $G = 2048$.

Platform	Intel(R) Xeon(R) Gold 6138		I5-6600K	I7-2600	I7-2600
f_i (GHz)	$f_1 = 2.00$		$f_2 = 3.90$	$f_3 = 3.40$	$f_4 = 3.40$
Algorithm	CA Fast Log-SCS		Fast SSCL [113]	CA Fast SSCL [47]	CA SCL [112]
LLR bit-width	32		8	32	32
AVX register bit-width	32	512	256		
List size L or stack size S	S =128		L =32		
D (μs)	274	79	577	433	3300
$\bar{D}$ (μs)	-	79	2252	368	2805
T (Mbps)	6.29	21.81	2.99	3.98	0.52
$\bar{T}$ (Mbps)	-	21.81	0.77	4.68	0.61
Memory (KB)	1096		82	280	280

proposed AVX implementation of the CA Fast Log-SCS decoder requires 13 times higher memory than the Fast SSCL decoder of [113], and 3 times higher memory than the CA Fast SSCL decoder of [47]. Hence, the improved latency and throughput of our Fast Log-SCS polar decoder is achieved at the cost of a higher memory requirement. However, it may be argued that the 1096 KB memory required is insignificant in the context of a high performance CPU.

5.5 Conclusions

In this chapter, we have proposed a novel Fast Log-SCS decoder for the 3GPP NR uplink polar code which reduces the complexity by 50%, compared to the state-of-the-art CA Fast polar SSCL decoder. We have also proposed an AVX-512 software implementation, which increases the normalised throughput by 6 times and reduces the normalised latency to 21% of the existing state-of-the-art AVX Fast SSCL polar decoder.

Chapter 6

Conclusions and Future Research

As shown in Figure 6.1, this thesis has designed low-latency and low-complexity decoding as well as demodulation schemes that satisfy the requirements of the Ultra-Reliable Low Latency Communication (URLLC) Fifth Generation (5G) mode. In particular, the advanced Arbitrarily Parallel Turbo Decoder (APTD) for turbo codes and the Improved Logarithmic Successive Cancellation Stack (Log-SCS) decoder for polar codes have been proposed in Chapter 2 and 4, respectively. These novel decoding algorithms exploit the unique code patterns of turbo or polar codes, and demonstrate their potential in the future applications of URLLC. In addition, Chapter 3 further proposes a concurrent Orthogonal Frequency-Division Multiplexing (OFDM) detection and turbo decoding scheme that further reduces the physical layer latency. Furthermore, practical high performance software implementation is considered in Chapter 5, which detailed the implementation of a fast polar decoder on x86 processors with Advanced Vector Extensions-512 (AVX-512) Single Instruction Multiple Data (SIMD) instructions. The summary of each chapter is as follows.

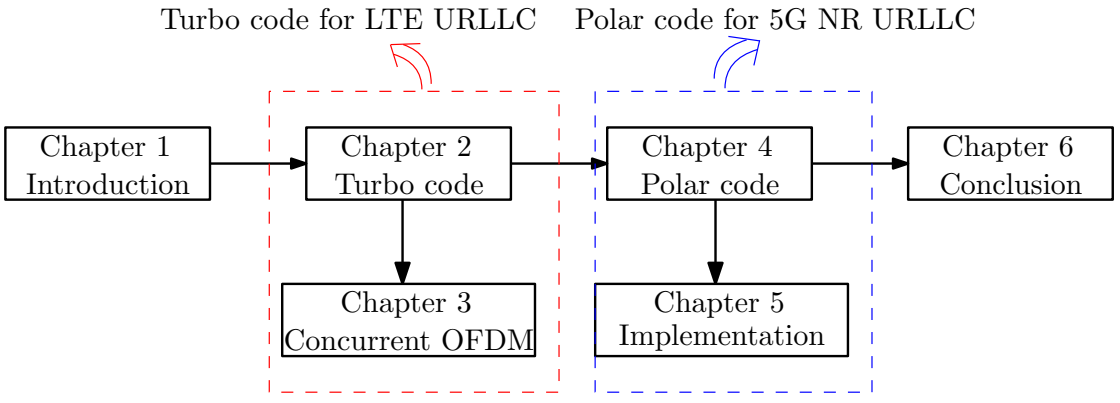


FIGURE 6.1: Thesis structure.

Chapter 2 started with a detailed overview of turbo encoding and decoding processes, which provides the background knowledge for the discussions throughout Chapters 2 and 3. By integrating the properties of the Long Term Evolution (LTE) State-of-the-art (SOTA) turbo decoder and the Fully-Parallel Turbo Decoder (FPTD), a novel APTD is introduced, which achieves low processing latencies while maintaining high reliability. The APTD algorithm allows the parallel operation of an arbitrary number P of processors, which is not limited to an integer factor of N, nor to N itself. The APTD employs several novel techniques in terms of the interleaver design, systematic information transmission, as well as the extrinsic information calculation, which further reduces the decoding complexity. Its particular benefits to short block length turbo codes further enhance its potential application in URLLC scenario. A comprehensive comparison of different turbo decoders is also summarised in Chapter 2.

While Chapter 2 was focused on the channel decoding process only, Chapter 3 extended the scope to optimize the entire receiver, proposing a concurrent receiver, demodulation, and decoding scheme, as shown in Figure 3.2. This approach reduces the decoding latency by threefold, and in the meantime, achieves the same error correction performance as if the turbo decoding process had only began after the reception of the OFDM symbol had been completed.

In addition to the turbo codes, polar codes are also investigated in the following two chapters. Chapter 4 first provides a detailed tutorial of the polar encoding and decoding process for the 5G uplink control channel. Following this, the improvements of the Log-SCS decoder have been proposed, which significantly reduces the decoding complexity by 7 times while achieving a similar error correction capability as the Log-SCL decoder.

Chapter 5 further considers the software implementation of the Improved Log-SCS polar decoder that has been proposed in Chapter 4. The proposed fast Log-SCS polar decoding algorithm and the corresponding software implementation are capable of attaining a decoding latency that is 80% lower than the SOTA Successive Cancellation List (SCL) polar decoder software implementation without loss of error correction performance. This is achieved by applying for simplified path-metric computations for the rate-0, rate-1 and repetition nodes in the proposed fast Log-SCS decoder, which reduces the decoding complexity by 50% on average. Furthermore, a 32-bit fixed point software implementation for x86 processors of the fast Log-SCS polar decoder is achieved as well, which maintains the same Block Error Rate (BLER) as that of the floating-point Log-SCS polar decoder. In addition, this software implementation is accelerated using SIMD instructions with AVX-512 for the first time, achieving a parallelism of 16 and satisfying the low-latency requirements of Software-Defined Radio (SDR) systems.

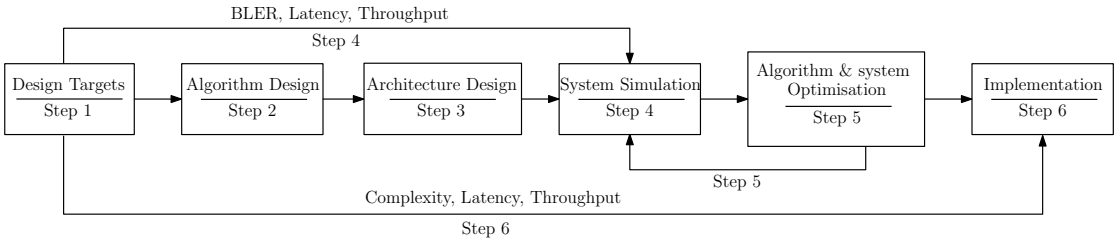


FIGURE 6.2: Design guidelines embraced by the thesis.

6.1 Design Guidelines

Figure 6.2 shows our guidelines for a typical algorithmic design with the aim of meeting particular requirements. More specifically, the first step is to identify and define the requirements and targets. In the case of the URLLC scenario considered in this treatise, our design targets included 1ms end-to-end latency, 10^{-5} BLER, low-complexity and high implementation flexibility. In particular, the LTE URLLC mode targets a 7.4 μs latency for turbo decoding, as discussed in Chapter 2. Furthermore, various block lengths and code rates in the 5G NR framework requires flexible implementation of channel decoders. These requirements guide the following design and gives the overall system characteristics.

Given the explicit requirements, an algorithm meeting the requirements is designed in Step 2. For example, in order to achieve the 7.4 μs latency requirement of the LTE URLLC mode, Chapter 2 proposes an APTD, which activates several processors in parallel compared to the SOTA turbo decoder and it is capable of supporting all the 188 legitimate block lengths.

Following a specific algorithm, a complete transmission and receive system needs to be built to examine whether the designed algorithm will be capable of satisfying the requirements identified in Step 1. For instance, a polar-coded QPSK-modulated transmitter is built in Chapter 4 to transmit polar coded blocks over AWGN channel, generating BLER results by Monte-Carlo simulation.

Then in Step 4, the system simulations quantifies the characteristics such as BLER, latency, complexity, etc., which allows the comparison with SOTA solutions, as demonstrated in Chapters 2 to 4. For example, from the BLER results of Chapter 2, we can conclude that the improved Log-SCS polar decoder achieves a superior error-correction capability than the SOTA Log-SCL decoder.

In general, not all the targets and requirements will be fulfilled in the first attempt. Therefore, Step 5 optimises the algorithm and system design in order to fill the gap between the current simulation results and the design target.

When the design targets are fulfilled after the optimisation, Step 6 implemented the proposed algorithm and system in software-defined radio or other practical implementation. We also need to characterise the throughput, latency, etc. For instance, Chapter 5 achieves a 32-bit fixed point software implementation for x86 processors of the fast Log-SCS polar decoder, which is capable of attaining a decoding latency that is 80% lower than the SOTA SCL polar decoder software implementation without loss of error correction performance and reduces the decoding complexity by 50% on average.

6.2 Future Work

Potential research interest inspired by our work of this thesis will be introduced in this section. First, the contention caused when interleaving multiple LLRs to the same Processing Element (PE) at the same time the second version of the APTD that we proposed in Chapter 2 will be addressed [22]. This contention problem arises when activating the number of PEs P that is not an integer factor of the block length N. We will address this by designing schedules that delay the interleaving of some extrinsic LLRs relative to the forward recursions in which they are generated. Our results seen in Figure 2.17 demonstrate that delaying the interleaving of extrinsic LLRs in this way has only a negligible impact upon the BLER performance. Furthermore, our future work will consider the practical hardware implementation of the proposed APTD algorithm in order to determine the throughput, latency, energy efficiency and hardware efficiency that can be achieved in practice [10, 13].

Second, the concurrent receiving architecture that we have proposed in Chapter 3 will have promising applications in different channel coding schemes and multicarrier communication systems [26–28]. For example, a concurrent receiving approach integrated with parallel LDPC decoding algorithm may achieve lower physical-layer latency in the 5G NR data channel. In addition, the complexity and latency trade-off may be further investigated for superior solutions. In addition to LDPC codes, we also aim to propose a novel design of polar codes for OFDM, whose salient feature is its diversity to tackle deep fading.

More generally, the relatively less-mature polar codes demands improvement in different aspects, especially in the reduction of latency and improvement of throughput. The improved Log-SCS decoder that we proposed in Chapter 4 have been software implemented in Chapter 5. As a complement to these, future work can consider hardware implementations in Field-Programmable Gate Array (FPGA) or Application Specific Integrated Circuit (ASIC), giving a low-complexity, low-latency implementation of the improved Log-SCS polar decoder for the potential application in 5G URLLC [9]. In addition, the BLER performance of the long block polar codes may be improved as well, in order to achieve the 10^{-5} reliability requirement of 5G URLLC.

In addition to the decoding schemes, the code construction for fading channels will be researched as well. For convenience of implementation, the 5G standard adopts a suboptimal solution for the code construction, which specifies a fixed polar code for different rates that is based on the β-expansion [57]. Therefore, the optimisation of the β-expansion technique for fading channels will be studied in the future.

Appendix A

A.1

After the culmulative FFT at the cth clock cycle, we have

$$\begin{aligned} Y_{z,c} &= \mathcal{F}\left\{\boldsymbol{h} * (\boldsymbol{x}\boldsymbol{I}_D) + \boldsymbol{n}\right\}_z \\ &= H_z \sum_{k=0}^{N} (\frac{1}{N} \sum_{n=0}^{N-1} X_n W_N^{nk}) d_k W_N^{-kz} + N_z, \end{aligned} \tag{A.1}$$

where $\boldsymbol{I}_D$ is first D columns of a diagonal matrix $\boldsymbol{I}_N$ with a size of $N \times N$ and

$$d_k = \begin{cases} 1, & \text{if } \frac{N}{C} c \geqslant k \\ 0, & \text{else} \end{cases}. \tag{A.2}$$

Therefore, we have

$$
\begin{aligned}
Y_{z,c} &= H_z \frac{1}{N} \sum_{k=0}^{N-1} \sum_{n=0}^{N-1} X_n W_N^{nk} d_k W_N^{-kz} + N_z \\
&= H_z \frac{1}{N} \sum_{k=0}^{N-1} \sum_{n=0}^{N-1} X_n W_N^{(n-z)k} d_k + N_z \\
&= H_z \frac{1}{N} \sum_{n=0}^{N-1} X_n \sum_{k=0}^{N-1} \left[W_N^{(n-z)k} d_k \right] + N_z \\
&= H_z \frac{1}{N} X_z \sum_{k=0}^{N-1} \left[W_N^0 d_k \right] + H_z \frac{1}{N} \sum_{n=0, n\neq z}^{N-1} X_n \sum_{k=0}^{N-1} \left[W_N^{(n-z)k} d_n \right] + N_z \\
&= \underbrace{H_z \frac{D}{N} X_z}_{\text{what we expect}} + \underbrace{H_z \frac{1}{N} \sum_{n=0, n\neq z}^{N-1} X_n \sum_{k=0}^{D-1} \left[W_N^{(n-z)k} \right]}_{\text{ICI}} + \underbrace{N_z}_{\text{noise}} \\
&= \frac{D}{N} \left(H_z X_z + \frac{H_z}{D} \sum_{n=0, n\neq z}^{N-1} \sum_{k=0}^{D-1} X_n W_N^{(n-z)k} + \frac{N}{D} N_z \right) \\
&= \frac{D}{N} \left(H_z X_z + \underbrace{\frac{H_z}{D} \sum_{n=0, n\neq z}^{N-1} \sum_{k=0}^{N-1} X_n W_N^{(n-z)k}}_{=0} - \frac{H_z}{D} \sum_{n=0, n\neq z}^{N-1} \sum_{k=D}^{N-1} X_n W_N^{(n-z)k} + \frac{N}{D} N_z \right) \\
&= \frac{D}{N} \left(H_z X_z - \frac{H_z}{D} \sum_{n=0, n\neq z}^{N-1} \sum_{k=D}^{N-1} X_n W_N^{(n-z)k} + \frac{N}{D} N_z \right)
\end{aligned} \tag{A.3}
$$

$$
\frac{P_{Noise}}{P_{signal}} = \frac{D * T_s}{N * T_s} \sum_{z=0}^{N-1} r_{noise}^2 = \frac{D}{N} \frac{N^2}{D^2} \sum_{z=0}^{N-1} N_z^2 = \frac{N}{D} N_0 \tag{A.4}
$$

$$\begin{aligned}\frac{P_{ICI}}{P_{signal}} &= \frac{D * T_s}{N * T_s} \sum_{z=0}^{N-1} r_{ici}^2 \\ &= \frac{D}{N} \sum_{z=0}^{N-1} \left\| \frac{H_z}{D} \sum_{n=0, n\neq z}^{N-1} \sum_{k=D}^{N-1} X_n W_N^{(n-z)k} \right\|^2 \\ &= \frac{D}{N} \sum_{z=0}^{N-1} \frac{\|H_z\|^2}{D^2} N(N-D) \\ &= \frac{N-D}{D} \sum_{k=0}^{N-1} \|H_k\|^2. \end{aligned} \tag{A.5}$$

Now, we obtain the relationship between the signal power and the power of ICI together with noise, as

$$\frac{P_{ICI+Noise}}{P_{signal}} = \frac{N}{D} N_0 + \frac{N-D}{D} \sum_{k=0}^{N-1} \|H_k\|^2. \tag{A.6}$$

Therefore, given the unit signal power, the variance of the ICI and noise can be expressed as

$$\begin{aligned}\sigma_r &= \frac{\frac{N}{D} N_0 + \frac{N-D}{D} \sum_{k=0}^{N-1} \|H_k\|^2}{2} \\ &= \frac{N + (N-D)\gamma \sum_{k=0}^{N-1} \|H_k\|^2}{2D\gamma}. \end{aligned} \tag{A.7}$$

Glossary

3GPP	3rd Generation Partnership Project
4G	Fourth Generation
5G	Fifth Generation
ADC	Analogue to Digital Converter
AP	*a posteriori*
APTD	Arbitrarily Parallel Turbo Decoder
ARP	Almost Regular Permutation
ASIC	Application Specific Integrated Circuit
AVX-512	Advanced Vector Extensions-512
AWGN	Additive White Gaussian Noise
B-DMCs	Binary-input Discrete Memoryless Channels
BER	Bit Error Ratio
BLER	Block Error Rate
BP	Belief Propagation
CA	CRC Aided
CCR	Computational Complexity Reduction
CDMA	Code Division Multiple Access
CIR	Channel Impulse Response
CP	Cyclic Prefix
CRC	Cyclic Redundancy Check
DAC	Digital to Analogue Converter
DFT	Discrete Fourier Transform
eMBB	enhanced Mobile Broadband
eNB	evolved Node B
ETU	Extended Typical Urban model
FAR	False Alarm Rate
FD	Frequency Domain
FFT	Fast Fourier Transform
FPGA	Field-Programmable Gate Array
FPTD	Fully-Parallel Turbo Decoder

GCD	Greatest Common Divisor
ICI	Inter-Carrier Interference
IDFT	Inverse Discrete Fourier Transform
IFFT	Inverse Fast Fourier Transform
LDPC	Low-Density Parity-Check
LLR	Log-likelihood ratio
Log-BCJR	Logarithmic Bahl-Cocke-Jelinek-Raviv
Log-SCL	Logarithmic Successive Cancellation List
Log-SCS	Logarithmic Successive Cancellation Stack
LR	Likelihood ratio
LTE	Long Term Evolution
MAP	Maximum *a posteriori*
MI	Mutual Information
MIMO	Multiple-Input and Multiple-Output
mMTC	massive Machine-Type Communications
MQAM	M-ary Quadrature Amplitude Modulation
NR	New Radio
OFDM	Orthogonal Frequency-Division Multiplexing
PC	Parity Check
PDCCH	Physical Downlink Control Channel
PE	Processing Element
PM	Path Metric
PPV	Polyanskyi-Poor-Verdù
PSC	Parallel to Serial Converter
PUCCH	Physical Uplink Control Channel
PUSCH	Physical Uplink Shared Channel
QAM	Quadrature Amplitude Modulation
QPP	Quadratic Permutation Polynomial
QPSK	Quaternary Phase Shift Keying
RF	Radio Frequency
SC	Successive Cancellation
SCL	Successive Cancellation List
SCS	Successive Cancellation Stack
SDR	Software-Defined Radio
SIMD	Single Instruction Multiple Data
SNR	Signal to Noise Ratio
SOTA	State-of-the-art
SPC	Serial to Parallel Converter
SSC	Simplified Successive Cancellation
SSCL	Simplified Successive Cancellation List
sTTI	shortened Transmission Time Interval

TD	Time Domain
TSMC	Taiwan Semiconductor Manufacturing Company
UE	User Equipment
URLLC	Ultra-Reliable Low Latency Communication
XOR	eXclusive-OR

Bibliography

[1] C. E. Shannon, "A mathematical theory of communication," *Bell system technical journal*, vol. 27, no. 3, pp. 379–423, 1948.

[2] C. Berrou and A. Glavieux, "Near optimum error correcting coding and decoding: Turbo-codes," *IEEE Transactions on Communications*, vol. 44, no. 10, pp. 1261–1271, 1996.

[3] C. Berrou and A. Glavieux, "Turbo codes," *Encyclopedia of Telecommunications*, 2003.

[4] L. Hanzo, T. Liew, B. Yeap, R. Tee, and S. X. Ng, *Turbo coding, turbo equalisation and space-time coding: EXIT-chart-aided near-capacity designs for wireless channels*. John Wiley & Sons, 2011.

[5] A. Viterbi, "Error bounds for convolutional codes and an asymptotically optimum decoding algorithm," *IEEE transactions on Information Theory*, vol. 13, no. 2, pp. 260–269, 1967.

[6] E-UTRA, "Multiplexing and channel coding," *3rd Generation Partnership Project Std. 3GPP TS*, vol. 36, p. V8, 2008.

[7] R. Gallager, "Low-density parity-check codes," *IRE Transactions on information theory*, vol. 8, no. 1, pp. 21–28, 1962.

[8] E. Arikan, "Channel polarization: A method for constructing capacity-achieving codes for symmetric binary-input memoryless channels," *IEEE Transactions on Information Theory*, vol. 55, no. 7, pp. 3051–3073, 2009.

[9] M. Shafi, A. F. Molisch, P. J. Smith, T. Haustein, P. Zhu, P. De Silva, F. Tufvesson, A. Benjebbour, and G. Wunder, "5G: A tutorial overview of standards, trials, challenges, deployment, and practice," *IEEE journal on selected areas in communications*, vol. 35, no. 6, pp. 1201–1221, 2017.

[10] T. Fehrenbach, R. Datta, B. Göktepe, T. Wirth, and C. Helge, "URLLC services in 5G-low latency enhancements for LTE," *Accepted for publication at IEEE Vehicular Technology Conference (VTC)*, Fall 2018.

[11] B. Tahir, S. Schwarz, and M. Rupp, "BER comparison between convolutional, turbo, LDPC, and polar codes," in *Telecommunications (ICT), 2017 24th International Conference on*, pp. 1–7, IEEE, 2017.

[12] T. Richardson and S. Kudekar, "Design of low-density parity check codes for 5G new radio," *IEEE Communications Magazine*, vol. 56, no. 3, pp. 28–34, 2018.

[13] 3GPP RP-171489, "Work item on ultra reliable low latency communication for LTE," June 2017.

[14] X. Zhang, "Latency reduction with short processing time and short TTI length," in *Intelligent Signal Processing and Communication Systems (ISPACS), 2017 International Symposium on*, pp. 545–549, IEEE, 2017.

[15] H. Ji, S. Park, J. Yeo, Y. Kim, J. Lee, and B. Shim, "Introduction to ultra reliable and low latency communications in 5G," *Computing Research Repository (CoRR) abs/1704.05565*, 2017.

[16] J. C. S. Arenas, T. Dudda, and L. Falconetti, "Ultra-low latency in next generation lte radio access," in *SCC 2017; 11th International ITG Conference on Systems, Communications and Coding; Proceedings of*, pp. 1–6, VDE, 2017.

[17] 3GPP RP-161299, "Work item on shortened TTI and processing time for LTE," June 2016.

[18] N. A. Johansson, Y.-P. E. Wang, E. Eriksson, and M. Hessler, "Radio access for ultra-reliable and low-latency 5G communications," in *Communication Workshop (ICCW), IEEE International Conference on*, pp. 1184–1189, IEEE, 2015.

[19] P. Robertson, E. Villebrun, and P. Hoeher, "A comparison of optimal and sub-optimal MAP decoding algorithms operating in the log domain," in *IEEE International Conference on Communications, ICC'95 Seattle,'Gateway to Globalization',*, vol. 2, pp. 1009–1013, IEEE, 1995.

[20] M. Bickerstaff, L. Davis, C. Thomas, D. Garrett, and C. Nicol, *A 24Mb/s radix-4 logMAP turbo decoder for 3GPP-HSDPA mobile wireless*. PhD thesis, IEEE, 2003.

[21] Y. Zhang and K. K. Parhi, "High-throughput radix-4 logMAP turbo decoder architecture," in *2006 Fortieth Asilomar Conference on Signals, Systems and Computers*, 2006.

[22] S. Yoon and Y. Bar-Ness, "A parallel MAP algorithm for low latency turbo decoding," *IEEE Communications Letters*, vol. 6, no. 7, pp. 288–290, 2002.

[23] T. Ilnseher, F. Kienle, C. Weis, and N. Wehn, "A 2.15 GBit/s turbo code decoder for LTE Advanced base station applications," in *7th International Symposium on Turbo Codes and Iterative Information Processing (ISTC)*, pp. 21–25, IEEE, 2012.

[24] R. G. Maunder, "A fully-parallel turbo decoding algorithm," *IEEE Transactions on Communications*, vol. 63, no. 8, pp. 2762–2775, 2015.

[25] A. S. Barbulescu and S. S. Pietrobon, "Interleaver design for turbo codes," *Electronics Letters*, vol. 30, no. 25, pp. 2107–2108, 1994.

[26] L. Hanzo, S. X. Ng, W. Webb, and T. Keller, *Quadrature amplitude modulation: From basics to adaptive trellis-coded, turbo-equalised and space-time coded OFDM, CDMA and MC-CDMA systems*. IEEE Press-John Wiley, 2004.

[27] L. Hanzo, M. Münster, B. Choi, and T. Keller, *OFDM and MC-CDMA for broadband multi-user communications, WLANs and broadcasting*. John Wiley & Sons, 2005.

[28] J. A. Bingham, "Multicarrier modulation for data transmission: An idea whose time has come," *IEEE Communications magazine*, vol. 28, no. 5, pp. 5–14, 1990.

[29] L. Xu, J. Yang, D. Huang, and A. Cantoni, "Exploiting cyclic prefix for Turbo-OFDM receiver design," *IEEE Access*, vol. 5, pp. 15762–15775, 2017.

[30] I. Shubhi and Y. Sanada, "Joint turbo decoding for overloaded MIMO-OFDM systems," *IEEE Transactions on Vehicular Technology*, vol. 66, no. 1, pp. 433–442, 2017.

[31] K. Niu, K. Chen, J. Lin, and Q. Zhang, "Polar codes: Primary concepts and practical decoding algorithms," *IEEE Communications magazine*, vol. 52, no. 7, pp. 192–203, 2014.

[32] Huawei, HiSilicon, "R1-1608862: Polar code construction for NR," *3GPP TSG RAN WG1 Meeting #86bis*, no. 10, 2016.

[33] C. Leroux, I. Tal, A. Vardy, and W. J. Gross, "Hardware architectures for successive cancellation decoding of polar codes," in *Acoustics, Speech and Signal Processing (ICASSP), 2011 IEEE International Conference on*, pp. 1665–1668, IEEE, 2011.

[34] E. Arikan, "A performance comparison of polar codes and Reed-Muller codes," *IEEE Communications Letters*, vol. 12, no. 6, 2008.

[35] B. Yuan and K. K. Parhi, "Early stopping criteria for energy-efficient low-latency belief-propagation polar code decoders," *IEEE Transactions on Signal Processing*, vol. 62, no. 24, pp. 6496–6506, 2014.

[36] Y. Zhang, A. Liu, X. Pan, Z. Ye, and C. Gong, "A modified belief propagation polar decoder," *IEEE Communications Letters*, vol. 18, no. 7, pp. 1091–1094, 2014.

[37] A. Alamdar-Yazdi and F. R. Kschischang, "A simplified successive-cancellation decoder for polar codes," *IEEE communications letters*, vol. 15, no. 12, pp. 1378–1380, 2011.

[38] C. Leroux, A. J. Raymond, G. Sarkis, and W. J. Gross, "A semi-parallel successive-cancellation decoder for polar codes," *IEEE Transactions on Signal Processing*, vol. 61, no. 2, pp. 289–299, 2013.

[39] I. Tal and A. Vardy, "List decoding of polar codes," in *Information Theory Proceedings (ISIT), 2011 IEEE International Symposium on*, pp. 1–5, IEEE, 2011.

[40] I. Tal and A. Vardy, "List decoding of polar codes," *IEEE Transactions on Information Theory*, vol. 61, no. 5, pp. 2213–2226, 2015.

[41] K. Chen, K. Niu, and J. Lin, "List successive cancellation decoding of polar codes," *Electronics letters*, vol. 48, no. 9, pp. 500–501, 2012.

[42] K. Niu and K. Chen, "Stack decoding of polar codes," *Electronics letters*, vol. 48, no. 12, pp. 695–697, 2012.

[43] P. Trifonov, "Efficient design and decoding of polar codes," *IEEE Transactions on Communications*, vol. 60, no. 11, pp. 3221–3227, 2012.

[44] K. Chen, K. Niu, and J. Lin, "Improved successive cancellation decoding of polar codes," *IEEE Transactions on Communications*, vol. 61, no. 8, pp. 3100–3107, 2013.

[45] V. Miloslavskaya and P. Trifonov, "Sequential decoding of polar codes," *IEEE Communications Letters*, vol. 18, no. 7, pp. 1127–1130, 2014.

[46] A. Balatsoukas-Stimming, M. B. Parizi, and A. Burg, "LLR-based successive cancellation list decoding of polar codes," *IEEE Transactions on Signal Processing*, vol. 63, no. 19, pp. 5165–5179, 2015.

[47] G. Sarkis, P. Giard, A. Vardy, C. Thibeault, and W. J. Gross, "Fast list decoders for polar codes," *IEEE Journal on Selected Areas in Communications*, vol. 34, no. 2, pp. 318–328, 2016.

[48] Y. Fan, J. Chen, C. Xia, C.-y. Tsui, J. Jin, H. Shen, and B. Li, "Low-latency list decoding of polar codes with double thresholding," in *Acoustics, Speech and Signal Processing (ICASSP), 2015 IEEE International Conference on*, pp. 1042–1046, IEEE, 2015.

[49] J. Lin, C. Xiong, and Z. Yan, "A reduced latency list decoding algorithm for polar codes," in *Signal Processing Systems (SiPS), 2014 IEEE Workshop on*, pp. 1–6, IEEE, 2014.

[50] G. Sarkis, P. Giard, A. Vardy, C. Thibeault, and W. J. Gross, "Fast polar decoders: Algorithm and implementation," *IEEE Journal on Selected Areas in Communications*, vol. 32, no. 5, pp. 946–957, 2014.

[51] H. Aurora, C. Condo, and W. J. Gross, "Low-complexity software stack decoding of polar codes," in *Circuits and Systems (ISCAS), 2018 IEEE International Symposium on*, pp. 1–5, IEEE, 2018.

[52] J. Sun and O. Y. Takeshita, "Interleavers for turbo codes using permutation polynomials over integer rings," *IEEE Transactions on Information Theory*, vol. 51, no. 1, pp. 101–119, 2005.

[53] O. Y. Takeshita, "On maximum contention-free interleavers and permutation polynomials over integer rings," *IEEE Transactions on Information Theory*, vol. 52, no. 3, pp. 1249–1253, 2006.

[54] TSGRANGRA, Network, "Evolved universal terrestrial radio access (E-UTRA); multiplexing and channel coding," *3rd Generation Partnership Project (3GPP)*, vol. TS 36, 2009.

[55] D. Soldani, Y. J. Guo, B. Barani, P. Mogensen, I. Chih-Lin, and S. K. Das, "5G for ultra-reliable low-latency communications," *IEEE Network*, vol. 32, no. 2, pp. 6–7, 2018.

[56] G. P. Fettweis, "The tactile internet: applications and challenges," *IEEE Vehicular Technology Magazine*, vol. 9, no. 1, pp. 64–70, 2014.

[57] 3GPP TS 38.212 V15.1.1, "NR Multiplexing and channel coding," *3rd Generation Partnership Project Std. 3GPP*, 2018.

[58] S. A. Hashemi, C. Condo, and W. J. Gross, "Fast and flexible successive-cancellation list decoders for polar codes," *IEEE Transactions on Signal Processing*, vol. 65, no. 21, pp. 5756–5769, 2017.

[59] B. Le Gal, C. Leroux, and C. Jego, "Multi-gb/s software decoding of polar codes," *IEEE transactions on signal processing*, vol. 63, no. 2, pp. 349–359, 2015.

[60] Altera, "3GPP LTE turbo reference design," Jan 2011.

[61] A. Li, L. Xiang, T. Chen, R. G. Maunder, B. M. Al-Hashimi, and L. Hanzo, "VLSI implementation of fully parallel LTE turbo decoders," *IEEE Access*, vol. 4, pp. 323–346, 2016.

[62] E. Boutillon, W. J. Gross, and P. G. Gulak, "VLSI architectures for the MAP algorithm," *IEEE Transactions on Communications*, vol. 51, no. 2, pp. 175–185, 2003.

[63] J. Vogt and A. Finger, "Improving the Max-Log-MAP turbo decoder," *Electronics Letters*, vol. 36, no. 23, pp. 1937–1939, 2000.

[64] C. E. Shannon, "A mathematical theory of communication," *ACM SIGMOBILE Mobile Computing and Communications Review*, vol. 5, no. 1, pp. 3–55, 2001.

[65] L. Kong, S. X. Ng, R. G. Maunder, and L. Hanzo, "Maximum-throughput irregular distributed space-time code for near-capacity cooperative communications," *IEEE Transactions on Vehicular Technology*, vol. 59, no. 3, pp. 1511–1517, 2010.

[66] J. Hagenauer, "The exit chart-introduction to extrinsic information transfer in iterative processing," in *Signal Processing Conference, 2004 12th European*, pp. 1541–1548, IEEE, 2004.

[67] R. Shrestha and R. P. Paily, "High-throughput turbo decoder with parallel architecture for LTE wireless communication standards," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 61, no. 9, pp. 2699–2710, 2014.

[68] H. Shariatmadari, S. Iraji, R. Jantti, P. Popovski, Z. Li, and M. A. Uusitalo, "Fifth-generation control channel design: Achieving ultrareliable low-latency communications," *IEEE Vehicular Technology Magazine*, vol. 13, no. 2, pp. 84–93, 2018.

[69] A. Aissioui, A. Ksentini, A. M. Gueroui, and T. Taleb, "On enabling 5G automotive systems using follow me edge-cloud concept," *IEEE Transactions on Vehicular Technology*, vol. 67, pp. 5302–5316, June 2018.

[70] Y. Polyanskiy, H. V. Poor, and S. Verdú, "Channel coding rate in the finite blocklength regime," *IEEE Transactions on Information Theory*, vol. 56, no. 5, pp. 2307–2359, 2010.

[71] M. Condoluci, M. Dohler, G. Araniti, A. Molinaro, and K. Zheng, "Toward 5G densenets: architectural advances for effective machine-type communications over femtocells," *IEEE Communications Magazine*, vol. 53, no. 1, pp. 134–141, 2015.

[72] M. Luvisotto, Z. Pang, and D. Dzung, "Ultra high performance wireless control for critical applications: challenges and directions," *IEEE Transactions on Industrial Informatics*, vol. 13, no. 3, pp. 1448–1459, 2017.

[73] P.-H. Chiu, P.-H. Tseng, and K.-T. Feng, "Interactive mobile augmented reality system for image and hand motion tracking," *IEEE Transactions on Vehicular Technology*, vol. 67, no. 10, pp. 9995–10009, 2018.

[74] M. Mozaffari, W. Saad, M. Bennis, and M. Debbah, "Unmanned aerial vehicle with underlaid device-to-device communications: performance and tradeoffs," *IEEE Transactions on Wireless Communications*, vol. 15, no. 6, pp. 3949–3963, 2016.

[75] M. Bennis, M. Debbah, and H. V. Poor, "Ultra-reliable and low-latency wireless communication: Tail, risk and scale," *arXiv preprint arXiv:1801.01270*, 2018.

[76] M. S. Elbamby, C. Perfecto, M. Bennis, and K. Doppler, "Toward low-latency and ultra-reliable virtual reality," *IEEE Network*, vol. 32, no. 2, pp. 78–84, 2018.

[77] B. Soret, P. Mogensen, K. I. Pedersen, and M. C. Aguayo-Torres, "Fundamental tradeoffs among reliability, latency and throughput in cellular networks," in *Globecom Workshops (GC Wkshps), 2014*, pp. 1391–1396, IEEE, 2014.

[78] D. Schneider, "The microsecond market," *IEEE spectrum*, vol. 6, no. 49, pp. 66–81, 2012.

[79] M. Simsek, A. Aijaz, M. Dohler, J. Sachs, and G. Fettweis, "5G-enabled tactile internet," *IEEE Journal on Selected Areas in Communications*, vol. 34, no. 3, pp. 460–473, 2016.

[80] G. P. Fettweis, "A 5G wireless communications vision," *Microwave Journal*, vol. 55, no. 12, pp. 24–36, 2012.

[81] J. W. Cooley and J. W. Tukey, "An algorithm for the machine calculation of complex fourier series," *Mathematics of computation*, vol. 19, no. 90, pp. 297–301, 1965.

[82] W. T. Cochran, J. W. Cooley, D. L. Favin, H. D. Helms, R. A. Kaenel, W. W. Lang, G. Maling, D. E. Nelson, C. M. Rader, and P. D. Welch, "What is the fast fourier transform?," *Proceedings of the IEEE*, vol. 55, no. 10, pp. 1664–1674, 1967.

[83] S. Weinstein and P. Ebert, "Data transmission by frequency-division multiplexing using the discrete fourier transform," *IEEE transactions on Communication Technology*, vol. 19, no. 5, pp. 628–634, 1971.

[84] S. Darlington, "On digital single-sideband modulators," *IEEE Transactions on Circuit Theory*, vol. 17, no. 3, pp. 409–414, 1970.

[85] B. Hirosaki, "An orthogonally multiplexed QAM system using the discrete fourier transform," *IEEE Transactions on Communications*, vol. 29, no. 7, pp. 982–989, 1981.

[86] Access, Evolved Universal Terrestrial Radio, "Base station (BS) radio transmission and reception," *3GPP TS 36.104 V14*, vol. 3, 2009.

[87] E. Arikan and E. Telatar, "On the rate of channel polarization," in *2009 IEEE International Symposium on Information Theory*, pp. 1493–1495, IEEE, 2009.

[88] B. Li, H. Shen, and D. Tse, "An adaptive successive cancellation list decoder for polar codes with cyclic redundancy check," *IEEE Communications Letters*, vol. 16, no. 12, pp. 2044–2047, 2012.

[89] C. Xiong, J. Lin, and Z. Yan, "Symbol-based successive cancellation list decoder for polar codes," in *Signal Processing Systems (SiPS), 2014 IEEE Workshop on*, pp. 1–6, IEEE, 2014.

[90] Z. Zhang, L. Zhang, X. Wang, C. Zhong, and H. V. Poor, "A split-reduced successive cancellation list decoder for polar codes," *IEEE Journal on Selected Areas in Communications*, vol. 34, no. 2, pp. 292–302, 2016.

[91] H. Zhou, C. Zhang, W. Song, S. Xu, and X. You, "Segmented CRC-aided SC list polar decoding," in *Vehicular Technology Conference (VTC Spring), 2016 IEEE 83rd*, pp. 1–5, IEEE, 2016.

[92] W. Song, H. Zhou, Y. Zhao, S. Zhang, X. You, and C. Zhang, "Low-complexity segmented CRC-aided SC stack decoder for polar codes," in *2016 50th Asilomar Conference on Signals, Systems and Computers*, pp. 1189–1193, Nov 2016.

[93] X. Liang, H. Zhou, Z. Wang, X. You, and C. Zhang, "Segmented successive cancellation list polar decoding with joint BCH-CRC codes," in *2017 51st Asilomar Conference on Signals, Systems, and Computers*, pp. 1509–1513, Oct 2017.

[94] S. Li, L. Lu, Y. Deng, J. Liu, and T. Huang, "A Reused-Public-Path successive cancellation list decoding for polar codes with CRC," *IEEE Communications Letters*, vol. 21, pp. 2566–2569, Dec 2017.

[95] P. Giard and A. Burg, "Fast-SSC-flip decoding of polar codes," in *Wireless Communications and Networking Conference Workshops (WCNCW), 2018 IEEE*, pp. 73–77, IEEE, 2018.

[96] X. Liu, S. Wu, X. Xu, J. Jiao, and Q. Zhang, "Improved polar SCL decoding by exploiting the error correction capability of CRC," *IEEE Access*, vol. 7, pp. 7032–7040, 2019.

[97] P. Chen, B. Bai, Z. Ren, J. Wang, and S. Sun, "Hash-Polar codes with application to 5G," *IEEE Access*, pp. 1–1, 2019.

[98] K. Niu and K. Chen, "CRC-aided decoding of polar codes," *IEEE Communications Letters*, vol. 16, no. 10, pp. 1668–1671, 2012.

[99] U. U. Fayyaz and J. R. Barry, "Low-complexity soft-output decoding of polar codes," *IEEE Journal on Selected Areas in Communications*, vol. 32, no. 5, pp. 958–966, 2014.

[100] T. Murata and H. Ochiai, "On design of CRC codes for polar codes with successive cancellation list decoding," in *Information Theory (ISIT), 2017 IEEE International Symposium on*, pp. 1868–1872, IEEE, 2017.

[101] Q. Zhang, A. Liu, X. Pan, and K. Pan, "CRC code design for list decoding of polar codes," *IEEE Communications Letters*, vol. 21, pp. 1229–1232, June 2017.

[102] J. Chen, Y. Chen, K. Jayasinghe, D. Du, and J. Tan, "Distributing CRC bits to aid polar decoding," in *2017 IEEE Globecom Workshops (GC Wkshps)*, pp. 1–6, Dec 2017.

[103] F. Cheng, A. Liu, Y. Zhang, and J. Ren, "CRC location design for polar codes," *IEEE Communications Letters*, vol. 22, pp. 2202–2205, Nov 2018.

[104] P. Chen, M. Xu, B. Bai, and J. Wang, "Design and performance of polar codes for 5G communication under high mobility scenarios," in *Vehicular Technology Conference (VTC Spring), 2017 IEEE 85th*, pp. 1–5, IEEE, 2017.

[105] D. Hui, M. Breschel, and Y. Blankenship, "Interleaved CRC for polar codes," in *2018 IEEE 87th Vehicular Technology Conference (VTC Spring)*, pp. 1–5, June 2018.

[106] Z. Babar, Z. B. K. Egilmez, L. Xiang, D. Chandra, R. G. Maunder, S. X. Ng, and L. Hanzo, "Polar codes and their quantum-domain counterparts," *IEEE Communications Surveys & Tutorials*, 2019.

[107] T. Erseghe, "Coding in the finite-blocklength regime: Bounds based on laplace integrals and their asymptotic approximations," *IEEE Transactions on Information Theory*, vol. 62, no. 12, pp. 6854–6883, 2016.

[108] Nokia, Shanghai-Bell, Alcatel-Lucent, "R1-1705860: Polar codes for control channels," *3GPP TSG-RAN WG1 #88bis Meeting*, vol. 16, no. 10, 2017.

[109] D. Hui, S. Sandberg, Y. Blankenship, M. Andersson, and L. Grosjean, "Channel coding in 5g new radio: A tutorial overview and performance comparison with 4g lte," *ieee vehicular technology magazine*, vol. 13, no. 4, pp. 60–69, 2018.

[110] G. Sarkis and W. J. Gross, "Increasing the throughput of polar decoders," *IEEE Communications Letters*, vol. 17, no. 4, pp. 725–728, 2013.

[111] P. Giard, G. Sarkis, C. Thibeault, and W. J. Gross, "Fast software polar decoders," in *Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on*, pp. 7555–7559, IEEE, 2014.

[112] G. Sarkis, P. Giard, A. Vardy, C. Thibeault, and W. J. Gross, "Increasing the speed of polar list decoders," in *Signal Processing Systems (SiPS), 2014 IEEE Workshop on*, pp. 1–6, IEEE, 2014.

[113] M. Léonardon, A. Cassagne, C. Leroux, C. Jégo, L.-P. Hamelin, and Y. Savaria, "Fast and flexible software polar list decoders," *Springer Journal of Signal Processing Systems (JSPS)*, pp. 1–16, January 2019.

[114] D. Abrahams and A. Gurtovoy, *C++ Template Metaprogramming: Concepts, Tools, and Techniques from Boost and Beyond (C++ in Depth Series)*. Addison-Wesley Professional, 2004.

[115] L. Xiang, Z. B. K. Egilmez, R. G. Maunder, and L. Hanzo, "Crc-aided logarithmic stack decoding of polar codes for ultra reliable low latency communication in 3gpp new radio," *IEEE Access*, vol. 7, pp. 28559–28573, 2019.

[116] Y. Wu and B. D. Woerner, "The influence of quantization and fixed point arithmetic upon the BER performance of turbo codes," in *Vehicular Technology Conference, 1999 IEEE 49th*, vol. 2, pp. 1683–1687, IEEE, 1999.

Subject Index

Author Index