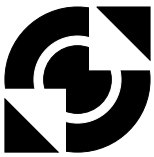


Signals & Systems

Turbo Multiuser Detection Architectures

M.Sc. Thesis

Gerben Heinen

University of Twente
Department of Electrical Engineering,
Mathematics & Computer Science (EEMCS)
Signals & Systems Group (SAS)
P.O. Box 217
7500 AE Enschede
The Netherlands

Report Number:	SAS03.035
Report Date:	December 4, 2003
Period of Work:	01/02/2003 – 10/12/2003
Thesis Committee:	Prof. Dr. ir. C.H. Slump
	ir. F.W. Hoeksema
	ir. R. Schiphorst
	ir. K.L. Hofstra
	ir. J. Potman

Abstract

The discovery of Turbo Codes in 1996 by Berrou et. al. proved to be a huge boost for the research of channel coding. The Turbo Principle behind turbo codes was found to be applicable in other areas. One of these areas is Multiuser Detection. In this thesis, Turbo Multiuser Detection is investigated in order to answer two main questions. The questions concern the performance gain that is obtained when turbo multiuser detection is used instead of non-turbo multiuser detection and the convergence behavior of turbo multiuser detection.

The performance gain is determined by comparing the bit-error-rate (BER) chart of a turbo multiuser detection architecture with the BER chart of a non-turbo multiuser detector. It was found that turbo multiuser detection results in a dramatical performance gain when $E_b/N_0 > 3\,\mathrm{dB}$ and more than one iteration is performed.

The convergence behavior of turbo multiuser detection is analyzed with the help of EXIT charts. EXIT charts are recently proposed by S. ten Brink as a tool to analyze the convergence behavior of turbo architectures. EXIT charts are discussed in this thesis. An EXIT chart of a turbo multiuser detection architecture is created. From this chart, the minimum number of iterations to obtain the lowest BER possible are found.

EXIT charts are also used to analyze the difference of iterating a-posteriori and extrinsic information in a turbo architecture. The analysis shows that EXIT charts of a-posteriori information give results, which contradict the results of simulations on turbo architectures.

Acknowledgements

For the electrical engineering curriculum every student has to perform a Individual Research Assignment. Before I started thinking about the subject of this assignment, I already got interested in signal processing and cellular wireless mobile communications, because of some courses I took on these subjects. I contacted Prof. Dr. ir. C.H. Slump of the Signals & Systems laboratory and he told me to contact Fokke Hoeksema and Roel Schiphorst. With them I worked on RAKE receivers for DS-CDMA systems for my individual research assignment. This experience led me to contact them again when I was looking for a subject for my master's thesis. They offered me some assignments, from which I choose 'Turbo Multiuser Detection Architectures' as the subject of my thesis. Together with Fokke and Roel, Jordy Potman and Klaas Hofstra became my supervisors. The results of my investigation from February 2003 until December 2003, are presented in this thesis.

First of all I wish to thank Prof. Dr. ir. C.H. Slump for allowing me to perform my master assignment in the Signals & Systems group. I wish to thank Fokke, Roel, Jordy and Klaas for providing guidance, support and comments during my master's assignment, but most of all I would like to thank them for letting me work with them as a colleague. Whenever I encountered a problem, I found difficult to solve immediately, they offered me guidance. Further I would like to thank the people of the Signals & Systems group for providing the nice work environment and facilities that made me able to perform my work. Last but certainly not least, I wish to thank my family, friends and roommates for their accompany and support during my studies.

Enschede, December 4, 2003

Gerben Heinen

Contents

List of Figures

List of Tables

1

Introduction

In this thesis the turbo principle, in particulary applied to turbo multiuser detection, is described and analyzed. The turbo principle is based on two techniques; concatenated coding and iterative decoding. The turbo principle was first used for turbo coding, but it was found that it could also be used for other decoding processes, like turbo multiuser detection and turbo equalization. In the turbo principle information is iterated between decoders several times, before making a bit decision. Applying the turbo principle to decoding problems has shown to give good performance, approaching the Shannon limit, while the overall system's complexity is not increased.

1.1 Background

In 1948 *Claude Shannon* introduced his now famous theorem on information [34]. Besides other things, he stated that, under certain conditions, it is possible to send data over a transmission line with an arbitrarily reliability by using channel coding. However, there is a limit to the capacity of the channel which depends on the signal power and bandwidth of the transmitted data bits and the noise power of the channel. Shannon did tell us about this capacity, but he did not tell us how to reach it.

For over 50 years scientist tried to find a code, that would make Shannon's promises come true. A lot of good codes were found, but they still did not come very close to the limit Shannon introduced. Eventually the search lead to some desperate remarks like '*All codes are good, except the ones we can think of*' made by various people in the coding research community. Just when people started to get desperate about ever finding this super-code, Berrou and Glavieux published a paper in 1993 in which they showed a coding technique that approached the Shannon limit very closely [6]. The coding technique was baptized with the name 'Turbo-Coding' and the corresponding codes were called 'Turbo-Codes'.

Instead of making research on codes obsolete, the discovery of turbo-

coding started, what is called by many authors, *the renaissance of channel coding*. Codes that were not considered for more than three decades became interesting again due to the discovery of turbo-coding. As we will see later the term 'turbo', does not really apply to the encoding, but more to the decoding. Later was found that this turbo principle, was also applicable in other areas of signal processing, like equalization and multi-user detection. Remarkable is that turbo-coding is build on concatenated coding and iterative decoding, two techniques that were already known for some time.

1.2 Capacity theory

To understand the need for applying a coding scheme, Shannon's theorem is discussed. The discussion in this section is based on [34].

Shannon's theorem on the capacity of an additive white gaussian noise (AWGN) channel, states that the maximum reliable transmission rate is given by:

$$\frac{C}{W} = \log_2\left(1 + \frac{P}{N_0 W}\right) = \log_2\left(1 + \frac{E_b}{N_0}\frac{R}{W}\right) \text{ bits} \tag{1.1}$$

where

$$\begin{aligned} C &= \text{channel capacity, [bits/s]} \\ W &= \text{transmission bandwidth, [Hz]} \\ P &= E_b R = \text{signal power, [W]} \\ N_0 &= \text{single-sided noise power spectral density, [W/Hz]} \\ E_b &= \text{energy per bit of the received signal, [J]} \\ R &= \text{data rate, [bits/s]} \end{aligned}$$

So the capacity C is the maximum rate at which information can be sent over a channel with arbitrarily high reliability, if the source is suitably matched to the channel [34]. So in an ideal situation the transmission rate R is equal to the channel capacity C. If we thus set $R = C$ in equation 1.1, we have for the ideal system

$$\frac{C}{W} = \log_2\left[1 + \frac{E_b}{N_0}\left(\frac{C}{W}\right)\right] \tag{1.2}$$

Solving equation 1.2 for E_b/N_0, we obtain a relation between E_b/N_0 and $C/W = R/W$

$$\frac{E_b}{N_0} = \frac{2^{C/W} - 1}{C/W} \tag{1.3}$$

This equation is plotted in figure 1.1

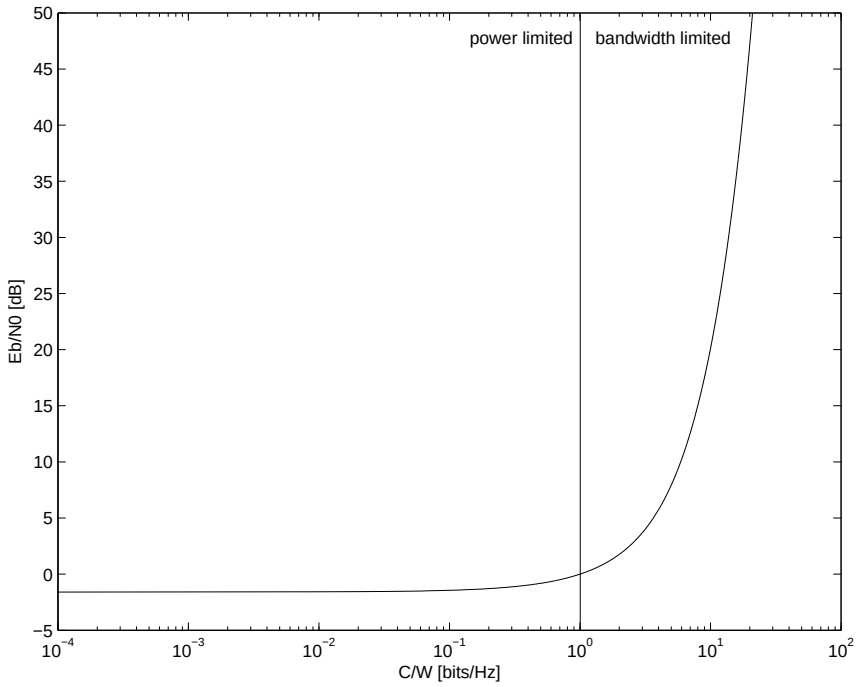


Figure 1.1: Power-bandwidth tradeoff for error-free transmission through noisy, bandwidth-limited channels.

Shannon proved that if the data rate R is smaller than the channel capacity C, it is possible to obtain error free transmission by using coding. The following equations apply,

$$R \leq W \log_2 \left(1 + \frac{P}{N_0 W}\right) \tag{1.4}$$

and

$$\frac{E_b}{N_0} \geq \frac{2^{C/W} - 1}{C/W} \tag{1.5}$$

Using these equations in figure 1.1, we see that at points below and to the right of the curve, no amount of coding will achieve total reliable communication. At points above and to the left of the curve error-free transmission is possible, but perhaps at very high costs.

In figure 1.1 the region where $R/W > 1$ is called the bandwidth-limited region and the region where $R/W < 1$ is called the power-limited region. This means that if the number of bits/s/Hz is greater than unity, the scheme is efficient in terms of utilizing bandwidth, while when bits/s/Hz is smaller than unity, the scheme is efficient in terms of power utilization. Interesting is to see that when $R/W \to 0$ (i.e., infinite bandwidth), the limiting signal to noise ratio (E_b/N_0) becomes $\ln(2)$ or about $-1.6dB$.

In the discussion above on channel capacity, arbitrarily long blocks of bits were assumed. In a practical case, however, finite block lengths are used. The Shannon curve does not give the minimum probability of error in this case, although for every finite block length scheme there must exist a minimum probability of error below which no coding scheme can perform better. In these cases normally increasing E_b/N_0 or increasing the block length, results in a lower bit error probability.

Also it must be noticed that in the regions below or to the right of the Shannon curve, communications are possible, although not without creating non-repairable bit errors. This has not to be a problem, since in most practical cases an acceptable amount of bit errors has to be achieved, instead of no bit errors at all.

1.3 Multiuser Detection

In mobile communication systems, the "ether", that serves as a medium for sending signals[1], is a scarce good. So when more than one user is present in this "medium", it has to share it with other users. Multiuser detection addresses the problem of reliable demodulation of the signal of the desired user in the presence of multiple-access interference (MAI) created by the reception of signals from many other simultaneous users. To be able to recover the signal of each single user from the signals of other users, the users must be separated from each other in some kind of way. This separation can be done in the time domain, in the frequency domain or in the code domain.

When users are separated in the time domain, each user has its own time slot in which it is allowed to transmit its data. The signal of an user can interfere with other users because of inter-symbol inference(ISI). A system that uses the time domain for separation of users is called a Time Division Multiple Access (TDMA) system.

When users are separated from each other in the frequency domain, each user has its own private part of the bandwidth on which it can continuously send its data. A user can interfere with the other users when a doppler shift into the other users' frequency band occurs. Also interference can be caused at the transmitter or receiver because of filter limitations. A system that uses the frequency domain for separation of users is called a Frequency Division Multiple Access (FDMA) system.

Finally, users can be separated from each other in the code domain. In the code domain all users have a unique codeword assigned to them. To avoid terminology problems in later chapters, we will refer to these codewords as spreadwords. These spreadwords contain a number of spread bits or chips.

[1] experiments done by Michealson and Morley have shown that the "ether" as a medium for electro-magnetic signals does not exist [15]. The term is used here to aid the explanation of multiuser detection

Spreadwords are orthogonal among each other, have a higher rate than the data bits and have a length which is called the spreading factor. A data bit that a user sends, is multiplied with its spreadword and the result is sent over the channel. At the receiver, the signal is filtered with a filter matched to the spreadword of the required user. With this system all users can send at the same time and in the same frequency band. For a more elaborate explanation of these terms, the reader should refer to [31]. A system that uses the code domain for separation of users is called a Code Division Multiple Access (CDMA) system.

The systems above can be combined to create hybrid systems. For example, CDMA can be combined with FDMA to form a FD-CDMA system. In such a system a group of users can be assigned to a frequency band, in which they use CDMA to separate themselves, while another group can do the same with the same spreading codes in a different frequency band.

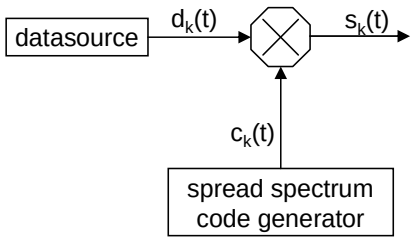


Figure 1.2: Transmitter model for a direct spread CDMA transmitter

Since CDMA is used in the remainder of this thesis, we elaborate some more on it. In section 6.2 this discussion is continued in a more thoroughly manner. A transmitter diagram for a direct sequence CDMA (DS-CDMA) transmitter is shown in figure 1.2. The spreadword occupies a much larger bandwidth than the bandwidth of the data stream, the signal power of this spreadword, however, is unity. So in a DS-CDMA system the signal power of the data is spread over a large bandwidth. At the receiver the signal of the required user is received, but because of other users in the area the signal is 'polluted' with their signal together with noise from the channel. Because of the unique spreadword of the required user, the signal can be de-spread and thus the sent data stream can be retrieved. In figure 1.3 the idealized situation in the frequency domain is shown. In figure 6.5 of section 6.2.1 the situation in the time domain is shown. In this section the DS-CDMA system is further discussed. In a practical situation the spreadwords of the interfering users won't be fully orthogonal because, so perfect de-spreading can not be obtained. The spreadwords are not full orthogonal, beacuse there only exist a limited number of spreadwords which are fully orthogonal and the number of users is usually larger than this number of

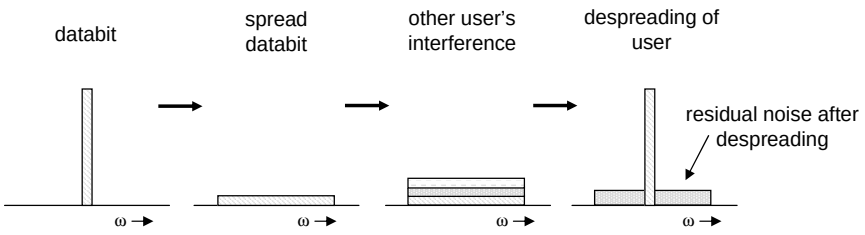


Figure 1.3: DS-CDMA behaviour in the frequency domain

spreadwords. When multi-path is present in the channel, the orthogonality of the spreadwords is even more affected. Together with noise added by the channel, makes obtaining the required user a challenging task.

1.4 Thesis description

The purpose of the research in this thesis is *to investigate the turbo principle applied to multiuser detection.* Two questions are of interest during this investigation. The first question is: 'what is the amount of performance gain that can be achieved by using turbo-multiuser-detection instead of 'classical'-multiuser-detection ('classical'=no turbo) and under what conditions and costs can these performance gains be achieved?'. The performance is evaluated with bit-error-rate (BER) charts of these two systems. The second question is: 'what is the convergence behavior of a turbo multiuser detection architecture?'. The convergence behavior is analyzed with EXIT charts. An EXIT chart is a newly proposed tool by S. ten Brink, to analyze the convergence behavior of turbo architectures. The usage of EXIT charts to analyze turbo-multiuser-detection architectures in this thesis is done for the first time, to the best knowledge of the author. The EXIT charts are used to determine how well the turbo-multiuser-detection architecture can converge to a low BER under different AWGN channel conditions. The number of users, correlation between spreadwords and used convolutional code are kept constant in this thesis, although they could be varied to analyze their influence on the convergence behavior. From the EXIT charts the number of iterations, to get the lowest BER that is possible, are determined. The discussion of EXIT charts and the results obtained in this thesis, will be used in the AWGN project [11].

In figure 1.4 on page 8 the organization of this thesis is represented in a schematic form. The chapter numbers next to the blocks in the figure, indicate in which chapter the subject is discussed. Chapters 2, 3, 4 and 6 describe on a system-level, turbo architectures. In these chapters, the main subject 'Turbo Multiuser Detection' is divided into two paths: the turbo-principle path and the multiuser-detection path. In the turbo-principle path,

first channel coding is discussed in chapter 2. In particular convolutional channel codes are investigated, since these codes are used in turbo architectures. The Viterbi decoder is presented here, since it resembles the MAP decoder, which is used in all turbo architectures. The discussion of the Viterbi decoder will aid in the understanding of convolutional-code-decoding. In chapter 3 the turbo principle is introduced with the help of turbo codes. Turbo code architectures contain decoding-algorithms, that accept and produce log-likelihood ratios (LLR), which are also used in turbo multiuser detection architectures. In this chapter two turbo code architectures, the PCCC and SCCC architectures, are introduced. These architectures are introduced to verify the correct implementation of the decoding-algorithms, since simulations of these architectures have already been performed in literature. Multiuser detection, which was briefly discussed in chapter 1, is then combined with the turbo principle in chapter 4. In this chapter a commonly used turbo multiuser detection architecture and two, for this thesis invented, turbo multiuser detection architectures are introduced and discussed. Only the commonly used architecture is simulated, since no time was available to simulate the other two architectures. In chapter 6 the decoding- and multiuser detection algorithms that are used in the turbo code and turbo multiuser detection architectures are discussed. These algorithms are the MAP decoder, the Max-Log-MAP decoder, the Log-Map decoder and the soft-interference-cancellation multiuser detector.

Analysis tools for turbo architectures are the well-known BER charts and EXIT charts. Chapter 5 describes the EXIT chart analysis-tool for turbo architectures. First E_b/N_0 regions in the BER chart of turbo code architectures are identified and given a name, based on the EXIT charts of these architectures. EXIT charts are found to be especially useful in the water-fall region of BER charts. Next transfer charts are discussed, which are needed to create EXIT charts. Simulation setups, that are used to create EXIT charts, are introduced.

In chapter 7 the system category and analysis-tools category are combined to perform simulations. BER and EXIT charts of the PCCC and SCCC architectures are compared with charts obtained in other literature to verify the correct implementation of the decoding-algorithms. Next the BER chart of the turbo multiuser detection architecture is compared with simulations in literature to verify the correct implementation of the multiuser detection algorithm.

When all these simulations are performed, the main questions of this thesis, described at the start of this section, are answered. The first question is answered by comparing the BER chart of the turbo multiuser detection architecture with the BER chart of a non-turbo multiuser detector. Areas of E_b/N_0 where the turbo multiuser detection architecture outperforms a non-turbo multiuser detector, and the conditions that should be met in these areas are identified and discussed. The second question is answered with an

EXIT chart that is made for the turbo multiuser detection algorithm. For different values of E_b/N_0 the convergence behavior and the number of useful iterations is analyzed.

Since a limited amount of time is available for the research in this thesis, some constraints on the system are assumed. The channel considered, is modelled as an AWGN channel. Multipath and fading are properties of real-world channels, but were not considered in this thesis. These constraints directly influence the multiuser detector. Only binary phase shift keying (BPSK) is used as modulation technique. Other modulation techniques like quadrature phase shift keying (QPSK) or quadrature amplitude modulation (QAM) were not considered, although the system can be expanded to include these modulation techniques. In real-world CDMA systems, synchronization techniques are used to synchronize all the users. In this thesis, perfect synchronization between all the users in the CDMA system is assumed.

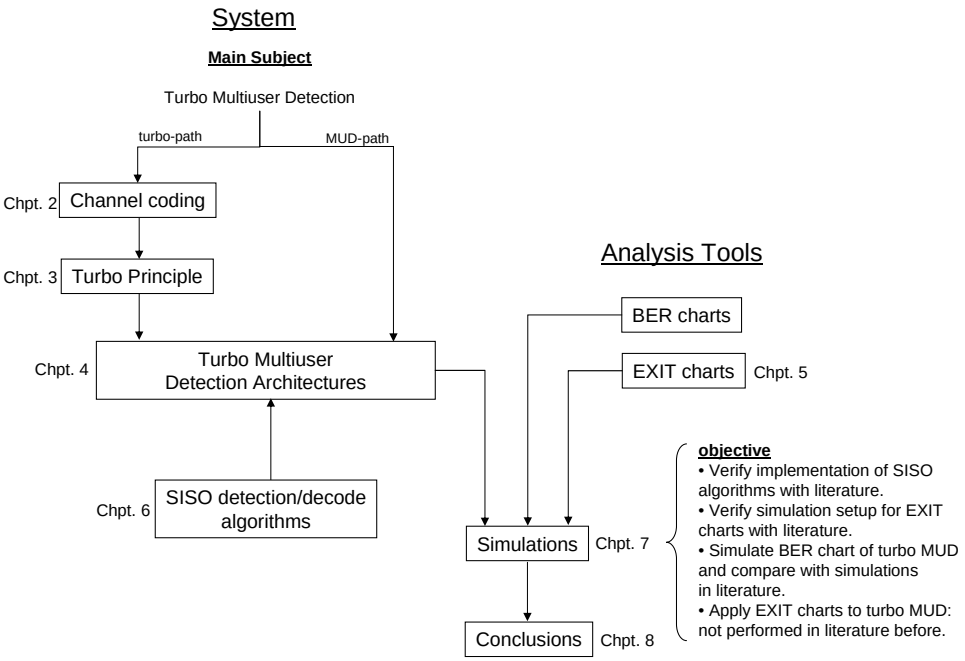


Figure 1.4: Schematically representation of the organization of this thesis

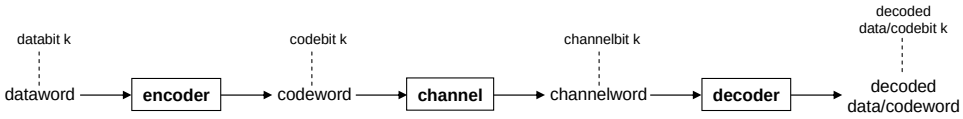


Figure 1.5: Nomenclature for coding

1.5 Nomenclature definition

In the following chapters different stadia in the encoding of a raw data stream are discussed. To be able to do this, a nomenclature for the input and output bits of different stadia are presented. In figure 1.5 the nomenclature for encoding is presented. The input for an encoder are datawords, which consists of databits. The output of an encoder are codewords, which consists of code bits. When the codebits are sent over a channel, the channel outputs channelwords, which consists of channelbits. After decoding the decoded data- or codewords are obtained, which consists of decoded data- or codebits. The variable k is used to denote the a bit.

In figure 1.6 the nomenclature for DS-CDMA spreading is introduced. An unspread sequence is presented as input to a spreader. The spreader spreads the input with a spreadword, which consists of chips. The output of the spreader is a spread sequence.

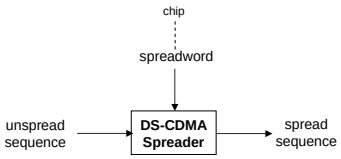


Figure 1.6: Nomenclature for spreading

2

Channel Coding

Shannon proved that an arbitrarily reliable communication over an AWGN channel can be achieved by using coding techniques. Since his theorem, many different coding schemes have been developed. Each scheme has its advantages and disadvantages, e.g. a coding scheme might achieve very low bit error probabilities but can have a high computational complexity.

The family of codes that are used to detect and correct bit errors are called *Forward Error Correcting* (*FEC*) codes. They can be split up into two categories: convolutional- and block codes. Convolutional codes are used on serial data, one or a few bits at a time, while block codes are used on long message blocks, which are typically build up out of more than 100 bytes of data [10].

The first FEC code was a single error correcting Hamming block code. Until that point convolutional error correcting was not seriously considered, until it got a huge boost when Viterbi launched his now famous maximum likelihood sequence estimation (MLSE) algorithm for decoding a convolutional code [16]. The Viterbi algorithm can be used for decoding/analyzing any process that is a markov finite state machine. Other famous block codes are BCH (Bose-Chaudhuri-Hochquenghem) codes and special subsets of them, like Reed-Solomon (RS) codes.

In turbo architectures convolutional codes are used, so in section 2.1 we will focus on convolutional codes. The Viterbi decoder will also be discussed in section 2.2, because of its resemblance with a maximum a-posteriori (MAP) decoder, which is discussed in chapter 6 and is used in turbo architectures.

2.1 Convolutional encoders

The following description of convolutional encoders is taken from different sources, like e.g. [25]. The information is combined and presented in a form suitable for this thesis.

A convolutional encoder is a markov finite state machine and takes as input databits and outputs codebits. An encoder can be represented in different ways, with a generator matrix, a generator polynomial, a logic table, a state diagram, a tree diagram or a trellis diagram. In literature a convolutional encoder is normally represented with its generator polynomial, state diagram and/or its trellis diagram. Convolutional encoders are normally implemented by a set of linear shift registers and modulo-2 adders. The code rate of an encoder is given by k/n, where k is the number of input databits and n is the number of output codebits. Figure 2.1 illustrates a commonly used shift register circuit that generates a rate 1/2 convolutional code. Other encoders can be created in a straight forward manner, as will be seen in the forthcoming text. Databits are shifted into the shift registers

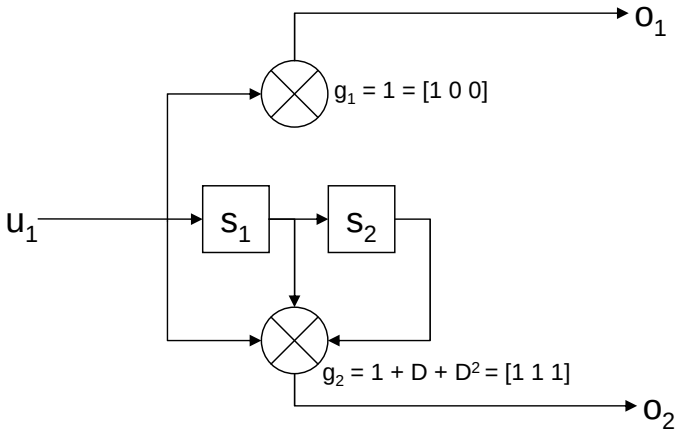


Figure 2.1: Rate 1/2 systematic convolutional encoder

via connection u_1. Every time a new bit is inserted into the shift register, the bits already in the register are shifted one position to the right. Memory elements of the shift register are connected to a modulo-2 adder, which produces the output codebits. There are two modulo-2 adders, so two output codebits are produced for one input databit, hence we have a rate 1/2 convolutional encoder. Which elements of the shift register are connected to the modulo-2 adders, is determined by the generator polynomial. The generator polynomial for output code bit o_2 is $g_2 = 1 + D + D^2$. This formula states that the shift register elements with delay zero ($D^0 = 1$), delay one ($D^1 = D$) and delay two (D^2) must be connected to the adder to obtain the output o_2. Since in the convolutional encoder of figure 2.1 codebit o_1 is directly connected to the input databit u_1, this encoder is called a *Systematic Convolutional Encoder* (SC). Convolutional Encoders with no input databits directly connected to the output code bits, are called *Non-Systematic Convolutional Encoders* (NSC). The encoder in figure 2.1 has a

Memory Constraint Length $\eta = 2$. When the number of inputs are added to the memory constraint length, the *Constraint Length* ν of the encoder is found. For the encoder in figure 2.1, $\nu = 3$. The entire generator polynomial $G(D)$ for the SC encoder in figure 2.1 is,

$$G(D) = \left[1, 1 + D + D^2\right]$$

The generator polynomial has a part before and after the comma. The '1' means that the first output should be directly connected to the input, and the second output should be connected to the shift register with connections given by $1 + D + D^2$. Because the largest delay element in the polynomial is D^2, it can be seen that the encoder has a $\eta = 2$ and since there is only one polynomial in equation , there is only one input and thus $\nu = 3$.

When a convolutional encoder has a large constraint length, the codewords it produces are more complex and thus more powerful, because a large constraint length results in codewords which have a large Hamming distance. A large code rate will also produce more complex codewords, although the amount of redundancy added with a higher code rate is also higher.

Another class of convolutional encoders are *Recursive Convolutional Encoders* (RSC). These encoders have a feedback polynomial, which connects some of the elements of the shift register through a modulo-2 adder to the input of the shift register. See figure 2.2 for an arbitrarily example of a rate 1/2 RSC encoder with a generator matrix $G(D)$,

$$G(D) = \left[1, \frac{1 + D + D^2}{1 + D^2}\right]$$

RSC encoders can also be systematic or non-systematic. An RSC encoder

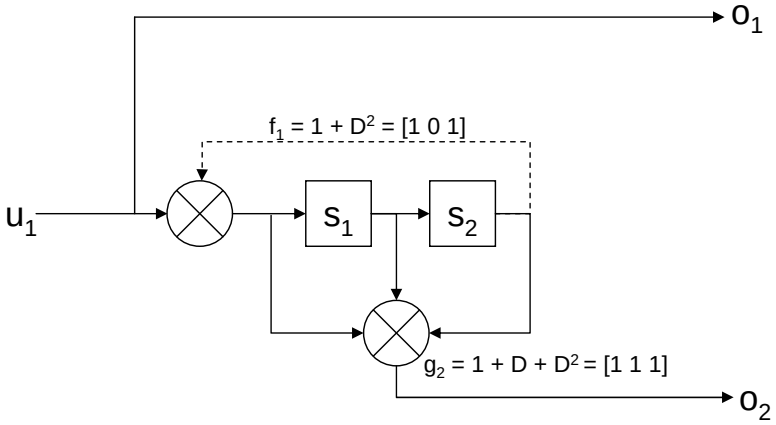


Figure 2.2: Rate 1/2 recursive systematic convolutional encoder

has a infinite impulse response (IIR), because of its feedback connections

[8]. This results in an infinite response of ones and zeros, when an input stream is given that consists of '0's with only one '1'. In this way an input dataword with a very low Hamming distance, results in a codeword with an infinite Hamming distance, if you wait infinitely long. The only way to terminate the RSC encoder is to get it back into its all-zero state. This can be achieved by sending it a suitable terminating dataword. A non-recursive encoder has a finite impulse response (FIR) and will create codewords with smaller Hamming distances [8].

The convolutional encoder can also be represented as a state diagram, which shows for every input into the shift register, to which state the shift register will go next. The state diagram of the encoder in figure 2.1 is shown in figure 2.3. A transition caused by a 0 as input is shown by the solid lines, while a transition caused by a 1 as input is show by the dashed lines. The lines are called branches. The outputs of the encoder is also shown for every transition or branch. The first bit is output o_1 and the second bit output o_2. From the state diagram it becomes clear why the codewords produced

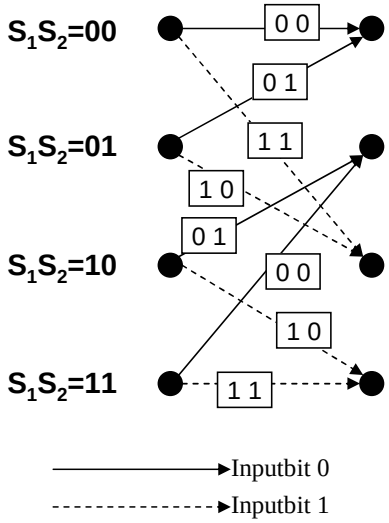


Figure 2.3: State diagram of a rate 1/2 systematic convolutional encoder

by a convolutional encoder can be used as a FEC code. The number of possible jumps from one state to another state is limited. Thus the databits in a dataword encoded with a convolutional encoder follow a certain path through the state diagram of the encoder. This path can be represented by a trellis diagram. Figure 2.4 shows the trellis for the databit stream '00110' which is encoded with the convolutional encoder of figure 2.1. The codeword output for the input dataword can be found by looking at the output codeword at each branch. For the example in figure 2.4, the output is '0000111000'. The output codebits can be sent over a channel using an

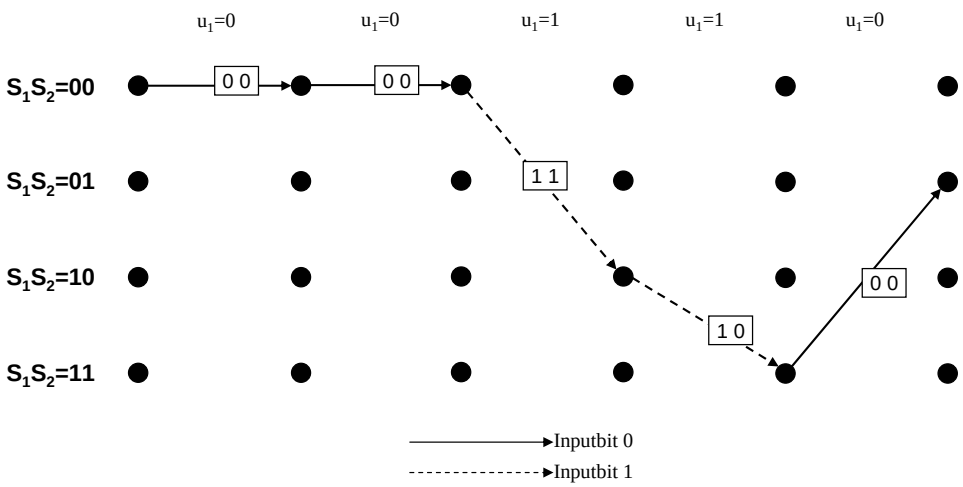


Figure 2.4: Trellis diagram of data bitstream '00110' encoded with a rate 1/2 systematic convolutional encoder

arbitrarily modulation scheme.

Once an input databit stream is encoded to a codebit output stream, the encoder ends in a certain state. It is possible to add input databits, that will terminate the encoder to a certain state, e.g. state 0. When an encoder is not terminated, is is called truncated. When an encoder is terminated to a state, the decoder needs smaller block-lengths, to decode the codebits with the same reliability as for a truncated encoder.

2.2 Viterbi decoder

When a codeword is sent over a channel, after modulation, a receiver receives the channelword and has to decode it to obtain the corresponding dataword. However, the channelword is corrupted by channel imperfections (noise, fading, inter symbol interference, etc.). The first practical decoder for decoding a channelword to a dataword, was introduced by Viterbi [16]. The Viterbi decoder is a *Maximum Likelihood Sequence Estimation* (MLSE) decoder. This means that the decoder tries to find the most likely path that the channelword has followed through the trellis diagram, like in figure 2.4. So the Viterbi decoder does not minimize the bit error rate (BER), but it minimizes the error of finding a wrong path.

The Viterbi decoder exists in two forms, namely the hard-decision and soft-decision form. The hard-decision Viterbi decoder receives hard decided channelbits from the demodulator, while the soft-decision decoder receives floating point values from, for example, a matched filter which quantizes channelbits in more than 2 possible values. The soft-decision decoder out-

performs the hard-decision decoder when it comes to minimizing the BER [14].

To understand the Viterbi decoder, the hard-decision version is first considered. We saw, that the encoder follows a specific path through the trellis in order to encode a dataword, by jumping from state to state. The Viterbi decoder uses the fact that a state transition from one state to another can only consist out of a finite set of possible jumps. Figure 2.5 shows an all-zero codeword, that is received with errors and is entered in a Viterbi decoder, which determines the most likely path. In the figure the dataword,

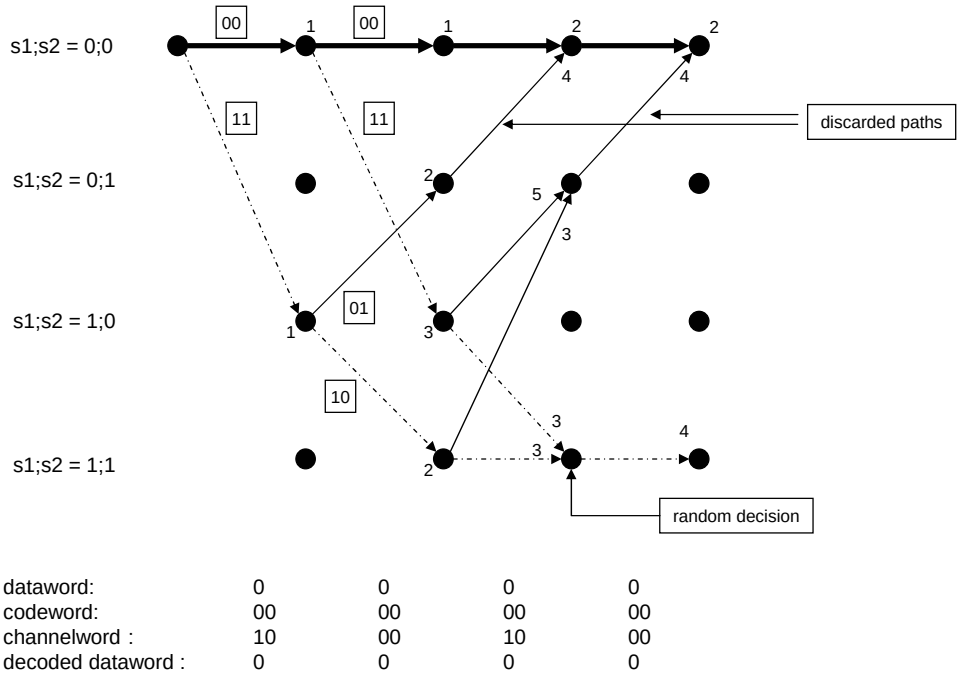


Figure 2.5: Trellis diagram of an all-zero dataword in a hard Viterbi decoder

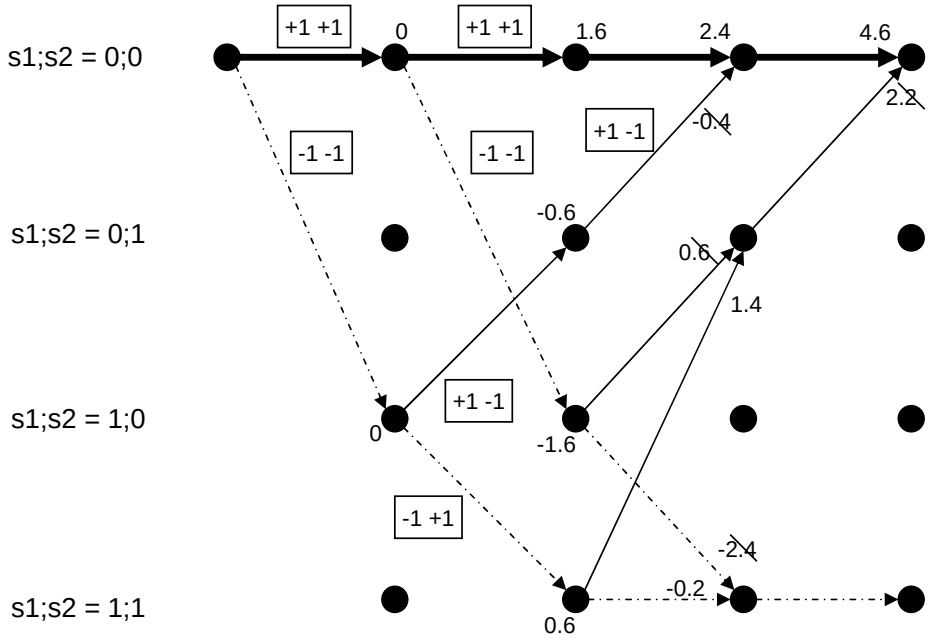
codeword, channelword and the decoded dataword are shown. We assume that the encoder started in state ($S_1S_2 = 00$). The two first codebits are '00'. Due to channel errors, the receiver received the channelbits as '10'. It is known from the state diagram, that the encoder could only have gone to state ($S_1S_2 = 00$) or state ($S_1S_2 = 10$) from its beginning state ($S_1S_2 = 00$). At this point there are thus two possible paths in the trellis that could have been followed by the encoder. A transition to state ($S_1S_2 = 00$) would have occurred when two '00' channelbits would have been received and a transition to state ($S_1S_2 = 10$) would have occurred if two '11' channelbits had been received. We calculate the difference between the received channelbits and the two possible channelbits, to get a metric for the two different paths that we are considering. The difference we calculate is called the branch-

metric. We assign the branch-metric to the state where the transition from one state leads to and do that for each possible transition. The branch-metric can be computed in different ways, where the calculation method should be selected based upon the channel. In the hard-decision example of figure 2.5, the Hamming distance is used as branch-metric. The Hamming distance between the channelbits '00' and '10' is 1. The Hamming distance between channelbits '11' and '10' is also 1. Then the next two channelbits are considered. For state ($S_1S_2 = 00$) the only possible next states are again state ($S_1S_2 = 00$) and state ($S_1S_2 = 10$). The next two channelbits we received are '00', for which we can again calculate the Hamming distances between the two possible bits '00' and '11' that could have caused the transition. The Hamming distance between '00' and '00' is 0, the Hamming distance between '00' and '11' is 2. We now add these distances to the already acquired distance 1 of state ($S_1S_2 = 00$) in the first step and put them at the next two possible states (see figure 2.5). The other state ($S_1S_2 = 10$), which got a branch-metric assigned to it, can only go to the next states ($S_1S_2 = 01$) and ($S_1S_2 = 11$). These transitions can only occur when code bits '01' or '10' were transmitted. The Hamming distances between these two possible transitions and the received channelbits is 1. This branch-metric is again added to the already acquired distance of the path and assigned to the next state. When we continue to the next transition, we see that the path from states ($S_1S_2 = 00$) to ($S_1S_2 = 00$) and the path from state 01 to 00, merge together. We now only keep the path with the lowest accumulated metric, which is the path from state ($S_1S_2 = 00$) to ($S_1S_2 = 00$). Since we want to have the path with the lowest accumulated metric in the end, we can prove that the path which had the highest metric at this point, will never get a lower metric than the other path [25]. The path that 'survives' is called the survivor path. In figure 2.5 we also see that two paths merge with the same accumulated metric. In that case the path that survives can be chosen arbitrarily. Each of these two paths would have continued the same way and so they both would have accumulated the same metric in the end. These two paths would have had the same likelihood and since the Viterbi decoder decides between paths based upon maximum likelihood, it cannot determine which of these two paths would have been the better one. On each point in time, the Viterbi decoder has a number of paths that is the same as the number of states in the state diagram.

When the Viterbi decoder has to make a decision about the data bits at some time, it examines all the paths it has tracked until then. The path with the lowest accumulated metric is chosen to be the most likely path. It then outputs the data bits which correspond with this path, by tracing the path back to its initial state.

The soft-decision Viterbi algorithm operates in the same way, only the branch-metric is calculated differently. The soft-decision algorithm receives channelbits from a matched filter. The received channelbits are not com-

pared with the codebits, but with the modulated codebits. To illustrate this, we examine an example with BPSK modulation. Figure 2.6 shows a trellis in a soft-decision Viterbi decoder. We assume that the Viterbi



dataword:	+1	+1	+1	+1
codeword:	+1+1	+1+1	+1+1	+1+1
channelword :	+0.8 -0.8	+0.5 +1.1	+0.9 -0.1	+0.7 +1.5
decoded dataword :	+1	+1	+1	+1

Figure 2.6: Trellis diagram of an all-zero dataword in a soft Viterbi decoder

encoder started in state ($S_1S_2 = 00$). From the matched filter two channelbits with values $+0.8$ and -0.8 are received. Again the only possible two next states are ($S_1S_2 = 00$) and ($S_1S_2 = 10$). The transition from state ($S_1S_2 = 00$) to state ($S_1S_2 = 00$) could only have produced the channelbits $+1$,$+1$. The first received channelbit is multiplied with the first possible channelbit and the second channelbit with the second possible channelbit and the two results are added, which gives the branch-metric. So we have $(0.8 \times +1) + (-0.8 \times +1) = 0$. This branch-metric is again assigned to the state where the branch leads to. This way of calculating branch-metrics can be seen as a bonus/penalty system. When a channelbit matches the possible channelbit in a high degree, the branch metric is assigned a high value, while when it doesn't match it might get a lower value or even the

value can be subtracted from it. So when a decision about the data bits has to be made, the path with the largest accumulated metric is chosen to be the correct path.

Until now the point where a decision on the data bits was made, was chosen arbitrarily. In section 1.2 it was concluded that the best results in terms of BER are obtained when block lengths are very high. Since this also implies very long delays, it is not practical in some situations. In literature about Viterbi decoders the time of decision on the data bits is done after 5ν or 5 times the constraint length of the encoder.

2.3 Conclusions

In this chapter convolutional codes were introduced. In section 2.1 a rate 1/2 systematic non-recursive convolutional encoder and a rate 1/2 recursive systematic encoder were introduced. The terminology for describing a convolutional encoder was introduced, like generator polynomial and the state transition diagram. In section 2.2 two Viterbi decoders were introduced, which can decode channelwords to their corresponding datawords.

In the next chapter the results of this chapter will be used to describe turbo codes and the turbo principle.

3

Turbo Coding and The Turbo Principle

In 1993 Berrou et. al. discovered a new channel code, which offered superior performance over already existing codes [6]. This new channel code was found by combining two already known techniques: concatenated coding and iterative decoding. However, their approach was different. They gave this new code the name 'Turbo Code'. Later it was found that 'the Turbo Principle' could also be used on other decoding problems.

Section 3.1 explains concatenated coding and iterative decoding and explores these techniques to obtain turbo coding and decoding. The contents of this section are mainly based on [8], but with some adaptations to fit in this thesis. Section 3.2 discusses two types of turbo encoders and explains their structure. Section 3.3 gives the decoder structures needed to decode the codewords generated by the turbo encoders of section 3.2. Section 3.4 discusses interleavers and the properties they need to have in a turbo architecture. Section 3.5 explains the turbo principle and why it can be used on many different decoding areas.

3.1 Concatenated Coding and Iterative Decoding

In section 2.1 it was concluded that the power of a FEC code to correct a series of errors in a bitstream, depends on the constraint length ν of the encoder. A larger constraint length will result in a more powerful code. Increasing the constraint length of the encoder, however, will increase the complexity of the decoder exponentially. To overcome this shortcoming, concatenation of codes can be used [8]. See figure 3.1 for the principle of concatenated coding. In concatenated coding multiple 'simple' encoders are connected to each other. The first encoder is called the outer coder and is applied first/removed last. The last encoder is called the inner coder and is applied last/removed first. The codewords created at the output are much

more complex than the codewords that would have been created by a single encoder. Decoding is done by the 'simple' decoders of the used codes, also concatenated to each other.

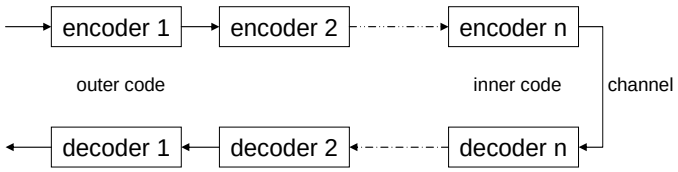


Figure 3.1: Concatenated Coding and Decoding

The most dramatical drawback of this way of coding is the phenomenon of *error propagation*. When a decoder makes a decoding error, due to an overwhelming amount of errors imposed on the channelwords, the next decoder receives the wrongly decoded channelwords and might not be able to correct these extra errors. It even might impose more errors onto the channelwords. At the end, none of the decoders might be able to correct the errors and the result is a wrongly decoded dataword.

To avoid this problem of error propagation, the channel errors should be spread over the codewords. To achieve this we can use an interleaver. An interleaver permutes the order of bits in a data stream in a known way. At the receiver a de-interleaver puts the bits back in their original order. A burst of errors imposed on the data stream by the channel, is in this way spread over the entire data stream. Interleavers are discussed in section 3.4.

The interleaver can now be connected between two encoders in figure 3.1, resulting in the system shown in figure 3.2.

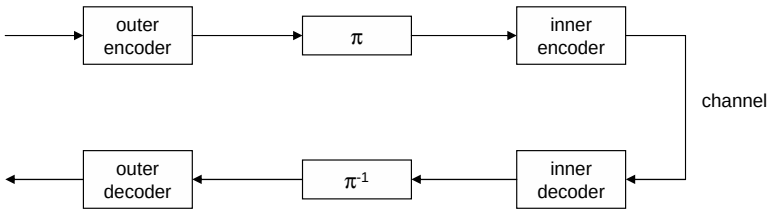


Figure 3.2: Concatenated Coding with an interleaver between the encoders

With the system in figure 3.2, turbo decoding can be explained. We take for the two encoders in figure 3.2 two block encoders. A block encoder produces a codeword of length n, of which the first k bits consist of the original dataword and the last n-k bits are parity bits. The outer encoder has data length k_1 and code length n_1. The inner encoder has data length k_2 and code length n_2. The block interleaver has a dimension of k_2 rows and n_1 columns. The code generated with this concatenated encoder can be

arranged in data and parity bits as in figure 3.3. The rows of the upper left k_2 by k_1 sub-matrix in figure 3.3 are the input to the outer encoder. The outer encoder takes k_1 input bits (one row) and generates $n_1 - k_1$ parity bits, which are added to the end of the row. Now the rows of length n_1 are the input for the interleaver. The output of the interleaver is fed to the inner encoder. This means that the columns of the upper k_2 by n_1 sub-matrix are the input to the inner encoder. The inner encoder takes k_2 input bits (the columns of the upper sub-matrix) and generates $n_2 - k_2$ parity bits, which are added below the column. Now the rows in figure 3.3 are the data and parity bits of the outer encoder, while the columns are the data and parity bits of the inner encoder.

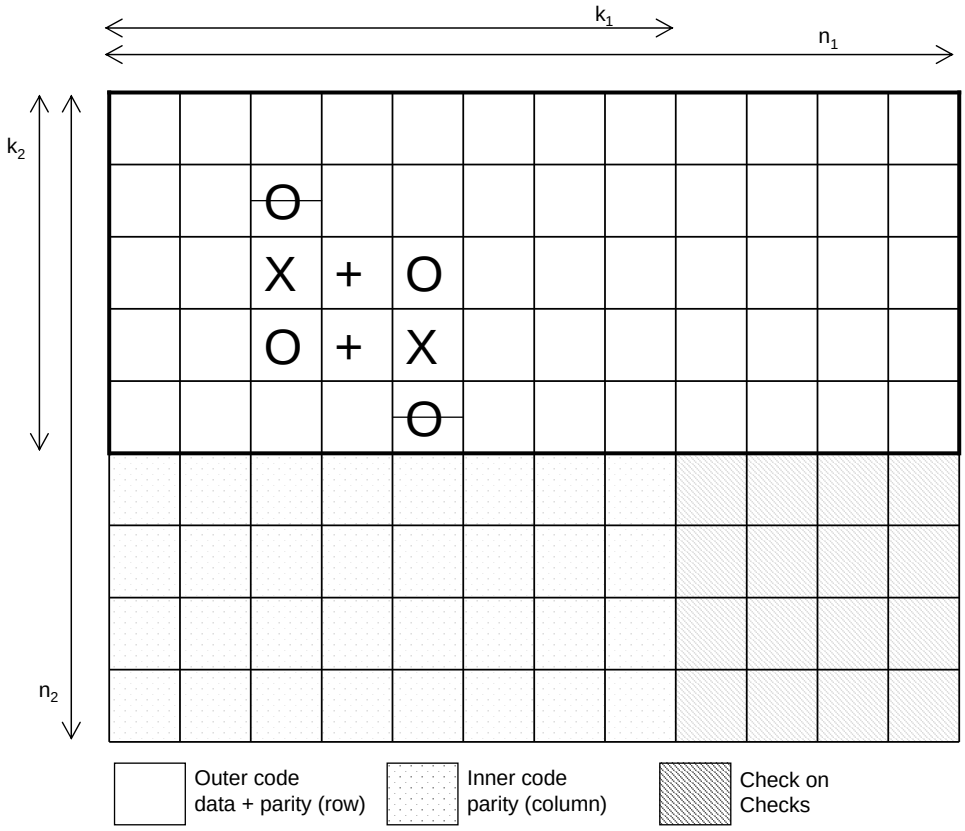


Figure 3.3: Arrangement of data and parity bits of the code in figure 3.2. O are received errors, X are errors created by the inner decoder (n_1,k_1), + are errors created by the outer decoder (n_2,k_2).

After the inner encoder has finished, the resulting codebit stream is sent over the channel by reading the matrix in figure 3.3 column by column. In figure 3.3 the 'O's show a pattern of errors in the received channelbits. The

channelbits are presented first to the *inner decoder* (see figure 3.2). The output of the inner decoder is de-interleaved and presented to the *outer decoder*. We assume that both decoders can only correct one error. First the inner encoder starts decoding the columns of the matrix in figure 3.3. The first two columns don't have any errors in them, so no problems there. However, the third column has two errors. Because the inner decoder can only correct *one* error, it can not correct the error in this column. It will correct only one error and introduce a new error denoted by the 'X'. When it reaches the fifth column, the same problem occurs. When the inner decoder is finished, the bit stream is de-interleaved and presented to the outer decoder. Because of the de-interleaving, the outer decoder starts decoding the rows of the matrix in figure 3.3. The first two rows give no problems, since no errors are present here (anymore). The third row, however, has got one channel error and one error caused by the inner decoder. Since the outer decoder also can correct only one error, it will not be able to correct all the errors and may even cause an extra error, denoted by the '+' in figure 3.3. The same problem occurs with the fourth row.

We have seen that this way of decoding, where the bit stream only passes each decoder once, results in decoded databits which still have errors in it. We also notice that, if we first would have decoded the stream with the outer decoder, there would have been a correct decoding, since the rows do not contain more than one error. The inner decoder can then also decode the bitstream without any extra errors, since there are no more than one error in the columns left. The bitstream can then be passed to the outer encoder, to correct more errors, and so on, and so on.

Here we have encountered the principle of turbo decoding. By letting the same channelbits iterate over the two decoders until no errors are left anymore, a better decoding can be achieved. However in the given example, there still is error propagation. Therefor we need one extra ingredient to make turbo decoding work.

In the example, hard-decision decoding was used. If we would have used soft-decision decoding, the decoding performance would have increased dramatically. Soft decision was already introduced in section 2.2. We re-interpret 'soft information' as the reliability of the hard bit. If one of the decoders outputs a low reliability for a decoded data or codebit, the other decoder might decide otherwise on this data or codebit, to achieve a better decoding result.

When we use soft information for the decoding of the bit, we need a decoder that uses and creates soft information. The soft Viterbi decoder in section 2.2, used soft information to decode the channelwords, but did not create soft information about the codebits or databits. A decoder that does this, is called a Soft In-Soft Out (SISO) decoder. For now we are going to assume we have decoder that is capable of that. Such a decoder is described in chapter 6.

3.2 Turbo Encoders

The contents of this section are mostly derived from [4]. In the previous section we used a serial concatenated encoder. In the original paper about Turbo-codes of Berrou [6], a parallel concatenated encoder architecture was used to create the turbo codes. In figure 3.4 a parallel turbo encoder is shown and in figure 3.5 a serial turbo encoder is shown.

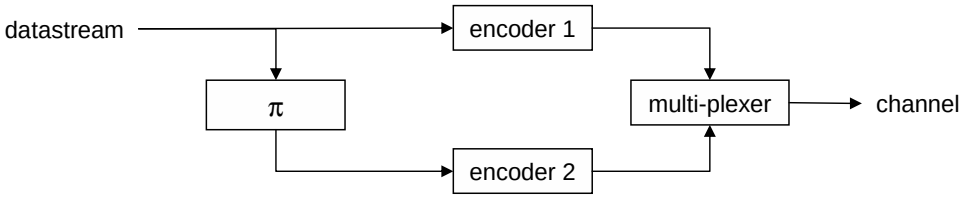


Figure 3.4: Generic parallel turbo encoder

Parallel codes were already exhaustively researched before Berrou published his papers. The main performance increase achieved by Berrou's encoder was due to the used interleaver.

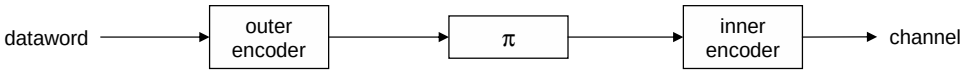


Figure 3.5: Generic serial turbo encoder

The encoders in figure 3.4 and 3.5 can be any type of encoder. The encoders most commonly used are recursive systematic encoders (RSC), because they combine the qualities of a systematic encoder together with the superior performance of a non-systematic encoder (NSC)[8].

Figure 3.6 gives the parallel turbo encoder, when two systematic encoders are used. The systematic or data bits and only the parity bits of the encoders are passed to the MUX, because the two encoder would both produce the systematic bit. It is possible to puncture the parity bits of the two encoder to create a rate 1/2 encoder. When puncturing is used not all the parity bits of every encoder is send, but only the half of the parity bits. So with the first systematic bit, only the first parity bit of the first encoder is sent and with the second systematic bit only the second parity bit of the second encoder is sent. When with every systematic bits, two parity bits are sent, the resulting encoder has a rate 1/3.

The performance of a turbo encoder depends on the feedback and parity polynomials used for the RSC encoder and the type of interleaver that is used [6], [3], [5], [26].

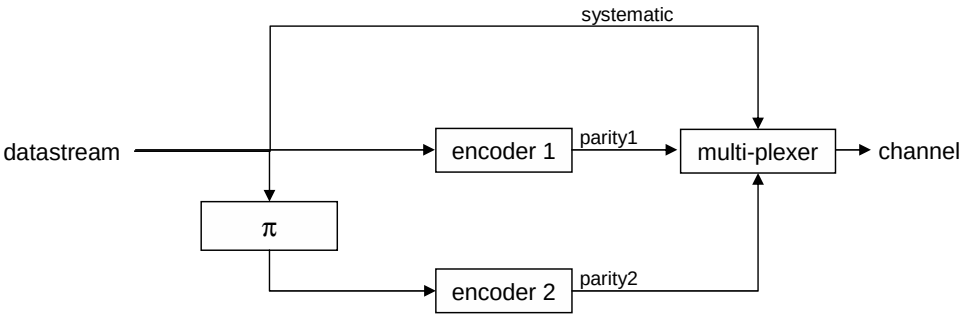


Figure 3.6: Systematic parallel turbo encoder

In section 2.1 we saw that a RSC encoder can produce codewords with a large Hamming distance, but there is also a possible input sequence which lets the encoder return to its all-zero state. This results in a codeword with a small Hamming distance. The interleaver has to permute the order of the input data in such a way, that if *one* encoder gets an input that results in a codeword with a small Hamming distance, the other will get the permuted sequence which results in a codeword with a large Hamming distance. Using a properly designed interleaver will result in at least one of the encoders outputting a codeword with a large Hamming distance. However, there is a probability that a parallel concatenated code will output a codeword with a small Hamming distance. This results in an error floor of a parallel concatenated code at high E_b/N_0. A serial code is less sensitive to codewords with small Hamming distances and thus does not show this error floor [4]. However, for certain values of E_b/N_0 the PCCC architectures outperforms the SCCC architecture as can be seen in chapter 7.

An optimal set of generator polynomials have been investigated in [5].

3.3 Turbo Decoders

For the serial turbo encoder and parallel turbo encoder, different decoders are needed. In figure 3.7 a parallel turbo decoder is shown and in figure 3.8 a serial turbo decoder is shown. These decoders are derived from [4]. The SISO decoders take as input Log-Likelihood Ratios (LLR) of the databits and codebits, which provide soft-information about the bits. The input LLR's are normally referred to as *a-priori information*, while the output LLR's are normally referred to as *a-posteriori information*. However, in figure 3.7 the decoder outputs extrinsic information. The difference between extrinsic and a-posteriori information will be discussed in the remainder of this section. For both the codebits and databits output LLR's are created.

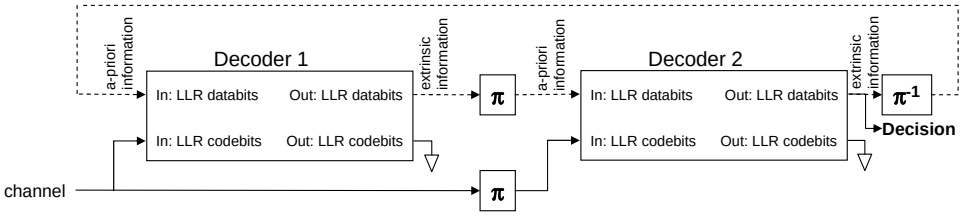


Figure 3.7: Decoder for the parallel concatenated code in figure 3.4

The LLR (Λ) of a databit $d_k \epsilon \{0, 1\}$ given a channelbit y_k is defined as

$$\Lambda(d_k) \triangleq \ln \frac{P(d_k = 0|y_k)}{P(d_k = 1|y_k)} \tag{3.1}$$

The LLR is the conditional probability of the bit being a '0' divided by the conditional probability that it is a '1'.

In the parallel turbo decoder, decoder 1 decodes the stream from encoder 1 and decoder 2 decodes the stream from encoder 2. In a parallel turbo decoder architecture the order in which the decoders operate on the received elements is arbitrarily. However, care has to be taken on interleaving or de-interleaving the produced LLR's before they are delivered to the next decoder. Both decoders in a parallel turbo decoder, take as codebit LLR input the channelbits for every iteration. The output LLR of the databits are properly (de-)interleaved and provided as databit LLR input to the next decoder. The codebit LLR output is not used. When an LLR input is not connected to an LLR output or no LLR output is yet available, the input LLR is set to zero. After some iterations a decision is made about the databits. The decision is performed on a single bit d_k and is decided to have been a '0' when $\Lambda(d_k) > 0$ and decided to have been a '1' when $\Lambda(d_k) < 0$. In the serial turbo decoder the inner decoder decodes the stream

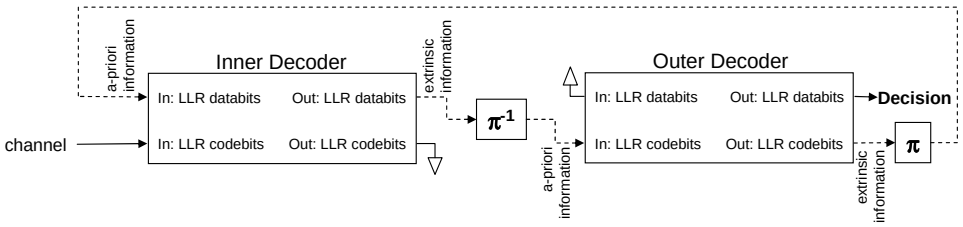


Figure 3.8: Decoder for the serial concatenated code in figure 3.5

of the inner encoder and the outer decoder decodes the stream of the outer encoder. The order in which these decoders are placed is not arbitrarily in a serial turbo decoder: the inner decoder has to decode the stream first,

followed by the outer decoder, because the output of the inner encoder in figure 3.5 is sent over the channel. The inner decoder takes as codebit LLR input the channelbits and as databit LLR input the codebit LLR output of the previous decoder. When the last is not yet available, the input is set to zero. The codebit LLR output of the inner decoder is not used. Its databit LLR output is properly de-interleaved and given as codebit LLR input to the outer decoder, because the codebits from the outer encoder are used as databits for the inner encoder. After some iterations the databit LLR output of the outer decoder is used to make a decision about the data bits. This decision is done in the same way as with the parallel decoder.

From now on, we are going to refer to the parallel turbo code encoder and decoder as a Parallel Concatenated Convolutional Code (PCCC) architecture and to the serial turbo code encoder and decoder as a Serial Concatenated Convolutional Code (SCCC) architecture.

A-posteriori information of a SISO decoder can be build up out of two components, namely the *intrinsic* information and the *extrinsic* information. This can be seen by applying the Bayes' rule to equation 3.1,

$$\begin{aligned}
\Lambda(d_k) &\triangleq \ln \frac{P(d_k = 0|y_k)}{P(d_k = 1|y_k)} \\
&= \ln \frac{P(d_k = 0 \bigcap y_k)/P(y_k)}{P(d_k = 1 \bigcap y_k)/P(y_k)} \\
&= \ln \frac{P(y_k|d_k = 0) \cdot P(d_k = 0)/P(y_k)}{P(y_k|d_k = 1) \cdot P(d_k = 1)/P(y_k)} \\
&= \underbrace{\ln \frac{P(y_k|d_k = 0)}{P(y_k|d_k = 1)}}_{intrinsic} + \underbrace{\ln \frac{P(d_k = 0)}{P(d_k = 1)}}_{extrinsic}
\end{aligned} \tag{3.2}$$

The intrinsic information is the information that is delivered to the decoder by the channel. The extrinsic information is the information obtained by the decoding of the code, so the information supplied by the known trellis of the encoder, and is not conditioned on the channel. The extrinsic information is the information that should be passed to the next decoder, since this information is not available to the next decoder, while the intrinsic information is already available at the next decoder. When the decoders are delivered new and unknown information about the data- and codebits, it can make a better decision about the code or data bits. However, after a number of iterations the extrinsic information of the two decoders will become more and more correlated, resulting in less performance increase [26], [8]. In figures 3.7 and 3.8 it is assumed that the LLR outputs of the decoders are only build up out of extrinsic information. Extrinsic and intrinsic information will be more extensively discussed in chapter 6. In chapter 7, transfer charts, discussed in chapter 5, will be used to analyze extrinsic and a-posteriori information.

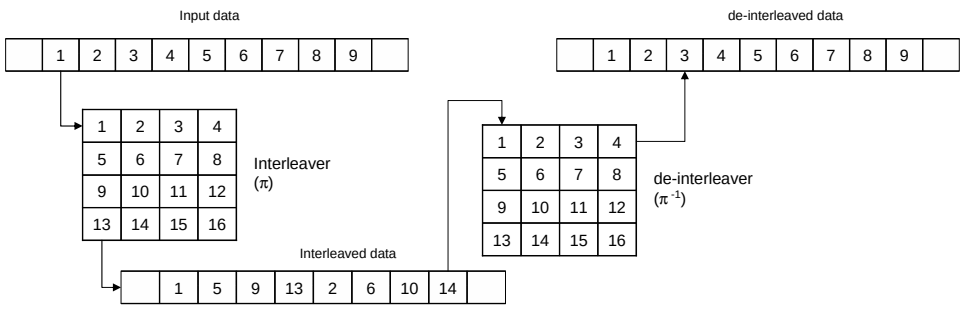


Figure 3.9: Block interleaver and de-interleaver operation

Setting LLR values to zero corresponds to $P(d_k = 1) = P(d_k = 0) = 0.5$.

3.4 Interleaving

In section 3.1 a block interleaver was already discussed. In this section, we elaborate a little more on interleavers.

The two main criteria in the design of an interleaver are 1) the minimum distance spectrum of the output and 2) the correlation between its input and output sequence. The interleaver has to create a large minimum distance spectrum and the correlations should be as low as possible. The second criteria is some times referred to as the *iterative decoding suitability* (IDS). When the input and output stream of the interleaver are less correlated the performance of the iterative decoder improves in terms of BER. The choice of a good interleaver is especially important when small blocks are used [24].

An interleaver is a permutation $i \mapsto \pi(i)$ that changes the order of a data stream. Each interleaver has its corresponding de-interleaver (π^{-1}), that is able to restore the interleaved stream to its original order [24].

The two types of interleavers that are considered are *block interleavers* and *pseudo-random interleavers*.

A block interleaver is a matrix with a number of rows and columns. The data stream that has to be permuted, is written in this matrix row by row. The permutation is obtained by reading the data out of the matrix column by column. A block interleaver with more rows and columns is better able to fulfill the two criteria, than one with less rows and columns. Figure 3.9 illustrates a block interleaver.

A pseudo-random interleaver permutes the data by generating randomly a permutation matrix, in which the new order of the bits in the input stream is stored. When a longer input stream is used, the interleaver will be able to fulfill the two criteria even better.

Since pseudo-random interleavers fulfill the two criteria better than a block interleaver, therefore only pseudo-random interleavers are considered

in the remainder of this paper.

3.5 The Turbo Principle

From the discussion in section 3.1, it is clear that in fact the 'turbo' in turbo-codes, does not apply to the code itself, but to the iterative way of decoding. The title *Turbo* is taken from the principle of the turbo engine. Inside a turbo engine, exhaust gasses are used to blow more air into the engine. Together with more fuel this gives more power to the engine. Since in turbo decoding, the output of the decoding process is also re-used to give more 'power' to the system in total, they gave it the name *turbo-codes*.

In the turbo engine, however the output of the engine is given as an input back to it. In turbo coding the input of a decoder should not be its own output, since it already knows its own output and so there is no gain to achieve. The turbo concept can thus be used when there are two or more sources with information about the databit that should be decoded. The information contained in these two information sources, should be as uncorrelated as possible, to obtain the largest improvement in the decoding process. The obtained information from each of these sources is then iterated between the decoders of the information sources, until the correlation between the information is at its maximum. This can be determined real-time or the decoding process can be set at a fixed number of iterations.

In turbo coding the two independent information sources are the coded original data-stream and the coded interleaved data-stream. The interleaver has to guarantee the orthogonality between these two information sources. Orthogonality can also be achieved by using two different encoders, which produce different codes.

So the turbo principle can be used anywhere where a decision about a system is needed and there exist two different uncorrelated sources of information about that system. When this is the case, a large performance increase can be achieved. The drawback is that iterating more than once over a set of decoders, requires more computational power.

3.6 Conclusions

In this chapter the turbo principle was introduced. Concatenated coding and iterative decoding were discussed in section 3.1. From these two techniques turbo codes were derived. Two turbo code architectures were introduced: the parallel concatenated convolutional code (PCCC) architecture and the serial concatenated convolutional code (SCCC) architecture. For these architectures, the encoders and decoders were discussed. Interleavers used in a turbo architecture were shortly discussed in section 3.4. In chapter 7 simulations are performed on an SCCC and a PCCC architecture to check the

correct implementation of the decoding-algorithms. The simulation setup will be the same as in [4] to make comparisons.

In section 3.5 the turbo principle was finally derived. The turbo principle is applicable in a decoding situation, where two or more information sources are available about the decoded information source, e.g. in parallel turbo coding these two information resources are the original coded datawords and the interleaved coded datawords.

4

Turbo Multiuser Detection

In chapter 3 the turbo principle was discussed and it was found that it could also be used for other decoding problems. In turbo multiuser detection (turbo-MUD), concatenated architectures of a multiuser detector and a convolutional code are commonly used [1][22][17][18][32][20]. By iterating the soft output of the convolutional code decoder back to the multiuser detector, a turbo multiuser detector architecture is created. In section 4.1 a standard turbo-MUD architecture is presented. In section 4.2 other turbo-MUD architectures are presented.

4.1 Serial Concatenated Multiuser Detector and Convolutional Code

The most commonly used turbo multiuser detector is a soft-input/soft-output multiuser-detector concatenated serially to a convolutional-code decoder [1][22][17][18][32][20].

This architecture is shown in figure 4.1 for K users. Datawords of every user are encoded with the same convolution encoder. The output of the convolutional encoder is interleaved with a pseudo-random interleaver which is different for each user. The output of the interleaver is spread with a unique spreadword. The channel is modelled as FIR filter, to be able to expand it to a multipath channel, with additive white gaussian noise. The spread sequence is sent over this channel with filter coefficients g_u where $u = 1..K$. The signal at the receiver is a superposition of the spread sequences of all users and gaussian distributed white noise.

At the receiver the soft-in/soft-out multiuser detector receives the spread channelwords. Together with the a-priori log-likelihood ratio input of the code bits, the multiuser detector outputs log-likelihood values for the codebits of every user present in the system. During the first iteration, when no a-priori information is yet present, the a-priori input is set to zero. The log-likelihood output values are de-interleaved and passed to the convolutional

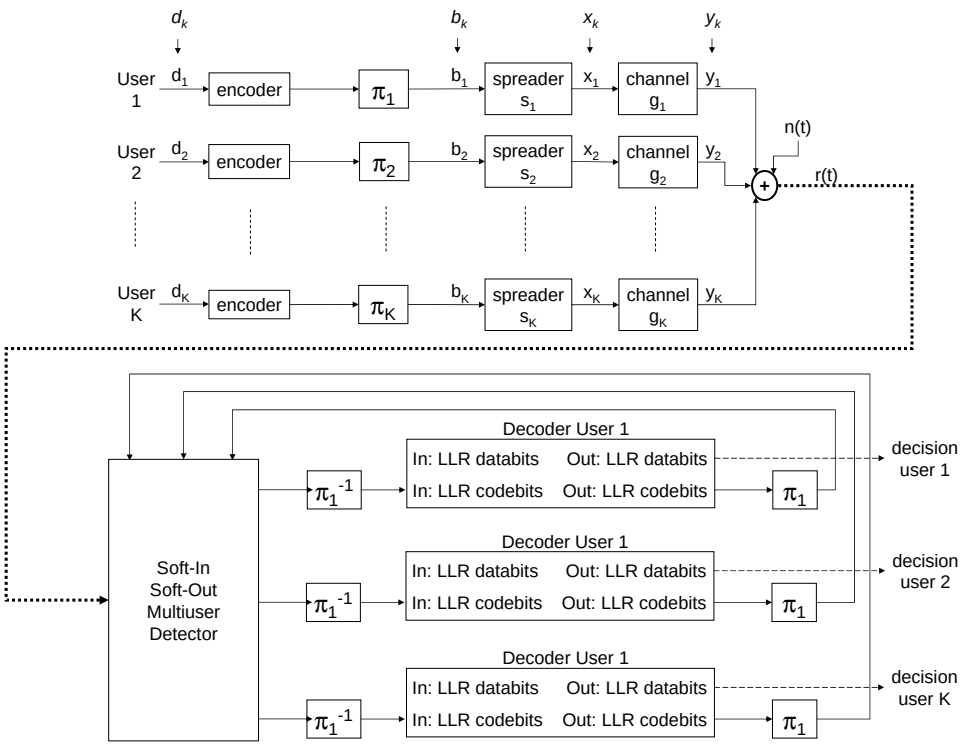


Figure 4.1: Serial concatenated Multiuser Detector and Convolutional Code for K users

code decoder as code bit log-likelihoods. The log-likelihood output of the decoder's code bits are interleaved and passed to the multiuser detector to improve its detection during the next iteration. The log-likelihood output of the decoder's databits can be used to estimate the transmitted data bits of every user.

The presented turbo multiuser detection architecture is principally the same as the SCCC architecture presented in section 3.2 and section 3.3. In the turbo multiuser detection architecture the inner code is replaced by the channel and the inner decoder is the multiuser detector.

X. Wang and V.Poor in [20] don't use the a-posteriori values of the log-likelihoods to pass from block to block (as is done in figure 4.1), but the extrinsic values (see figure 1 in [20]). The soft-in soft-out multiuser detector that is used in this thesis (section 6.2.1) is a soft-interference cancellation detector, which needs the best possible estimates of the codebits, which are the a-posteriori values. In chapter 7 simulations will be performed that prove that using a-posteriori information gives better results than extrinsic information.

4.2 Hybrid Turbo Multiuser Detection

The iterative structure of a turbo architecture makes it possible to create hybrid systems, which contain PCCC and SCCC architectures. The only consideration that has to be taken is that each decoder gets the right input. Only two hybrid systems are presented here, although many more could be invented. The EXIT charts in section 5.3 make it possible to analyze these architectures. In chapter 7 simulations will be performed on these settings.

Hybrid System 1: Multiuser Detector with two serial concatenated codes

The system in figure 4.2 is created by adding one extra convolutional encoder to the system in figure 4.1. For simplicity only one user is assumed. A similar architecture, from which the architecture in figure 4.2 is derived,

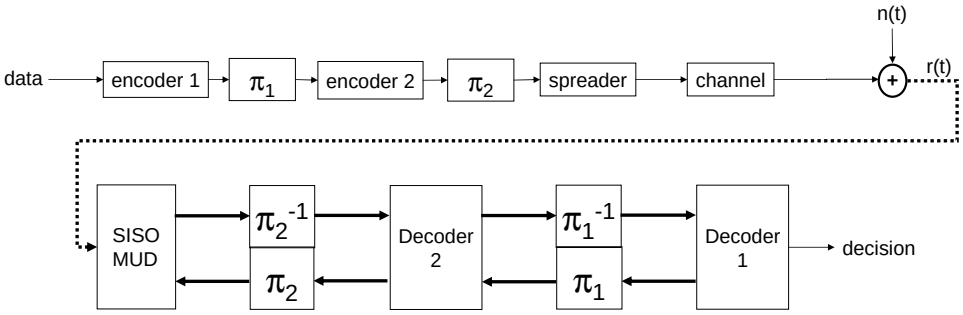


Figure 4.2: Serial concatenated Multiuser Detector and Two Convolutional Codes [30]

is investigated by M. Tüchler in [30]. At the receiver soft information can be iterated between the SISO Mud and Decoder 2 and between Decoder 2 and Decoder 1. Care should be taken to feed the correct information to the decoder. The code bits from the SISO MUD go to the code bits input of Decoder 2, while the data bits output of Decoder 2 go to the code bits input of Decoder 1.

Hybrid System 2: Multiuser Detector with parallel concatenated code

The system in figure 4.3 is created by connecting the SISO multiuser detector together with a parallel concatenated code. This hybrid system is invented by the author of this thesis. For simplicity only one user is assumed. This architecture is more flexible than the multiuser detector with two serial concatenated codes. At the receiver soft information can be iterated between all the processes. The only constraint is that the multiuser detector needs to run first. When iterating information care needs to be taken that every

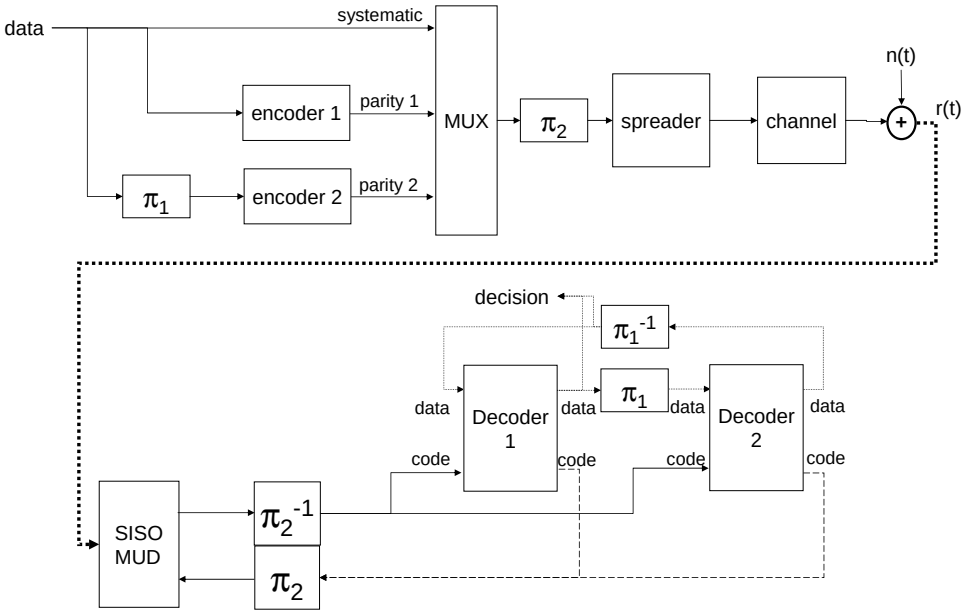


Figure 4.3: Multiuser Detector and Two Parallel Concatenated Convolutional Codes

block gets the correct interleaved version of the bits it needs. The decision on the databits can be made from the LLR databit output of decoder 1 or decoder 2. The a-priori input for the SISO MUD can come from the LLR codebit output of decoder 1 or decoder 2.

4.3 Conclusions

In this chapter three turbo multiuser detection architectures were given. The architecture in section 4.1 is the most common used in literature. The two architectures in section 4.2 were invented for this thesis, to illustrate the hybrid nature of the turbo principle. Because of time constraints the hybrid architectures are not further considered in this thesis. The turbo multiuser detection architecture of section 4.1 is simulated in chapter 7. A BER chart of this architecture is created and compared to the BER chart obtained in literature. This comparison is made to verify the implementation of the SISO multiuser detector. Next the BER chart is used to compare the turbo multiuser detection architecture to a non-turbo multiuser detector, to answer the first question of this thesis given in chapter 1. An EXIT chart is made for the turbo multiuser detection architecture to answer the second question of this thesis, also given in chapter 1. An EXIT chart is an analysis tool for turbo architectures and is discussed in chapter 5.

5

Convergence Behavior and EXIT charts

Iterative or turbo architectures are notoriously complex and hard to analyze with conventional BER charts. A recently proposed tool by S. ten Brink in [28] to analyze the convergence behavior of a turbo architecture is an extrinsic information transfer (EXIT) chart. In an EXIT chart the decoding blocks are treated as 'black boxes' of which the input/output transfer function of the extrinsic information is known. With a transfer chart the interplay between the blocks can be analyzed. This has proven to be very useful in regions of low SNR, since in these regions the convergence behavior of a turbo architecture is hard to analyze with a 'traditional' BER chart. An EXIT chart is build up out of two transfer charts. In section 5.1 the convergence behavior of turbo architectures is discussed. In section 5.2 transfer charts are introduced, which are needed to create an EXIT chart. In section 5.3 the EXIT charts are discussed.

5.1 Convergence Behavior

In turbo architectures, the iterating of log-likelihood ratios, makes it very hard to analyze the behavior of the architecture. When the bit error rate at a certain signal to noise ratio of such a architecture is wanted, simulations have to be performed, which take a lot of time and computational power. Also it is hard to see if the decoder blocks in a turbo architecture are matched to each other, or that they perhaps are a very bad combination. In figure 5.1 a sketch is drawn of the BER as a function of E_b/N_0 for an SCCC and a PCCC architecture. These results can be found in different papers on turbo coding, like [4] and [27]. For a PCCC architecture three regions can be defined.
I.) *Pinch-off Region*: In this region the BER remains constant and iterating does not improve the BER significantly.

II.) *Waterfall Region*: In this region the BER decreases dramatically when the signal to noise ratio is increased.
III.) *Wide-open Region*: The BER is at a very low value and more iterations will not improve the BER.
An SCCC architecture does not have the error floor that a PCCC architecture has. The chosen nomenclature for these regions can be understood when in section 5.3 EXIT charts are discussed.

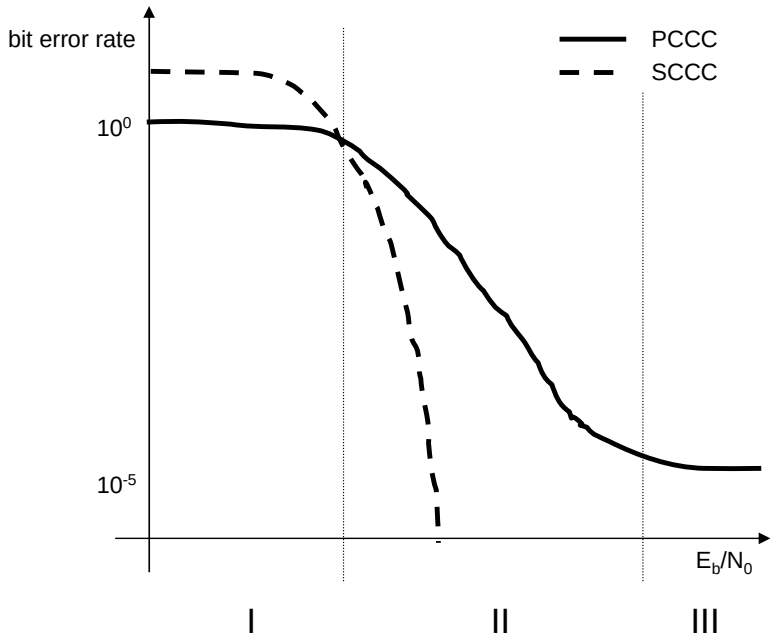


Figure 5.1: Bit error rate for SCCC and PCCC codes

5.2 Transfer Charts

An EXIT chart is created with two transfer charts from decoder blocks. A transfer chart, relates the a-priori to the a-posteriori information contents of a decoder block. This section discusses the creation of such a transfer chart. Its contents are taken from [29] and adapted to fit in this thesis.

The mutual information contents of the a-priori and a-posteriori knowledge is described by the mutual information $I(a;b)$ [13]. Consider a noisy channel over which a symbol a_i from an alphabet A is sent and a symbol b_j from an alphabet B is received. The probability that a_i was sent is $P(a_i)$ (a-priori) and the probability that an a_i is decoded at the decoder given the received b_j is $P(a_i|b_j)$. The mutual information $I(a_i;b_j)$ is defined as the difference between the uncertainty of the symbol a_i before and after transmission [13].

Consider an AWGN channel over which discrete-time signals are received,

$$\xi = x + n; \tag{5.1}$$

the conditional probability density functions then can be written as,

$$p(\xi|X = x) = \frac{1}{\sqrt{2\pi}\sigma_n} e^{-((\xi-x)^2)/2\sigma_n^2)} \tag{5.2}$$

where the binary random values X denotes the transmitted bits $x\epsilon\{-1, +1\}$. The corresponding LLR values Λ are calculated as,

$$\Lambda = \ln \frac{p(\xi|X = +1)}{p(\xi|X = -1)} \tag{5.3}$$

which can be simplified with equation 5.2 to,

$$\Lambda = \frac{2}{\sigma_n^2} \cdot \xi = \frac{2}{\sigma_n^2} \cdot (x + n) \tag{5.4}$$

The noise variable n is gaussian distributed with mean zero and variance $\sigma_n^2 = N_0/2$. We can formulate equation 5.4 also as,

$$\Lambda = \mu_\Lambda \cdot x + n_\Lambda \tag{5.5}$$

The following expression can be found for μ_Λ,

$$\begin{aligned} \mu_\Lambda &= E[\Lambda] \\ &= \sum_i E[\Lambda|x_i] \cdot p_i^x \\ &= \sum_i \frac{2}{\sigma_n^2} \cdot p_i^x \\ &= \frac{2}{\sigma_n^2} \end{aligned} \tag{5.6}$$

where $p_i^x = 0.5$ is the probability of x. n_Λ is gaussian distributed with mean zero and variance,

$$\begin{aligned} \sigma_\Lambda^2 &= E\{\Lambda^2\} - E\{\Lambda\}^2 \\ &= E\left\{\left(\frac{2}{\sigma_n^2}x + \frac{2}{\sigma_n^2}n\right)^2\right\} - \left(\frac{2}{\sigma_n^2}\right)^2 \\ &= E\left\{\left(\frac{2}{\sigma_n^2}\right)^2 x^2 + 2\left(\frac{2}{\sigma_n^2}\right)^2 xn + \left(\frac{2}{\sigma_n^2}\right)^2 n^2\right\} - \left(\frac{2}{\sigma_n^2}\right)^2 \\ &= \left(\frac{2}{\sigma_n^2}\right)^2 + \left(\frac{2}{\sigma_n^2}\right)^2 \sigma_n^2 - \left(\frac{2}{\sigma_n^2}\right)^2 \\ &= \frac{4}{\sigma_n^2} \end{aligned} \tag{5.7}$$

where we made us of the known properties: $x^2 = 1$, $E\{n\} = 0$ and $E\{n^2\} = \sigma_n^2$. We see that the mean and variance of Λ are connected by,

$$\mu_\Lambda = \frac{\sigma_\Lambda^2}{2} \tag{5.8}$$

So the mean of the log-likelihood ratio is the half of its variance. This relation will prove to be useful for modelling the a-priori information contents later in this section.

In a turbo architecture the mutual information between a log-likelihood value and a sent symbol is wanted. Assuming that a_i is a binary send symbol and $a_i \epsilon \{-1, +1\}$ and Λ_i is the log-likelihood value of this symbol at the receiver, the mutual information $I(a_i; \Lambda)$ is [13],

$$\begin{aligned} I(a_i; \Lambda_i) &= \frac{1}{2} \sum_{a_i = \{-1,+1\}} \int_{-\infty}^{+\infty} P(\Lambda_i | a_i) \\ &\quad \log_2 \left[\frac{2 \cdot P(\Lambda_i | a_i)}{P(\Lambda_i | a = -1) + P(\Lambda_i | a = +1)} \right] d\xi \end{aligned} \tag{5.9}$$

and

$$0 \leq I(a_i; \Lambda_i) \leq 1 \tag{5.10}$$

The probability density functions (pdf) $P(\Lambda_i | a_i)$, $P(\Lambda_i | a = -1)$ and $P(\Lambda_i | a = +1)$ in equation 5.9 can be evaluated numerically by Monte Carlo simulations. However, since we are using BPSK modulation over an AWGN channel in this paper, we can simplify the equation, because the pdf's have the following two properties [30], they are,

$$\textit{symmetric: } p(\Lambda_i | a_i = +1) = p(-\Lambda_i | a_i = -1) \tag{5.11}$$

and

$$\textit{consistent: } p(\Lambda_i | a_i = +1) = p(-\Lambda_i | a_i = +1) \cdot e^{\Lambda_i} \tag{5.12}$$

With these properties equation 5.9 simplifies to,

$$I(a_i; \Lambda_i) = 1 - \int_{-\infty}^{+\infty} p(\Lambda_i | a_i = +1) \cdot \log_2(1 + e^{-\Lambda_i}) \tag{5.13}$$

This equation is the expectation $E[1 - \log_2(1 + e^{-\Lambda_i})]$ over the pdf $p(\Lambda_i | a_i = +1)$,

$$I(a_i; \Lambda_i) = E\left[1 - \log_2(1 + e^{-\Lambda_i})\right] \tag{5.14}$$

By evoking the ergodic assumption, this expectation is arbitrarily closely approximated by the time average [12],

$$I(a_i; \Lambda_i) \approx 1 - \frac{1}{N} \sum_{n=1}^{N} \log_2(1 + e^{-a_i \cdot \Lambda_i}) \tag{5.15}$$

In [33], it was observed that the LLR output of a SISO block, approaches a gaussian distribution for an increasing number of iterations. We can thus use equation 5.15 to calculate the mutual information of the LLR output of the databits,

$$I_{d_N}(d_k;\Lambda_{d_k}) \triangleq 1 - \frac{1}{N}\sum_{n=1}^{N}\log_2(1+e^{-d_k\cdot\Lambda_{d_k}}) \tag{5.16}$$

so the mutual information I_{d_N} of N databits d_k can be calculated with N values of databit d_k and their corresponding LLR's Λ_{d_k} at the output of the decoder.

The mutual information of the code bits can be obtained by averaging over M code bits that are mapped on one information bit,

$$I_{c_N}(c_{i,m};\Lambda_{c_{i,m}}) = \frac{1}{M}\sum_{m=1}^{M} I(c_{i,m};\Lambda_{c_{i,m}}) \tag{5.17}$$

with

$$I(c_{i,m};\Lambda_{c_{i,m}}) \triangleq 1 - \frac{1}{N}\sum_{n=1}^{N}\log_2(1+e^{-c_{i,m}\cdot\Lambda_{c_{i,m}}}) \tag{5.18}$$

where $c_{i,m=\{1..M\}}$ is the m^{th} codebit of the M codebits mapped on a_i and $\Lambda_{c_{i,m}}$ is the corresponding log-likelihood value. So the mutual information I_{c_N} can be calculated by calculating the mutual information $I(c_{i,m};\Lambda_{c_{i,m}})$ for N codebits $c_{i,m}$ and then averaging $I(c_{i,m};\Lambda_{c_{i,m}})$ for every codebit.

To create the transfer chart the mutual information of the a-priori input I_A and the extrinsic output I_E are related to each other by the transfer function T [28]. For an inner decoder in an SCCC architecture or for both the decoders in a PCCC architecture (see section 3.2 and 3.3), I_E is not only a function of I_A, but also of the signal to noise ratio E_b/N_0 of the channel, so

$$I_E = T(I_A, E_b/N_0) \tag{5.19}$$

The transfer chart of a decoder is obtained by supplying its a-priori LLR input with known mutual information and measuring the mutual information of the desired output. For the a-priori LLR values it has been observed by simulation [33] that, 1) For large interleavers the a-priori values are highly uncorrelated from the channel observations and 2) the extrinsic LLR output approaches a gaussian like distribution with an increasing number of iterations. Since the extrinsic LLR output becomes the a-priori LLR input, we can model the a-priori input A with the above observations as,

$$A = \mu_A \cdot x + \eta_A \tag{5.20}$$

where x is a known transmitted systematic bit and η_A is a gaussian distributed variable with mean zero and variance σ_A^2. Thus for the a-priori

LLR input the relation between its mean and variance is the same as in equation 5.8. When we combine equations 5.2, 5.8 and 5.13 we can obtain a relation between the variance of a gaussian distribution and its mutual information,

$$I_A(\sigma_A) = 1 - \int_{-\infty}^{+\infty} \frac{1}{\sqrt{2\pi}\sigma_A} e^{-((\xi-(\sigma_A^2/2)\cdot x)^2)/2\sigma_A^2)} \cdot \log_2(1+e^{-\Lambda_i}) d\Lambda_i \quad (5.21)$$

For abbreviation we define [29],

$$J(\sigma) = I_A(\sigma_A = \sigma) \quad (5.22)$$

with

$$\lim_{\sigma\to 0} J(\sigma) = 0 \ \textit{and} \ \lim_{\sigma\to\infty} J(\sigma) = 1 \quad (5.23)$$

Equation 5.21 can not be expressed in a closed form. However, it is monotonically increasing [29], so it is reversible,

$$\sigma_A = J^{-1}(I_A) \quad (5.24)$$

With this equation we can find for a desired mutual information of the a-priori input, the variance, and thus mean with the help of equation 5.8, of the gaussian input. In figure 5.2 equation 5.21 is plotted[1].

Figure 5.3 shows the simulation setup to generate the transfer chart for the inner decoder in an SCCC architecture and for both the decoders in a PCCC architecture. Randomly generated databits are encoded to codebits. The rate and generator polynomials can be chosen arbitrarily, however the decoder should be the corresponding SISO decoder of this encoder. The codebits are sent over an AWGN channel with a certain E_b/N_0 for which the EXIT chart is desired. The channelbits are converted to LLR values with the help of equation 5.4, by multiplying the received channelbits with $2/\sigma_n^2$. The LLR values for the codebits are offered to the decoder as codebit LLR input. The a-priori information is created by multiplying the databits with a gaussian variable. This gaussian variable has a mean μ_A and σ_A that corresponds to a certain mutual information I_A. The σ_A that corresponds to a certain mutual information I_A can be calculated with equation 5.24. With equation 5.8 the mean μ_A can also be found.

Both these streams are offered to the decoder. The LLR output of the codebits is discarded. The mutual information of the LLR data output is calculated according to equation 5.16.

Figure 5.4 shows the simulation setup to generate the transfer chart for the outer decoder in a SCCC architecture or for the decoder in the turbo

[1] A numerical evaluation of equation 5.21 gives numerical problems for certain ranges of σ. Figure 5.2 is therefor plotted with Wolframs' Research Mathematica, which fortunately indicated the regions where numerical problems occur, so that different calculation methods could be used [Hoeksema].

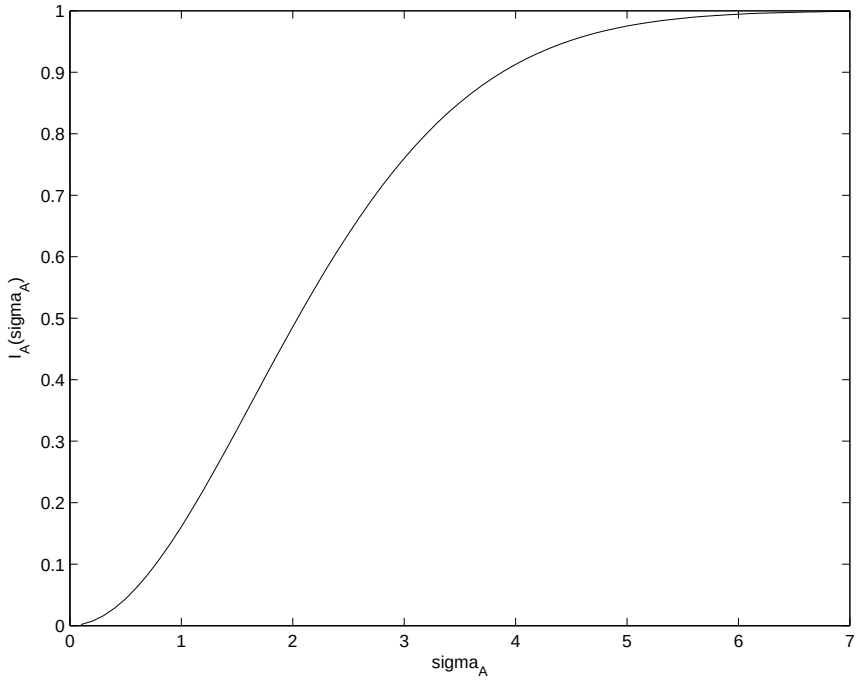


Figure 5.2: A-priori mutual information as a function of σ

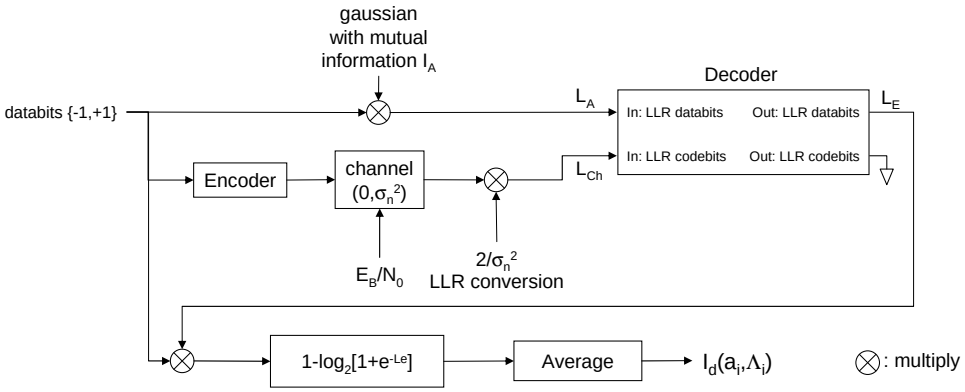


Figure 5.3: Simulation setup for generating transfer charts of inner decoders in an SCCC architecture and for both decoders in a PCCC architecture

multiuser detector of figure 4.1. This decoder only receives LLR values of its codebits as input. Randomly generated databits are encoded to codebits. A rate 1/2 encoder is used for practical purposes, because only two codebits need to be combined. The codebits are multiplied with a gaussian distributed variable with a certain mutual information. The mutual

information is determined in the same was as for the inner decoder. These codebits with a certain mutual information are offered to the decoder. The LLR data input of the decoder is set to zero, as it would be in an SCCC architecture. The mutual information of the LLR codebit output is calculated according to equation 5.18.

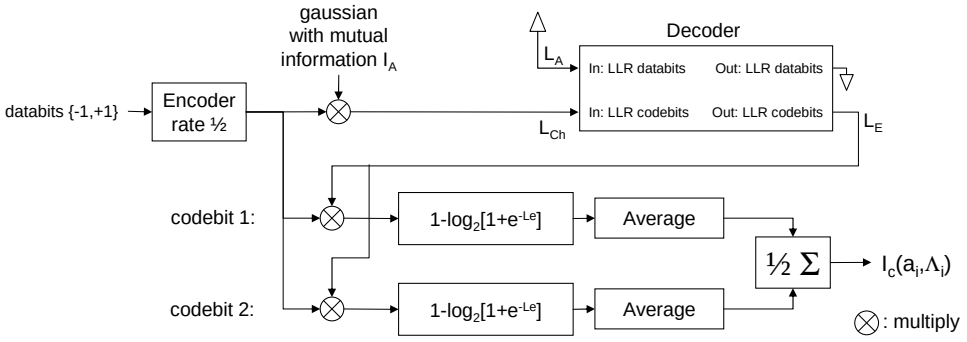


Figure 5.4: Simulation setup for generating transfer charts for outer decoders in an SCCC architecture

Figure 5.5 shows the simulation setup to generate the transfer chart for a multiuser detector for an AWGN channel with variance σ_n^2. The simulation setup is for a system with four users. It is assumed that the spreading codes of the users have an equal correlation coefficient. This assumption is made, so that the a-priori input of every user of the MUD detector can be supplied with the same value for the mutual information. The mutual information outputs I_{d1}, I_{d2}, I_{d3} and I_{d4} should all have about the same values at every iteration.

5.3 EXIT Charts

An EXIT chart consists of two transfer charts from decoder blocks. The two transfer charts are plotted in one figure, where one of the transfer charts has its axes swapped. Since the output of one block is the input of the other in an iterative architecture, the information *trajectory* of a turbo architecture can be analyzed.

For a PCCC architecture, the extrinsic output of the first decoder is interleaved and given as a-priori input to the second decoder. The interleaving operation does not change the mutual information [29]. The extrinsic output of the second decoder is de-interleaved and given to the first decoder as a-priori input, again the de-interleaving does not change the mutual information. In figure 5.6 an example EXIT chart of a PCCC architecture is shown. The transfer chart of decoder 1 is directly plotted in the EXIT chart, which is the upper line: on the x-axis its a-priori input and on the y-axis

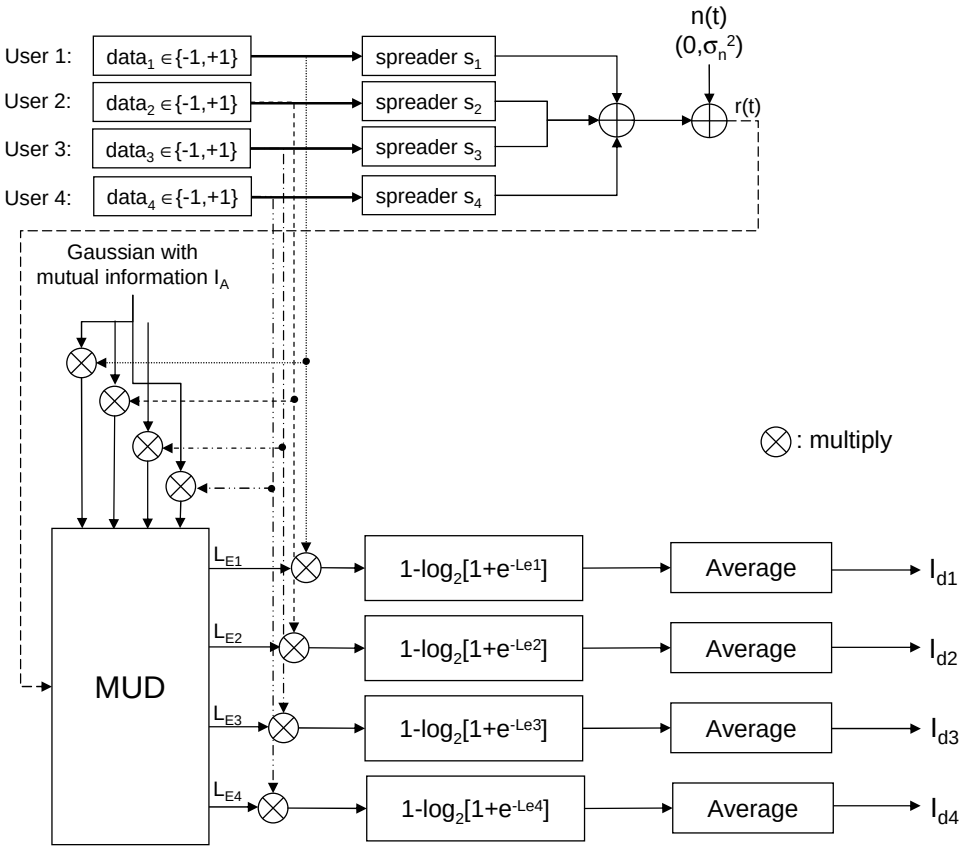


Figure 5.5: Simulation setup for generating the transfer chart for a multiuser detector with four users and a AWGN channel with variance σ_n^2

its extrinsic output. The axes of the transfer chart of the second decoder are swapped and plotted in the same EXIT chart: on the y-axis its a-priori input and on the x-axis its extrinsic output. Now the extrinsic mutual information transfer of the two decoders can be followed through the chart, because the output of one decoder, becomes the input of the other.

We define $I_{A_{m,n}}$ and $I_{E_{m,n}}$ as respectively the a-priori information of the m^{th} decoder during the n^{th} iteration and the a-posteriori information of the m^{th} decoder during the n^{th} iteration and T_m is the information transfer function of decoder m. At the first iteration $I_{A_{1,0}} = 0$. With fixed E_b/N_0 the extrinsic output of the first decoder becomes $I_{E_{1,0}} = T_1(I_{A_{1,0}})$. The extrinsic mutual information output is then given to the second decoder as a-priori input, $I_{A_{2,0}} = I_{E_{1,0}}$. The extrinsic mutual information output of the second decoder then becomes $I_{E_{2,0}} = T_2(I_{A_{2,0}})$ or $I_{E_{2,0}} = T_2(T_1(I_{A_{1,0}}))$. This can be generalized by $I_{A_{2,n}} = I_{E_{1,n}}$, $I_{A_{1,n+1}} = I_{E_{2,n}}$, $I_{E_{1,n}} = T_1(I_{A_{1,n}})$ and $I_{E_{2,n}} = T_2(I_{A_{2,n}})$. Note that interleaving does not change the mutual

information.

Iterating over the two decoders is only useful, when at each iteration some extra information is gained. So iterations proceed as long as $I_{E_{2,n+1}} > I_{E_{2,n}}$. With $I_{E_{2,n+1}} = T_2(T_1(I_{E_{2,n}}))$ or $T_2^{-1}(I_{E_{2,n+1}}) = T_1(I_{E_{2,n}})$ we can formulate this as $T_1(I_{E_{2,n}}) > T_2^{-1}(I_{E_{2,n+1}})$. So iterations stop when $I_{E_{2,n+1}} = I_{E_{2,n}}$ or $T_1(I_{E_{2,n}}) = T_2^{-1}(I_{E_{2,n+1}})$. This situation corresponds to an intersection of the transfer charts of the two decoders in the EXIT chart.

In figure 5.6 the transfer charts at $-1.5dB$ for the two decoders in the lower left corner of the figure, intersect quite early in the iteration process, while the transfer chart at $-1dB$ never intersect before a mutual information of 1 is reached.

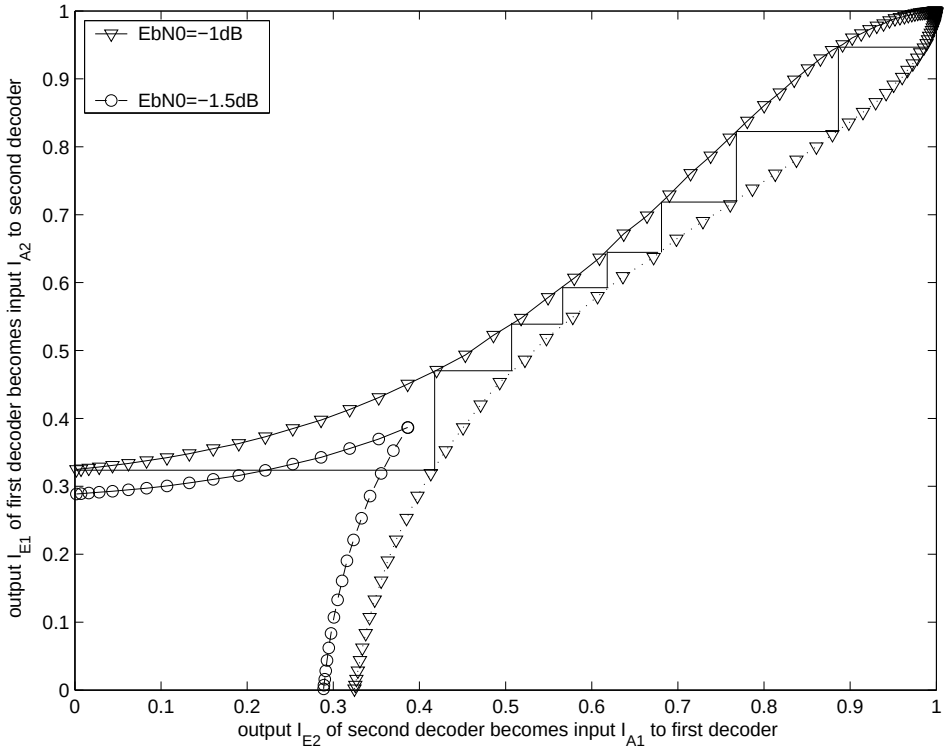


Figure 5.6: Simulated trajectories of iterative decoding at $E_b/N_0 = -1.5dB$ and $-1dB$ (symmetric PCC rate 1/3, interleaver size 16.384 systematic bits

For low E_b/N_0 values the two transfer charts in an EXIT chart become closer to each other. When they are so close that they just intersect at some point, the turbo architecture will not be able to converge to a point where a mutual information of 1 is reached. The EXIT chart can thus be used to analyze a turbo architecture's convergence behavior for different E_b/N_0.

From the above, the nomenclature for the different regions that is used

in section 5.1 can be understood. In the *pinch-off region* the transfer charts of the two decoders, intersect at some stage, so the turbo architecture is pinched off. In the *waterfall region* the EXIT chart just opens up. When the chart is 'opened' up a little more the turbo architecture iterates even faster to $I_E = 1$. In the *wide-open region* the transfer charts are very far apart and are thus wide open.

A point of interest is the BER that can be reached when $I_A \approx 1$ and $I_E \approx 1$. In [29] equation (31) gives a relation between the bit error probability P_b, the mutual informations I_A and I_E and E_b/N_0 of the channel. This formula is valid to calculate the BER from I_A and I_E in regions of low E_b/N_0. Readers are referred to the paper for more information.

5.4 Conclusions

In this chapter EXIT charts were introduced. An EXIT chart can be used as an analysis-tool for the convergence behavior of a turbo architecture. An EXIT chart is made from two transfer charts of the algorithms in a turbo architecture. Transfer charts are made of individual algorithms, without their implementation in the architecture where they are used.

In section 5.1 the convergence behavior of turbo architectures were divided in three regions: the pinch-off region, the waterfall region and the wide-open region. The names of these regions are obtained from EXIT charts. In section 5.2 simulation setups for creating the transfer charts were shown. With the transfer charts an EXIT chart was created for these turbo architectures. EXIT charts were shown to be a useful tool to determine when a turbo architecture is just able to improve the BER by iterating.

In chapter 7 EXIT charts for the SCCC and PCCC architectures of chapter 3 are created to verify the tool with which the EXIT charts are created.

Next an EXIT chart for the turbo multiuser detector of chapter 4 is created, to answer the second question of this thesis, given in chapter 1.

6

Soft-input Soft-output (SISO) Algorithms

As stated in section 3.3 a decoder in a turbo architecture, must accept LLR soft information about the code and data bits and produce LLR soft information about the code and data bits. These decoder are sometimes referred to as A-Posteriori Probability (APP) decoders.

The most famous convolutional code decoder invented in 1974 by Bahl, Cocke, Jelinek and Raviv is the Maximum A-Posteriori (MAP) decoder [2]. It is sometimes referred to as the *BCJR* algorithm, based upon the first letters of the inventors names. This algorithm can be used to decode convolutional and block-codes by minimizing the bit error rate. However, until the discovery of turbo codes it was not used often, because of its complexity. It also only performs slightly better (in a non-turbo configuration) than the Viterbi algorithm. When turbo-codes were discovered, it became interesting again. The MAP algorithm is discussed in section 6.1.1.

A lot of work has been done through the years to reduce the complexity of the MAP algorithm. This resulted in the Log-MAP algorithm, which has the same performance as the MAP algorithm, and the suboptimal Max-Log-Map algorithm. The Log-MAP algorithm transforms the calculations of the MAP algorithm to the logarithmic domain, which makes computations much simpler, while no approximations are made. The Max-Log-MAP algorithm makes an approximation in some of its calculations, which makes it suboptimal. The Log-MAP decoder is discussed in section 6.1.3 and the Max-Log-MAP decoder in section 6.1.2.

For turbo multiuser detection a multi user detector is needed which accepts and creates LLR soft information of all the code bits of every user available. Such a detector is discussed in section 6.2.1. This detector is derived from [20].

6.1 Convolutional Code Decoders

6.1.1 MAP Decoder

The following section is mainly based on [14], but adapted to fit this thesis. We assume a binary system with BPSK modulation: a databit '0' is represented by a BPSK-modulated databit '+1' and a databit '1' is represented by a BPSK-modulated databit '-1'. The BPSK-modulated databits will from now on be referred to as 'databits'.

To understand the MAP algorithm, a mind-experiment is set up. In this experiment a stream of databits is sent over an AWGN channel and received with some kind of antenna at a receiver. First we only send a stream of '-1' databits over the channel. When we measure the received channelbits at the receiver for a certain period, the conditional probability density function (pdf) of the '-1' databits can be created. This conditional pdf is denoted with $p(y_k|d_k = -1)$, where y_k is the received channelbit and d_k is the sent databit. We do the same for a stream of '+1' databits to obtain the conditional pdf $p(y_k|d_k = +1)$. Figure 6.1 shows these measurements.

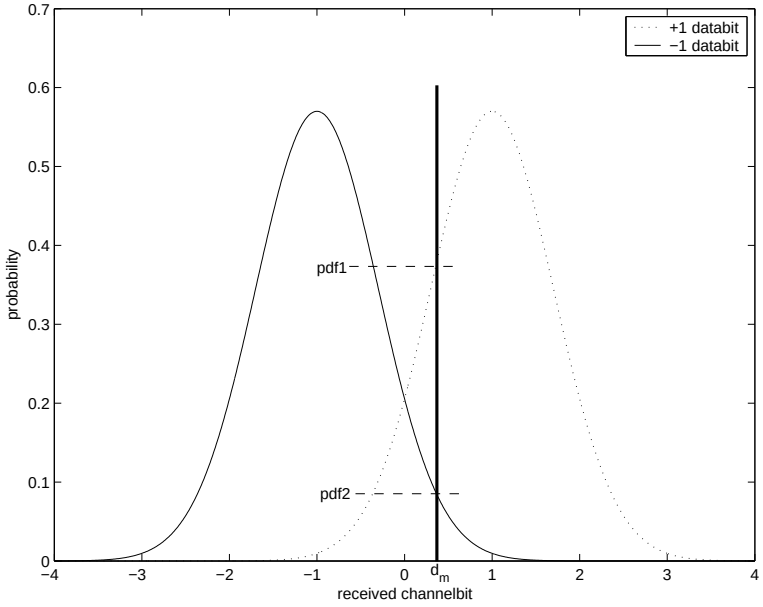


Figure 6.1: a-priori probability density function of BPSK elements

After creating these pdf's a random stream of databits is sent over the channel. At the receiver a measurement d_m of a received channelbit is made. The measurement d_m corresponds with two values on the pdf's in figure 6.1, namely pdf1 and pdf2. The maximum likelihood rule states that the databit corresponding to the received channelbit, is the element that has the highest

likelihood. In this case pdf1 has the highest likelihood and thus the received element is decided to be a '+1' databit. This rule corresponds to looking at the sign of the received channelbit to determine the sent databit. This decision rule is called the a-priori rule, since the decision is already known without any channelbits being received..

Another rule which can be used to decide what databits was sent is the maximum a-posteriori (MAP) rule. Instead of determining the largest a-priori likelihood function $p(y_k|d_k = \pm 1)$, the largest a-posteriori probability $p(d_k = \pm 1|y)$ is determined. We see here that the elements are first received, before a decision is made on what was received. Also y is a block of received channelbits, so a decision on d_k is based on a block of channelbits.

Let us define two hypotheses H_1 and H_2, where if hypothesis H_1 is true the databit is a '+1' and if hypothesis H_2 is true the databit is a '-1'. Equation 6.1 gives the MAP decision rule

$$P(d_k = +1|y) \gtrless_{H_2}^{H_1} P(d_k = -1|y) \tag{6.1}$$

so if $P(d_k = +1|y)$ is larger than $P(d_k = -1|y)$, hypothesis H_1 is assumed to be true and hypothesis H_2 is true in the other case. This equation is the basis for a MAP decoder. In section 3.3 the log-likelihood ratio was defined. If we rewrite equation 6.1, we see why this ratio can be useful

$$\frac{P(d_k = +1|y)}{P(d_k = -1|y)} \gtrless_{H_2}^{H_1} 1 \tag{6.2}$$

The derivation of the MAP decoder from this equation is done with the use of the Bayes' rule. The Bayes' rule is shown in equation 6.3, where $|$ is the conditional symbol and $\cap$ is the 'AND' symbol.

$$P(A \cap B) = P(A|B)P(B) = P(B|A)P(A) \tag{6.3}$$

The Bayes' rule allows us to rewrite equation 6.1 in two different ways, as is shown in equation 6.4 and equation 6.5.

$$\frac{P(d_k = +1 \cap y)}{P(y)} \gtrless_{H_2}^{H_1} \frac{P(d_k = -1 \cap y)}{P(y)} \tag{6.4}$$

and

$$\frac{P(y|d_k = +1)P(d_k = +1)}{P(y)} \gtrless_{H_2}^{H_1} \frac{P(y|d_k = -1)P(d_k = -1)}{P(y)} \tag{6.5}$$

In [14] equation 6.4 is used to determine the MAP receiver, while in [26] equation 6.5 is used to obtain the MAP decoder. The choice of formula does not matter. For no apparent reason we use equation 6.4 as a base to start the derivations of the MAP decoder.

Another observation which can be made from equation 6.5, is that this MAP rule converts to the ML rule, when $P(d_k = +1) = P(d_k = -1) = 0.5$, so when the bit probabilities are known.

We rewrite equation 6.4 to give us the log-likelihood ratio $L(d_k|y)$,

$$\Lambda(d_k|y_k) \triangleq ln\left(\frac{P(d_k = +1 \cap y_k)}{P(d_k = -1 \cap y_k)}\right) \tag{6.6}$$

The decision rule now becomes,

$$\Lambda(d_k|y_k) \gtrless_{H_2}^{H_1} 0 \tag{6.7}$$

If the LLR $\Lambda(d_k|y_k)$ is positive the probability of a '+1' was larger than the probability of a '-1' and vice-versa.

To obtain an expression for the log-likelihood in equation 6.6, we have to take a look at the trellis of the used code. See figure 6.2 for the trellis of an RSC code with constraint length $\nu = 3$. A state transition shown by a solid line, is caused by an input element of '+1', while a broken line shows the transitions caused by an input element '-1'.

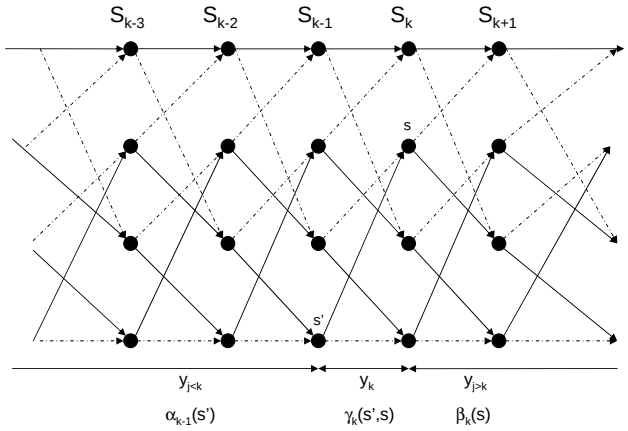


Figure 6.2: MAP decoder Trellis for RSC $\nu = 3$ code

In figure 6.2 the states at discrete time points S_{k-3}, S_{k-2}, S_{k-1}, S_k and S_{k+1} are shown. A state transition from state s' to one of its two possible next states s determines which bit was sent. So the probability of an '-1' input element, is the probability of all the state transitions caused by a '-1' (the broken lines in figure 6.2). The same holds for an '+1' as input element. We can therefor rewrite equation 6.6 as,

$$\Lambda(d_k|y) = ln\left(\frac{\sum_{(s',s)\Longrightarrow d_k=+1} P(S_{k-1} = s' \cap S_k = s \cap y)}{\sum_{(s',s)\Longrightarrow d_k=-1} P(S_{k-1} = s' \cap S_k = s \cap y)}\right) \tag{6.8}$$

where $(s', s) \Longrightarrow d_k = +1$ is the set of transition from previous state S_{k-1} to current state S_k that can occur for $d_k = +1$, vice versa for $d_k = -1$.

Until now we have assumed that we wanted to know the log-likelihood of the data bit. In our SISO decoder we also want to know the log-likelihood of the code bits. When, for example, we want to know the log-likelihood $\Lambda(c_1|y)$ of the code bit c_1, formula 6.8 becomes

$$\Lambda(c_1|y) = ln\left(\frac{\sum_{(s',s)\Longrightarrow c_1=+1} P(S_{k-1} = s' \cap S_k = s \cap y)}{\sum_{(s',s)\Longrightarrow c_1=-1} P(S_{k-1} = s' \cap S_k = s \cap y)}\right) \tag{6.9}$$

So the summation is over all branches where $c_1 = +1$ in the numerator and $c_1 = -1$ in the denominator.

From now on we will write $P(S_{k-1} = s' \cap S_k = s \cap y)$ as $P(s' \cap s \cap y)$. We split up the received channelword in three parts: the channelbits that belong to the present transition $s' \Longrightarrow s$ is called y_k, the sequence before the present transition is called $y_{j<k}$ and the sequence after the present transition is called $y_{j>k}$. In figure 6.2 these three parts are shown. We now can write for the individual probability $P(s' \cap s \cap y)$,

$$P(s' \cap s \cap y) = P(s' \cap s \cap y_{j<k} \cap y_k \cap y_{j>k}) \tag{6.10}$$

If we assume that the channel is memoryless, we know that the sequence $y_{j>k}$, that still has to be received, only depends on the current state s and not on the previous state s'. Assuming a memoryless channel also implies that the already received sequence $y_{j<k}$ only depends on the previous state s' and not on the current state s. Together with Bayes' rule in equation 6.3, we can expand equation 6.10 into

$$\begin{aligned} P(s' \cap s \cap y) &= P(y_{j>k}|s' \cap s \cap y_{j<k} \cap y_k) \cdot P(s' \cap s \cap y_{j<k} \cap y_k) \\ &= P(y_{j>k}|s) \cdot P(s' \cap s \cap y_{j<k} \cap y_k) \\ &= P(y_{j>k}|s) \cdot P(y_k \cap s|s' \cap y_{j<k}) \cdot P(s' \cap y_{j<k}) \\ &= P(y_{j>k}|s) \cdot P(y_k \cap s|s') \cdot P(s' \cap y_{j<k}) \end{aligned} \tag{6.11}$$

On the second line of equation 6.11 we substituted $P(y_{j>k}|s' \cap s \cap y_{j<k} \cap y_k)$ with $P(y_{j>k}|s)$, because the conditional probability of $y_{j>k}$ does not depend on anything that happened before state s. Subsequently the same is done for the conditional probability of $y_k \cap s$.

Remember that each state transition is caused by N codebits, where N is the rate of the encoder. So the variable y_k consists of N channelbits. The k subscript is still used to denote these N channelbits, to make it easier to see to which databit d_k these N codebits belong.

From equation 6.11 we see that we have written the probability $P(s' \cap s \cap y)$ as a set of multiplications of probabilities of events happening on, before and after time k. We rewrite equation 6.11 as,

$$P(s' \cap s \cap y) = \beta_k(s) \cdot \gamma_k(s', s) \cdot \alpha_{k-1}(s') \tag{6.12}$$

where

$$\alpha_{k-1}(s') \triangleq P(s' \cap y_{j<k}) \tag{6.13}$$

which is the probability that the state at time k-1 is s' and the received sequence up to that point is $y_{j<k}$.

$$\beta_k(s) \triangleq P(y_{j>k}|s) \tag{6.14}$$

which is the probability that after the trellis state s, the future received sequence is $y_{j>k}$.

$$\gamma_k(s', s) \triangleq P(y_k \cap s|s') \tag{6.15}$$

which is the probability that the state at time k-1 is s' and the state at time k is s, given the received element y_k.

Figure 6.2 shows the meaning of $\alpha_{k-1}(s')$, $\beta_k(s)$ and $\gamma_k(s', s)$. These values can be found by a recursive algorithm, which will be outlined in the next part of this section. When all the values are found, they can be used in equation 6.6 to calculate the log-likelihood for the databit(s) and codebit(s).

Forward Recursive Calculation of $\alpha_k(s)$
To obtain the algorithm for recursively calculating $\alpha_k(s)$, we rewrite equation 6.13 to,

$$\begin{aligned} \alpha_k(s) &= P(s \cap y_{j<k+1}) \\ &= P(s \cap y_{j<k} \cap y_k) \\ &= \sum_{alls'} P(s' \cap s \cap y_{j<k} \cap y_k) \end{aligned} \tag{6.16}$$

together with Bayes rule and the assumption that the channel is memoryless, we can follow the same procedure as with equation 6.11,

$$\begin{aligned} \alpha_k(s) &= \sum_{alls'} P(s \cap y_k|s' \cap y_{j<k}) \cdot P(s' \cap y_{j<k}) \\ &= \sum_{alls'} P(s \cap y_k|s') \cdot P(s' \cap y_{j<k}) \\ &= \sum_{alls'} \gamma_k(s', s) \cdot \alpha_{k-1}(s') \end{aligned} \tag{6.17}$$

From equation 6.17 we see that the value of $\alpha_k(s)$ can be calculated recursively once the value for $\gamma_k(s', s)$ is known. We only need some initial conditions,

$$\begin{aligned} \alpha_0(S_0 = 0) &= 1 \\ \alpha_0(S_0 = s) &= 0 \qquad \text{for all } s \neq 0 \end{aligned} \tag{6.18}$$

where we assume that the encoder started in state 0, so that state is the only possible start state.

Backward recursive calculation of $\beta_k(s)$ values
To recursively calculate the values of $\beta_k(s)$, we follow the same procedure as for the $\alpha_k(s)$ values. We can write for $\beta_{k-1}(s')$,

$$\beta_{k-1}(s') = P(y_{j>k-1}|s') \tag{6.19}$$

Again assuming the channel is memoryless and using Bayes' rule, we obtain,

$$\begin{aligned}
\beta_{k-1}(s') &= \sum_{\text{all s}} P(y_{j>k-1} \cap s|s') \qquad (6.20)\\
&= \sum_{\text{all s}} P(y_k \cap y_{j>k} \cap s|s')\\
&= \sum_{\text{all s}} P(y_{j>k}|s' \cap s \cap y_k) \cdot P(s \cap y_k|s')\\
&= \sum_{\text{all s}} P(y_{j>k}|s) \cdot P(s \cap y_k|s')\\
&= \sum_{\text{all s}} \beta_k(s) \cdot \gamma_k(s', s)
\end{aligned}$$

We see again that the values of $\beta_k(s)$ can be calculated backward recursively once the value of $\gamma_k(s', s)$ are known.

The initial values of $\beta_k(s)$ are a little bit more difficult to find. Berrou [6] used the same initial conditions for $\beta_k(s)$ as for $\alpha_k(s)$. However from equation 6.14, we saw that $\beta_k(s)$ is the probability of the data stream that is received after state s. Only after the last symbol N has been received, no further sequence will be received anymore. We can find an answer, when we consider $\beta_{N-1}(s)$ in equation 6.14,

$$\begin{aligned}
\beta_{N-1}(s') &= P(y_N|s') \qquad (6.21)\\
&= \sum_{\text{all s}} P(y_N \cap s|s')\\
&= \sum_{\text{all s}} \gamma_N(s', s)
\end{aligned}$$

From the recursion of equation 6.20 we find,

$$\beta_{N-1}(s') = \sum_{\text{all s}} \beta_N(s) \cdot \gamma_N(s', s) \tag{6.22}$$

If we want both equations to be satisfied, we must have,

$$\beta_N(s) = 1 \qquad \text{for all } s \tag{6.23}$$

If the trellis is terminated in state 0, then the values of γ will take care of the fact that for the last 2ν state transitions, there can only be one possible path

from the current to the next state. However implementing $\beta_N(s=1)=1$, $\beta_N(s\neq 1)=0$ and calculating γ normally from the inputs is much easier to implement and therefor used more often.

The calculation of $\gamma_k(s',s)$
We saw that the calculations of the $\alpha_k(s)$ and $\beta_k(s)$ values all require the value of $\gamma_k(s',s)$. We now determine how to calculate this value from equation 6.15 and Bayes' rule,

$$\begin{aligned}\gamma_k(s',s) &= P(y_k \cap s|s') \\ &= P(y_k|s' \cap s)\cdot P(s|s') \\ &= P(y_k|s' \cap s)\cdot P(d_k)\end{aligned} \tag{6.24}$$

where d_k is the data bit causing the transition from s' to s and $P(d_k)$ is the a-priori probability of this data bit.

To find an expression for $P(d_k)$ in equation 6.24, we use the log-likelihood $\Lambda(d_k)$ of the data bit probability $P(d_k)$,

$$\Lambda(d_k) \triangleq ln\left(\frac{P(d_k=+1)}{P(d_k=-1)}\right) \tag{6.25}$$

we can write for $P(d_k=+1)$,

$$\begin{aligned}e^{\Lambda(d_k)} &= \frac{P(d_k=+1)}{1-P(d_k=+1)} \\ P(d_k=+1) &= \frac{e^{\Lambda(d_k)}}{1+e^{\Lambda(d_k)}} \\ &= \frac{1}{1+e^{-\Lambda(d_k)}}\end{aligned} \tag{6.26}$$

Similarly for,

$$\begin{aligned}P(d_k=-1) &= \frac{1}{1+e^{+\Lambda(d_k)}} \\ &= \frac{e^{-\Lambda(d_k)}}{1+e^{-\Lambda(d_k)}}\end{aligned} \tag{6.27}$$

From these two equations an expression for $P(d_k)$ can be derived in two different ways.

$$P(d_k) = \frac{e^{d_k\Lambda(d_k)}}{1+e^{d_k\Lambda(d_k)}} \tag{6.28}$$

and

$$P(d_k) = \frac{e^{[(d_k+1)/2]\Lambda(d_k)}}{1+e^{\Lambda(d_k)}} \tag{6.29}$$

In both equation we made us of $d_k \epsilon \{-1,+1\}$. Equation 6.28 can be rewritten in a term independent of d_k and a term dependent on d_k. This is useful,

when during the calculation of the LLR, the term independent of d_k will cancel out in the numerator and denominator,

$$\begin{aligned} P(d_k) &= \left(\frac{e^{-\Lambda(d_k)/2}}{1+e^{-\Lambda(d_k)}}\right) \cdot e^{d_k \Lambda(d_k)/2} \\ &= C_1 \cdot e^{d_k \Lambda(d_k)/2} \end{aligned} \tag{6.30}$$

where

$$C_1 = \left(\frac{e^{-\Lambda(d_k)/2}}{1+e^{-\Lambda(d_k)}}\right) \tag{6.31}$$

For equation 6.29 the same can be done.

$$\begin{aligned} P(d_k) &= \left(\frac{1}{1+e^{\Lambda(d_k)}}\right) \cdot e^{[(d_k+1)/2]\Lambda(d_k)} \\ &= C_2 \cdot e^{[(d_k+1)/2]\Lambda(d_k)} \end{aligned} \tag{6.32}$$

where

$$C_2 = \left(\frac{1}{1+e^{\Lambda(d_k)}}\right) \tag{6.33}$$

We see that C_1 and C_2 do not depend on d_k.

Now the same can be done for the term $P(y_k|s' \cap s)$ in equation 6.24. We define a codeword x_k that consists of n codebits and is created by the encoding of one data bit d_k. Also we define a channelword y_k that is the received x_k after transmission over an AWGN channel. Since the transition from state s' to state s is caused by the codeword x_k, we can also write $P(y_k|s' \cap s)$ as $P(y_k|x_k)$. So assuming the channel is memoryless, we can write,

$$P(y_k|s' \cap s) = P(y_k|x_k) = \prod_{l=1}^{n} P(y_{kl}|x_{kl}) \tag{6.34}$$

where y_{kl} and $x_{kl}\epsilon\{-1,+1\}$ are respectively channelbit l and codebit l of the channel- and codeword.

We can express $P(y_{kl}|x_{kl})$ in terms of the log-likelihood by using the same approach as we did in formulas 6.25 to 6.32.

$$\Lambda(y_{kl}|x_{kl}) \triangleq ln\left(\frac{P(y_{kl}|x_{kl}=+1)}{P(y_{kl}|x_{kl}=-1)}\right) \tag{6.35}$$

and subsequently we find

$$\begin{aligned} P(y_{kl}|x_{kl}) &= \left(\frac{e^{-\Lambda(y_{kl}|x_{kl})/2}}{1+e^{-\Lambda(y_{kl}|x_{kl})}}\right) \cdot e^{x_{kl}\cdot\Lambda(y_{kl}|x_{kl})/2} \\ &= C_3 \cdot e^{x_{kl}\cdot\Lambda(y_{kl}|x_{kl})/2} \end{aligned} \tag{6.36}$$

where

$$C_3 = \left(\frac{e^{-\Lambda(y_{kl}|x_{kl})/2}}{1 + e^{-\Lambda(y_{kl}|x_{kl})}} \right) \tag{6.37}$$

and

$$\begin{aligned} P(y_{kl}|x_{kl}) &= \left(\frac{1}{1 + e^{\Lambda(y_{kl}|x_{kl})}} \right) \cdot e^{[(x_{kl}+1)/2] \cdot \Lambda(y_{kl}|x_{kl})} \\ &= C_4 \cdot e^{[(x_{kl}+1)/2] \cdot \Lambda(y_{kl}|x_{kl})} \end{aligned} \tag{6.38}$$

where

$$C_4 = \left(\frac{1}{1 + e^{\Lambda(y_{kl}|x_{kl})}} \right) \tag{6.39}$$

We now use equations 6.30, 6.34 and 6.36 in equation 6.24 to obtain an expression for $\gamma_k(s', s)$

$$\gamma_k(s', s) = C \cdot e^{d_k \Lambda(d_k)/2} \cdot \prod_{l=1}^{n} e^{x_{kl} \cdot \Lambda(y_{kl}|x_{kl})/2} \tag{6.40}$$

where

$$C = C_1 \cdot C_3 \tag{6.41}$$

Now we can take equation 6.8 and equation 6.12 and use our previous results to obtain an expression for the log-likelihood ratio $\Lambda(d_k|y)$ of the data bit d_k,

$$\begin{aligned} \Lambda(d_k|y) &= ln \left(\frac{\sum_{(s',s) \Longrightarrow d_k=+1} P(S_{k-1} = s' \cap S_k = s \cap y)}{\sum_{(s',s) \Longrightarrow d_k=-1} P(S_{k-1} = s' \cap S_k = s \cap y)} \right) \qquad (6.42) \\ &= ln \left(\frac{\sum_{(s',s) \Longrightarrow d_k=+1} \alpha_{k-1}(s') \cdot \beta_k(s) \cdot \gamma_k(s', s)}{\sum_{(s',s) \Longrightarrow d_k=-1} \alpha_{k-1}(s') \cdot \beta_k(s) \cdot \gamma_k(s', s)} \right) \\ &= ln \left(\frac{\sum_{(s',s) \Longrightarrow d_k=+1} \alpha_{k-1}(s') \cdot \beta_k(s) \cdot C \cdot e^{d_k \Lambda(d_k)/2} \cdot \prod_{l=1}^{n} e^{x_{kl} \cdot \Lambda(y_{kl}|x_{kl})/2}}{\sum_{(s',s) \Longrightarrow d_k=-1} \alpha_{k-1}(s') \cdot \beta_k(s) \cdot C \cdot e^{d_k \Lambda(d_k)/2} \cdot \prod_{l=1}^{n} e^{x_{kl} \cdot \Lambda(y_{kl}|x_{kl})/2}} \right) \\ &= ln \left(\frac{\sum_{(s',s) \Longrightarrow d_k=+1} \alpha_{k-1}(s') \cdot \beta_k(s) \cdot e^{d_k \Lambda(d_k)/2} \cdot \prod_{l=1}^{n} e^{x_{kl} \cdot \Lambda(y_{kl}|x_{kl})/2}}{\sum_{(s',s) \Longrightarrow d_k=-1} \alpha_{k-1}(s') \cdot \beta_k(s) \cdot e^{d_k \Lambda(d_k)/2} \cdot \prod_{l=1}^{n} e^{x_{kl} \cdot \Lambda(y_{kl}|x_{kl})/2}} \right) \end{aligned}$$

We can rewrite this as

$$\begin{aligned} \Lambda(d_k|y) &= ln \left(e^{\Lambda(d_k)} \cdot \frac{\sum_{(s',s) \Longrightarrow d_k=+1} \alpha_{k-1}(s') \cdot \beta_k(s) \cdot \prod_{l=1}^{n} e^{x_{kl} \cdot \Lambda(y_{kl}|x_{kl})/2}}{\sum_{(s',s) \Longrightarrow d_k=-1} \alpha_{k-1}(s') \cdot \beta_k(s) \cdot \prod_{l=1}^{n} e^{x_{kl} \cdot \Lambda(y_{kl}|x_{kl})/2}} \right) \\ &= ln \left(e^{\Lambda(d_k)} \right) + ln \left(\frac{\sum_{(s',s) \Longrightarrow d_k=+1} \alpha_{k-1}(s') \cdot \beta_k(s) \cdot \prod_{l=1}^{n} e^{x_{kl} \cdot \Lambda(y_{kl}|x_{kl})/2}}{\sum_{(s',s) \Longrightarrow d_k=-1} \alpha_{k-1}(s') \cdot \beta_k(s) \cdot \prod_{l=1}^{n} e^{x_{kl} \cdot \Lambda(y_{kl}|x_{kl})/2}} \right) \\ &= \Lambda(d_k) + ln \left(\frac{\sum_{(s',s) \Longrightarrow d_k=+1} \alpha_{k-1}(s') \cdot \beta_k(s) \cdot \prod_{l=1}^{n} e^{x_{kl} \cdot \Lambda(y_{kl}|x_{kl})/2}}{\sum_{(s',s) \Longrightarrow d_k=-1} \alpha_{k-1}(s') \cdot \beta_k(s) \cdot \prod_{l=1}^{n} e^{x_{kl} \cdot \Lambda(y_{kl}|x_{kl})/2}} \right) \qquad (6.43) \\ &= \Lambda(d_k) + \Lambda_e(d_k|y) \qquad (6.44) \end{aligned}$$

with

$$\begin{aligned}\Lambda_e(d_k|y) &= ln\left(\frac{\sum_{(s',s)\Longrightarrow d_k=+1}\alpha_{k-1}(s')\cdot\beta_k(s)\cdot\prod_{l=1}^{n}e^{x_{kl}\cdot\Lambda(y_{kl}|x_{kl})/2}}{\sum_{(s',s)\Longrightarrow d_k=-1}\alpha_{k-1}(s')\cdot\beta_k(s)\cdot\prod_{l=1}^{n}e^{x_{kl}\cdot\Lambda(y_{kl}|x_{kl})/2}}\right)\\ &= ln\left(\sum_{(s',s)\Longrightarrow d_k=+1}\alpha_{k-1}(s')\cdot\beta_k(s)\cdot\prod_{l=1}^{n}e^{x_{kl}\cdot\Lambda(y_{kl}|x_{kl})/2}\right)-\\ &\quad ln\left(\sum_{(s',s)\Longrightarrow d_k=-1}\alpha_{k-1}(s')\cdot\beta_k(s)\cdot\prod_{l=1}^{n}e^{x_{kl}\cdot\Lambda(y_{kl}|x_{kl})/2}\right)\end{aligned}\tag{6.45}$$

In equation 6.43 the data bit d_k is in the numerator always $+1$ and in the denominator always -1. It can therefor be put in front of the fraction. In equation 6.45 $\Lambda_e(d_k|y)$ is the extrinsic information, as was discussed in equation 3.2 in section 3.3. The extrinsic information is passed to the next decoder. Keep in mind that for the calculation of $\alpha_{k-1}(s')$ and $\beta_k(s)$, $\gamma_k(s',s)$ as in equation 6.40 has to be calculated. The constant factor C, however, doesn't need to be calculated, since it will cancel out in the log-likelihood calculation.

The same can be done for the log-likelihood of a code bit x_{km}

$$\begin{aligned}\Lambda(x_{km}|y) &= ln\left(\frac{\sum_{(s',s)\Longrightarrow x_{km}=+1}\alpha_{k-1}(s')\cdot\beta_k(s)\cdot e^{d_k\Lambda(d_k)/2}\cdot\prod_{l=1}^{n}e^{x_{kl}\cdot\Lambda(y_{kl}|x_{kl})/2}}{\sum_{(s',s)\Longrightarrow x_{km}=-1}\alpha_{k-1}(s')\cdot\beta_k(s)\cdot e^{d_k\Lambda(d_k)/2}\cdot\prod_{l=1}^{n}e^{x_{kl}\cdot\Lambda(y_{kl}|x_{kl})/2}}\right)\\ &= ln\left(e^{\Lambda(y_{km}|x_{km})}\cdot\frac{\sum_{(s',s)\Longrightarrow c_k=+1}\alpha_{k-1}(s')\cdot\beta_k(s)\cdot e^{d_k\Lambda(d_k)/2}\cdot\prod_{l=1,l\neq m}^{n}e^{x_{kl}\cdot\Lambda(y_{kl}|x_{kl})/2}}{\sum_{(s',s)\Longrightarrow c_k=-1}\alpha_{k-1}(s')\cdot\beta_k(s)\cdot e^{d_k\Lambda(d_k)/2}\cdot\prod_{l=1,l\neq m}^{n}e^{x_{kl}\cdot\Lambda(y_{kl}|x_{kl})/2}}\right)\\ &= \Lambda(y_{km}|x_{km})+ln\left(\frac{\sum_{(s',s)\Longrightarrow c_k=+1}\alpha_{k-1}(s')\cdot\beta_k(s)\cdot e^{d_k\Lambda(d_k)/2}\cdot\prod_{l=1,l\neq m}^{n}e^{x_{kl}\cdot\Lambda(y_{kl}|x_{kl})/2}}{\sum_{(s',s)\Longrightarrow c_k=-1}\alpha_{k-1}(s')\cdot\beta_k(s)\cdot e^{d_k\Lambda(d_k)/2}\cdot\prod_{l=1,l\neq m}^{n}e^{x_{kl}\cdot\Lambda(y_{kl}|x_{kl})/2}}\right)\end{aligned}\tag{6.46}$$

Here the extrinsic information is also in the fraction, so that part is the only part that needs to be calculated for the next decoder.

Special case: code bit LLR input from AWGN channel

In a PCCC architecture, both the decoders take their code elements input from the channel. In an SCCC architecture only the inner decoder takes input from the channel. For a memoryless AWGN channel with BPSK modulation we find for $P(y_{kl}|x_{kl})$,

$$P(y_{kl}|x_{kl})=\frac{1}{\sqrt{2\pi}\sigma}\exp\left(-\frac{E_b}{2\sigma^2}(y_{kl}-ax_{kl})^2\right)\tag{6.47}$$

where E_b is the transmitted energy per bit, σ^2 is the noise variance of the channel and a is the fading amplitude (a=1 for non-fading channels and $0<a<1$ for flat fading channels).

When we use this equation in equation 6.35 we get for the log-likelihood $\Lambda(y_{kl}|x_{kl})$,

$$\begin{aligned}
\Lambda(y_{kl}|x_{kl}) &= ln\left(\frac{P(y_{kl}|x_{kl}=+1)}{P(y_{kl}|x_{kl}=-1)}\right) \\
&= ln\left(\frac{\frac{1}{\sqrt{2\pi}\sigma}\exp\left(-\frac{E_b}{2\sigma^2}(y_{kl}-a)^2\right)}{\frac{1}{\sqrt{2\pi}\sigma}\exp\left(-\frac{E_b}{2\sigma^2}(y_{kl}+a)^2\right)}\right) \\
&= -\frac{E_b}{2\sigma^2}(y_{kl}-a)^2+\frac{E_b}{2\sigma^2}(y_{kl}+a)^2 \\
&= \frac{2E_b a}{\sigma^2}y_{kl} \qquad (6.48)
\end{aligned}$$

In this formula it is clear that the channel elements should be multiplied by a constant, based upon channel characteristics, to get the log-likelihood for the decoder. A simpler version of this equation was also found in section 5.2 in equation 5.4.

6.1.2 Max-Log-MAP Decoder

The calculations of the standard MAP decoder need a lot of computations, because of the exponentials in the formulas. To solve the computational complexity problem, the Max-Log-MAP algorithm transfers the calculations of $\alpha_{k-1}(s')$, $\beta_k(s)$ and $\gamma_k(s',s)$ into the logarithmic domain. We define $A_k(s)$, $B_k(s)$ and $\Gamma_k(s',s)$,

$$A_k(s) \triangleq \ln(\alpha_k(s)) \qquad (6.49)$$

$$B_k(s) \triangleq \ln(\beta_k(s)) \qquad (6.50)$$

$$\Gamma_k(s',s) \triangleq \ln(\gamma_k(s',s)) \qquad (6.51)$$

The formulas of the conventional MAP algorithm for $\alpha_{k-1}(s')$, $\beta_k(s)$ and $\gamma_k(s',s)$ can be substituted into the equations 6.49, 6.50 and 6.51. By doing so, we can rewrite equation 6.49,

$$\begin{aligned}
A_k(s) &\triangleq \ln(\alpha_k(s)) \\
&= \ln\left(\sum_{\text{all s'}}\alpha_{k-1}(s')\gamma_k(s',s)\right) \\
&= \ln\left(\sum_{\text{all s'}}\exp[A_{k-1}(s')+\Gamma_k(s',s)]\right) \qquad (6.52)
\end{aligned}$$

For $B_k(s)$ we can find the same expression,

$$\begin{aligned} B_k(s) &\triangleq \ln(\beta_k(s)) \\ &= \ln\left(\sum_{\text{all s'}} \beta_{k+1}(s)\gamma_k(s',s)\right) \\ &= \ln\left(\sum_{\text{all s'}} \exp[B_{k+1}(s) + \Gamma_k(s',s)]\right) \end{aligned} \tag{6.53}$$

The calculation of $\Gamma_k(s',s)$ is as follows,

$$\begin{aligned} \Gamma_k(s',s) &\triangleq \ln(\gamma_k(s',s)) \\ &= ln\left(C \cdot e^{d_k\Lambda(d_k)/2} \cdot \prod_{l=1}^{n} e^{x_{kl}\cdot\Lambda(y_{kl}|x_{kl})/2}\right) \\ &= \widehat{C} + \frac{1}{2}d_k\Lambda(d_k) + \frac{1}{2}x_{kl}\Lambda(y_{kl}|x_{kl}) \end{aligned} \tag{6.54}$$

with

$$\widehat{C} = \ln(C) \tag{6.55}$$

The exponential in equations 6.52 and 6.53 can be avoided by making use of the following simplification.

$$ln\left(\sum_i e^{x_i}\right) \triangleq \max_i\{x_i\} \tag{6.56}$$

This simplification becomes more clear, when we write it as,

$$\begin{aligned} \ln\left(\sum_i e^{x_i}\right) &= \ln\left(e^{x_n} + \sum_{i,i\neq n} e^{x_i}\right) \\ &= \ln\left(e^{x_n}(1 + \sum_{i,i\neq n} e^{x_i - x_n}\right) \\ &= \ln\left(e^{x_n}\right) + \ln\left((1 + \sum_{i,i\neq n} e^{\overbrace{x_i - x_n}^{<0}}\right) \\ &\approx \ln\left(e^{x_n}\right) = x_n \text{ ,when: } x_n = \max_i x_i \end{aligned}$$

The approximation can only be made when $(x_i - x_n)$ is small enough and thus the exponential can be neglected.

Applying this simplification to equations 6.52 and 6.53 results in

$$A_k(s) \approx \max_{s'}(A_{k-1}(s') + \Gamma_k(s',s)) \tag{6.57}$$

For $B_k(s)$ we can find the same expression,

$$B_k(s) \approx \max_s (B_{k+1}(s) + \Gamma_k(s', s)) \tag{6.58}$$

In equation 6.57 we see that a new value $A_k(s)$ is calculated by adding the branch metric $\Gamma_k(s', s)$ to the previous $A_{k-1}(s')$. From the two paths reaching a state, only the state that has the highest metric is kept. This is the same as the Viterbi algorithm, since one path is being selected as the survivor and the other is discarded. The same holds for the calculation of $B_k(s)$.

For the LLR in equation 6.42, we can now find

$$\begin{aligned}
\Lambda(d_k|y) &= ln\left(\frac{\sum_{(s',s)\Longrightarrow d_k=+1} \alpha_{k-1}(s') \cdot \beta_k(s) \cdot \gamma_k(s', s)}{\sum_{(s',s)\Longrightarrow d_k=-1} \alpha_{k-1}(s') \cdot \beta_k(s) \cdot \gamma_k(s', s)}\right) \\
&= ln\left(\frac{\sum_{(s',s)\Longrightarrow d_k=+1} \exp(A_{k-1}(s') + B_k(s) + \Gamma_k(s', s))}{\sum_{(s',s)\Longrightarrow d_k=-1} \exp(A_{k-1}(s') + B_k(s) + \Gamma_k(s', s))}\right) \\
&\approx \max_{(s',s)\Longrightarrow d_k=+1} (A_{k-1}(s') + B_k(s) + \Gamma_k(s', s)) \\
&\quad - \max_{(s',s)\Longrightarrow d_k=-1} (A_{k-1}(s') + B_k(s) + \Gamma_k(s', s)) \qquad (6.59) \\
&= \max_{(s',s)\Longrightarrow d_k=+1} (A_{k-1}(s') + B_k(s) + \frac{1}{2} d_k \Lambda(d_k) + \frac{1}{2} x_{kl} \Lambda(y_{kl}|x_{kl})) \\
&\quad - \max_{(s',s)\Longrightarrow d_k=-1} (A_{k-1}(s') + B_k(s) + \frac{1}{2} d_k \Lambda(d_k) + \frac{1}{2} x_{kl} \Lambda(y_{kl}|x_{kl}))
\end{aligned}$$

We obtain the expression for the LLR with the extrinsic information term like in equation 6.45,

$$\begin{aligned}
\Lambda_e(d_k|y) &= \max_{(s',s)\Longrightarrow d_k=+1} (A_{k-1}(s') + B_k(s) + \sum_{l=1}^{n} \frac{1}{2} x_{kl} \Lambda(y_{kl}|x_{kl})) \\
&\quad - \max_{(s',s)\Longrightarrow d_k=-1} (A_{k-1}(s') + B_k(s) + \sum_{l=1}^{n} \frac{1}{2} x_{kl} \Lambda(y_{kl}|x_{kl}))
\end{aligned} \tag{6.60}$$

6.1.3 Log-MAP Decoder

The Max-Log-MAP algorithm uses the approximation of equation 6.56, which makes it suboptimal. However, this equation can be made more exact by using the equation,

$$\begin{aligned}
\ln(e^{x_1} + e^{x_2}) &= \ln\left[e^{x_1}(1 + e^{x_2 - x_1})\right] \\
&= x_1 + \ln\left(1 + e^{x_2 - x_1}\right)
\end{aligned} \tag{6.61}$$

If instead taking x_1 out of the logarithm in equation 6.61, the largest of x_1 and x_2 is taken out of the algorithm,

$$\begin{aligned} \ln(e^{x_1} + e^{x_2}) &= \max(x_1, x_2) + \ln\left(1 + e^{-|x_1 - x_2|}\right) \\ &= \max(x_1, x_2) + f_c(\delta) \\ &= g(x_1, x_2) \end{aligned} \tag{6.62}$$

and

$$f_c(\delta) = \ln\left(1 + e^{|x_1 - x_2|}\right) \tag{6.63}$$

Equation 6.62 only accepts two arguments, while there are normally more than two transitions caused by a $d_k = +1$ or $d_k = -1$. This problem can be solved by nesting the $g(x_1, x_2)$ function.

$$\ln\left(\sum_{i=1}^{I} e^{x_i}\right) = g(x_I, g(x_{I-1}, \ldots, g(x_3, g(x_2, x_1))) \ldots) \tag{6.64}$$

From equation 6.62 and equation 6.64, we see that the Log-Map algorithm is in essential the same as the Max-Log-MAP algorithm. To obtain the Log-MAP algorithm from the Max-Log-MAP algorithm, for every MAX operation a correction term $f_c(\delta)$ has to be added to the maximum value.

It has been found in [14] that the correction term $f_c(\delta)$ does not need to be calculated every time, but only for eight values between 0 and 5.

6.1.4 Performance and Complexity of MAP decoders

If the SISO decoders are grouped according to performance with the best decoder first and worst decoder last, it would be MAP, Log-MAP and Max-Log-MAP [23]. The complexity of implementing the decoders, from large to small, gives the same group. The MAP decoder is never considered in literature for implementation. This is because of its very high complexity and because the Log-MAP decoder gives the same performance for a lesser complexity, as will be seen later.

Simulations done in [23] show that the Log-MAP and MAP decoder perform equally best for bit-noise ratios ranging from 1.0dB to 2.5dB. Max-Log-MAP performs 0.5dB worse than Log-MAP.

The complexity of the SISO decoders is given in [23], however these results have not been checked with the SISO decoders in this thesis. These results are copied in table 6.1.

6.2 SISO Multiuser Detectors

The soft multiuser detector of figure 4.1 in chapter 4 needs to accept a-priori LLR's of the codebits and needs to produce a-posteriori LLR's of the

Operation	Max-Log-MAP	Log-MAP
max ops	$5\text{x}2^{\nu} - 2$	$5\text{x}2^{\nu} - 2$
additions	$10\text{x}2^{\nu} + 11$	$15\text{x}2^{\nu} + 9$
mult. by $\pm$ 1	8	8

Table 6.1: Complexity of different SISO decoders (ν is the constraint length of the encoder) taken from [23]

codebits for every user. It can do this by using the knowledge from the received signal, the multi path channel and the spreadword of the desired user or users. There exist a lot of multiuser detectors, like the de-correlating detector and the decision driven detectors, who all produce hard decisions of the received bits. For detailed description of multiuser detectors see the master's thesis of J. Potman [21]. The existing detectors need to be adapted to accept and produce soft information.

All multiuser detectors have a bank of matched filters in common. For every user in the system there is a matched filter, which is matched to the spreadword of that user. The blocks that follow the bank of matched filters determine the kind of multiuser detector.

6.2.1 SISO Soft Cancellation Multiuser Detector

The multiuser detector investigated in this thesis, was developed by X. Wang and V. Poor in [20]. It consists of a bank of matched filters followed by a soft interference cancellation stage and a Minimum Mean Square Error (MMSE) filter and finally the calculation of the soft outputs. See figure 6.3 for the block diagram of the multiuser detector. This multiuser detector is used in

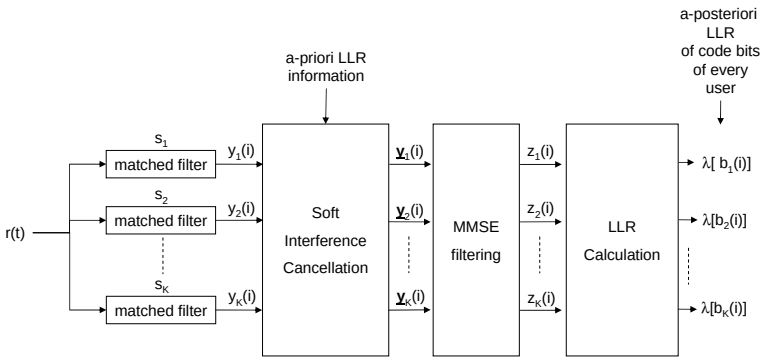


Figure 6.3: SISO Multiuser Detector block for a turbo multiuser detection architecture

the architectures of figures 4.1, 4.2 and 4.3 of chapter 4. The matched filters are matched to the spreadwords of the users in the system. The output of

the matched filters is led to a soft cancellation block, which uses the a-priori information of the code bits of every user to remove the interference of the other users. The output of the soft cancellation block is led to a MMSE filter, which filters for the bit of the desired user. From the output of the MMSE filter, the LLR of the codebits of each user is calculated. This is the a-posteriori information of the codebit, which is passed to the convolutional code decoder as the a-priori information. The soft-interference cancelling multiuser detector of Wang and Poor in [20] is described in more detail in the remainder of this section.

System Description

In section 1.3 the DS-CDMA receiver is briefly discussed. We continue here to give a mathematical description. The transmitter and channel that are used are shown in figure 6.4. There are K users in the system. We consider

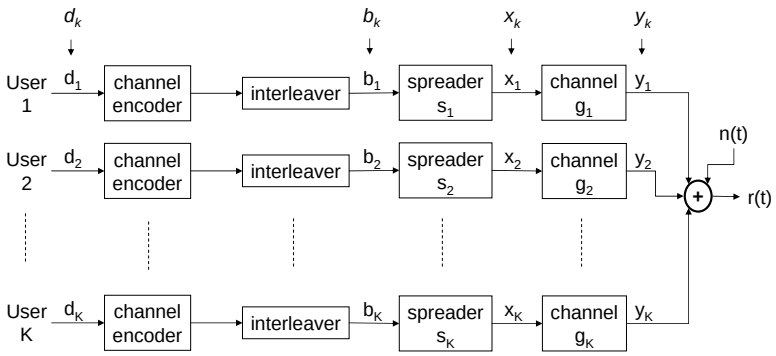


Figure 6.4: CDMA transmitter and channel

the signals $b_1, b_2, \ldots, b_K$ after the interleaver, where b_k is the coded data symbol of the k^{th} user and $b_k \in \{-1, +1\}$. This signal is spread with normalized spreadwords $s_1, s_2, \ldots, s_K$, where s_k is the spreadword of the k^{th} user and $s_k \in \{-1, +1\}$. s_k is build up out of chips, where each chip has duration T_C. The transmitted signal due to the k^{th} user is given by,

$$x_k(t) = A_k \sum_{i=0}^{M-1} b_k(i) s_k(t - iT) \tag{6.65}$$

where M is the number of symbols b_k per user per frame, T is the duration of a symbol b_k, A_k is the amplitude of the k^{th} user. It is assumed that $s_k(t)$ is only supported on the interval $[0, T]$ and has unit energy. See figure 6.5 for an illustration of the spreading. The k^{th} user's signal travels through the channel $g_k(t)$, which has a baseband impulse response,

$$g_k(t) = \sum_{l=0}^{L_k-1} g_{kl} \delta(t - \tau_{kl}) \tag{6.66}$$

where L_k are the number of paths in the k^{th} user's channel, g_{kl} is the complex gain of the l^{th} path of the k^{th} user's channel and τ_{kl} is the delay of the l^{th} path of the k^{th} user's channel and can be defined in the terms of chip or symbol durations depending on the path. See figure 6.6 for what $g_k(t)$ would look like for some arbitrary values of g_{kl} and τ_{kl}. For the signal y_k we can

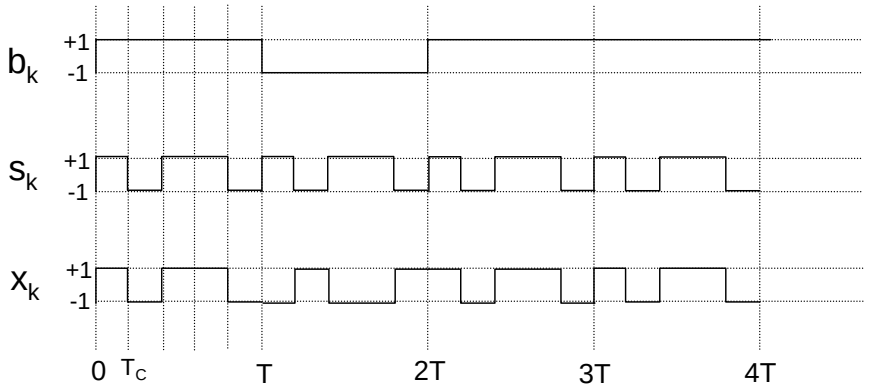


Figure 6.5: DS-CDMA behavior in the time domain

now write,

$$\begin{aligned} y_k(t) &= x_k(t) \otimes g_k(t) \\ &= A_k \sum_{i=0}^{M-1} b_k(i) \sum_{l=0}^{L_k-1} g_{kl} s_k(t - iT - \tau_{kl}) \end{aligned} \tag{6.67}$$

where $\otimes$ denotes convolution. The signal $r(t)$ at the receiver is a superposition of the signals $y_1(t), \ldots, y_K(t)$ together with additive white gaussian noise (AWGN) $n(t)$ with zero mean, unit power spectral density and variance σ^2,

$$\begin{aligned} r(t) &= \sum_{k=1}^{K} y_k(t) + \sigma n(t) \\ &= \sum_{k=1}^{K} A_k \sum_{i=0}^{M-1} b_k(i) \sum_{l=0}^{L_k-1} g_{kl} s_k(t - iT - \tau_{kl}) + \sigma n(t) \end{aligned} \tag{6.68}$$

Single path, no delay, synchronous CDMA channel model
The signal $r(t)$ is received at the receiver and given as input to the bank of matched filters. We first assume a very simple channel for every user,

$$g_k(t) = \delta(t) \text{ ,for k=1,...,K} \tag{6.69}$$

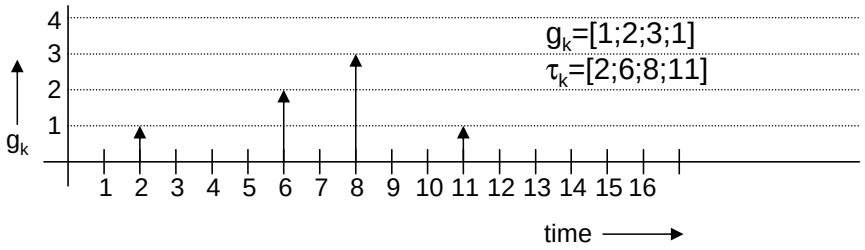


Figure 6.6: Example of a channel model with filter coefficients $g_k(t)$

we now can write the received signal $r(t)$ as,

$$r(t) = \sum_{k=1}^{K} A_k \sum_{i=0}^{M-1} b_k(i) s_k(t - iT) + \sigma n(t) \tag{6.70}$$

The output $\underline{y}_k(i)$ of the k^{th} matched filter for one bit b_k becomes,

$$\begin{aligned}
\underline{y}_k(i) &\triangleq \int_0^T s_k(t) r(t) dt, \text{ k=1,...,K} \\
&= \int_0^T s_k(t) \left[\sum_{j=1}^{K} A_j b_j(i) s_j(t) + \sigma n(t) \right] dt \\
&= \sum_{j=1}^{K} A_j b_j(i) \int_0^T s_j(t) s_k(t) + \sigma n_k \qquad (6.71) \\
&= \sum_{j=1}^{K} A_j b_j(i) \rho_{jk} + \sigma n_k \\
&= A_k b_k(i) + \sum_{j \neq k} A_j b_j(i) \rho_{jk} + \sigma n_k \qquad (6.72)
\end{aligned}$$

where,

$$\rho_{jk} = \int_0^T s_j(t) s_k(t) \tag{6.73}$$

The term $\sigma n(t)$ in equation 6.71 is not affected by the matched filter, because of convolution properties of white noise with unit spectral density. The following equations summarize these properties,

$$E[< n, s >] = 0$$

$$E[< n, s >^2] = \| s \|^2 = 1$$

Equation 6.72 can be written in vector notation. We define the following,

$$\underline{\mathbf{y}}(i) = [y_1, \ldots, y_K]^T$$

$$\underline{\mathbf{b}}(i) = [b_1, \ldots, b_K]^T$$

$$\mathbf{A} = diag\{A_1, \ldots, A_K\}$$

$$\mathbf{R} = \{\rho_{jk}\} = \begin{bmatrix} \rho_{11} & \rho_{21} & \cdots & \rho_{K1} \\ \rho_{12} & \rho_{22} & \cdots & \rho_{K2} \\ \vdots & \vdots & \vdots & \vdots \\ \rho_{1K} & \rho_{2K} & \cdots & \rho_{KK} \end{bmatrix}$$

$$\mathbf{n} = n_k$$

$\mathbf{n}$ is a zero-mean Gaussian random vector independent of $\mathbf{b}$ with covariance matrix equal to,

$$E[\mathbf{n}\mathbf{n}^T] = \sigma^2\mathbf{R}$$

We now can write equation 6.72 in vector notation,

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_K \end{bmatrix} = \begin{bmatrix} \rho_{11} & \rho_{21} & \cdots & \rho_{K1} \\ \rho_{12} & \rho_{22} & \cdots & \rho_{K2} \\ \vdots & \vdots & \vdots & \vdots \\ \rho_{1K} & \rho_{2K} & \cdots & \rho_{KK} \end{bmatrix} \begin{bmatrix} A_1 & 0 & \ldots & 0 \\ 0 & A_2 & \ldots & 0 \\ \vdots & \vdots & \ddots & 0 \\ 0 & 0 & 0 & A_K \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_K \end{bmatrix} + \begin{bmatrix} n_1 \\ n_2 \\ \vdots \\ n_K \end{bmatrix}$$

$$\underline{\mathbf{y}}(i) = \mathbf{R}\mathbf{A}\underline{\mathbf{b}}(i) + \sigma\mathbf{n} \tag{6.74}$$

By using the bank of matched filters, we replaced $y(t)$ with $\underline{\mathbf{y}}(i)$, this can be done without loss of optimality [31].

Soft Interference Cancellation

Now we have derived a sufficient model for our channel, we can start to derive the soft interference cancellation block. This is done by calculating soft estimates of the code bits of all the users from the a-priori information and then substracting these estimates from the received signal vector $\mathbf{y}$.

In order to calculate a soft estimate of a bit, we recall the log-likelihood ratio,

$$\Lambda_k(b_k) = \log \frac{P(b_k = +1)}{P(b_k = -1)} \tag{6.75}$$

Rewriting this equation gives an expression for $P(b_k = +1)$ and $P(b_k = -1)$. First for $P(b_k = +1)$,

$$\begin{aligned} e^{\Lambda_k(b_k)} &= \frac{P(b_k = +1)}{1 - P(b_k = +1)} \\ P(b_k = +1) &= \frac{e^{\Lambda_k(b_k)}}{1 + e^{\Lambda_k(b_k)}} \\ &= \frac{1}{1 + e^{-\Lambda_k(b_k)}} \end{aligned} \tag{6.76}$$

where in the first equation $P(b_k = -1)$ is replaced with $(1 - P(b_k = +1))$. The same can be done for $P(b_k = -1)$,

$$\begin{aligned} e_k^{\Lambda}(b_k) &= \frac{1 - P(b_k = -1)}{P(b_k = -1)} \\ P(b_k = -1) &= \frac{1}{1 + e^{\Lambda_k(b_k)}} \\ &= \frac{e^{-\Lambda_k(b_k)}}{1 + e^{-\Lambda_k(b_k)}} \end{aligned} \tag{6.77}$$

From equation 6.76 and 6.77 we can derive a more general equation for $P(b_k = b_j)$ $(b_j \in \{+1, -1\})$,

$$P(b_k = b_j) = \frac{e^{b_j \Lambda_k(b_j)}}{1 + e^{b_j \Lambda_k(b_j)}} \tag{6.78}$$

With some manipulations and the use of the following equations,

$$\sinh(z) = \frac{1}{2}(e^z - e^{-z}) \tag{6.79}$$

$$\cosh(z) = \frac{1}{2}(e^z + e^{-z}) \tag{6.80}$$

$$\begin{aligned} \tanh(z) &= \frac{\sinh(z)}{\cosh(z)} \\ &= \frac{e^z - e^{-z}}{e^z + e^{-z}} \\ &= \frac{e^{2z} - 1}{e^{2z} + 1} \end{aligned} \tag{6.81}$$

we can derive for equation 6.78,

$$\begin{aligned} P(b_k) &= \frac{e^{b_k \Lambda_k(b_k)}}{1 + e^{b_k \Lambda_k(b_k)}} \frac{e^{-\frac{1}{2} b_k \Lambda_k(b_k)}}{e^{-\frac{1}{2} b_k \Lambda_k(b_k)}} \\ &= \frac{e^{\frac{1}{2} b_k \Lambda_k(b_k)}}{e^{-\frac{1}{2} b_k \Lambda_k(b_k)} + e^{\frac{1}{2} b_k \Lambda_k(b_k)}} \\ &= \frac{\frac{1}{2}(b_k + 1) e^{\frac{1}{2}\Lambda_k(b_k)} + \frac{1}{2}(1 - b_k) e^{-\frac{1}{2}\Lambda_k(b_k)}}{e^{-\frac{1}{2}\Lambda_k(b_k)} + e^{\frac{1}{2}\Lambda_k(b_k)}} \\ &= \frac{\frac{1}{2}\left(e^{\frac{1}{2}\Lambda_k(b_k)} + e^{-\frac{1}{2}\Lambda_k(b_k)}\right) + b_k \frac{1}{2}\left(e^{\frac{1}{2}\Lambda_k(b_k)} - e^{-\frac{1}{2}\Lambda_k(b_k)}\right)}{e^{-\frac{1}{2}\Lambda_k(b_k)} + e^{\frac{1}{2}\Lambda_k(b_k)}} \\ &= \frac{\cosh\left[\frac{1}{2}\Lambda_k(b_k)\right] + \sinh\left[\frac{1}{2}\Lambda_k(b_k)\right]}{2\cosh\left[\frac{1}{2}\Lambda_k(b_k)\right]} \\ &= \frac{\cosh\left[\frac{1}{2}\Lambda_k(b_k)\right] + \cosh\left[\frac{1}{2}\Lambda_k(b_k)\right]\tanh\left[\frac{1}{2}\Lambda_k(b_k)\right]}{2\cosh\left[\frac{1}{2}\Lambda_k(b_k)\right]} \\ &= \frac{1}{2}\left[1 + b_k \tanh\left(\frac{1}{2}\Lambda_k(b_k)\right)\right] \end{aligned} \tag{6.82}$$

In equation 6.82, equality 3 can be written because $b_k \in \{-1, +1\}$. Equality 5 and 6 follows from equations 6.79, 6.80 and 6.81.

With the expression for the bit probability, a soft estimate $\tilde{b}_j(i)$ of code bit i can be made for all the users,

$$\begin{aligned}
\tilde{b}_j(i) &\triangleq \sum_{b_j \in \{-1,+1\}} b_j P(b_j) \\
&= \sum_{b_j \in \{-1,+1\}} \frac{b_j}{2}\left[1 + b_j \tanh\left(\frac{1}{2}\Lambda_j[b_j(i)]\right)\right] \\
&= \frac{1}{2}\left[1 + \tanh\left(\frac{1}{2}\Lambda_j[b_j(i)]\right)\right] - \frac{1}{2}\left[1 - \tanh\left(\frac{1}{2}\Lambda_j[b_j(i)]\right)\right] \\
&= \frac{1}{2} + \frac{1}{2}\tanh\left(\frac{1}{2}\Lambda_j[b_j(i)]\right) - \frac{1}{2} + \frac{1}{2}\tanh\left(\frac{1}{2}\Lambda_j[b_j(i)]\right) \\
&= \tanh\left(\frac{1}{2}\Lambda_j[b_j(i)]\right) \qquad (6.83)
\end{aligned}$$

We now order the soft bit estimates of code bit i of all users into a vector,

$$\tilde{\mathbf{b}}(i) = \left[\tilde{b}_1(i), \ldots, \tilde{b}_K(i)\right]^T \qquad (6.84)$$

From this vector we derive a vector $\tilde{\mathbf{b}}^k(i)$ with the k^{th} vector set to zero,

$$\tilde{\mathbf{b}}^k(i) = \left[\tilde{b}_1(i), \ldots, \tilde{b}_{k-1}(i), 0, \tilde{b}_{k+1}(i), \ldots, \tilde{b}_K(i)\right]^T \qquad (6.85)$$

The soft interference calculation is now performed on the output of every matched filter in 6.74. This is done by substracting the cancellation vector in 6.85 from the matched filter output, to obtain the soft interference cancellation output $\underline{\mathbf{y}}_k(i)$,

$$\begin{aligned}
\underline{\mathbf{y}}_k(i) &\triangleq \underline{\mathbf{y}}(i) - \mathbf{RA}\tilde{\mathbf{b}}^k(i) \\
&= \mathbf{RA}[\mathbf{b}(i) - \tilde{\mathbf{b}}^k(i)] + \sigma\mathbf{n} \qquad (6.86)
\end{aligned}$$

Minimum Mean Square Error filtering

To suppress resulting interference in the output $\underline{\mathbf{y}}_k(i)$ of the soft interference cancellation stage, an instantaneous MMSE filter is applied. This filter is constructed to minimize the mean-square error between the i^{th} code bit of user k, $b_k(i)$. The output of this filter is,

$$z_k(i) = \underline{\omega}_k(i)^T \underline{\mathbf{y}}_k(i) \qquad (6.87)$$

where $\underline{\omega}_k(i) \in \Re^K$ are the filter taps of user k,

$$\underline{\omega}_k(i) = \left[\omega^1(i), \ldots, \omega^K(i)\right] \tag{6.88}$$

To minimize the mean-square error between the filter output and the code bit $b_k(i)$, $\underline{\omega}_k(i)$ must satisfy the following equation,

$$\begin{aligned} \underline{\omega}_k(i) &= \arg\min_{\underline{\omega}\in\Re^K} E\left\{\left[b_k(i) - \underline{\omega}^T \underline{\mathbf{y}}_k(i)\right]^2\right\} \\ &= \arg\min_{\underline{\omega}\in\Re^K} \underline{\omega}^T E\left\{\underline{\mathbf{y}}_k(i)\underline{\mathbf{y}}_k(i)^T\right\}\underline{\omega} \\ &\quad -2\underline{\omega}^T E\left\{b_k(i)\underline{\mathbf{y}}_k(i)\right\} \end{aligned} \tag{6.89}$$

To minimize the term in equation 6.89, we differentiate to $\underline{\omega}$,

$$\frac{\partial \underline{\omega}_k(i)}{\partial \underline{\omega}} = 2\underline{\omega}^T E\left\{\underline{\mathbf{y}}_k(i)\underline{\mathbf{y}}_k(i)^T\right\} - 2E\left\{b_k(i)\underline{\mathbf{y}}_k(i)\right\} \tag{6.90}$$

and set the equation to zero,

$$2\underline{\omega}^T E\left\{\underline{\mathbf{y}}_k(i)\underline{\mathbf{y}}_k(i)^T\right\} - 2E\left\{b_k(i)\underline{\mathbf{y}}_k(i)\right\} = 0$$

$$\begin{aligned} \underline{\omega}_k(i) &= \frac{E\left\{b_k(i)\underline{\mathbf{y}}_k(i)\right\}}{E\left\{\underline{\mathbf{y}}_k(i)\underline{\mathbf{y}}_k(i)^T\right\}} \\ &= E\left\{b_k(i)\underline{\mathbf{y}}_k(i)\right\} E\left\{\underline{\mathbf{y}}_k(i)\underline{\mathbf{y}}_k(i)^T\right\}^{-1} \end{aligned} \tag{6.91}$$

We use equation 6.86 to find an expression for the expectation terms in equation 6.91,

$$E\left\{\underline{\mathbf{y}}_k(i)\underline{\mathbf{y}}_k(i)^T\right\} = \mathbf{RA}cov\left\{\mathbf{b}(i) - \tilde{\mathbf{b}}^k(i)\right\}\mathbf{AR} + \sigma^2\mathbf{R} \tag{6.92}$$

where,

$$\begin{aligned} cov\left\{\mathbf{b}(i) - \tilde{\mathbf{b}}^k(i)\right\} &= diag[var\{b_1(i)\}, \ldots, var\{b_{k-1}(i)\}, 1 \\ &\quad , var\{b_{k+1}(i)\}, \ldots, var\{b_K(i)\}] \end{aligned} \tag{6.93}$$

and

$$\begin{aligned} var\{b_k(i) &= E\{b_k(i)^2\} - [E\{b_k(i)\}]^2 \\ &= 1 - \tilde{\mathbf{b}}_k(i) \end{aligned} \tag{6.94}$$

thus,

$$\begin{aligned} cov\left\{\mathbf{b}(i) - \tilde{\mathbf{b}}^k(i)\right\} &= diag[1 - \tilde{\mathbf{b}}_1(i), \ldots, 1 - \tilde{\mathbf{b}}_{k-1}(i), 1 \\ &\quad , 1 - \tilde{\mathbf{b}}_{k+1}(i), \ldots, 1 - \tilde{\mathbf{b}}_K(i)] \end{aligned} \tag{6.95}$$

We define the following term and rewrite it,

$$\begin{aligned}\underline{V}_k(i) &\triangleq \mathbf{A}cov\left\{\mathbf{b}(i)-\tilde{\mathbf{b}}^k(i)\right\}\mathbf{A} \\ &= A_k^2\underline{\mathbf{e}}_j\underline{\mathbf{e}}_k^T+\sum_{j\neq k}A_j^2[1-\tilde{\mathbf{b}}_j(i)^2]\underline{\mathbf{e}}_j\underline{\mathbf{e}}_k^T \end{aligned} \quad (6.96)$$

where $\underline{\mathbf{e}}_k$ is a column-vector with all zeros except the k^{th} element, which is 1. The term $\underline{\mathbf{e}}_j\underline{\mathbf{e}}_k^T$ gives a matrix of dimension KxK with all zeros, except the k^{th} element on the diagonal, which is a 1. The term $A_k^2\underline{\mathbf{e}}_j\underline{\mathbf{e}}_k^T$ is outside the summation because in the matrix $cov\left\{\mathbf{b}(i)-\tilde{\mathbf{b}}^k(i)\right\}$ the k^{th} element on the diagonal is a 1. For the other expectation term in equation 6.91,

$$\begin{aligned}E\left\{b_k(i)\underline{\mathbf{y}}_k(i)\right\} &= \mathbf{RA}E\left\{b_k(i)\left[\mathbf{b}(i)-\tilde{\mathbf{b}}^k(i)\right]\right\} \\ &= \mathbf{RA}\underline{\mathbf{e}}_k \end{aligned} \quad (6.97)$$

The last expression in equation 6.97 is due to the expectancy of the term $\mathbf{b}(i)-\tilde{\mathbf{b}}^k(i)$, which is a vector with all zeros, except for the k^{th} position, where the expected value of $b_k(i)\in\{-1,+1\}$ should be. This vector is exactly $\underline{\mathbf{e}}_k$.

Now we can substitute equations 6.92, 6.96 and 6.97 into equation 6.91,

$$\underline{\omega}_k(i) = \mathbf{RA}\underline{\mathbf{e}}_k\left[\mathbf{R}\underline{V}_k(i)\mathbf{R}+\sigma^2\mathbf{R}\right]^{-1} \quad (6.98)$$

$$= \mathbf{A}_k\mathbf{R}^{-1}\left[\underline{V}_k(i)+\sigma^2\mathbf{R}^{-1}\right]^{-1}\underline{\mathbf{e}}_k \quad (6.99)$$

Now substituting equation 6.98 and 6.86 into equation 6.87, gives us an expresion for the MMSE filter output,

$$\begin{aligned}z_k(i) &= \mathbf{A}_k\mathbf{R}^{-1}\left[\underline{V}_k(i)+\sigma^2\mathbf{R}^{-1}\right]^{-1}\underline{\mathbf{e}}_k\left[\underline{\mathbf{y}}(i)-\mathbf{RA}\tilde{\mathbf{b}}^k(i)\right] \\ &= \mathbf{A}_k\underline{\mathbf{e}}_k\left[\underline{V}_k(i)+\sigma^2\mathbf{R}^{-1}\right]^{-1}\left[\underline{\mathbf{R}}^{-1}\underline{\mathbf{y}}(i)-\mathbf{A}\tilde{\mathbf{b}}^k(i)\right] \end{aligned} \quad (6.100)$$

In equation 6.100 the term $\underline{\mathbf{R}}^{-1}\underline{\mathbf{y}}(i)$ is the output of a decorrelating multiuser detector.

Log-likelihood calculation

From the output of the MMSE filter, the LLR of the code bit i of every user can be calculated. In [19] it is shown that in the output of a MMSE filter, the residual interference and noise, is well approximated by a Gaussian distribution. So we can see the output $z_k(i)$ of the MMSE filter as an additive white Gaussian noise channel with input bit $b_k(i)$,

$$z_k(i)=\mu_k(i)b_k(i)+\eta_k(i) \quad (6.101)$$

where $\mu_k(i)$ is the amplitude at the output of the k^{th} user's signal and $\eta_k(i) \sim (0, \nu_k^2(i))$ is the Gaussian noise at the output with variance $\nu_k^2(i)$. When we rewrite the formula to an expression for $\eta_k(i)$, we get,

$$\eta_k(i) = z_k(i) - \mu_k(i)b_k(i) \tag{6.102}$$

Since $\eta_k(i)$ is gaussian distributed the term right of the equal sign is also gaussian distributed. For a gaussian distribution the probability function is known. In the equation above the term $b_k(i)$ enables us to write the probability function as a conditional probability,

$$P[z_k(i)|b_k(i)] = \frac{1}{\sigma\sqrt{2\pi}} \exp\left\{\frac{-[z_k(i) - \mu_k(i)b_k(i)]^2}{2\nu_k^2(i)}\right\} \tag{6.103}$$

With the conditional probability we can write an expression for the log-likelihood ratio $\Lambda_{out}[b_k(i)]$ from the output of the MMSE filter,

$$\begin{aligned}
\Lambda_{out}[b_k(i)] &= \log \frac{P[z_k(i)|b_k(i) = +1]}{P[z_k(i)|b_k(i) = -1]} \\
&= \log \frac{\exp\left\{\frac{-[z_k(i)-\mu_k(i)]^2}{2\nu_k^2(i)}\right\}}{\exp\left\{\frac{-[z_k(i)+\mu_k(i)]^2}{2\nu_k^2(i)}\right\}} \\
&= \log\left[\exp\left\{\frac{-[z_k(i) - \mu_k(i)]^2}{2\nu_k^2(i)}\right\}\right] + \log\left[\exp\left\{\frac{[z_k(i) + \mu_k(i)]^2}{2\nu_k^2(i)}\right\}\right] \\
&= \frac{-[z_k(i) - \mu_k(i)]^2}{2\nu_k^2(i)} + \frac{[z_k(i) + \mu_k(i)]^2}{2\nu_k^2(i)} \\
&= \frac{2z_k(i)\mu_k(i)}{\nu_k^2(i)}
\end{aligned} \tag{6.104}$$

To calculate $\Lambda_{out}[b_k(i)]$ we need an expression for $\mu_k(i)$ and $\nu_k^2(i)$. $\mu_k(i)$ can be found by using equation 6.87 and 6.86,

$$\begin{aligned}
\mu_k(i) &= E\{z_k(i)b_k(i)\} \\
&= E\left\{\mathbf{A}_k\underline{\mathbf{e}}_k^T\left[\underline{V}_k(i) + \sigma^2\mathbf{R}^{-1}\right]^{-1}\left[\underline{\mathbf{R}}^{-1}\left(\mathbf{RA}[\mathbf{b}(i) - \tilde{\mathbf{b}}^k(i)] + \sigma\mathbf{n}\right)\right]b_k(i)\right\} \\
&= \mathbf{A}_k\underline{\mathbf{e}}_k^T\left[\underline{V}_k(i) + \sigma^2\mathbf{R}^{-1}\right]^{-1} E\left\{\underline{\mathbf{A}}b_k(i)[\mathbf{b}(i) - \tilde{\mathbf{b}}^k(i)] + b_k(i)\mathbf{R}^{-1}\sigma\mathbf{n}\right\} \\
&= \mathbf{A}_k^2\underline{\mathbf{e}}_k^T\left[\underline{V}_k(i) + \sigma^2\mathbf{R}^{-1}\right]^{-1}\underline{\mathbf{e}}_k \\
&= \mathbf{A}_k^2\left[\left[\underline{V}_k(i) + \sigma^2\mathbf{R}^{-1}\right]^{-1}\right]_{kk}
\end{aligned} \tag{6.105}$$

The expectancy term in the third step of equation 6.105 contains two terms who are added. The first term was also used in equation 6.97 and the same steps are followed here. The second term is zero, because the expectancy of

the noise term is zero, as was given by definition. For $\nu_k^2(i)$ we find,

$$\begin{aligned}
\nu_k^2(i) &= var\{z_k(i)\} \\
&= E\{z_k(i)^2\} - \mu_k(i)^2 \\
&= \underline{\omega}_k(i) E\{y_k(i) y_k(i)^T\} \underline{\omega}_k(i) - \mu_k(i)^2 \\
&= \mathbf{A}_k^2 \underline{\mathbf{e}}_k^T \left[\underline{V}_k(i) + \sigma^2 \mathbf{R}^{-1}\right]^{-1} \underline{\mathbf{e}}_k - \mu_k(i)^2 \\
&= \mu_k(i) - \mu_k(i)^2
\end{aligned} \tag{6.106}$$

We can use equation 6.106 in equation 6.104,

$$\begin{aligned}
\Lambda_{out}[b_k(i)] &= \frac{2 z_k(i) \mu_k(i)}{\mu_k(i) - \mu_k(i)^2} \\
&= \frac{2 z_k(i)}{1 - \mu_k(i)}
\end{aligned} \tag{6.107}$$

MMSE filter output special cases

The MMSE filter output in 6.100 can be viewed in the context of some special cases, where there is no a-priori input and when there is perfect a-priori input. No a-priori information means that $\Lambda_j[b_j(i)] = 0$. In this case $\tilde{\mathbf{b}}^k(i) = \underline{0}$ and thus $\underline{V}_k(i) = \mathbf{A}^2$. Then 6.100 becomes,

$$\begin{aligned}
z_k(i) &= \mathbf{A}_k \underline{\mathbf{e}}_k \left[\mathbf{A}^2 + \sigma^2 \mathbf{R}^{-1}\right]^{-1} \left[\underline{\mathbf{R}}^{-1} \underline{\mathbf{y}}(i)\right] \\
&= \mathbf{A}_k \underline{\mathbf{e}}_k \left[\mathbf{R} + \sigma^2 \mathbf{A}^{-2}\right]^{-1} \underline{\mathbf{y}}(i)
\end{aligned} \tag{6.108}$$

which is simply the output of a lineair MMSE filter [31].

The other case, when there is perfect a-priori information, results in $\Lambda_j[b_j(i)] = \pm\infty$. In this case $\tilde{\mathbf{b}}^k(i) = [b_1(i), \ldots, b_{k-1}(i), 0, b_{k+1}(i), \ldots, b_K(i)]$, so there are perfect estimates of the bits $b_k(i)$ of all the other users. This results in $\underline{V}_k(i) = \mathbf{A}_k^2 \underline{\mathbf{e}}_k \underline{\mathbf{e}}_k^T$. Substituting these results in 6.98,

$$\begin{aligned}
\underline{\omega}_k(i) &= \mathbf{A}_k \mathbf{R}^{-1} \left[\mathbf{A}_k^2 \underline{\mathbf{e}}_k \underline{\mathbf{e}}_k^T + \sigma^2 \mathbf{R}^{-1}\right]^{-1} \underline{\mathbf{e}}_k \\
&= \frac{\mathbf{A}_k}{\mathbf{A}_k^2 + \sigma^2} \underline{\mathbf{e}}_k
\end{aligned} \tag{6.109}$$

which then results in the output $z_k(i)$,

$$\begin{aligned}
z_k(i) &= \underline{\omega}_k(i)^T \underline{\mathbf{y}}_k(i) \\
&= \frac{\mathbf{A}_k}{\mathbf{A}_k^2 + \sigma^2} \underline{\mathbf{e}}_k \underline{\mathbf{y}}_k(i) \\
&= \frac{\mathbf{A}_k}{\mathbf{A}_k^2 + \sigma^2} \underline{\mathbf{e}}_k \left[y_k(i) - \sum_{j \neq k} \mathbf{A}_j \rho_{kj} b_j(i) \right]
\end{aligned} \tag{6.110}$$

we see that in this case, the output of the MMSE filter is just a scaled version of the k^{th} user's matched filter output after perfect soft interference cancellation. The soft interference calculation is perfect, since perfect estimates of all other users' bits are available.

6.2.2 Soft Cancellation MUD Complexity

The computational complexity of the multi user detector is mostly determined by outer and inner vector products. The computation of $z_k(i)$ in equation 6.100 involves two vector inner products, one for computating the decorrelating output and one in computing the final $z_k(i)$ output. In [20] the a recursive method is offered for calculating the inverse of a matrix. This methode has two calculations which both involve K vector outer products. So with the recursive methode, the dominant computation per user per symbol involves two vector inner and two vector outer products. The complexity of the SISO multi user detector is then $O(K^2)$.

6.3 Conclusions

In this chapter the main building blocks for the turbo architectures in chapters 3 and 4 were introduced and described.

In section 6.1 three convolutional code decoders were described; the MAP decoder, the Max-Log-MAP decoder and the Log-MAP decoder. These decoders are used in the PCCC and SCCC turbo architecture of section 3.3. They are also used in the turbo multiuser detectors of section 4.1 and section 4.2. Performance differences between the decoders were investigated in [23] and are not repeated in this thesis.

In section 6.2 a SISO turbo multiuser detector was introduced and described. It was developed by X. Wang and V. Poor in [20] and is used in the turbo multiuser architectures of section 4.1 and section 4.2.

In chapter 7 the building blocks of this chapter will be used to perform simulations on the PCCC, SCCC and turbo multiuser detection architectures.

7

Simulations

In this chapter simulations are performed on the SCCC, PCCC and turbo multiuser detection architectures presented in previous chapters.

The SCCC and PCCC architectures are presented in chapter 3. The used decoder is the Log-MAP decoder. BER charts and EXIT charts of these architectures are made, to compare them with simulations done in literature. BER charts of these architectures are simulated in [4]. EXIT charts of similar turbo code architectures are simulated in [29] and [28]. The simulations in this chapter will be done with the same settings as in [4]. When the BER charts correspond to the BER charts in [4], the implementation of the Log-MAP decoder is concluded to be correct. When the EXIT charts in this chapter are similar to the EXIT charts in [29] and [27], the tool for creating EXIT charts is considered to be implemented correctly.

The turbo multiuser detector architecture that is simulated, is presented in section 4.1. This architecture was also simulated by X.Wang and V.Poor in [20]. The simulations in this chapter will be done with the same settings as in [20] to be able to compare the results. When the results are the same, the turbo multiuser detector architecture is concluded to be implemented correctly.

To answer the first question of this thesis, given in section 1.4, simulation results of non-turbo multiuser detectors presented by R.M. Buehrer in [7] are re-printed in this thesis. These results are compared to the BER chart of the turbo multiuser detection architecture.

To answer the second question, an EXIT chart is made for the turbo multiuser detection architecture. With this EXIT chart, the convergence behavior of the architecture is analyzed.

In the simulations of the BER chart for the turbo multiuser detection architecture, the a-posteriori information seems to give better results than the extrinsic information. To analyze the difference between iterating extrinsic and a-posteriori information in a turbo architecture, transfer charts are made for the mutual information of extrinsic and a-posteriori information

from a Log-MAP decoder.

7.1 Implementation

The turbo architectures and their building blocks, that will be simulated, are implemented in C++. The DSP and communications library IT++ [9] is used to implement the architectures. From this library, predefined signal processing functions, like a spreader, an AWGN channel and pseudo-random interleaver, are used. The simulation software is described in the software manual that accompanies this document and with comments given in the source code. The document and the source code can be found on the accompanying cdrom. The simulations are performed on a system with an AMD 2.8Ghz processor with 2GB of memory. Simulation times varied from 15 minutes to one hour.

The E_b/N_0 ratio, used in the following sections as a parameter for the simulations, is always the signal to noise ratio of the databits. Thus E_b is the signal energy of the databits. The signal energy E_c of the codebits in the simulations is always 1 and is related to E_b as $E_c = R \cdot E_b$, where R is the code rate. The following procedure is used to calculate the variance σ_n^2 of the channel from (E_b/N_0)[dB], where (E_b/N_0)[dB] is E_b/N_0 in dB's,

$$\begin{aligned} E_b/N_0 &= 10^{(\frac{1}{10}(E_b/N_0[dB]))} \\ N_0 &= E_b/10^{(\frac{1}{10}E_b/N_0)} \\ &= E_c/R \cdot 10^{(\frac{1}{10}E_b/N_0)} \\ &= 1/R \cdot 10^{(\frac{1}{10}E_b/N_0)} \end{aligned}$$

and thus with BPSK modulation,

$$\sigma_n^2 = N_0/2 \tag{7.1}$$

When an EXIT chart is made, care has to be taken with the calculation of the noise variance of the channel. When, for example, the EXIT chart of the PCCC architecture of section 3.3 on page 26 is made, we take each decoder out of the architecture and use the simulation setup of figure 5.3 to make the transfer chart of that decoder. In the simulation setup of figure 5.3 the encoder of section 3.2 on page 25 now seems to have a code rate $R = 1/2$. However in the PCCC architecture, this decoder received channelbits which came from a channel, whose variance was calculated according to a code rate $R = 1/3$. So the channel variance in a transfer-chart simulation-setup, also has to be calculated with the code rate R of the turbo architecture where the decoder came from.

7.2 Verification Simulations

In this section the BER charts and EXIT charts for the SCCC and PCCC architecture are created. At the end of the section, conclusions are drawn based on these charts.

7.2.1 PCCC architecture BER chart

In this section the PCCC architecture discussed in section 3.2 and section 3.3 is simulated.

The PCCC encoder that is used for these simulations is shown in figure 3.6 on page 26. Its settings are shown in table 7.1. The PCCC decoder that

Encoder 1	type:	RSC
	rate:	1/2
	polynomial:	$\left[1, \frac{1+D+D^3+D^4}{1+D^3+D^4}\right]$
	ν:	5
	final state:	truncated
Encoder 2	type:	RSC
	rate:	1/2
	polynomial:	$\left[1, \frac{1+D+D^3+D^4}{1+D^3+D^4}\right]$
	ν:	5
	final state:	truncated
Interleaver	type:	pseudo-random
	size:	16384 bits
Multiplexer	no-puncturing	
Output	systematic,parity1,parity2	
PCCC encoder rate	1/3	
Channel	AWGN	
Iterations	15	

Table 7.1: Settings for the PCCC encoder in figure 3.6 used for simulations

is used is shown in figure 3.7 on page 27. The decoders are implemented as Log-MAP decoders of section 6.1.3. Simulations were performed on 100 blocks of 16384 databits. For each block the encoders are reset to their initial state 0. For each simulation all settings remain constant, except for the E_b/N_0 values of the channel, which serve as a parameter for the simulations.

The results of these simulations are shown in figure 7.1 on page 85, where the BER is plotted against the number of iterations and against the signal to noise ratio in [dB]. Because of time limitations, the simulations were not done until for every iteration a minimum number of bit errors was detected.

This can be seen in the simulation results, in the form of curves which do not continue to low BER values.

In figure 7.2 on page 86 the same simulation is shown as in figure 7.1 on page 85, only in this simulation a-posteriori information is iterated between the decoders, instead of extrinsic information.

7.2.2 PCCC architecture EXIT chart

In this section the EXIT chart of the PCCC architecture is presented. The settings of the decoders are given in table 7.1. The transfer charts are made with the simulation setup of figure 5.3 in section 5.2 by using 10^6 input databits. It should be noted that the systematic bits of one of the RSC encoders, in the PCCC encoder of figure 3.6, is punctured. The effective code rate of the RSC encoders therefor becomes $R = 1/3$. So when creating the transfer chart of a decoder in a PCCC architecture, every second systematic bit in the received channelword, should be put to zero. In this way the decoder gets the same amount of information as it would have gotten in the PCCC architecture. The transfer charts are made for $E_b/N_0 = -0.5\,\text{dB}$, $E_b/N_0 = 0.2\,\text{dB}$, $E_b/N_0 = 0.3\,\text{dB}$, $E_b/N_0 = 0.5\,\text{dB}$ and $E_b/N_0 = 1.0\,\text{dB}$. The EXIT chart is plotted in figure 7.3 on page 87. In this figure the decoding trajectory is drawn for $E_b/N_0 = 1.0\,\text{dB}$.

7.2.3 SCCC architecture BER chart

In this section the SCCC architecture discussed in section 3.2 and section 3.3 is simulated.

The SCCC encoder that is used for these simulations is shown in figure 3.5 on page 25. Its settings are shown in table 7.2. The SCCC decoder that is used is shown in figure 3.8 on page 27. The decoders are implemented as Log-MAP decoders of section 6.1.3. Simulations were performed on 100 blocks of 16384 databits. For each block the encoders are reset to their initial state 0. For each simulation all settings remain constant, except for the E_b/N_0 values of the channel, which serve as a parameter for the simulations.

The results of these simulations are shown in figure 7.4 on page 88, where the BER is plotted against the number of iterations and against the signal to noise ratio. Because of time limitations, the simulations were not done until for every iteration a minimum number of bit errors was detected. This can be seen in the simulation results, in the form of curves which do not continue to low BER values.

7.2.4 SCCC architecture EXIT chart

In this section the EXIT chart of the SCCC architecture is presented. The settings of the decoders are given in table 7.2. The transfer chart of the

Inner encoder	type:	RSC
	rate:	1/2
	polynomial:	$\left[1, \frac{1+D+D^3}{1+D}\right]$
	ν:	5
	final state:	truncated
Outer encoder	type:	SC
	rate:	1/2
	polynomial:	$\left[1, 1+D+D^3\right]$
	ν:	5
	final state:	truncated
Interleaver	type:	pseudo-random
	size:	16384 bits
SCCC encoder rate	1/4	
Channel	AWGN	
Iterations	15	

Table 7.2: Settings for the SCCC encoder in figure 3.5 used for simulations

inner decoder is made with the simulation setup of figure 5.3 in section 5.2 and the transfer chart of the outer decoder is made with the simulation setup of figure 5.4 in the same section. The transfer charts are made with 10^6 input databits. Since no puncturing is used in the SCCC architecture, no channelbits need to be put to zero, as was done with the EXIT chart of the PCCC architecture in section 7.2.2. The transfer charts are made for $E_b/N_0 = -0.5\,\text{dB}$, $E_b/N_0 = -0.3\,\text{dB}$, $E_b/N_0 = -0.1\,\text{dB}$ and $E_b/N_0 = 0.5\,\text{dB}$. The EXIT chart is plotted in figure 7.5 on page 89. In this figure the decoding trajectory for $E_b/N_0 = 0.5\,\text{dB}$ is drawn.

7.2.5 Conclusions

PCCC architecture BER chart

The simulations of the PCCC architecture in figure 7.1 on page 85 can be compared with the results in figure 6 of [4], since the simulation setup is exactly the same. In [4] a bit error probability of 10^{-4} is reached after 5 iterations with a signal to noise ratio $E_b/N_0 = 0.45\,\text{dB}$. In figure 7.1 a BER of 10^{-4} after 5 iteration is reached when $E_b/N_0 = 0.83\,\text{dB}$. Thus, the PCCC architecture in this thesis performs $\approx 0.40\,\text{dB}$ worse than in [4]. A possible explanation for this performance difference is that in [4] an '*Additive Sliding Window SISO (ASW-SISO)*' algorithm is used. However, the ASW-SISO algorithm should be sub-optimal compared to the Log-MAP algorithm of this thesis, since it operates on shorter block lengths. Because of this sub-optimality, it is expected that the ASW-SISO would perform worse than

the Log-MAP. A more detailed investigation of the ASW-SISO algorithm is needed, to explain the difference between the simulation results in this thesis and in [4].

In chapter 5 a bit-error-floor for a PCCC architecture was introduced. This error-floor is encountered in literature on PCCC architectures, like figure 8 in [4]. The simulations in this thesis do not show such an error floor. Simulations with more blocks of databits were performed to see if the error-floor would occur at lower BER. Even then the error-floor was not observed. These simulations are not reproduced in this thesis. Why there is no error-floor observed, is not known.

In figure 7.2 the simulations show that using a-posteriori information to iterate between the decoders, gives a lower performance in terms of BER, than when extrinsic information is used. So in a PCCC architecture, iterating extrinsic information gives better results in term of BER than iterating a-posteriori information.

PCCC architecture EXIT chart

The EXIT chart of the PCCC architecture used in this thesis, is given in figure 7.3 on page 87 for different E_b/N_0 values. Similar simulations have been done by S. ten Brink in [29]. The simulations in [29] are done for a PCCC architecture with code rate $R = 1/2$, while the PCCC architecture used in this thesis has a code rate $R = 1/3$. The EXIT chart for the PCCC architecture in this thesis, just opens up at $E_b/N_0 = 0.2\,\text{dB}$, however the opening is very narrow so iterating to a low BER is possible, but will take a lot of iterations. In figure 6 of [29] the EXIT chart just opens up at $E_b/N_0 = 0.7\,\text{dB}$. So the difference in code rate between the two architectures seems to result in a $0.5\,\text{dB}$ worse performance for the PCCC architecture in [29].

The iteration trajectory is drawn in figure 7.3 for $E_b/N_0 = 1.0\,\text{dB}$. After 4 iterations the trajectory has reached $I_E \approx 1$. In the BER chart of figure 7.1 on page 85 after 4 iterations at $E_b/N_0 = 1.0\,\text{dB}$, a BER of $\approx 2 \cdot 10^{-5}$ is reached. These findings correspond, since a low BER implies a large mutual information I_E.

SCCC architecture BER chart

The results of the simulations of the SCCC architecture in figure 7.4 on page 88 can not directly be compared with the results of the simulations of the SCCC architecture in figure 7 of [4]. In [4] the outer-encoder is non-systematic. The outer encoder of the SCCC architecture in this thesis is systematic. Although a direct comparison is hard to make, we see that the results differ only tenths of dB's. For example, in figure 7.4 at $E_b/N_0 = 0.30\,\text{dB}$ after almost 10 iterations a BER of 10^{-5} is reached. In [4] this BER

is reached after 10 iterations at $E_b/N_0 = 0.10\,\text{dB}$. At $E_b/N_0 = 0.00\,\text{dB}$, in [4] after 13 iterations a BER of 10^{-5} is reached. In figure 7.4 after 13 iterations at $E_b/N_0 = 0.03\,\text{dB}$ a BER of $\approx 2 \cdot 10^{-3}$ is reached. Thus, the SCCC architecture in [4] outperforms the SCCC architecture in this thesis. A performance difference was also observed for the PCCC architecture and it is assumed the same reasons, namely the usage of the ASW-SISO algorithm in [4], can account for it.

In figure 7.6 on page 90, E_b/N_0 curves are drawn for the SCCC and PCCC architectures of figures 7.1 and 7.4 in one plot. Figure 8 in [4] gives the same figure. The SCCC and PCCC architecture used in figure 8 of [4] have the same code rate and complexity, the SCCC and PCCC architecture in this thesis do not have the same rate and complexity. This makes a direct comparison of the SCCC and PCCC architectures in this thesis not really possible. We see that at $E_b/N_0 > 1\,\text{dB}$ after 3 iterations the PCCC architecture outperforms the SCCC architecture, although the PCCC encoder is rate 1/3 and the SCCC encoder is rate 1/4. In section 2.1 it was concluded that an encoder with a large code rate can produce more powerful codewords, however in the above situation the PCCC architecture with its lower code rate outperforms the SCCC architecture. At iteration 10 the SCCC architecture always outperforms the PCCC architecture for a BER as low as 10^{-5}.

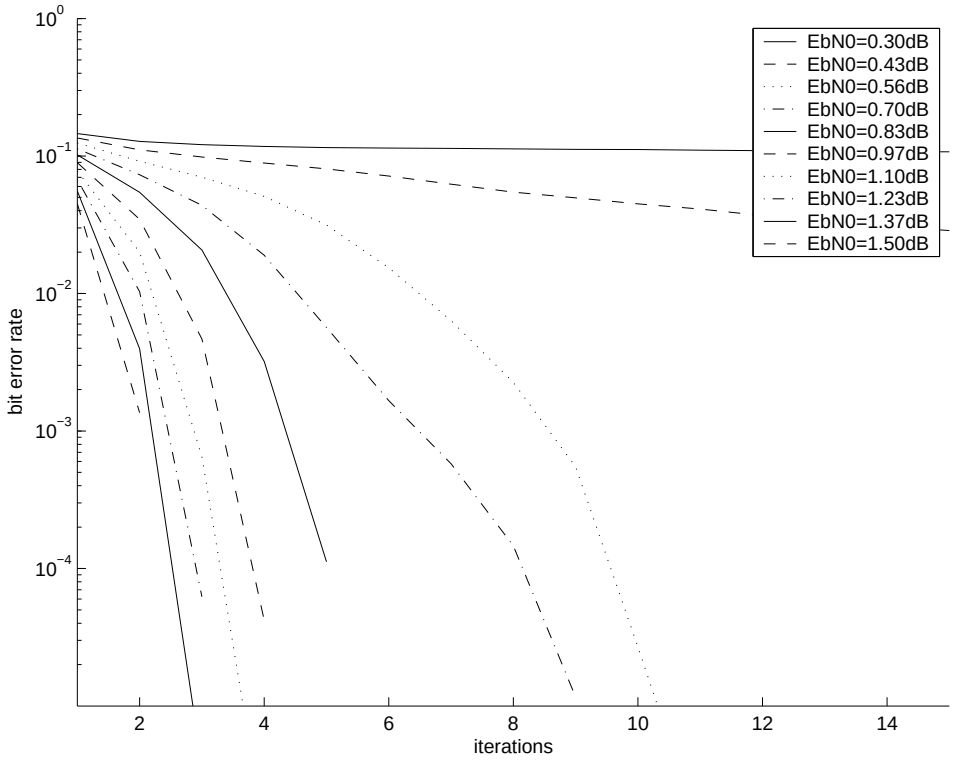
SCCC architecture EXIT chart

The EXIT chart of the SCCC architecture used in this thesis, is given in figure 7.5 on page 89 for different E_b/N_0 values. The transfer chart for the inner decoder is made with the same procedure as the transfer charts of the decoder in the PCCC architecture. The difference in shape is caused by the puncturing of the systematic bits for the decoder in a PCCC architecture. The transfer chart for the outer decoder is also made by S. ten Brink in [28]. In [28] the settings of the decoder are not exactly the same, however the shape of the transfer chart is comparable.

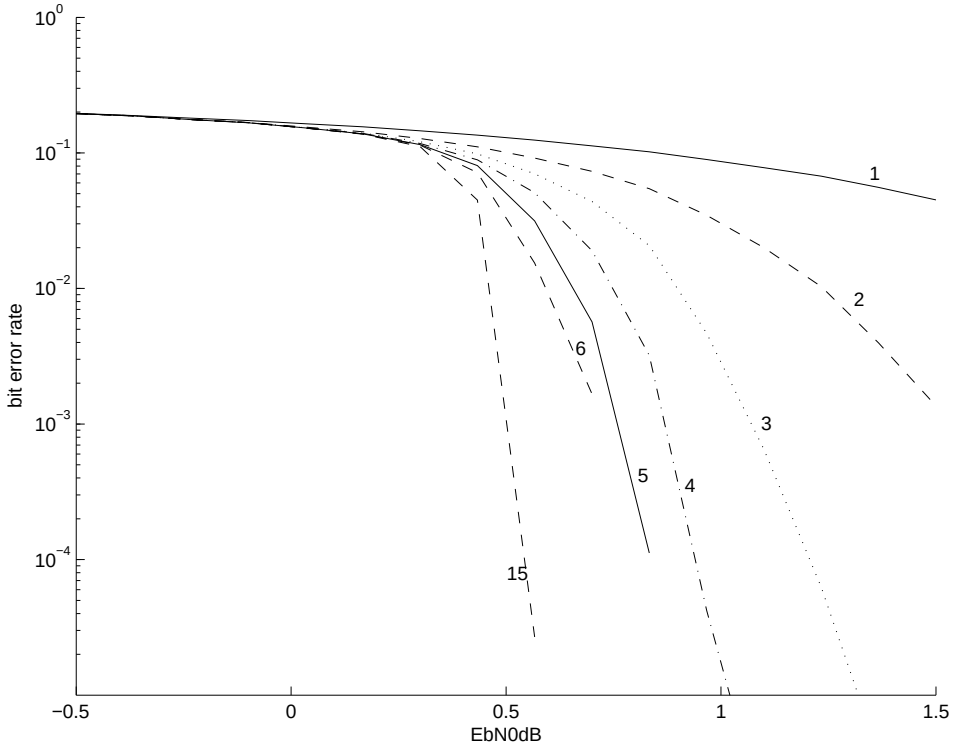
In figure 7.5 the EXIT chart of the SCCC architecture used in this thesis, just opens up at $E_b/N_0 = -0.1\,\text{dB}$. In the BER chart of the SCCC architecture in figure 7.4 on page 88, at $E_b/N_0 = -0.1\,\text{dB}$, after 15 iterations the BER decreased to $\approx 2 \cdot 10^{-3}$. The EXIT chart suggests that more iterations would have led to an even lower BER. For $E_b/N_0 = 0.5\,\text{dB}$ the decoding trajectory is given in the EXIT chart of figure 7.5. The trajectory reaches $I_E \approx 1$ after 6 iterations. In the BER chart of figure 7.4 at $E_b/N_0 = 0.5\,\text{dB}$ after 6 iterations a BER of $\approx 10^{-4}$ is reached. Iteration 7 is not shown in the BER chart, however when the curve of iteration 15 is examined, iteration 7 would have results in an even lower BER. These findings correspond, since a low BER implies a large mutual information I_E.

Because the BER charts of the PCCC and SCCC architectures in this

thesis show the same performance as in [4], we conclude that the implementations of the Log-MAP decoder is correct. We therefor can use the Log-MAP decoder for the simulations of the turbo multiuser detection architecture. Since the EXIT charts of these architectures also correspond with the simulations in [28] and [29], we conclude that the tool to make EXIT charts works correctly and we can use it to make an EXIT chart for the turbo multiuser detection algorithm.

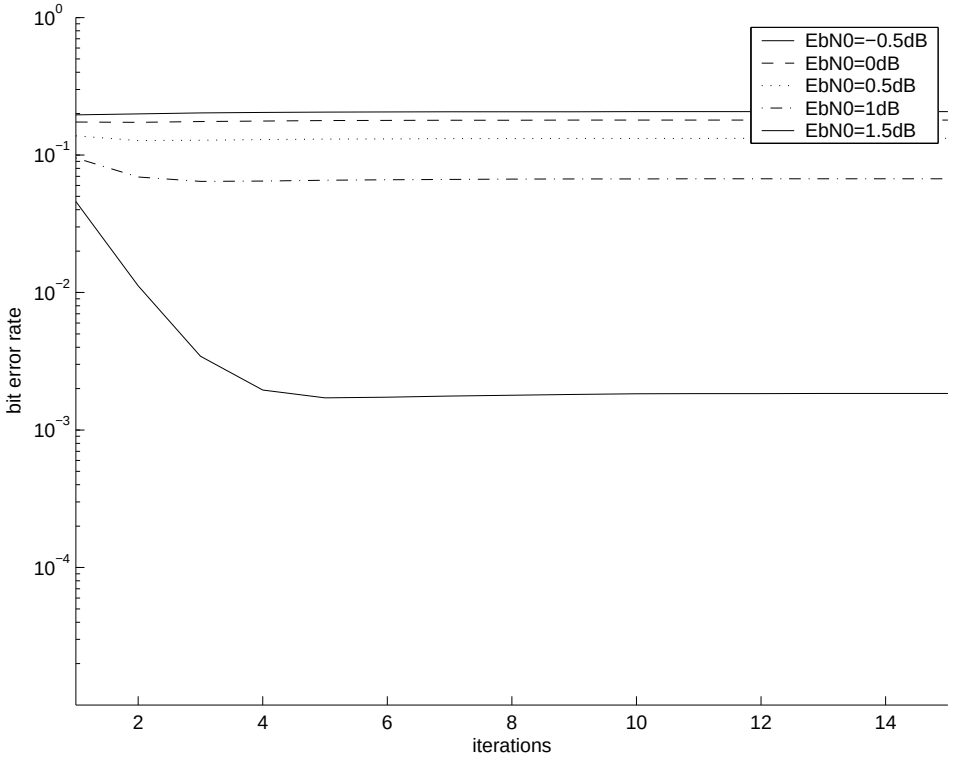


(a) bit error rate versus the number of iterations

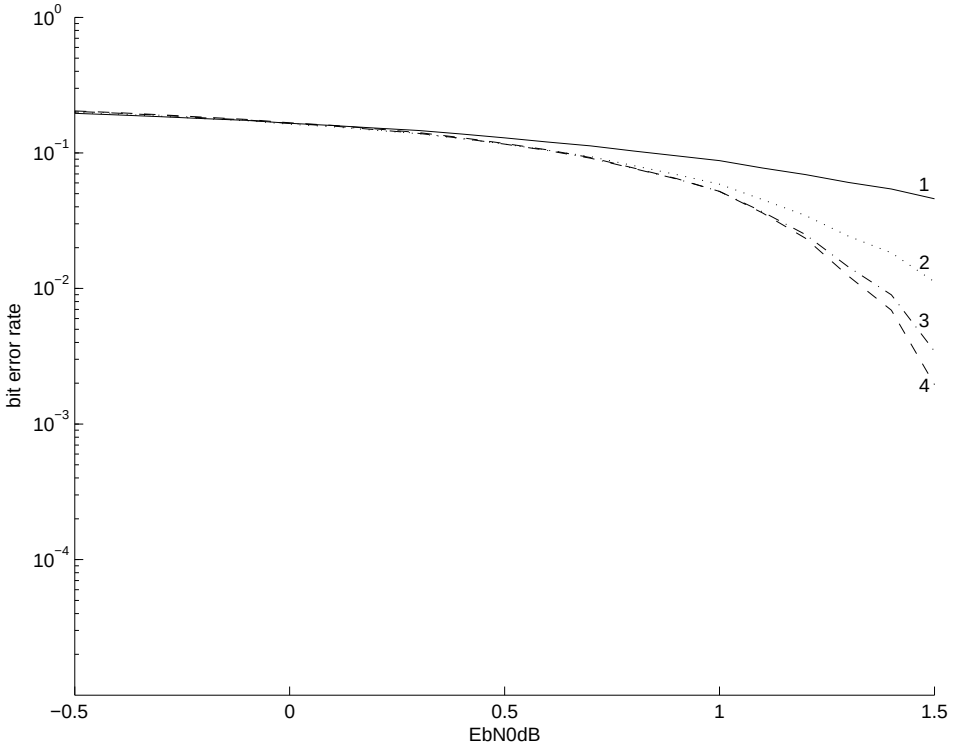


(b) bit error rate versus the signal to noise ratio, iteration numbers are next to the curves

Figure 7.1: Simulation results for a PCCC architecture, 100 blocks of 16384 data bits, interleaver size of 16384 bits



(a) bit error rate versus the number of iterations



(b) bit error rate versus the signal to noise ratio, iteration numbers are next to the curves

Figure 7.2: Simulation results for a PCCC architecture where a-posteriori instead of extrinsic information is iterated, 100 blocks of 16384 data bits, interleaver size of 16384 bits

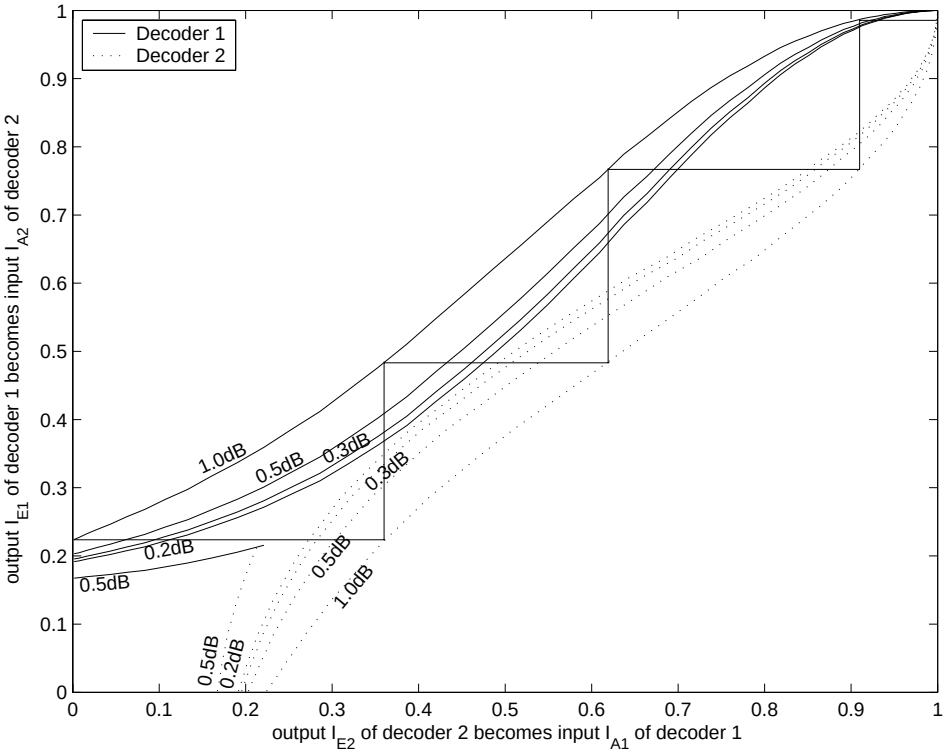
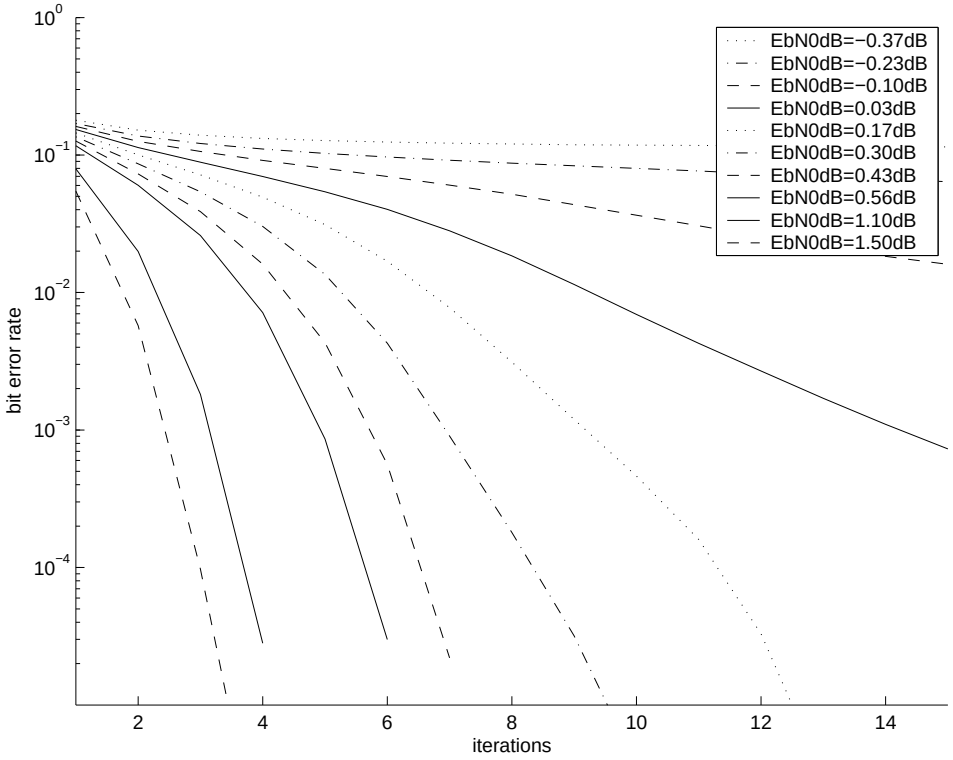
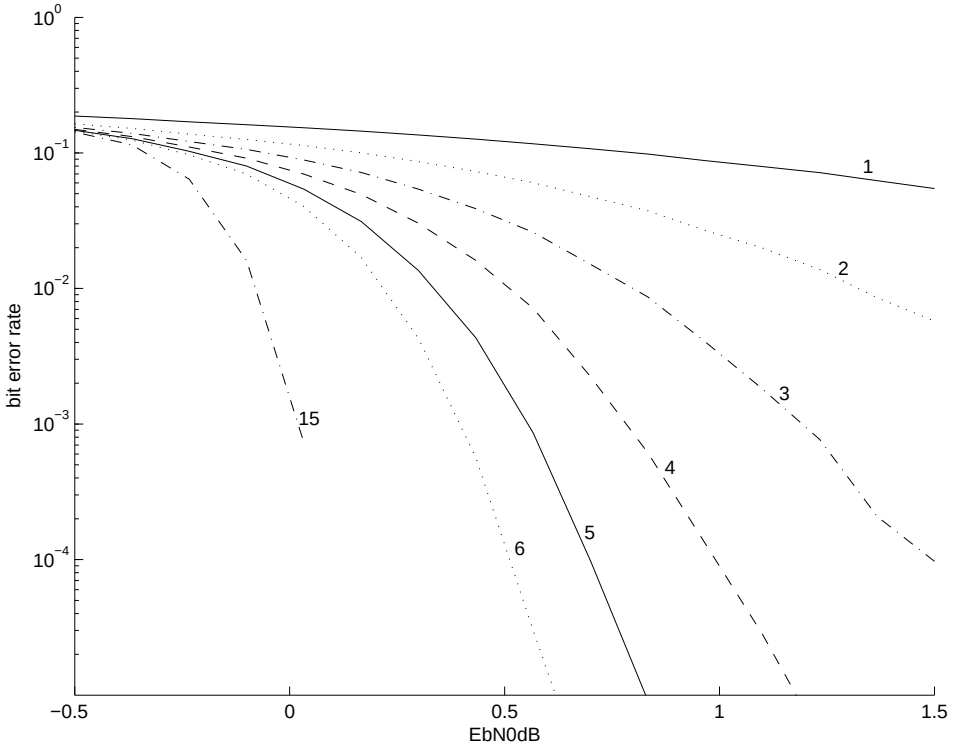


Figure 7.3: EXIT chart for the PCCC architecture. The E_b/N_0 values of the channel are plotted next to the curves of the decoders. The decoding trajectory for $E_b/N_0 = 1.0\,\mathrm{dB}$ is plotted in the figure.



(a) bit error rate versus the number of iterations



(b) bit error rate versus the signal to noise ratio, iteration numbers are next to the curves

Figure 7.4: Simulation results for a SCCC architecture, 100 blocks of 16384 data bits, interleaver size of 16384 bits

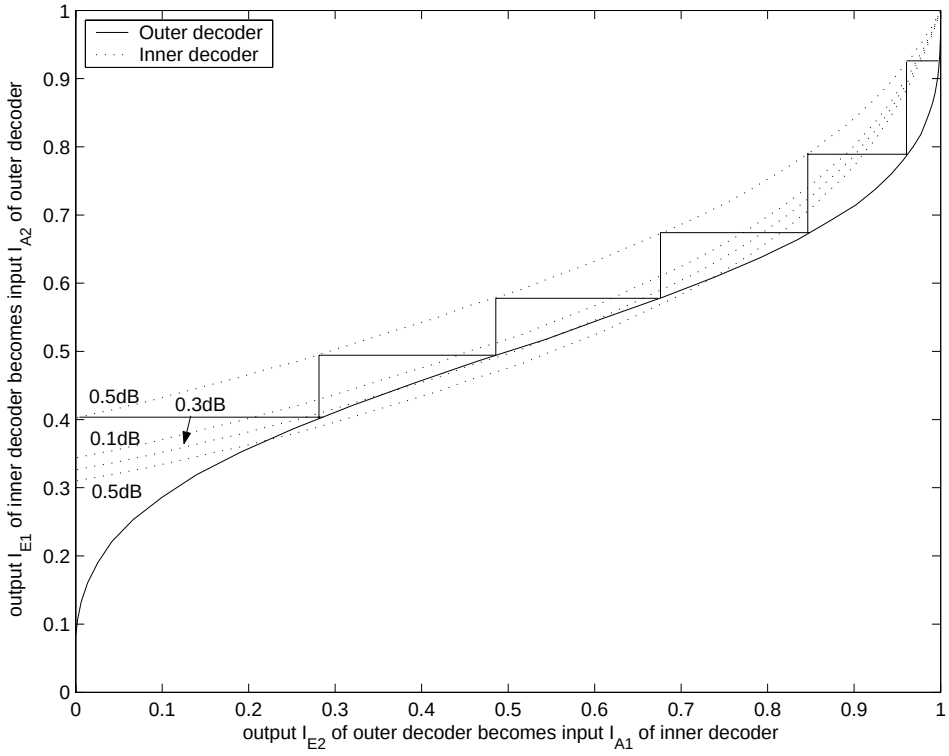


Figure 7.5: EXIT chart for the SCCC architecture. The E_b/N_0 values of the channel are plotted next to the curves of the inner decoder. The decoding trajectory for $E_b/N_0 = 0.5\,\mathrm{dB}$ is plotted in the figure.

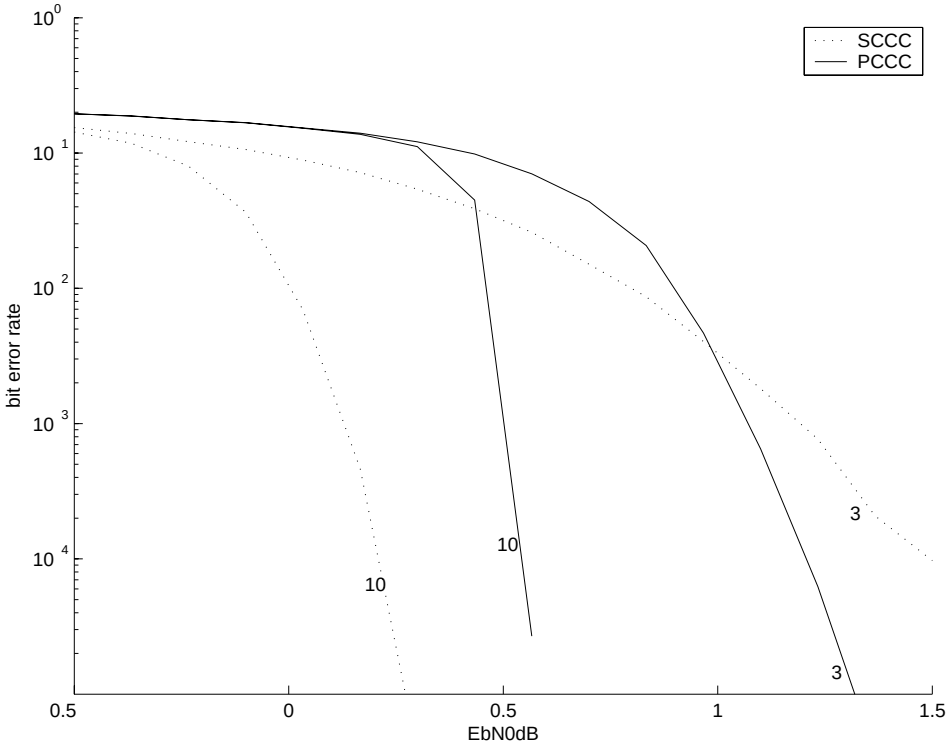


Figure 7.6: Comparison of SCCC and PCCC simulations. Iteration numbers are next to the lines.

7.3 Turbo Multiuser Detection Simulations

In this section a BER chart and an EXIT chart are created for the turbo multiuser detection architecture. At the end of the section conclusions are drawn based on these charts. These conclusions answer the main questions of this thesis, given in chapter 1.

7.3.1 Turbo Multiuser Detection Architecture BER chart

In this section the turbo multiuser detection architecture discussed in section 4.1 is simulated. The architecture is shown in figure 4.1 on page 34. The used soft-in/soft-out multiuser detector is the SISO multiuser detector of section 6.2.1 and is shown in figure 6.3 on page 64. The decoders are implemented as Log-MAP decoders of section 6.1.3. The settings of the transmitter are given in table 7.3. Simulations are performed on blocks of 65536 databits. In [20]

Encoder	type:	RSC
	rate:	1/2
	polynomial:	$\left[1, \frac{1+D+D^2+D^4}{1+D^3+D^4}\right]$
	ν:	5
	final state:	truncated
		(for every user the same)
Spreader	type:	direct spread CDMA
	spreadingsfactor:	16
	correlation coefficient ρ_{ij}:	0.75
Interleaver	type:	pseudo-random
	size:	6536 bits
		(different for every user)
Users	4, all have equal power	
Channel	AWGN	
Iterations	5	

Table 7.3: Settings for the transmitter of the users in the turbo multiuser detection arcitecture of figure 4.1 used for simulations

blocks of 128 databits were used. These small blocks are not used here, since the encoder is not terminated to a state. The influence of the block length was discussed in chapter 2. Simulations were stopped when at iteration 5 at least 4000 bit errors were simulated. More iterations did not increase the performance of the architecture. The results of these simulations are shown in figure 7.7 on page 96.

In the last part of section 4.1 the use of a-posteriori information instead of extrinsic information was discussed. In figure 7.8 on page 97 simulation results are given when the turbo multiuser detection architecture of figure 1

in [20] is used. In this architecture extrinsic information is iterated between blocks, instead of a-posteriori information as is done in figure 4.1 on page 34. The settings are also given in table 7.3.

In figure 7.7 on page 96 the upper dotted curve represents the BER curve for iteration 3 of a turbo multiuser detection architecture where extrinsic information is passed from the decoder to the SISO MUD and a-posteriori information is passed from the SISO MUD to the decoder.

7.3.2 Turbo Multiuser Detection Architecture EXIT chart

In this section the EXIT chart of the Turbo multiuser detection architecture is presented. The settings are the same as in that section, and thus given in table 7.3. The transfer chart of the SISO MUD is made with the simulation setup of figure 5.5 and the transfer chart of the decoder is made with the simulation setup for the outer decoder of figure 5.4. For the decoder two transfer charts are plotted; one for a decoder with a-posteriori output and one for a decoder with extrinsic output. This enables us to compare the convergence with extrinsic and a-posteriori output. The transfer charts are made with 10^6 input databits. Since no puncturing is used in the turbo multiuser detection architecture, no channelbits need to be put to zero, as was done with the EXIT chart of the PCCC architecture. The transfer charts are made for $E_b/N_0 = 1\,\text{dB}$, $E_b/N_0 = 2\,\text{dB}$, $E_b/N_0 = 3\,\text{dB}$ and $E_b/N_0 = 4\,\text{dB}$. The EXIT chart is plotted in figure 7.10 on page 99. In this figure the decoding trajectory is drawn for $E_b/N_0 = 4\,\text{dB}$.

7.3.3 Conclusions

SISO Multiuser Detector Implementation Verification

The simulation results of the turbo multiuser detector architecture in figure 7.7 on page 96 are also given in figure 3 of [20]. The difference between the architecture in this thesis and the architecture in [20] was described at the end of section 4.1 on page 33. The simulation results in figure 7.8 on page 97 show that the architecture presented in figure 1 of [20] does not iterate to a lower BER. For the first iteration the results are the same as in the figure of [20], but for more iterations the results differ. For 3 or more iterations the BER seems to converge to ≈ 0.15 in figure 7.8. An explanation for this can be given by examining equation 3.2 on page 28. Extrinsic information is created by subtracting the intrinsic information from the a-posteriori information. Extrinsic information contains therefor less information about the data- or codebit than a-posteriori information. Since the SISO multiuser detector performs soft-cancellation, as was explained in section 6.2.1 on page 64, it needs the best possible estimate of the codebit and that is provided by the a-posteriori information.

When the architecture of figure 4.1 is used, the simulation results can be compared with the simulation results in figure 3 of [20]. The single user curve in figure 7.7 is at a slightly lower BER when $E_b/N_0 = 0\,\text{dB}$ and at a slightly larger BER when $E_b/N_0 = 4\,\text{dB}$. The curves for iteration 1,2&3 when $E_b/N_0 = 4\,\text{dB}$ have a larger BER in figure 7.7 than in [20]. However, at iteration 4&5 the curves of figure 7.7 and [20] are at about the same BER.

The upper dotted curve in figure 7.7 is at a higher BER than the curve for iteration 3 of the architecture in figure 4.1. This dotted curve is made when extrinsic information is send from the decoder to the multiuser detector, and a-posteriori information is send from multiuser detector to the decoder. So although only extrinsic information is passed from decoder to multiuser detector, the architecture still performs worse than when all a-posteriori information was used. We conclude that using a-posteriori information in a turbo multiuser detection architecture gives better results. An explanation can be found in section 3.3 on page 26. There it was said that a decoding block should never get information it already has. The decoder in a turbo multiuser detection architecture, however, has no other information than the information about the codebits. So offering it less information will result in a worse performance. So, it could be concluded that in the SCCC architecture, the outer decoder should also receive a-posteriori information to obtain a performance increase. However simulations showed that this is *not* true and a worse performance is obtained. These simulations are not presented in this thesis. No explanation can be given for this behavior. The discussion about a-posteriori and extrinsic information is continued in section 7.4.

The results above show that using a-posteriori information to iterate between the blocks in a turbo multiuser detection architecture gives better results than using extrinsic information as is done in [20]. When we assume that in [20] also a-posteriori information was used (this is not reported there), to obtain the simulation results, we can conclude that the implementation of the SISO multiuser detection architecture works correctly.

Turbo & non-Turbo Multiuser Detection Comparison

In figure 7.9 on page 98, simulation results are shown for non-turbo multiuser detectors. This chart has been copied from [7] and is also used in [21]. The multiuser detector in figure 7.9 that is comparable with the SISO multiuser detector in this thesis is the '*Successive Interference Canceller (Succ IC)*'. For a detailed description of this multiuser detector refer to [21]. The two multiuser detectors will be compared at $E_b/N_0 = 4\,\text{dB}$. In the turbo multiuser detector architecture the SISO multiuser detector is followed by a convolutional code decoder. During the first iteration, no a-priori information is available at the SISO MUD, so it acts like a set of matched-filters. Since the multiuser detector in the turbo multiuser detection architecture can not operate on its own, we have to compare the entire turbo multiuser detector

architecture with the successive interference cancelling multiuser detector. The simulations of figure 7.9 were done with 10 users and a spreading factor 31, while the simulations in this thesis are done with 4 users and a spreading factor of 16. Also the correlation factors of the spreading sequences of the non-turbo multiuser detector are not given in [7], however in our simulations of the turbo multiuser detector they are $\rho_{ij} = 0.75$ and obviously quite large.

When we compare figure 7.9 and figure 7.7, we see that the turbo architecture only performs better than the *Succ IC* at $E_b/N_0 > 3\,\text{dB}$. For this performance increase the turbo architecture has to iterate at least two times. When only one iteration is used, the successive IC and all the other multiuser detectors of figure 7.9 outperform the turbo architecture for all E_b/N_0, even though the number of users in the non-turbo simulations (U=10) is higher than in the turbo simulations (U=4). Because of the convolutional code, the single user bound of the turbo architecture descends much faster to low BER at increasing E_b/N_0 than the single user bound of the successive IC. Simulations on the turbo architecture have not been done for $E_b/N_0 > 4\,\text{dB}$, however when the BER curves are extrapolated to higher E_b/N_0, we see that iterating 5 times in a turbo architecture gives a dramatically increase in performance in terms of BER.

Overall we can conclude that the turbo multiuser detector of this thesis gives an increased performance over the non-turbo multiuser detectors of figure 7.9 when $E_b/N_0 > 3\,\text{dB}$ and more than one iteration is performed. When these conditions are fulfilled, the turbo multiuser detector will give a high performance increase in terms of BER, compared to the non-turbo multiuser detector. However, the performance increase in terms of the BER should be able to make up for the increase in complexity and required computational power. In [21] the complexity of the *Succ IC* is given on page 30. The complexity of the SISO multiuser detector in this thesis is given in section 6.2.2 and the complexity of the Log-MAP decoder is given in section 6.1.4. In case the complexity increase of a turbo architecture poses no problem and the BER needs to be very low or the number of users in an area is very high, a turbo multiuser detector is a better choice than a non-turbo multiuser detector.

Turbo Multiuser Detection Architecture EXIT chart

The EXIT chart of the turbo multiuser detection architecture used in this thesis, is given in figure 7.10 on page 99 for different E_b/N_0 values. Similar simulation results were *not* found in literature. In figure 7.10 two curves for the decoder are drawn, one for the decoder with a-posteriori output and one for the decoder with extrinsic output. In section 7.3.1 it was concluded, that iterating a-posteriori information gives better results than extrinsic information. In figure 7.10 the decoder transfer chart for extrinsic information, intersects the transfer chart of the SISO multiuser detector earlier than the

transfer chart of the a-posteriori information. Thus the EXIT charts confirms that a-posteriori information gives better results in the turbo multiuser detection algorithm.

The a-posteriori transfer chart intersects the SISO multiuser detector transfer chart for low E_b/N_0 values earlier than for higher values. This also can be see in the BER chart of figure 7.7 on page 96, when at low E_b/N_0 iterating does not improve BER significantly.

Although the transfer chart of the decoder reaches the point where $I_E \approx 1$, the transfer chart of the SISO multiuser detector does not reach this point for the simulated E_b/N_0 values. This implies that the turbo multiuser detector does not converge to a low BER of $\approx 10^{-5}$ for every value for E_b/N_0 as was seen with the PCCC and SCCC architectures of the previous sections, provided that the curves in the EXIT chart of the PCCC and SCCC architecture do not intersect before this point is reached. In the EXIT chart the transfer charts of the SISO multiuser detector are vertically shifted upwards for higher E_b/N_0 values. If the E_b/N_0 value becomes high enough, it is expected that the transfer chart of the SISO multiuser detector will also reach $I_E \approx 1$.

In figure 7.10 the iterative trajectory for $E_b/N_0 = 4\,\mathrm{dB}$ is given. When this trajectory is quickly examined, it appears that the two transfer charts intersect after 3 iterations. However, in figure 7.7 on page 96 the fourth and fifth iteration still give a lower BER than the third iteration. In the EXIT chart of figure 7.10 the transfer chart of the decoder, makes a very steep rise at the end of the trajectory. Here, a small addition of mutual information by the SISO multiuser detector, will give a large addition of mutual information by the decoder.

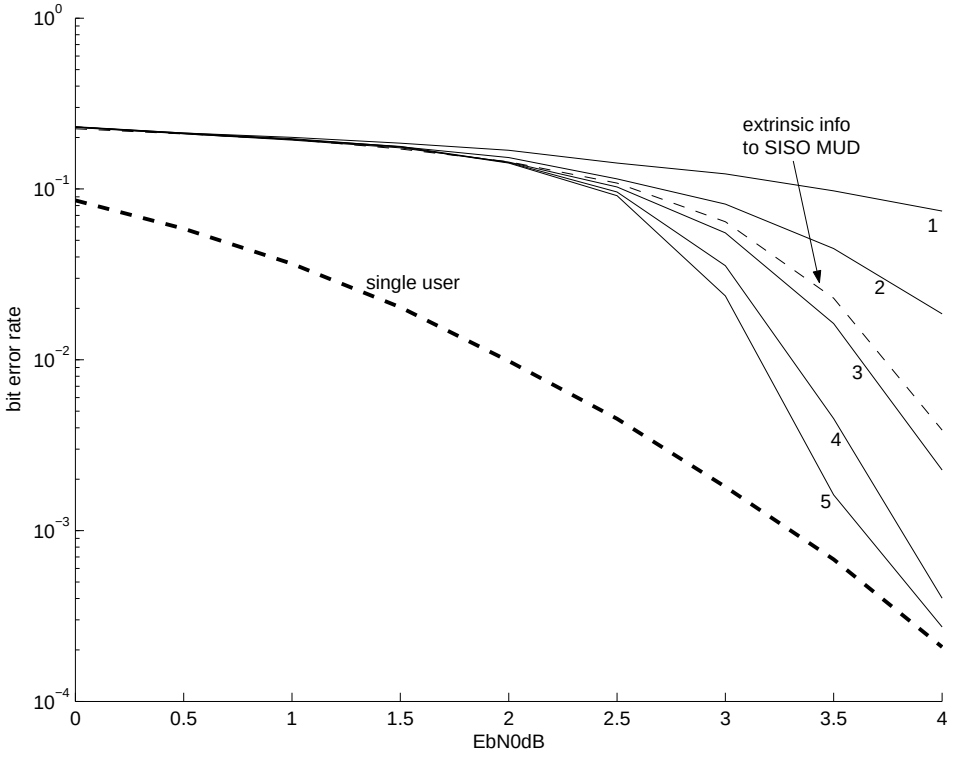


Figure 7.7: Simulation results for a turbo multiuser detection architecture, 65536 data bits per block, interleaver size of 65536 bits, 4000 errors per iteration, $\rho_{ij} = 0.75$ and 4 users. Iteration numbers are next to the lines. The dotted curve is iteration 3 in an architecture where extrinsic instead of a-posteriori information is passed from the decoder to the SISO MUD

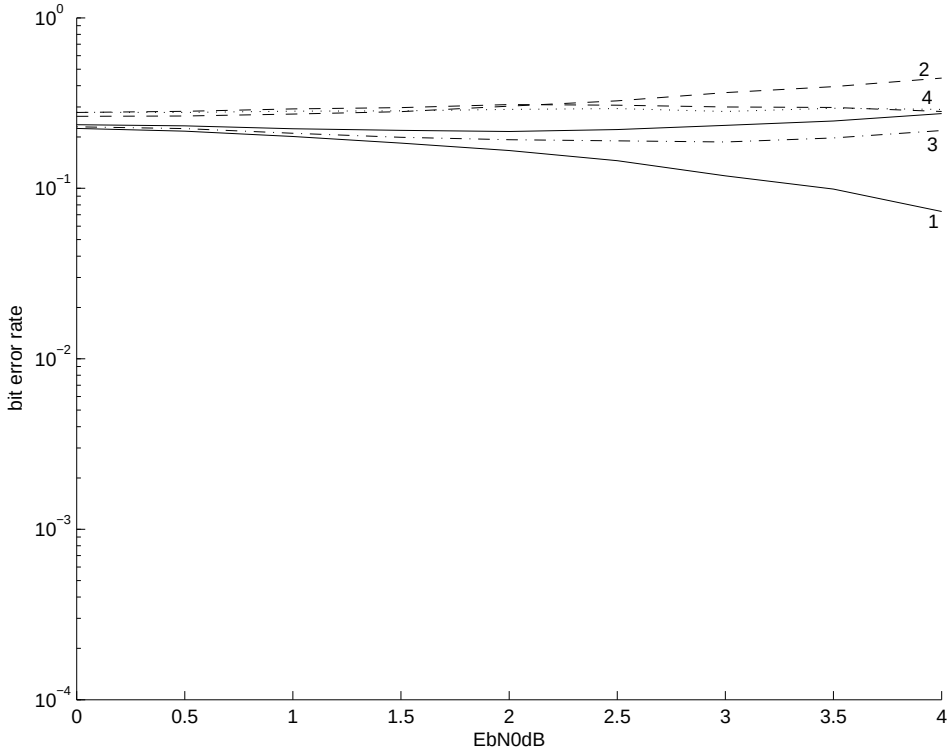


Figure 7.8: Simulation results for a turbo multiuser detection scheme where extrinsic information is iterated between the blocks. Iteration numbers are next to the curves

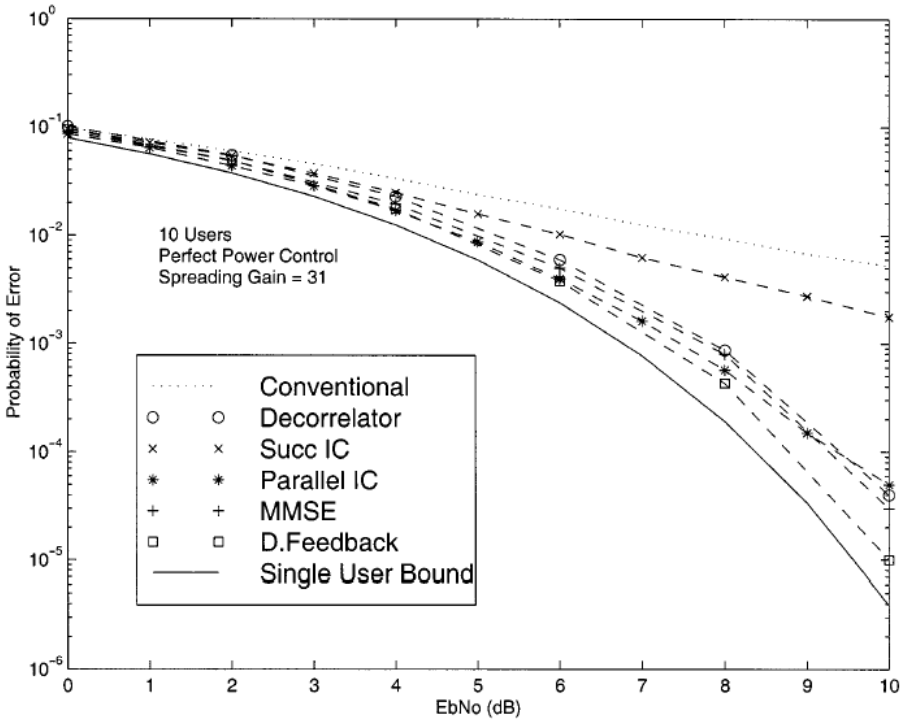


Figure 7.9: Simulation results for non-turbo multiuser detection architectures obtained from [7] and used in [21]

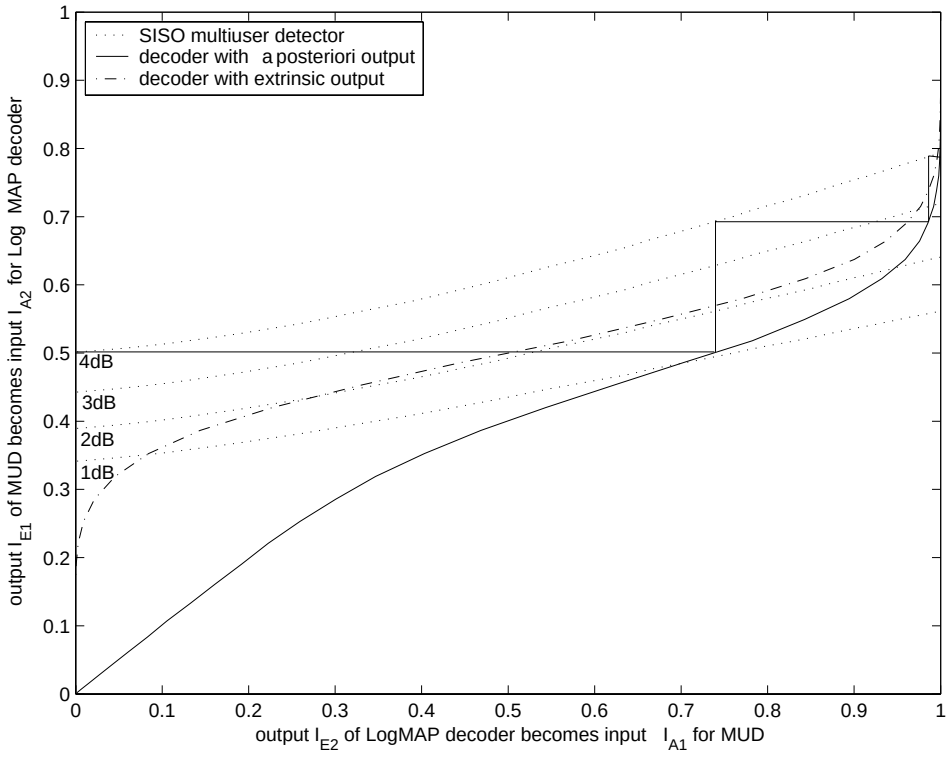


Figure 7.10: EXIT chart for the turbo multiuser detection architecture. The E_b/N_0 values of the channel are plotted below the curves of the SISO multiuser detector. Two transfer charts for the decoder are given; one where the decoder outputs a-posteriori information and one where the decoder output extrinsic information.

7.4 A-posteriori & Extrinsic Transfer Charts

Because of the discussion about a-posteriori and extrinsic information in previous sections, these subjects are further investigated. In this section transfer charts of the a-posteriori and extrinsic information output of a Log-MAP decoder are presented. With these transfer charts the difference of the information contents of a-posteriori and extrinsic information can be evaluated. At the end of the section conclusions will be given.

7.4.1 A-posteriori & Extrinsic Transfer Charts

In figure 7.11 on page 101 the transfer function of an outer decoder in an SCCC architecture is plotted for an output of extrinsic and a-posteriori information. The setting of the outer decoder is the same as the setting of the decoder in table 7.1.

In figure 7.12 on page 102 the transfer function of an inner decoder in an SCCC architecture is plotted for an output of extrinsic and a-posteriori information. The setting of the inner decoder is the same as the setting of the decoder in table 7.2.

7.4.2 Conclusions

In figure 7.11 on page 101 the transfer function of an outer decoder in an SCCC architecture is plotted for an output of extrinsic and a-posteriori information. In figure 7.12 on page 102 the transfer function of an inner decoder in an SCCC architecture is plotted for an output of extrinsic and a-posteriori information.

In all these figures the a-posteriori information has a higher mutual information value than the extrinsic information. This can be explained, by referring to equation 3.2 on page 28. The extrinsic information is made by subtracting the intrinsic information from the a-posteriori information, thus the extrinsic information contains less information about the databit d_k than the a-posteriori information.

In the figures the mutual information of the extrinsic and a-posteriori information converge to each other for large mutual information inputs. Thus, the extra information obtained for every iteration becomes less when more iterations are performed, as was concluded in chapter 5.

From the transfer charts of figure 7.12, it can be concluded that when the a-posteriori transfer charts are used to make an EXIT chart, the transfer charts will intersect not as early as when the transfer charts of the extrinsic information are used, because the transfer chart of the a-posteriori information is concave and the transfer chart of the extrinsic information is convex. From this, it can be concluded that using a-posteriori information to iterate between the decoders in a PCCC or an SCCC architecture will give a

better performance in terms of BER. However, in figure 7.2 on page 86 using a-posteriori information gives worse results in terms of BER than when extrinsic information is used, as in figure 7.1.

A possible explanation for this can be found in the procedure for calculating the mutual information of the output. To calculate the mutual information, the LLR output was assumed to be Gaussian distributed. A-posteriori information might not have a Gaussian distribution. To calculate the mutual information of the a-posteriori information, without assuming any distribution, equation 5.9 on page 40 can be used. In this formula the probability density function is needed to calculate the mutual information. Further investigations are needed on this.

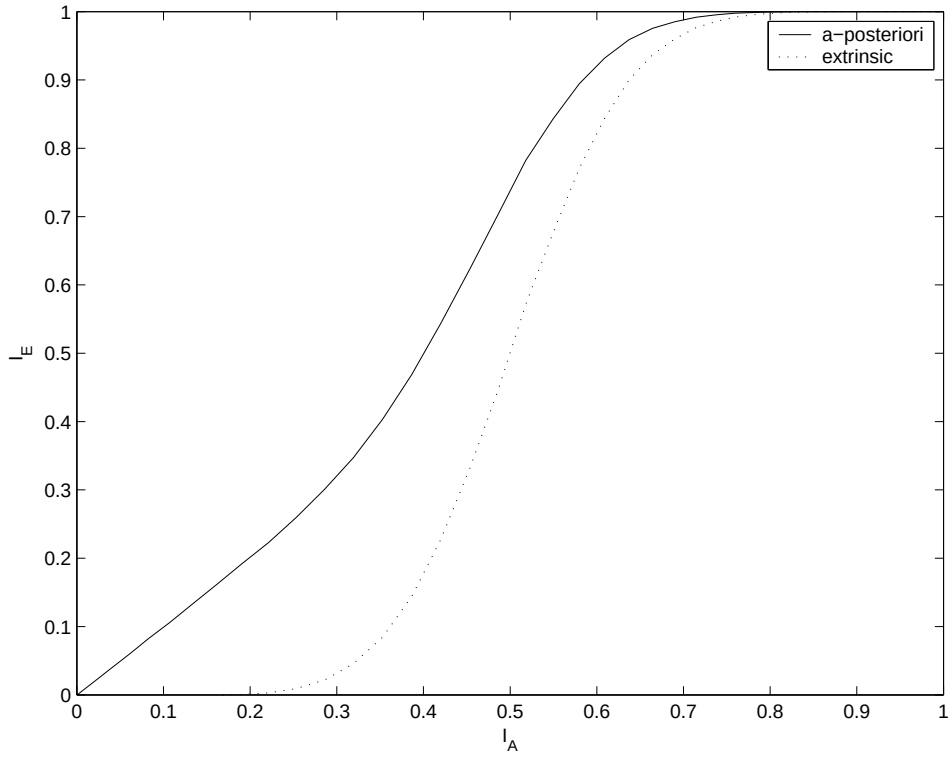
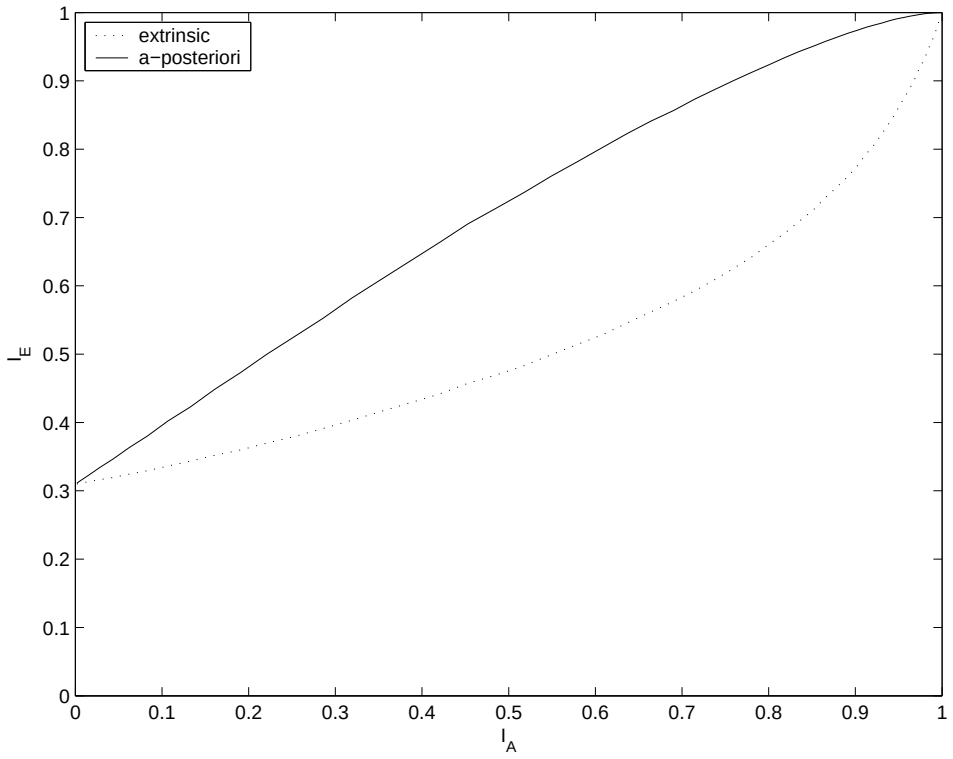
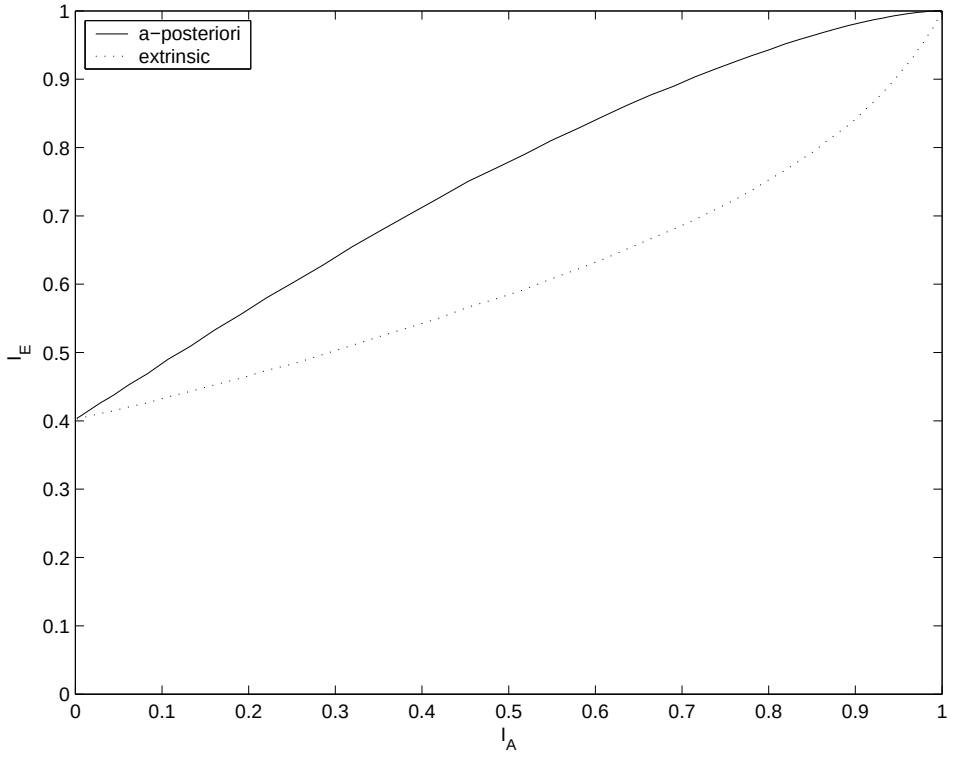


Figure 7.11: Transfer charts for an outer decoder in an SCCC architecture which outputs a-posteriori information and extrinsic information. The settings are the same as for the decoder in table 7.1.



(a) transfer chart for $E_b/N_0 = -0.5\,\mathrm{dB}$ of the channel



(b) transfer chart for $E_b/N_0 = 0.5\,\mathrm{dB}$ of the channel

Figure 7.12: Transfer charts for an inner decoder in an SCCC architecture which outputs a-posteriori information and extrinsic information. The settings are the same as for the inner decoder in table 7.2.

7.5 Conclusions

In this chapter simulations were performed on the PCCC and SCCC architectures of chapter 3 to verify the correct implementation of the used algorithms. Also simulations were performed on the turbo multiuser detection architecture of chapter 4 to answer the main questions of this thesis, which were formulated in chapter 1. Since simulations showed very different behavior of the architectures when a-posteriori or extrinsic information was used, transfer charts were made to analyze this difference.

In section 7.2 simulations were performed on the PCCC and SCCC architectures to verify the correctness of the implementations of the Log-MAP decoder and the EXIT chart tool. BER charts were used to make comparisons with BER charts from the same architectures in other literature. Since the BER charts in this thesis correspond to the BER charts in other literature, it was concluded that the implementation of the Log-MAP decoder is correct. EXIT charts were used to make comparisons with EXIT charts for the same architectures in other literature. Since the EXIT charts in this thesis were similar to EXIT charts in literature, and that the conclusions drawn from EXIT charts and BER charts corresponded with each other, it was concluded that the tool to make EXIT charts works correctly.

With the EXIT chart for the PCCC architecture, it was concluded that the minimum value for E_b/N_0, to make it iterate to a low BER, is 0.2 dB. For the SCCC architecture this minimum value for E_b/N_0 was found to be -0.1 dB.

In section 7.3 simulations were performed on the turbo multiuser detection architecture. BER charts were used to make comparisons with simulation results in literature. Since the BER charts in this thesis correspond to the BER chart in literature, it was concluded that the implementation of the SISO multiuser detector is correct. EXIT charts for a turbo multiuser detection architecture were created to analyze the convergence behavior. EXIT charts were applied to turbo multiuser detection for the first time.

From the BER charts, it was concluded that a-posteriori information needs to be iterated between the decoding blocks, instead of extrinsic information as is done in [20]. The BER charts were also used to compare the turbo multiuser detection architecture with a non-turbo multiuser detector. It was found that the turbo multiuser detection architecture outperforms the non-turbo multiuser detector dramatically, when $E_b/N_0 > 3$ dB and more than one iteration is performed. The comparisons were performed for the *Succ IC*, which is similar to SISO multiuser detector in this thesis.

The EXIT charts showed that for lower values of E_b/N_0, the transfer charts of an EXIT chart intersect earlier. The EXIT charts also showed that the turbo multiuser detection architecture does not reach the point

where $I_E \approx 1$, for the simulated values of E_b/N_0. The PCCC and SCCC architectures were found to always be able to converge to this point, provided that the transfer charts in the EXIT chart of the PCCC or SCCC architecture do not intersect before this point is reached. It is assumed that for higher E_b/N_0 the turbo multiuser detection architecture also will converge to $I_E \approx 1$, since for higher E_b/N_0 the transfer chart of the SISO multiuser detector shifts upwards. In the EXIT chart it was also seen, that the transfer chart for a decoder that outputs extrinsic information, intersects the transfer chart of the SISO multiuser detector for a lower input mutual information value, than for a decoder that outputs a-posteriori information. This corresponds with the conclusions that a-posteriori information gives a better performance in terms of BER than extrinsic information, which were taken from the BER charts.

From the transfer charts with a-posteriori and extrinsic information curves for a decoder, it seemed that using a-posteriori information to iterate between the blocks of a PCCC or an SCCC architecture gives better results in terms of BER than using extrinsic information. This, however, does not comply with BER simulations of a-posteriori information. A possible explanation is that the mutual information of the LLR values is calculated with a Gaussian distribution imposed on these values. A-posteriori information might not have a Gaussian assumption. However, the EXIT chart of the turbo multiuser detector architecture seems to be more correct, when the transfer chart for a-posteriori information of the decoder is used. These findings contradict each other. This behavior can not be explained without further investigations.

8

Conclusions and Recommendations

In this chapter conclusions are drawn for the main questions of this thesis described in chapter 1. Also conclusions will be drawn for some other questions that arose during the investigation in this thesis. Next, some recommendations are made for further research on the topic of turbo multiuser detection.

8.1 Conclusions

In this thesis *the turbo principle applied to multiuser detection* was investigated. Two main questions were formulated in chapter 1, which are answered successively below.

The first question was: 'what is the amount of performance gain that can be achieved by using turbo-multiuser-detection instead of 'classical'-multiuser-detection ('classical'=no turbo) and under what conditions and costs can these performance gains be achieved?'. This question was answered by comparing the BER chart of the turbo multiuser detection architecture with the BER chart of a non-turbo multiuser detector. The BER chart showed that a turbo multiuser detection architecture outperforms a non-turbo multiuser detector dramatically for $E_b/N_0 > 3\,\mathrm{dB}$ and when more than one iteration is performed. The costs of this performance gain were not evaluated. However, a reference to literature, where the complexity of the non-turbo multiuser detector and a reference to the sections in this thesis where the complexity of the turbo multiuser detection architecture can be found, were given. The results of the simulations to answer this question can be found in section 7.3.1. More elaborate conclusions can be found in section 7.3.3.

The second question was: 'what is the convergence behavior of a turbo

multiuser detection architecture?'. The convergence behavior of the turbo multiuser detection architecture was analyzed with an EXIT chart. The EXIT chart showed that for lower values of E_b/N_0, the transfer charts of an EXIT chart intersect at lower input mutual information values and thus the BER of the architecture will not converge to a low value. The EXIT charts also showed that the turbo multiuser detection architecture does not reach the point where $I_E \approx 1$, for the simulated values of E_b/N_0. The PCCC and SCCC architectures were found to always be able to converge to this point, provided that the transfer charts in the EXIT chart of the PCCC or SCCC architecture do not intersect before this point is reached. It is assumed that for higher E_b/N_0 the turbo multiuser detection architecture also will converge to $I_E \approx 1$, since for higher E_b/N_0 the transfer chart of the SISO multiuser detector shifts upwards. The results of the simulations to answer this question can be found in section 7.3.2. More elaborate conclusions can be found in section 7.3.3.

For this thesis software was created that can be used to simulate turbo architectures. Also a tool to create EXIT charts was made. Simulations on PCCC and SCCC architectures were done, to verify the implementations of the algorithms used in the turbo multiuser detection architecture and to verify the tool to create EXIT charts. The implementations and the tool were found to be correct.

During the investigation of the turbo architectures, questions arose about the difference between a-posteriori and extrinsic information. Transfer charts of a Log-MAP decoder that outputs a-posteriori and extrinsic information were made to answer these questions. The transfer charts showed that using a-posteriori information in a PCCC or SCCC architecture should give better results in terms of BER. Simulations of these architectures contradict these findings. On the other hand, when the transfer chart of the decoder with a-posteriori output is used in the EXIT chart of the turbo multiuser detection architecture, the findings seem to correspond better with simulations. An explanation for this can not be given. More conclusions can be found in section 7.4.

8.2 Recommendations

- EXIT charts can be used to determine how many iterations of a turbo architecture are useful to obtain a low BER. This can be used in the AWGN project [11], where adaptive wireless networking is researched. EXIT charts can be made for a number of situations and stored in the decoder or can be perhaps made *semi-real-time* to adaptively alter the number of iterations. A total *real-time* implementation will not

be possible, since the mutual information is calculated from a block of LLR values. So a number of LLR values first have to be created before the calculation can be done, which causes a time delay. However, computing an EXIT chart *semi-real-time* might need too much computational power to be feasible.

- Simulations to obtain BER charts are very time consuming. Investigations could be performed to reduce the time needed for these simulations.

- No simulations were performed on the hybrid turbo multiuser detection architectures of section 4.2. Techniques developed in [30] could be used to create 3-dimensional EXIT charts for these architectures and determine the optimal order of activation.

- The EXIT charts of the turbo multiuser detection architecture were created with the assumption that the mutual information of every user increased with the same value at every iteration. This implies that the correlation coefficient between the spreadwords of all the users is about the same and that the power-control of the systems is perfect. Since in practical situations these assumptions might not hold, their influence on the EXIT chart could be investigated.

- The transfer charts of chapter 7 were created by assuming a Gaussian distribution on the output LLR values. The validity of this assumption could be checked by calculating the mutual information of the LLR values based on the probability density function. When the probability density function is known, equation 5.9 on page 40 can be used to calculate the mutual information.

- Other SISO multiuser detectors, that are suitable for usage in a turbo architecture, are presented in literature, e.g. [32][18][17][22][1]. Their performance could be evaluated by creating the appropriate EXIT charts.

- The settings for the encoders could be changed to find a code, that gives better results when used in a turbo multiuser detection architecture than the codes that are used in this thesis.

- For the PCCC architecture in this thesis, no bit-error floor was observed, although this error floor is observed in literature like [4]. Further investigations on this bit-error floor could be performed.

Bibliography

[1] P.D. Alexander, M.C. Reed, J.A. Asenstorfer, and C.B. Schlegel. Iterative multiuser interference reduction: Turbo cdma. *IEEE Trans on Comm.*, 47(7), July, 1999.

[2] L.R. Bahl, J. Cocke, F. Jelinek, and J. Raviv. Optimal decoding of linear codes for minimizing symbol error rate. *IEEE Trans. on Information Theory*, March 1974.

[3] A.S. Barbulescu and S.S. Pietrobon. Interleaver design for turbo codes. *Electr. Letters*, December 1994.

[4] S. Benedetto and G. Montorsi. Iterative decoding of serially concatenated convolutional codes. *Electr. Letters*, 32(13), June 1996.

[5] S. Benedetto and G. Montorsi. Design of parallel concatenated convolutional codes. *IEEE. Trans. Comm*, May 1996.

[6] C. Berrou and A. Glavieux. Near optimum error correcting coding and decoding: Turbo-codes. *IEEE Trans. Comm.*, 44, October 1996.

[7] R.M. Buehrer, N.S. Correal-Mendoza, and B.D. Woerner. A simulation comparision of multiuser receivers for cellular cdma. *IEEE Transactions on Vehicular Technology*, 4(49):1065–1085, July, 2000.

[8] A. Burr. Turbo-codes: the ultimate error control codes. *Electronics and Communication Engineering Journal*, August 2001.

[9] Open Source Community. It++ 3.7.1 c++ library. `http://itpp.sourceforge.net`, 2003.

[10] C. Flemming. A tutorial on convolutional coding with viterbi decoding. `http://pweb.netcom.com/%7Echip.f/Viterbi.html`, 2003.

[11] ir. F.W. Hoeksema, ir. J. Potman, and Prof. Dr. ir. C.H. Slump. Adaptive wireless networking (awgn) project. `http://www.sas.`

el.utwente.nl/research/wireless_communication/adaptive_w%ireless_networking/, 2003.

[12] J. Hagenauer. Log-likelihood ratios, mutual information and exit charts - a primer. *12th Joint Conference on Communications and Coding, Saas Fe, Switzerland*.

[13] R.W. Hamming. *Coding and Information Theory*. Prentice Hall, Inc., 1986.

[14] L. Hanzo, T.H. Liew, and B.L. Yeap. *Turbo Coding, Turbo Equalisation and Space-Time Coding for Transmission over fading channels*. John Wiley & Sons, 2002.

[15] S. Hawking. *A brief history of time*. Bantam Books, 1988.

[16] J.F. Hayes. The viterbi algorithm applied to digital data transmission. *IEEE Comm magazine*, 50th anniversary, May 2002.

[17] Q. Li, X. Wang, and C.N. Georghiades. Turbo multiuser detection for turbo-coded cdma in multipath fading channels. *IEEE Trans. on Vehicular Techn.*, 51(5), September, 2002.

[18] L. Liu, W.K. Leung, and L. Ping. Simple iterative chip-by-chip multiuser detection for cdma systems. *Department of Electronic Engineering, City University of Hong Kong*.

[19] H.V. Poor and S. Verdú. Probability of error in mmse multiuser detection. *IEEE Trans. Inform. Theory*, 43, May 1997.

[20] H.V. Poor and X. Wang. Iterative (turbo) soft interference cancellation and decoding for coded cdma. *IEEE Trans. Comm.*, 47(7), July 1999.

[21] J. Potman. Development of a multiuser detection testbed. *Master's thesis, University of Twente, Department of electrical engineering, Laboratory of Signals & Systems*.

[22] D. Reynolds and X. Wang. Turbo multiuser detection with unknown interferers. *IEEE Trans. on Comm.*, 50(4), April, 2002.

[23] P. Robertson, E. Villebrun, and P. Hoeher. A comparison of optimal and sub-optimal map decoding algorithms operating in the log domain. *Proc.ICC '95, Seattle, USA*, June 1995.

[24] H. R. Sadjadpour, N. J. A. Sloane, M. Salehi, and G. Nebe. Interleaver design for turbo codes.

[25] C. Schlegel and L. Perez. *Trellis and Turbo Coding*. IEEE Press, 2002.

[26] B. Sklar. A primer on turbo code concepts. *IEEE Comm. Magazine*, December 1997.

[27] S. ten Brink. Rate one-half code for approaching the shannon limit by 0.1db. *Electronics Letters*, 36(15), July 2000.

[28] S. ten Brink. Convergence of iterative decoding. *Electronics Letters*, 35(10), May 1999.

[29] S. ten Brink. Convergence behavior of iteratively decoded parallel concatenated codes. *IEEE Trans. Comm.*, 49(10), October 2001.

[30] M. Tuchler. Convergence prediction for iterative decoding of threefold concatenated systems. *Institute for Communications Engineering, Munich University of Technology, Germany.*

[31] S. Verdú. *Multiuser Detection.* Cambridge Univ. Press, 1998.

[32] C. Wang, J. Hsu, S. Shih, and J. Wen. A soft-input soft-output decorrelating block decision-feedback multiuser detector for turbo-coded dscdma systems. *Wireless Personal Communications*, 17.

[33] N. Wiberg. Codes and decoding on general graphs. *Ph.D. dissertation, Linkoping Univ. Sweden.*

[34] R.E. Ziemer and R.L. Peterson. *Introduction to digital communication.* Prentice Hall, Inc., 2001.