# Implementation of Parallel and Serial Concatenated Convolutional Codes

Yufei Wu

Dissertation submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy
in
Electrical Engineering

Brian D. Woerner, Chair
Peter M. Athanas
F. Gail Gray
Lee Johnson
Jeffrey H. Reed

April, 2000
Blacksburg, Virginia

Keywords: Wireless Communications, Channel Coding, Turbo Codes, Parallel Concatenated Convolutional Code, Serial Concatenated Convolutional Code
Copyright 2000, Yufei Wu

Design and Implementation of Parallel and Serial Concatenated Convolutional Codes

Yufei Wu

(ABSTRACT)

Parallel concatenated convolutional codes (PCCCs), called "turbo codes" by their discoverers, have been shown to perform close to the Shannon bound at bit error rates (BERs) between $10^{-4}$ and $10^{-6}$. Serial concatenated convolutional codes (SCCCs), which perform better than PCCCs at BERs lower than $10^{-6}$, were developed borrowing the same principles as PCCCs, including code concatenation, pseudorandom interleaving and iterative decoding.

The first part of this dissertation introduces the fundamentals of concatenated convolutional codes. The theoretical and simulated BER performance of PCCC and SCCC are discussed. Encoding and decoding structures are explained, with emphasis on the Log-MAP decoding algorithm and the general soft-input soft-output (SISO) decoding module. Sliding window techniques, which can be employed to reduce memory requirements, are also briefly discussed.

The second part of this dissertation presents four major contributions to the field of concatenated convolutional coding developed through this research. First, the effects of quantization and fixed point arithmetic on the decoding performance are studied. Analytic bounds and modular renormalization techniques are developed to improve the efficiency of SISO module implementation without compromising the performance. Second, a new stopping criterion, SDR, is discovered. It is found to perform well with lowest cost when evaluating its complexity and performance in comparison with existing criteria. Third, a new type-II code combining automatic repeat request (ARQ) technique is introduced which makes use of the related PCCC and SCCC. Fourth, a new code-assisted synchronization technique is presented, which uses a list approach to leverage the simplicity of the correlation technique and the soft information of the decoder. In particular, the variant that uses SDR criterion achieves superb performance with low complexity.

Finally, the third part of this dissertation discusses the FPGA-based implementation of the turbo decoder, which is the fruit of cooperation with fellow researchers.

# Acknowledgments

This work has been made possible by the generous support from MPRG Industrial Affiliates Foundation and the Defense Advanced Research Projects Agency (DARPA) through the GloMo II project.

I would like to first thank my advisor, Dr. Woerner, for his support and guidance, not only in my study and research, but in many other aspects of life. Without his encouragement I could have given up the Ph.D. program halfway. I am also obliged to other professors of my committee for their assistance and suggestions towards my research.

Next I would like to thank the students and staff at MPRG. It would be impossible for me to switch to wireless communication smoothly and complete the Ph.D. program on time without the high quality students around me and the interaction between them. So many MPRG students have helped me with simulation and software application that it is hard to name them all. Special thanks should be given to Dr. Matthew C. Valenti and Dr. T. Keith Blankenship. Dr. Valenti was my mentor on turbo codes when I first started. He generously shared many research ideas, papers, and other information with me, even after he left for University of West Virginia. Dr. Blankenship, my boyfriend and fellow researcher, provided a tremendous amount of emotional and technical support throughout the past three years. He never hesitated to help me kindly in every way he could and in every step I made.

I am also thankful to Jason Hess, a former student with Virginia Tech Information System Center (VISC), who implemented my turbo decoder design on a FPGA board for the GloMo II project.

Last but not least, I would like to thank my grandma, my parents, and my sister for their endless love and encouragement throughout my life.

# Contents

# List of Tables

# List of Figures

# Abbreviations

| | |
|---|---|
| **API** | application programming interface |
| **APP** | *a posteriori* probability |
| **ARQ** | automatic repeat request |
| **AWGN** | additive white Gaussian noise |
| **BCH** | Bose-Chaudhuri-Hocquenghem (code) |
| **BCJR** | Bahl-Cocke-Jelinek-Raviv (algorithm) |
| **BER** | bit error rate |
| **BSC** | binary symmetric channel |
| **CCM** | configurable computing machine |
| **CE** | cross entropy |
| **CH-UC** | unconstrained AWGN channel |
| **CH-BI** | binary-input constrained AWGN channel |
| **CH-BIBO** | binary-input binary-output channel |
| **CLB** | configurable logic block |
| **CRC** | cyclic redundancy check (code) |
| **BPSK** | binary phase shift keying |
| **CWEF** | conditional weight enumerating function |
| **DSP** | digital signal processing |
| **FER** | frame error rate |
| **FEC** | forward error correction |
| **FF** | flip-flop |
| **FPGA** | field programmable gate array |
| **HCCC** | hybrid concatenated convolutional code |
| **HDL** | hardware description language |
| **IOB** | input/output buffer |
| **IOWEF** | input-output weight enumerating function |
| **LLR** | log-likelihood ratio |
| **LUT** | lookup table |
| **MAP** | maximum *a posteriori* |
| **ML** | most likelihood, most likely |
| **NAK** | negative acknowledgment |

| | |
|---|---|
| **PC** | personal computer |
| **PCCC** | parallel concatenated convolutional code |
| **pdf** | probability density function |
| **PE** | processing element |
| **PLD** | programmable logic device |
| **PN** | pseudonoise |
| **RF** | radio frequency |
| **RS** | Reed-Solomon (code) |
| **RSC** | recursive systematic convolutional (code) |
| **SCCC** | serial concatenated convolutional code |
| **SIR** | signal-to-interference ratio |
| **SCR** | sign change ratio |
| **SDR** | sign difference ratio |
| **SISO** | soft-input soft-output |
| **SNR** | signal-to-noise ratio |
| **SPW** | Signal Processing WorkSystem |
| **SRAM** | static random access memory |
| **SLVA** | serial list output Viterbi algorithm |
| **SOVA** | soft-output Viterbi algorithm |
| **SW-SISO** | sliding window soft-input soft-output |
| **TDM** | time division multiplex |
| **TTCM** | turbo trellis coded modulation |
| **VA** | Viterbi algorithm |
| **VHDL** | very high speed integrated circuit (VHSIC) HDL |

# Chapter 1

# Introduction

Error-control coding, an outgrowth of Shannon's Information Theory, has developed over the past fifty years from a mathematical curiosity into a fundamental element of almost any system that transmits or stores digital information. Many early coding applications were developed for deep-space and satellite communication systems. With the emergence of digital cellular telephony, digital television, and high-density digital storage, coding technology promises to predominate not only in scientific and military applications, but also in numerous commercial applications.

## 1.1 Composition of Digital Communication Systems

Recently, there has been an increasing demand for efficient and reliable digital communication systems. Large-scale high-speed networks have grown to transmit voice, image, data and other types of information. The major concern of design engineers is to minimize the error probability at the receiver end by making wise use of the power and bandwidth resources, while keeping the system complexity reasonable to minimize cost.

A block diagram of a basic communication system is shown in Figure 1.1 [1]. The information source may be analog or digital in nature. An analog signal would be sampled and quantized before transmission through a digital system.

The information source usually contains redundancy. There are either dependencies between successive symbols, or the probability of the occurrence of each symbol is not uniform. Thus, the source encoder is used to remove the redundancy before

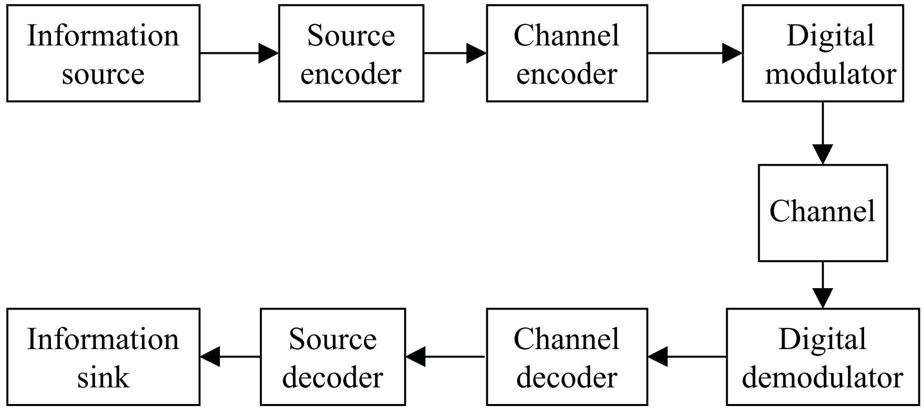



Figure 1.1: Block diagram of a communication system.

transmission so that the fewest number of bits are used to represent the information. The information source is transformed by the source encoder into a sequence of bits called the information sequence.

After the source encoder, the channel encoder purposely adds redundancy into the information sequence. Unlike the uncontrolled redundancy in the original information source which can not be used to improve system performance, the redundancy added by the channel encoder is generated in a *controlled* manner which increases reliability. Channel coding is a good way of achieving the necessary transmission fidelity with the available transmitter and receiver resources, such as power, bandwidth, and modulation technique.

Since bits are not suitable for transmission over a physical channel, the digital modulator is used to transform them into a continuous-time waveform for transmission. This waveform is sent over the physical channel. Typical transmission channels include wireline channels, fiber optic channels, wireless electromagnetic channels, underwater acoustic channels, and storage channels [2]. Whatever the medium, the transmitted signal will be distorted in a random manner by, e.g., the thermal noise generated by the electronic devices in radio frequency (RF) front end or the cosmic noise picked up by the antenna. At the receiver end, the digital demodulator processes the corrupted waveform and produces the estimation of the transmitted data.

The output of the demodulator is passed to the channel decoder which uses the redundancy and the knowledge of the channel code to detect and correct errors caused by the physical media.

Finally, the source decoder accepts the decoded bits and attempts to reconstruct the original information source with knowledge of the source encoding method.

Among all the above functional blocks, the channel encoding and decoding pair is our concern. Although coding and modulation are frequently treated together in highly bandlimited system using trellis coded modulation [3] [4] [5], coding and modulation are more commonly treated separately in energy-limited wireless systems.

## 1.2 Development of Channel Coding Techniques

The diagram in Figure 1.1 shows a one-way system, in that the transmission is strictly in the forward direction, from the transmitter to the receiver. In contrast to a two-way system which can use automatic repeat request (ARQ) with error detection and retransmission, the error control strategy for a one-way system must be forward error correction (FEC), which automatically corrects errors detected at the receiver. Most coded systems use some kind of FEC, even when the system is not strictly one-way [6]. FEC includes block codes, convolutional codes, as well as concatenated codes which build upon block and convolutional codes.

Block coding was the first coding technique developed. Block codes collect groups of $k_0$ information bits and independently map them to codewords of $n_0$ bits. Hamming discovered the code bearing his name, which was the first class of error correction codes to be described in a combinatorial, constructive manner [7]. After that, major breakthroughs were made with the discovery of Reed-Solomon (RS) codes and Bose-Chaudhuri-Hocquenghem (BCH) codes [8] [9] [10]. Binary BCH codes include the Hamming and Golay codes. However, it was discovered that binary BCH codes are asymptotically weak. On the other hand, RS codes, a subset of nonbinary BCH codes, are optimal in the maximum separable distance sense. However, good RS codes require an alphabet size that grows with the block length. Forney solved the asymptotic complexity issue by concatenating short random codes with long RS codes [11].

The first practical decoding algorithm for block codes was threshold decoding introduced by Reed [12]. After that, many methods were proposed, including solving simultaneous linear equations, sequential decoding, decoding low-density parity-check codes [13], and most importantly, algebraic decoding such as the Berlekamp-Massey algorithm for RS codes [14] [15] [16] [17].

Fundamentally different from block codes, convolutional codes have memory so that the mapping from the $k_0$ information bits to the $n_0$ code bits is a function of the past information bits. Convolutional codes were first invented by Elias [18], who proved that randomly chosen codes of this type were good.

Sequential decoding [19] was the first practical decoding algorithm discovered for long randomly chosen convolutional codes. This decoding method has a variable computation characteristic and is ultimately limited by the computational cutoff rate [20], where the number of operations becomes unbounded. Threshold decoding is a simple method which can be used for convolutional codes with certain constraints [21] [6]. An exceptional discovery was the Viterbi algorithm (VA) [22], which works extremely well when the constraint length is small. The VA has a fixed number of computations per step; therefore it is not limited by a computational cutoff rate. The VA, which minimizes sequence error probability, has gained a wide application not only in channel decoding, but also in many other fields which involve maximum likelihood estimation of the states of a Markov chain, e.g., speech recognition. As an alternative to the VA, the maximum *a posteriori* (MAP) algorithm, which minimizes bit error probability, was introduced by Bahl *et al.* to decode convolutional codes [23]. MAP decoding was not widely employed until the discovery of turbo codes because it involves more than twice the complexity, but provides similar performance to the VA when applied to convolutional codes.

It has been found that with similar complexity, larger coding gains can be achieved by code concatenation. Forney first introduced concatenation in 1966 [11], where an inner code and an outer code were used in cascade. A common structure is a powerful nonbinary RS outer code followed by a short constraint length inner convolutional code with soft-decision Viterbi decoding [24]. A symbol interleaver is used between the inner and outer code so that long bursty errors from the inner decoder are broken into separate blocks for the outer decoder [25]. In addition, iterative decoding was exploited to improve the performance of the concatenated codes. A general approach where both the inner and the outer code produce reliability information to help each other improve the performance, was proposed with the introduction of the soft-output Viterbi algorithm (SOVA) [26].

With an ingenious application of the existent ingredients, a major breakthrough was made by Berrou *et al.* with the discovery of "turbo codes" [27]. Turbo codes achieve performance close to the Shannon limit with the combination of two or more recursive convolutional codes, a pseudorandom interleaver, and a MAP iterative decoding algorithm. Turbo code performance about 0.5 dB away from Shannon capacity at BER= $10^{-5}$ is usually achievable with short constraint length, long block length, and 10 to 20 decoding iterations.

Prompted by the discovery of turbo codes, which are also called parallel concatenated convolutional codes (PCCCs), serial concatenated convolutional codes (SCCCs) and hybrid concatenated convolutional codes (HCCCs) were constructed with the same components to provide similar coding gains [28]. SCCCs and HCCCs can perform better than PCCCs at high signal-to-noise ratio, because of a superior distance profile. In addition to convolutional codes, block codes, such as Hamming codes and RS codes, can also be used as the constituent codes in the concatenation.

## 1.3 Outline of Dissertation

This research has attempted to bring the initial promise of turbo codes closer to practical reality. Both design and implementation issues for PCCCs and SCCCs are addressed.

In Chapter 2, the fundamentals of channel coding technology are presented. Starting with the computation of channel capacity, channel coding bounds are obtained for three typical channel models: unconstrained AWGN channels, binary-input constrained AWGN channels, and binary-input binary-output channels. Achieving these bounds is the ultimate goal of coding researchers. This problem can be stated in many ways, including achieving the minimum bit error probability for a given code rate, achieving the lowest possible signal-to-noise ratio to provide a certain performance, or achieving the largest coding gain possible for a given set of parameters. On the other hand, the bounds are the limits of any possible code, and can be used as a guide for selecting the coding parameters for a specified performance. For the given PCCC and SCCC structures, theoretical analysis is presented to explain the reasons they excel. Interleaver gain, a key feature of PCCCs and SCCCs in comparison to convolutional codes, is emphasized in this explanation. Following the analysis, design guidelines are summarized.

In Chapter 3, the conventional Log-MAP algorithm for PCCCs is presented. As an extension, an enhanced implementation of the general soft-input soft-output decoding technique is developed. Then, the procedures to decode PCCCs and SCCCs are shown for the general decoding module. For continuous transmission or blockwise transmission with very large frame sizes, a sliding window could be incorporated into the algorithm to reduce memory requirements and delay. Finally, a novel method is proposed to compute the backward metrics in a forward manner so that the storage requirement of the backward path metrics is reduced. A stability analysis is presented for the proposed scheme.

In Chapter 4, a comprehensive simulation study is presented to show the performance of PCCCs and SCCCs. The influence of various parameters, such as the number of iterations, frame size, code rate, and code generators, is illustrated with the examples presented. Also, the differences between PCCCs and SCCCs are clarified.

In Chapter 5, issues related to integer representation and fixed-point arithmetic are discussed. First the issue of quantization noise is addressed. A method is presented to minimize the quantization error by finding the optimal gain to scale the received signal properly. Simulations show that 8-bit quantization of the received signals is enough to provide performance without noticeable degradation. Then computation inside the SISO decoding module is examined. Analytical bounds on path metric differences and the soft output are derived, which are applied to find the minimal data width in the decoder. All the bounds are shown to be tight and accurate by simulations. In addition, a modular renormalization technique is introduced to replace subtractive normalization, which reduces the path metrics computation.

In Chapter 6, stopping criteria are examined. The judicious choice of stopping criteria can significantly save computational power by dynamically terminating the iteration of the decoder. A new SDR criterion is reported and its performance is compared with existing stopping criteria. Simulations show that it achieves BER/FER performance and average number of iterations similar to existing criteria with minimal computation and no storage requirements.

In Chapter 7, a new type-II code combining ARQ scheme is proposed which makes use of the related PCCCs and SCCCs. It is found that by reordering the bits in the encoder and using a special interleaver, the PCCC bits are actually a subset of the corresponding SCCC bits. This allows the ARQ system to transmit PCCC bits in the first transmission, and transmit the remaining SCCC bits only when requested. Thus the PCCC decoder can be used for the first transmission, with the SCCC decoder used for subsequent transmissions. Simulations show that the new scheme has excellent performance including BER, FER, throughput efficiency, and complexity load.

In Chapter 8, frame synchronization of the turbo coding system is discussed. A novel list synchronization technique is developed by refining the simple correlation method. Two variations, LLR sync and SDR sync, are discussed and tested. Their curves of false sync rate have much steeper slopes than those of conventional correlation synchronization technique. In addition, SDR sync is found to perform better than LLR sync with much lower computational complexity and delay, especially in the low BER region that turbo codes operate in (i.e., BER$< 10^{-3}$).

In Chapter 9, an FPGA implementation of the turbo decoder is presented, which is accomplished by cooperation with fellow researchers. The high level system design is described with emphasis on SPW modeling. The low level design flow and the implementation results are also shown. Four suggestions are provided to improve the performance of a future implementation.

Finally in Chapter 10, the contributions of this dissertation are summarized, and possible areas for future work to extend this research are outlined.

# Chapter 2

# Theoretic Foundation of Concatenated Codes

This chapter introduces the fundamental characteristics of PCCC and SCCC, with emphasis on the reasons that they outperform other coding techniques such as block codes and convolutional codes. First, performance bounds for coding techniques in general are established. Then the structures of PCCC and SCCC systems are introduced and an analysis of their performance is presented.

## 2.1 Channel Model

A communication system model is shown in Figure 2.2 to facilitate the analysis of channel coding. The modulator and the demodulator are included as a part of the composite channel [2]. If the channel noise is additive white Gaussian noise (AWGN), the modulator accepts binary inputs, and the demodulator makes hard decisions, then the composite channel is a binary symmetric channel (BSC). More generally, if the input of the modulator is a symbol selected from a finite, discrete alphabet, and the output of the demodulator is unquantized, then the composite channel is a discrete-input continuous-output channel. This channel model is often used for turbo codes, since the decoder requires a soft output from the demodulator.

In the system the information vector $\mathbf{u}$ of dimension $N$ is encoded into a code vector $\mathbf{c}$ of dimension $N_f$. The code bit is mapped to determine the channel input $X$. The channel output signal is $Y = X + Z$, where $Z$ is the additive channel noise. The terms $X$, $Y$, and $Z$ are treated as random variables, and their respective values

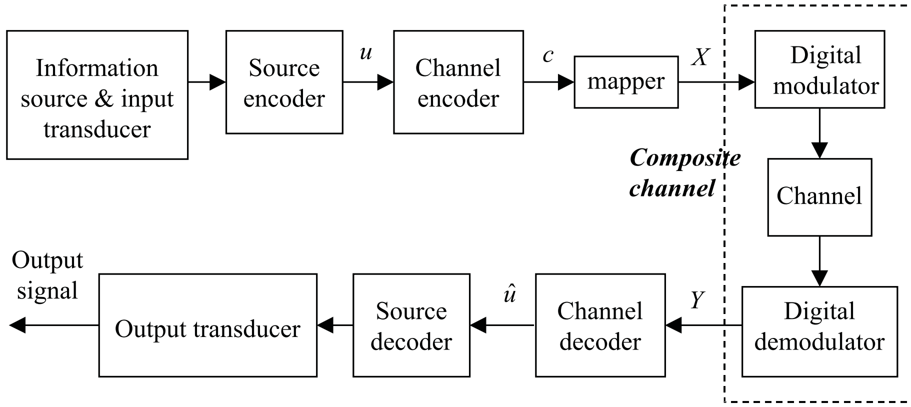



Figure 2.2: A general communication system.

are denoted by $x_i$, $y_i$, and $z_i$, where $y_i = x_i + z_i$.

## 2.2 Channel Capacity

In his 1948 paper, "A Mathematical Theory of Communication" [29], Claude Shannon introduced the concept of channel capacity $C$, which is defined as

$$C = \max_{p(x)} I(X, Y), \tag{2.1}$$

where $p(x)$ is the probability density functions (pdf) of $X$ and $I(X, Y)$ is the mutual information between $X$ and $Y$, given by

$$I(X, Y) = \begin{cases} \sum_x \sum_y p(x, y) \log_2 \frac{p(x,y)}{p(x)p(y)}, & \text{for a discrete channel,} \\ \int p(x, y) \log_2 \frac{p(x,y)}{p(x)p(y)}, & \text{for a continuous channel.} \end{cases} \tag{2.2}$$

where $p(y)$ is the pdf of $Y$ and $p(x, y)$ is the joint pdf of $X$ and $Y$. For a Gaussian channel, the capacity is

$$C = \frac{1}{2} \log_2 \left(1 + \frac{P}{\sigma^2}\right) \quad \text{bits per transmission,} \tag{2.3}$$

where $Z \sim \mathcal{N}(0, \sigma^2)$, and $P$ is the power constraint: $\frac{1}{N_f} \sum_{i=1}^{N_f} x_i^2 \leq P$, where $N_f$ is the number of symbols transmitted.

For a bandlimited AWGN channel with bandwidth $B$ and signal power $P$, the capacity is given by

$$C = B \log_2 \left(1 + \frac{P}{N_0 B}\right) \quad \text{bits/sec,} \tag{2.4}$$

where $N_0/2$ is the two-sided noise power spectral density. If perfect Nyquist signaling is assumed, then $P = E_s/\Delta T$, where $E_s$ is the average signal energy in each signaling interval of duration $\Delta T$. Conceptually, $C$ is the number of information bits per second which can be transmitted theoretically with arbitrarily low error rate over the channel. For a fixed bandwidth $B$, $C$ increases with an increase in the transmitted signal power. On the other hand, if $P$ is fixed, the capacity can be increased by increasing the bandwidth $B$. When $B \to \infty$, the channel capacity approaches its asymptotic value [2]:

$$C_\infty = \frac{P}{N_0 \ln 2} \quad \text{bits/sec.} \tag{2.5}$$

## 2.3 Channel Coding Tradeoffs

The **noisy channel coding theorem** [29] states the following: there exist channel codes (and decoders) that make it possible to achieve reliable communication with as small an error probability as desired, provided that the code rate $r < C$, where $C$ is the channel capacity; conversely, if $r > C$, it is impossible to make the probability of error tend towards zero with any code.

The channel coding theorem proves the existence of good codes, but it does not provide a way to construct them. From the proof of the theorem, it is known that a code with randomly chosen codeword sequences will turn out to be a good code with high probability. However, the complexity of its maximum likelihood (ML) decoder is proportional to the number of codewords. Thus, the problem amounts to finding a code with distance properties that approximate "random" codes, but with sufficient structure to allow efficient decoding.

It has been shown by Gallager [30] that for any discrete-input memoryless channel, there exists an $N_f$-symbol code of rate $r$ for which the word error probability with maximum likelihood decoding is bounded by

$$P_w(e) < \exp[-N_f \cdot \mathrm{E}(r)], \quad 0 \le r \le C, \tag{2.6}$$

where $\mathrm{E}(r)$ is a convex decreasing positive function of $r$. From this expression, the following three ways are found to provide reliable transmission, each of which is associated with some tradeoff.

1. Decrease the code rate $r = N/N_f$, as in Figure 2.3. However, for a given source rate, decreasing $r$ means increasing the transmission rate, and hence increasing

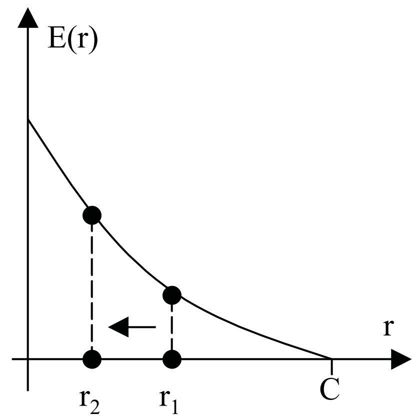



Figure 2.3: $E(r)$ increases when $r$ decreases.

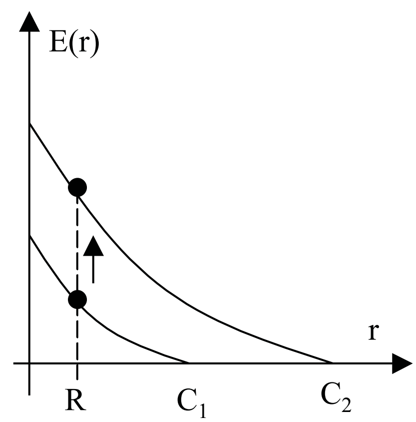



Figure 2.4: $E(r)$ increases when $C$ increases.

the required bandwidth.

2. Increase the channel capacity $C$, as in Figure 2.4. For a given rate $r$, increasing $C$ makes $E(r)$ increase. However, to increase the channel capacity for a given channel and a given bandwidth, the signal power $P$ has to be increased.

3. Increase the code length $N_f$, while keeping $r$ and $E(r)$ fixed, so that both the bandwidth and the power remain the same. In this case, the performance improves at the expenses of the decoding complexity. In fact, for randomly chosen codes and maximum likelihood decoding, the decoding complexity $J$ is on the order of the number of codewords, or,

$$\begin{aligned} J &\propto 2^N = 2^{N_f r} \\ &\propto \exp(N_f r) \end{aligned}$$

Consequently,

$$\begin{aligned} P_w(e) &< \exp[-N_f \mathrm{E}(r)] \\ &\propto \exp\left[-\ln J\left(\frac{\mathrm{E}(r)}{r}\right)\right] \\ &\propto J^{-\mathrm{E}(r)/r} \end{aligned} \tag{2.7}$$

This implies that the word error probability decreases only algebraically with respect to the decoding complexity.

In many cases, the bandwidth efficiency and the power efficiency cannot be compromised but higher decoding complexity is allowed. Fortunately, unlike random codes, good codes and decoding algorithms can be designed so that the decoding complexity increases linearly with $N$. Two examples are the convolutional codes (with the VA) and the concatenated codes (with MAP or SOVA), whose decoding complexity per information bit is decoupled from the frame size, i.e.,

$$J \propto N = N_f r$$

for the entire frame. Thus,

$$\begin{aligned} P_w(e) &< \exp[-N_f \mathrm{E}(r)] \\ &\propto \exp\left(-J\frac{\mathrm{E}(r)}{r}\right) \end{aligned} \tag{2.8}$$

This implies that the word error probability decreases exponentially with an increase in the decoding complexity. When it comes to the bit error probability $P_b(e)$, convolutional codes and concatenated codes behave differently because of the different mapping from the information weight to the codeword weight. It was found that $P_b(e)$ decreases much more dramatically with an increase of frame size for concatenated codes than for convolutional codes. Thus, when the frame size is sufficiently large ($N > 200$), concatenated codes have lower $P_b(e)$ than convolutional codes with the same decoding complexity.

## 2.4 Channel Coding Bounds

If the information bits are received independently with error probability $P_b(e)$, then the information capacity is $1 - H_b(e)$ [31], where

$$H_b(e) = -P_b(e)\log_2 P_b(e) - (1 - P_b(e))\log_2(1 - P_b(e)) \tag{2.9}$$

is the entropy of a binary source with error probability $P_b(e)$. After channel encoding with code rate $r$, each code bit contains $r(1 - H_b(e))$ information. Since $r(1 - H_b(e))$ cannot exceed channel capacity $C$, the code rate $r$ is bounded by the following:

$$r \leq \frac{C}{1 - H_b(e)}. \tag{2.10}$$

From this inequality, several theoretic coding bounds can be derived. In this dissertation three types of channels are discussed. The first is the unconstrained AWGN channel (CH-UC), where the coding alphabet is unconstrained and both the input and the output can have any distribution. The second is the binary-input constrained AWGN channel (CH-BI), where the input is binary, but the output can assume any value. The third is the binary-input binary-output channel (CH-BIBO), where both the input and the output are constrained to be binary. Since the channel noise is assumed to be AWGN, CH-BIBO is equivalent to a BSC.

### 2.4.1 $P_b(e)$ Bounds

#### CH-UC

Let $E_b$ be the average signal energy per binary symbol when channel coding is not used. Thus, when channel coding of rate $r$ is applied, the average power $P$ of the transmitted signal is $P = rE_b/T$, where $T$ is the bit time duration. When $N_0/2$ is the two sided power spectral density of the channel noise, and the bandwidth is $B$, the average noise power is $\sigma^2 = N_0 B$. Substituting these relationships into Equation (2.3) and assuming the Nyquist minimum bandwidth (i.e., $B = 1/2T$), the capacity of CH-UC is obtained:

$$\begin{aligned} C &= \frac{1}{2}\log_2\left(1 + \frac{rE_b}{N_0 BT}\right) \\ &= \frac{1}{2}\log_2\left(1 + \frac{2rE_b}{N_0}\right) \qquad \text{bits per transmission,} \end{aligned} \tag{2.11}$$

which is achieved with a Gaussian input. Considering Equation (2.10), the code rate is thus bounded by

$$r \leq \frac{\log_2\left(1 + \frac{2rE_b}{N_0}\right)}{2(1 - H_b(e))}. \tag{2.12}$$

When Equation (2.12) is satisfied with equality, the channel capacity bound is achieved for CH-UC. For code rates $r$ between 0 and 1, $P_b(e)$ limits are numerically calculated and plotted in Figure 2.5.

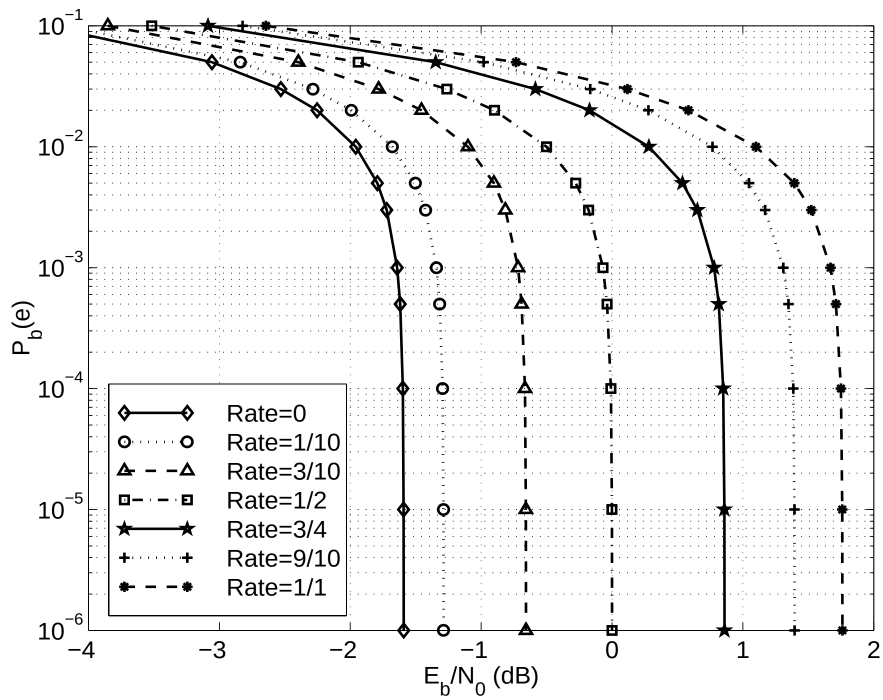



Figure 2.5: Bit error probability bounds for CH-UC with various code rates.

When $r \to 0$, the transmission bandwidth approaches infinity, and the lowest possible $E_b/N_0$ is achieved for transmission with arbitrarily low bit error probability. If $r \to 0$, Equation (2.12) yields

$$\begin{aligned}
\lim_{r\to 0}[1 - H_b(e)] &= \lim_{r\to 0} \frac{\log_2\left(1 + \frac{2rE_b}{N_0}\right)}{2r} \\
&= \lim_{r\to 0} \frac{d\left(\log_2\left(1 + \frac{2rE_b}{N_0}\right)\right)/dr}{d(2r)/dr} \\
&= \lim_{r\to 0} \frac{\frac{2E_b/N_0}{1+2rE_b/N_0}}{2\ln 2} \\
&= \frac{E_b}{(\ln 2)N_0} \qquad (2.13)
\end{aligned}$$

When $P_b(e) \to 0$, $1 - H_b(e) \to 1$. This implies that to achieve reliable transmission, the minimum $E_b/N_0$ for all coding schemes is $E_b/N_0 = \ln 2 (= -1.6 \text{ dB})$.

**CH-BI**

When the input of the AWGN channel is constrained to be binary, i.e., $(\sqrt{E_s}, -\sqrt{E_s})$, but the output is not limited, the channel capacity is found to be

$$C = \frac{2rE_b}{(\ln 2)N_0} - \frac{1}{\sqrt{2\pi}} \int \exp\left(\frac{-t^2}{2}\right) \cdot \log_2 \cosh\left(t\sqrt{\frac{2rE_b}{N_0}} + \frac{2rE_b}{N_0}\right) dt, \tag{2.14}$$

as shown in Appendix B.

Substituting Equation (2.14) into Equation (2.10) and taking the equality, a bound for the bit error probability versus $E_b/N_0$ is obtained for a given $r$. In Figure 2.6, the bounds on $P_b(e)$ for various code rates are plotted.

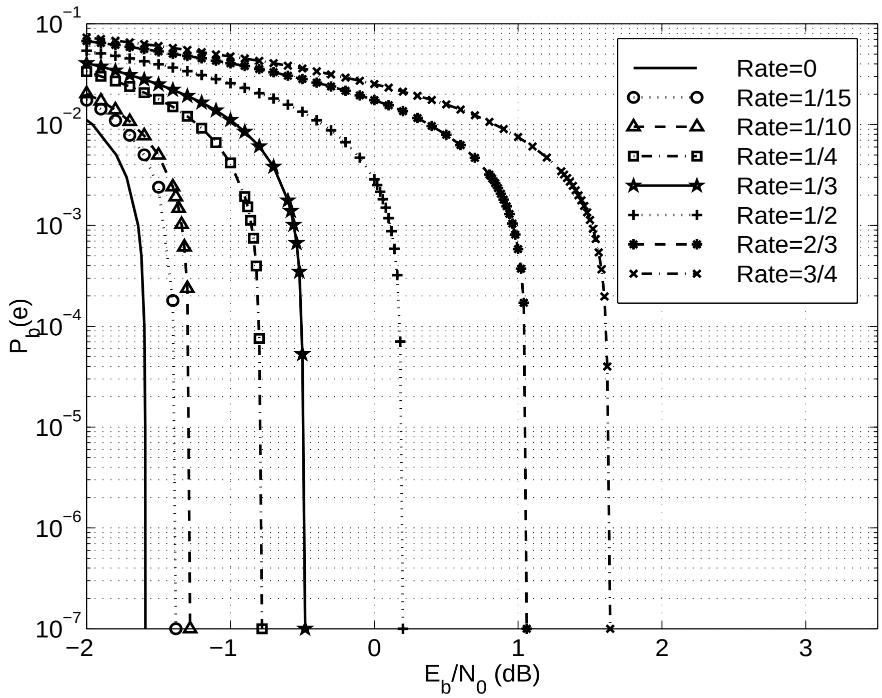



Figure 2.6: Bit error probability bounds for CH-BI with various code rates.

As $r \to 0$,

$$\begin{aligned} \lim_{r\to 0}[1 - H_b(e)] &= \lim_{r\to 0} \frac{C}{r} \\ &= \frac{2E_b}{(\ln 2)N_0} - \lim_{r\to 0} \frac{1}{\sqrt{2\pi}r} \int \exp\left(\frac{-t^2}{2}\right) \cdot \log_2 \cosh\left(t\sqrt{\frac{2rE_b}{N_0}} + \frac{2rE_b}{N_0}\right) dt. \end{aligned} \tag{2.15}$$

If we let

$$t' = t\sqrt{\frac{2rE_b}{N_0}} + \frac{2rE_b}{N_0},$$

then

$$\begin{aligned} \frac{dt'}{dr} &= t\sqrt{\frac{E_b}{2N_0}}r^{-1/2} + \frac{2E_b}{N_0} &(2.16)\\ \lim_{r\to 0} t' &= 0 &(2.17)\\ \lim_{r\to 0}\frac{\sinh t'}{\sqrt{r}} &= \lim_{r\to 0}\frac{(\cosh t')\left(t\sqrt{\frac{E_b}{2N_0}}r^{-1/2} + \frac{2E_b}{N_0}\right)}{0.5r^{-1/2}} = t\sqrt{\frac{2E_b}{N_0}}. &(2.18)\end{aligned}$$

Thus,

$$\begin{aligned} \lim_{r\to 0}[1 - H_b(e)] &= \frac{2E_b}{(\ln 2)N_0} - \frac{1}{\sqrt{2\pi}}\lim_{r\to 0}\int \exp\left(\frac{-t^2}{2}\right)\cdot \log_2 e \cdot \frac{\sinh t'}{\cosh t'} \\ &\quad \cdot \left(t\sqrt{\frac{E_b}{2N_0}}r^{-1/2} + \frac{2E_b}{N_0}\right) dt \\ &= \frac{2E_b}{(\ln 2)N_0} - \frac{1}{\sqrt{2\pi}}\int \exp\left(\frac{-t^2}{2}\right)\cdot \log_2 e \cdot t^2\frac{E_b}{N_0}dt \\ &= \frac{2E_b}{(\ln 2)N_0} - \frac{1}{\sqrt{2\pi}}\frac{E_b}{(\ln 2)N_0}\cdot\sqrt{2\pi} \\ &= \frac{E_b}{(\ln 2)N_0}, &(2.19)\end{aligned}$$

where $\int t^2 \exp\left(\frac{-t^2}{2}\right) dt = \sqrt{2\pi}$ is applied. Thus, analogous to CH-UC, the $E_b/N_0$ limit of CH-BI is also $E_b/N_0 = \ln 2 (= -1.6\text{ dB})$, as $r \to 0$.

### 2.4.2 Minimum $E_b/N_0$

Here the minimum $E_b/N_0$ required to achieve arbitrarily low bit error probability for any rate $r$ is considered. When $P_b(e) \to 0$ and thus $H_b(e) \to 0$, Equation (2.10) becomes

$$r \le C. \tag{2.20}$$

We consider this limiting case for the three channels of interest.

1. For CH-UC, Equation (2.20) implies

$$r \le \frac{1}{2}\log_2\left(1 + \frac{2rE_b}{N_0}\right). \tag{2.21}$$

   Taking the equality, we obtain

$$\frac{E_b}{N_0} = \frac{2^{2r}-1}{2r}. \tag{2.22}$$

2. For CH-BI, Equation (2.20) becomes

$$r \leq \frac{2rE_b}{(\ln 2)N_0} - \frac{1}{\sqrt{2\pi}} \int \exp\left(\frac{-t^2}{2}\right) \cdot \log_2 \cosh\left(t\sqrt{\frac{2rE_b}{N_0}} + \frac{2rE_b}{N_0}\right) dt. \quad (2.23)$$

3. For CH-BIBO, Equation (2.20) implies

$$r \leq C = 1 - H(P_s(e)) = 1 + P_s(e)\log_2 P_s(e) + (1 - P_s(e))\log_2(1 - P_s(e)), \quad (2.24)$$

with the binary channel symbol error probability

$$P_s(e) = Q\left(\sqrt{\frac{2rE_b}{N_0}}\right), \quad (2.25)$$

where

$$Q(x) = \frac{1}{\sqrt{2\pi}} \int_x^\infty \exp(-t^2/2)dt, \quad (2.26)$$

when BPSK is used for transmission.

Taking equalities in Equations (2.21), (2.23), and (2.24), the relationships between the minimum $E_b/N_0$ and $r$ can be found for errorless transmission. The resulting curves are drawn in Figure 2.7, where numerical calculation is used for Equations (2.23) and (2.24).

From Figure 2.7, we see that, at $r < 1/3$, the power requirement for CH-BI is approximately the same as that for CH-UC. The difference increases significantly for greater values of $r$; however, the difference between CH-BIBO and CH-BI decreases as $r$ increases. This can be interpreted as the gap between soft decision decoding and hard decision decoding. It can be concluded that, for most coding rates employed in practice, there is a $1.5 \sim 2$ dB gain by employing soft decision decoding instead of hard decision decoding.

### 2.4.3 Maximum Achievable Coding Gain

The coding gain of a system is the difference between the $E_b/N_0$ required to achieve a specified $P_b(e)$ without coding and the $E_b/N_0$ required to achieve the same $P_b(e)$ with coding. An upper bound on the coding gain of any channel coding scheme is found by letting $r \to 0$. Again, three types of channels are discussed below.

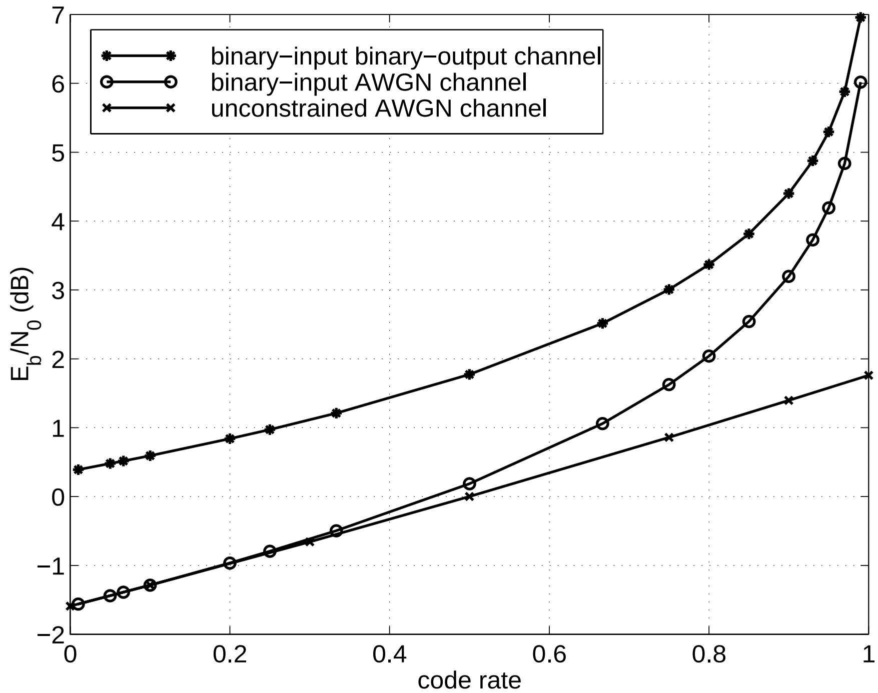



Figure 2.7: Minimum $E_b/N_0$ versus code rate $r$ for CH-UC, CH-BI and CH-BIBO.

1. For CH-UC, the information capacity limit is derived in Equation (2.13). Rewriting it, we obtain

$$\begin{aligned}\frac{E_b}{N_0} &= \ln 2(1 - H_b(e)) \\ &= \ln 2(1 + P_b(e)\log_2 P_b(e) + (1 - P_b(e))\log_2(1 - P_b(e))). \quad (2.27)\end{aligned}$$

2. Equation (2.19) indicates that CH-BI has the same performance as CH-UC when $r \to 0$.

3. For CH-BIBO,

$$\begin{aligned}\lim_{r\to 0}[1 - H_b(e)] &= \lim_{r\to 0}\frac{C}{r} \\ &= \lim_{r\to 0}\frac{1 + P_s(e)\log_2 P_s(e) + (1 - P_s(e))\log_2(1 - P_s(e))}{r} \\ &= \lim_{r\to 0}\Big[P_s'(e)\log_2 P_s(e) + P_s(e)\frac{1}{(\ln 2)P_s(e)}P_s'(e) \\ &\quad -P_s'(e)\log_2(1 - P_s(e)) + (1 - P_s(e))\frac{-P_s'(e)}{(\ln 2)(1 - P_s(e))}\Big] \\ &= \lim_{r\to 0} P_s'(e)(\log_2 P_s(e) - \log_2(1 - P_s(e)))\end{aligned}$$

$$
\begin{aligned}
&= \lim_{r\to 0} -\frac{1}{2}\sqrt{\frac{E_b}{\pi N_0}}\frac{\frac{P_s'(e)}{\ln 2}\left(\frac{1}{P_s(e)}+\frac{1}{1-P_s(e)}\right)}{0.5r^{-1/2}} \\
&= \lim_{r\to 0} -\frac{4}{\ln 2}\sqrt{\frac{E_b}{\pi N_0}}\frac{\left(-\frac{1}{2}\sqrt{\frac{E_b}{\pi N_0}}\exp\left(-\frac{rE_b}{N_0}\right)r^{-1/2}\right)}{r^{-1/2}} \\
&= \frac{2}{(\ln 2)\pi}\frac{E_b}{N_0}, \qquad (2.28)
\end{aligned}
$$

where BPSK modulation is assumed. Here $P_s(e)$ is expressed in Equation (2.25) and the following relationships are used in the derivation of Equation (2.28):

$$
\lim_{r\to 0} P_s(e) = \lim_{r\to 0} Q\left(\sqrt{\frac{2rE_b}{N_0}}\right) = \frac{1}{2}, \qquad (2.29)
$$

$$
\frac{dP_s(e)}{dr} = -\frac{1}{2}\sqrt{\frac{E_b}{\pi N_0}}\exp\left(-\frac{rE_b}{N_0}\right)r^{-1/2}. \qquad (2.30)
$$

Equation (2.28) shows that to achieve errorless transmission, i.e., $P_b(e) \to 0$ and thus $H_b(e) \to 0$, the $E_b/N_0$ lower bound is $E_b/N_0 = (\ln 2)\pi/2 (= 0.37$ dB).

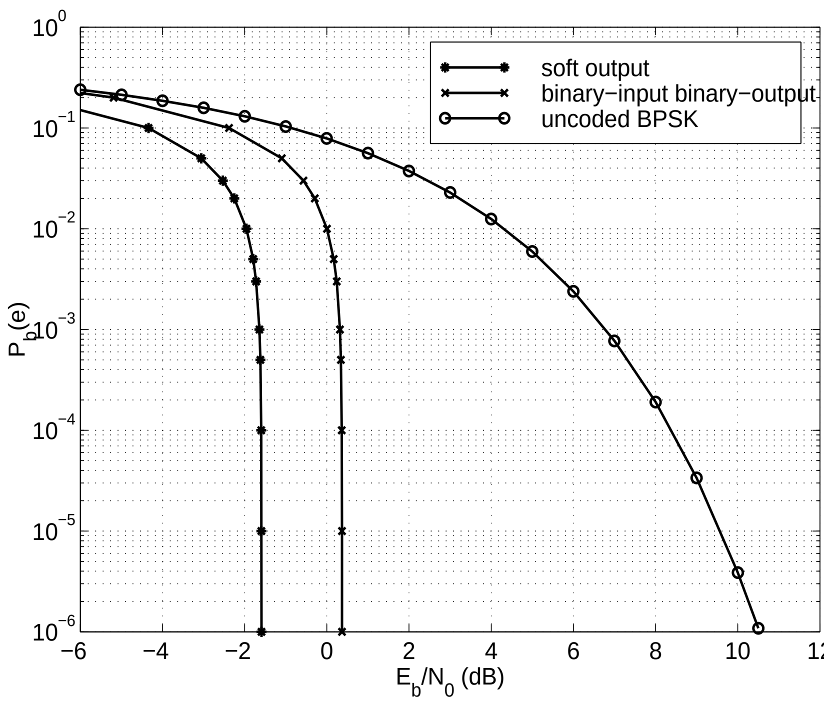



Figure 2.8: Lower bound of $P_b(e)$ for CH-UC/CH-BI and CH-BIBO.

Summarizing the above results, two curves are plotted in Figure 2.8 for Equations (2.27) and (2.28). They give the lowest possible bit error probability among all schemes for each channel type. The curve

$$P_b(e) = Q\left(\sqrt{\frac{2E_b}{N_0}}\right) \tag{2.31}$$

is also plotted for uncoded BPSK modulation as a comparison [1] [32].

In Figure 2.9, the largest possible coding gain for both channels are shown. Clearly, the coding gain decreases as $P_b(e)$ increases. There will be no coding gain, or even negative coding gain, when the channel is badly corrupted. Also, as $r \to 0$, CH-UC and CH-BI provide about 2 dB more coding gain than CH-BIBO.

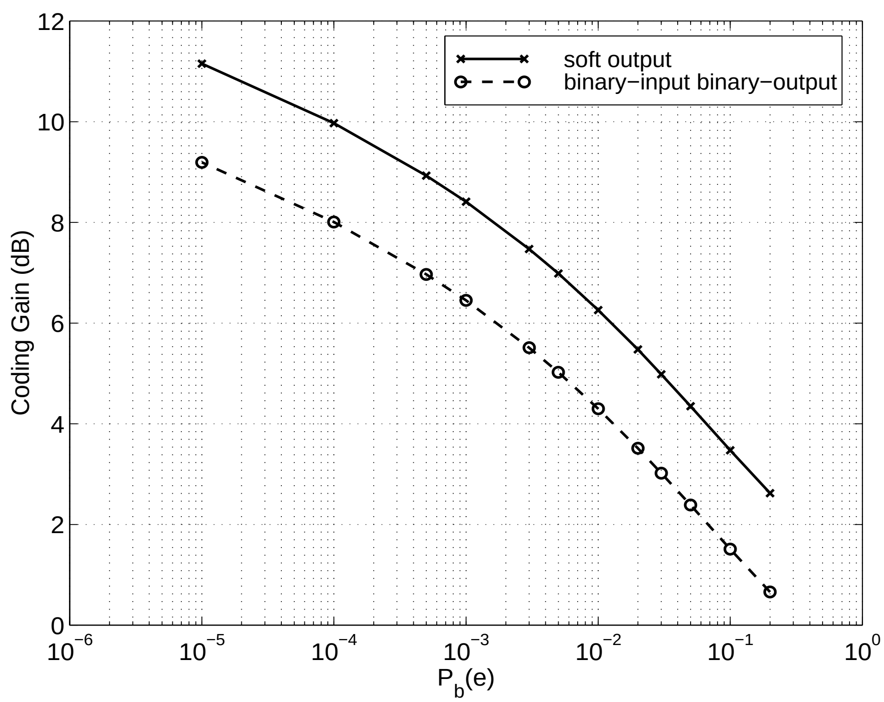



Figure 2.9: Upper bound of coding gain for CH-UC/CH-BI and CH-BIBO.

## 2.5 Theoretic Performance Bounds for PCCC and SCCC

The theoretical bounds presented in the above section are the ultimate limits that actual coding techniques attempt to approach. Since the channel coding theorem was proved by Claude Shannon [29], researchers have searched for codes which provide vanishingly small error probabilities at practical rates. Efforts have been invested in finding codes which appear random so as to provide a coding gain while retaining enough structure so that a decoder with reasonable complexity is possible. A major breakthrough was made by Berrou *et al.* [27], who discovered the so-called "turbo codes." Turbo codes were demonstrated to have performance within 0.5 dB of the Shannon limit around BER=$10^{-5}$ [33] [34].

Turbo codes, also called PCCCs, have generated a large body of research exploring their advantageous features and extending their principles to other structures [35] [36] [37] [38]. SCCCs were an important related discovery made by Benedetto *et al.* [39], which were found to outperform PCCCs at high $E_b/N_0$ in terms of both bit error probability and frame error probability.

### 2.5.1 Structure of PCCC

The original turbo code is the combination of two recursive systematic convolutional (RSC) codes, a pseudorandom interleaver, and an iterative MAP decoder.

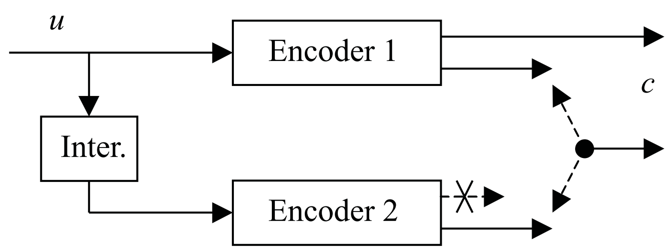



Figure 2.10: Turbo (PCCC) encoder.

The turbo encoder is illustrated in Figure 2.10. The input information bits feed the first encoder and enter the second encoder after being scrambled by the interleaver. The output sequence consists of the information bits and the parity bits of both encoders. Puncturing, which deletes some codeword bits according to a chosen pattern, can be used to increase the code rate of the overall code [27]. The encoder can be generalized to consist of $n_{ce}$ parallel cascaded constituent encoders joined by $(n_{ce} - 1)$ interleavers.

The function of the pseudorandom interleaver is to move the original $\alpha(k)$-th entry to the $k$-th position, where $\alpha(k)$, $k \in \{1, ..., N\}$, denotes the size-$N$ interleaver mapping. The interleaver has the effect of matching low-weight[1] parity sequences

[1]Weight is the number of nonzero bits in a codeword sequence. The weight distribution of a linear code is equivalent to the distance distribution, and determines the bit error probability performance.

with high-weight parity sequences in almost all cases, thus generating a code with very few low-weight output sequences. Usually the free distance[2] of turbo codes is not large. The excellent performance around BER=$10^{-4} \sim 10^{-6}$ is due instead to a drastic reduction in the number of nearest neighboring codeword sequences, in comparison to a conventional convolutional code [40].

Another important feature of turbo codes is the iterative decoder, which uses a soft-input soft-output MAP decoding algorithm [41]. The constituent codes are alternately decoded, and the "extrinsic" information at each stage is passed to the next decoding stage. In this way the information is shared between two constituent decoders. The iterative decoding procedure tends to converge as the number of iterations increases [42].

At almost any bandwidth efficiency (the ratio of bit rate to transmission bandwidth), performance about 0.5 dB away from the capacity limit at BER= $10^{-5}$ is achievable with short constraint length[3] turbo codes, very large frame sizes, and 10 to 20 decoding iterations. A major disadvantage of turbo codes is their long decoding delay, resulting from the large frame size and iterative decoding. Another disadvantage is the weaker performance at low BER (usually BER$< 10^{-6}$) because of their low free distance [40].

### 2.5.2 Structure of SCCC

Using the same components as turbo codes, such as constituent encoders, interleavers, rate converters (puncturer), and soft-input soft-output MAP decoders, another type of concatenated code, SCCCs, was proposed [39]. The superior performance of SCCCs has inspired much investigation and application in the coding field.

A serial concatenated code was first conceived by Forney [11]. It was shown that the probability of error for serial concatenated codes decreases exponentially as the frame size increases at rates less than capacity while decoding complexity increases only algebraically. The best known example is a RS outer code concatenated with a convolutional inner code separated by an interblock symbol interleaver. A SCCC is the result of combining the features of serial concatenated codes with those of turbo codes. Unlike the symbol interleaver between the RS and the convolutional code, a

[2]Free distance is the minimum Hamming distance between all pairs of complete codeword sequences.

[3]Constraint length is the number of past inputs affecting the current outputs. It is equal to one plus the number of stages in the encoder shift register.

bit interleaver is used in SCCCs to introduce randomness into the codeword sequence.

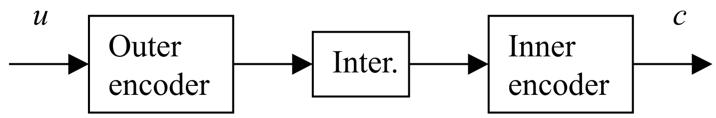



Figure 2.11: SCCC encoder.

The basic structure of a SCCC encoder is shown in Figure 2.11. The information bits are encoded by the outer encoder, whose output sequence is passed to the bit interleaver. The bit interleaver permutes the output of the outer encoder and then passes it as the input to the inner encoder. The output of the inner encoder is transmitted through the channel. The general structure of a SCCC encoder encompasses $n_{ce}$ serially cascaded constituent encoders separated by $(n_{ce} - 1)$ interleavers.

The pseudorandom interleaver has high probability to break the outer code output pattern which produces low-weight inner code output so as to provide inner code input that produces high-weight inner code output. Like PCCCs, SCCCs also create an overall code trellis with a huge number of states because of the extra dimension introduced by the bit interleaver; nonetheless, the trellis can be decoded with a relatively simple iterative MAP decoder.

Unlike PCCC whose $P_b(e)$ theoretically decreases at the rate of $N^{-1}$ as $N$ increases, $N$ being the interleaver size, the $P_b(e)$ of SCCC can decrease at a faster rate, e.g., $N^{-2}, N^{-3}, ...$, as $N$ increases. The error "floor" associated with PCCCs, where the bit error probability flattens, is therefore mitigated by SCCCs. This feature is illustrated by the simulation curves in Figure 2.12. A disadvantage of SCCCs is that they are more computationally complex to decode than PCCCs with constituent codes of the same memory size. Also, SCCC tends to have a higher bit error probability than PCCC at low $E_b/N_0$.

### 2.5.3 Performance Upper Bound and Deductions

In a manner similar to convolutional codes, it is possible to evaluate the distance structure and performance of turbo codes through the use of a weight enumerating function. For an $(N_f, N)$ code, the input-output weight enumerating function (IOWEF) is defined to be [43]

$$A(W, H) = \sum_{w=0}^{N} \sum_{h=0}^{N_f} A_{w,h} W^w H^h, \tag{2.32}$$

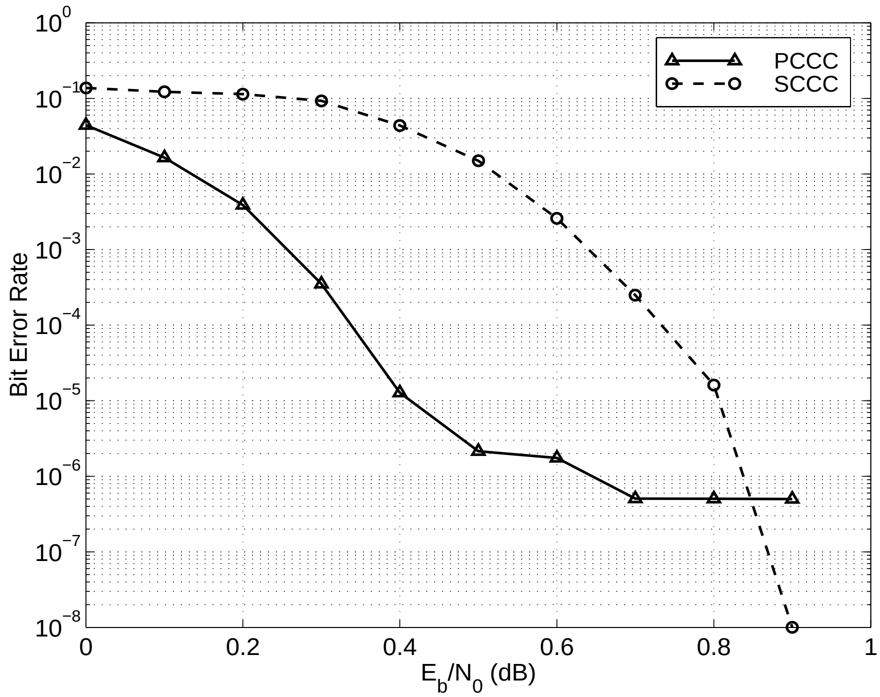



Figure 2.12: Performance comparison of PCCC ($g = (15, 17)_{octal}$) and SCCC ($g = (7, 5)_{octal}$): rate 1/3, frame size 5120, 10 iterations.

where $A_{w,h}$ is the number of codeword sequences with Hamming weight $h$ generated by input with weight $w$. Based on the code IOWEF, the application of the union bound yields the following upper bound to the codeword error probability when ML decoding is used:

$$P_w(e) \leq \sum_{h=d_{min}}^{N_f} \left( \sum_{w=1}^{N} A_{w,h} \right) Q\left( \sqrt{\frac{2hrE_b}{N_0}} \right), \tag{2.33}$$

where $d_{min}$ is the lowest Hamming weight of the nonzero codeword sequences. For linear codes such as block codes, convolutional codes, PCCCs and SCCCs, $d_{min}$ is also the smallest Hamming distance between any two distinct codeword sequences. The bit error probability is upper bounded by

$$\begin{aligned} P_b(e) &= \sum_w \frac{w}{N} P_w(e, w) \\ &\leq \sum_{h=d_{min}}^{N_f} B_h Q\left( \sqrt{\frac{2hrE_b}{N_0}} \right) \end{aligned} \tag{2.34}$$

where $P_w(e, w)$ is the codeword error probability of codeword sequences generated by input sequences of weight $w$,

$$B_h = \sum_{w=1}^{N} \frac{w}{N} A_{w,h}, \tag{2.35}$$

and $NB_h$ is the total input weight of all input sequences that yield codeword sequences of weight $h$.

Equation (2.34) suggests that there are two possible methods to reduce the bit error probability:

- Reduce the word error probability by increasing $d_{min}$. This is the strategy taken by convolutional codes and block codes. In addition, a well-designed convolutional code keeps the number of minimum distance codeword sequences as small as possible.

- Reduce the multiplicities $B_h$ of the most significant terms, which correspond to the lowest output weights. This accounts for the success of turbo codes.

For convolutional codes, if an information sequence $\mathbf{u}$ produces a codeword sequence $\mathbf{c}$, its delayed version $d\mathbf{u}$ produces $d\mathbf{c}$, which has the same weight as $\mathbf{c}$. Asymptotically, $B_h$ approaches the total information weight of all codeword sequences with weight $h$ that are generated by any information sequence with the first bit equal to one. When $N$ increases, $B_h$ does not decrease accordingly. This makes the interleaver gain impossible. On the other hand, $B_h$ increases rapidly with $h$. At low $E_b/N_0$, the large multiplicities make the contribution of the higher distance spectral lines[4] to $P_b(e)$ greater than the free distance asymptote. This is why convolutional codes do not have a steep BER slope at low $E_b/N_0$.

However, because of the pseudorandom interleaver between two constituent codes, turbo codes are time-varying. Hence, $\mathbf{u}$ and $d\mathbf{u}$ produce different codewords with different weight. Low-weight parity sequences of the first constituent code (RSC1) tend to be matched to high-weight parity sequences of the second constituent code (RSC2). For any small $h$, $B_h \ll 1$. This type of distance spectrum is referred to as "spectrally thin". For very large interleavers, spectral thinning results in few low-weight codeword sequences, but a large number of codeword sequences of the average weight. Unlike convolutional codes, large frame sizes help turbo codes achieve a sparse distance spectrum. As the interleaver size approaches infinity, the thinning of

[4]The distance spectrum of a code is the number of codeword sequences with a certain weight.

the spectrum enables the free distance asymptote to dominate the performance at low $E_b/N_0$. This is why turbo codes can achieve performance close to the Shannon limit at low to medium $E_b/N_0$.

At medium to high $E_b/N_0$, the performance of turbo codes approaches the free distance asymptote

$$P_b(e) \approx \frac{1}{N} Q\left(\sqrt{\frac{2d_{fp} r E_b}{N_0}}\right) \sum_{w=1}^{N} w A_{w,d_{fp}} \tag{2.36}$$

where $N$ is the interleaver size and $d_{fp}$ is the free distance of turbo codes. Since $d_{fp}$ is relatively small, the free distance asymptote is relatively flat. This results in the error floor of turbo codes. If a convolutional code has large $d_{fp}$, its performance would eventually surpass that of a turbo code at high $E_b/N_0$ because of its steeper free distance asymptote.

One way to lower the error floor is to increase the interleaver size $N$ while preserving $d_{fp}$ and $\sum_{w=1}^{N} w A_{w,d_{fp}}$. This will lower the error floor without changing the slope. On the other hand, if $N$ is fixed, $d_{fp}$ can be increased to provide a steeper asymptote, while preserving $\sum_{w=1}^{N} w A_{w,d_{fp}}$.

In the following sections, union bound analysis following the work of Benedetto, Montorsi, and Divsalar [43] [39] is used to explore the performance characteristics of PCCC and SCCC.

### 2.5.4 Uniform Interleaver

To compute the upper bounds of the word and bit error probabilities of turbo codes, the conditional weight enumerating function (CWEF) of the concatenated code is needed. The CWEF is given by

$$A(w, H) = \sum_{h=0}^{N_f} A_{w,h} H^h. \tag{2.37}$$

The CWEF describes the weight distribution of codeword sequences generated by the information sequences of a given weight $w$. For codes with large block sizes, the complexity of computing the CWEF becomes unmanageable due to the extra dimension added by the interleaver. Since the interleaver size is equal to the frame size, which is usually very large, it is almost impossible to exhaust the possible combinations of the inputs to both constituent encoders. The input of the second encoder depends not only on the information sequence, but also on the interleaver mapping. Thus, the knowledge of the input alone is insufficient to yield the weight of the codeword sequence from the second encoder.

To facilitate the theoretical analysis of turbo codes, the concept of a uniform interleaver [44] was introduced. The **uniform interleaver** is a probabilistic device that represents the average behavior of all deterministic bit interleavers for a given length $N$. To each sequence of weight $w$, it associates all distinct sequences of weight $w$ and length $N$ with equal probability $1/\binom{N}{w}$. Using a uniform interleaver, each input word of weight $w$ at the input of the first encoder $C_1$ is mapped to all distinct permutations with equal probability. Hence, all possible codeword sequences corresponding to the input weight $w$ will be generated after passing the second encoder $C_2$. In consequence, all input words of the same weight will produce the same set of codewords out of $C_2$. Therefore, the CWEFs of $C_1$ and $C_2$ become independent and can be multiplied to yield the overall CWEF

$$A^{C_p}(w, H) = \frac{A^{C_1}(w, H) \times A^{C_2}(w, H)}{\binom{N}{w}} \tag{2.38}$$

of the parallel concatenated code for a given input weight $w$.

The IOWEF of a PCCC obtained using the uniform interleaver is equal to the IOWEF averaged over all possible interleavers. The performance upper bound for PCCC computed using the IOWEF relative to the uniform interleaver coincides with the average of the performance upper bounds obtainable with the whole class of interleavers [45]. Thus, for any $E_b/N_0$, the performance obtained by the uniform interleaver is achievable by at least one deterministic interleaver.

### 2.5.5 Performance Upper Bound for PCCC

Assuming a uniform interleaver of length $N$ which lies between two rate 1/2 constituent codes, by Equation (2.38), the PCCC has a CWEF with the following coefficients for any given $w$ [43]:

$$A_{w,h}^{C_p} = \sum_{h_1} \frac{A_{w,h_1}^{C_1} \times A_{w,h-h_1}^{C_2}}{\binom{N}{w}}. \tag{2.39}$$

Note that the codeword sequence weight of $C_2$ should not include the weight of the systematic bits since they are not transmitted. Based on Equation (2.34), the bit error probability is thus

$$P_b(e) \leq \sum_h \left( \sum_w \frac{w}{N} A^{C_p}_{w,h} \right) Q\left( \sqrt{\frac{2hrE_b}{N_0}} \right). \tag{2.40}$$

Since turbo codes are linear, the analysis can be performed by assuming the transmission of the all-zero codeword sequence. An error event occurs whenever a path leaves the all-zero state and remerges with the all-zero state at a later time. Define $A_{w,h,n_i}$ to be the number of codeword sequences with input weight $w$, codeword sequence weight $h$, and $n_i$ concatenated error events, where $i = 1, 2$ is the index of the constituent code. When the interleaver size $N$ is much larger than the constraint length, the coefficient $A_{w,h}$ of the constituent code can be approximated by

$$A_{w,h} \approx \sum_{j=1}^{n_{max}} \binom{N}{j} A_{w,h,j}, \tag{2.41}$$

where $n_{max}$ is the largest number of error events concatenated in a codeword sequence of weight $h$ generated by an input sequence of weight $w$. The assumption of large $N$ permits neglecting the length of error events when they are comparatively short, so that there are $\binom{N}{j}$ ways to arrange $j$ error events. Using Equation (2.41), (2.39) can be written as

$$A^{C_p}_{w,h} = \sum_{n_1=1}^{n_{1,max}} \sum_{n_2=1}^{n_{2,max}} \frac{\binom{N}{n_1}\binom{N}{n_2}}{\binom{N}{w}} \sum_{h_1} A^{C_1}_{w,h_1,n_1} \times A^{C_2}_{w,h-h_1,n_2}. \tag{2.42}$$

When $N \gg n$, the approximation

$$\binom{N}{n} \approx \frac{N^n}{n!} \tag{2.43}$$

can be applied to Equation (2.42) to obtain

$$A^{C_p}_{w,h} = \sum_{n_1=1}^{n_{1,max}} \sum_{n_2=1}^{n_{2,max}} \frac{w!}{n_1!n_2!} N^{n_1+n_2-w} \sum_{h_1} A^{C_1}_{w,h_1,n_1} \times A^{C_2}_{w,h-h_1,n_2}. \tag{2.44}$$

Substituting Equation (2.44) into Equation (2.40), we have

$$P_b(e) \leq \sum_h \left( \sum_w ww! \sum_{n_1=1}^{n_{1,max}} \sum_{n_2=1}^{n_{2,max}} \frac{1}{n_1!n_2!} N^{n_1+n_2-w-1} \sum_{h_1} A^{C_1}_{w,h_1,n_1} \times A^{C_2}_{w,h-h_1,n_2} \right) Q\left( \sqrt{\frac{2hrE_b}{N_0}} \right). \tag{2.45}$$

As $N \to \infty$, for each weight $h$, the dominant term is the one with the largest exponent of $N$. Define the largest exponent to be

$$\varphi(h) = \max_{n_1,n_2,w} \{n_1(h) + n_2(h) - w(h) - 1\}, \tag{2.46}$$

where $n_1(h)$ and $n_2(h)$ are the number of error events concatenated in a weight-$h$ codeword sequence generated by a weight-$w$ information sequence for codes one and two, respectively. Hence, for large $N$, the bit error probability can be approximated by

$$P_b(e) \approx \sum_h C_h N^{\varphi(h)} Q\left(\sqrt{\frac{2hrE_b}{N_0}}\right), \tag{2.47}$$

where $C_h$ is a constant independent of $N$. If $\varphi(h) < 0$, $P_b(e)$ decreases with the increase of $N$. This is called **interleaver gain** and is a very important feature of turbo codes. The slope of $P_b(e)$ versus $E_b/N_0$ is determined by $\varphi(h)$. A smaller $\varphi(h)$ gives a larger interleaver gain, when $\varphi(h) < 0$. Special attention is paid to two important values of $\varphi(h)$,

- $\varphi(d_{fp})$, where $h = d_{fp}$ is the free distance of the PCCC. This exponent of $N$ dominates at high $E_b/N_0$.
- The largest value $\varphi_{max} = \max_h \varphi(h)$, which is the dominant term as $N \to \infty$.

1. $\varphi(d_{fp})$
   When $h = d_{fp}$, the input must result in a single error event for both constituent codes, or, $n_1(d_{fp}) = 1$, $n_2(d_{fp}) = 1$. So

   $$\varphi(d_{fp}) = n_1(d_{fp}) + n_2(d_{fp}) - w_{fp} - 1 = 1 - w_{fp},$$

   where $w_{fp}$ is the minimum input weight among those producing the free distance codeword sequences of PCCC. Since a negative exponent of $N$ is required for an interleaver gain, $w_{fp}$ must be greater than one. For nonrecursive convolutional codes and block codes, an input sequence of minimum weight one produces the output sequence of the smallest weight. Thus $w_{fp} = 1$, $\varphi(d_{fp}) = 0$, and therefore there is no interleaver gain at high $E_b/N_0$. For recursive convolutional codes, an input of weight one will produce a codeword sequence of infinite weight. Thus, the smallest output weight event must be produced by an input of weight greater than one, or $w_{fp} \geq 2$ [44]. As a result, $\varphi(d_{fp}) \leq -1$. For the recursive convolutional encoders of rate $1/n_0$, $w_{fp} = 2$ [44], which implies $\varphi(d_{fp}) = -1$.

This indicates an interleaver gain of $N^{-1}$ at high $E_b/N_0$. Hence, the constituent encoders must be recursive to provide interleaver gain at high $E_b/N_0$.

2. $\varphi_{max} = \max_{n_1,n_2,w,h}\{n_1(h) + n_2(h) - w(h) - 1\}$
For a given $w$, the largest values of $n_1$ and $n_2$ can be achieved by

$$n_{i,max} = \left\lfloor \frac{w}{w_{i,min}} \right\rfloor, \quad i = 1, 2, \tag{2.48}$$

where $w_{i,min}$ is the minimum weight of the input sequence producing an error event of the $i$-th constituent code. Again, we will see that a recursive implementation is crucial for the interleaver gain.

For nonrecursive convolutional codes and block codes, we have $w_{i,min} = 1$, and consequently $n_{i,max} = w$. That is, if every input sequence with weight one generates a finite-weight error event, then an input sequence with weight $w$ will generate $w$ error events at most, corresponding to the concatenation of $w$ error events of input weight one. This will also occur at $C_2$ when a uniform interleaver is assumed since the uniform interleaver generates all possible permutations of the input sequence. So $\varphi_{max} = w - 1 \geq 0$, and it is impossible to obtain any interleaver gain.

For recursive convolutional codes, the minimum possible weight of input sequences that generate error events is $w_{i,min} = 2$. In this case, $n_{i,max} \leq \lfloor w/2 \rfloor, i = 1, 2$. The maximum exponent is

$$\begin{aligned} \varphi_{max} &\leq 2\left\lfloor \frac{w}{2} \right\rfloor - w - 1 \\ &= \begin{cases} -1, & w \text{ is even.} \\ -2, & w \text{ is odd.} \end{cases} \end{aligned} \tag{2.49}$$

This implies the existence of interleaver gain when $N \to \infty$. Particularly, $\varphi_{max} = -1$ for rate $1/n_0$ recursive convolutional codes. Since the multiplicity for the input of weight $w = 2$ is the largest among those producing similar output weight, its error events dominate the error probability. Asymptotically, the bit error probability is

$$P_b(e) \approx N^{-1} N_{1f,eff} N_{2f,eff} Q\left(\sqrt{\frac{2d_{fp,eff} r E_b}{N_0}}\right), \tag{2.50}$$

where $d_{fp,eff}$ is the **effective free distance** of the PCCC [46], which is defined as the minimum weight of the codeword sequences generated by input sequences of weight 2, and $N_{if,eff}$ is the multiplicity of error events of the $i$-th constituent code with output weight equal to the effective free distance. The effective free distance of the constituent codes should be maximized to minimize $P_b(e)$.

### 2.5.6 Conclusions on PCCC Design

From the above analysis, the following conclusions can be drawn for designing a good parallel concatenated code:

- Both constituent encoders must be recursive convolutional encoders.
- The optimization of the recursive constituent encoders should maximize their effective free distance and minimize their multiplicities.
- Among codes with the largest effective free distance, those codes maximizing $d_3$ (minimum weight of codeword sequences produced by an input sequence with $w = 3$), then $d_4$ (minimum weight of codeword sequences produced by an input sequence with $w = 4$), and so on, should be chosen.

To achieve good performance, it is important that the constituent codes are recursive, but not necessarily systematic. It is for the simplicity in the encoder and decoder that systematic codes, and hence recursive systematic convolutional codes, are usually chosen.

For a rate $1/n_0$ recursive convolutional encoder with memory $m$, the effective free distance is upper bounded by [44] [45]

$$d_{f,eff} \leq 2 + (n_0 - 1)(2^{m-1} + 2).$$

Furthermore, equality holds if and only if the generating matrix of the code has the form

$$g(D) = \left[1, \frac{P_1(D)}{Q(D)}, \cdots, \frac{P_{n_0-1}(D)}{Q(D)}\right],$$

where $Q(D)$ is a primitive polynomial over GF(2) of degree $m$. $P_1(D), \cdots, P_{n_0-1}(D)$ are polynomials different than $Q(D)$ with the form $(1 + ... + b_i D^i + ... + D^m)$, $i = 1, ..., m-1$, $b_i \in (0, 1)$.

### 2.5.7 Performance Upper Bound for SCCC

Since SCCCs have the same components as PCCC, they can be analyzed in a similar manner. In particular, the uniform interleaver concept can still be applied.

Let the superscripts $o$ and $i$ refer to the outer and inner constituent codes, respectively, and $N$ refer to the interleaver size. An information sequence of length $Nr^o$ passes the outer encoder of rate $r^o$, which produces a codeword sequence of length $N$. The output of the outer encoder is permuted and used as the input to the inner encoder. The inner code of rate $r^i$ produces an output of length $N/r^i$. The output of the inner encoder is transmitted through the channel. The overall code rate $r$ equals $r^o r^i$.

Similar to PCCC, let $A^{C_s}_{w,h}$ be the IOWEF coefficients of the SCCC. Let $A_{w,h,j}$ represent the number of codeword sequences of weight $h$ with $j$ concatenated error events produced by information sequences of weight $w$. According to Equation (2.34), the upper bound on the bit error probability is [39]

$$P_b(e) \leq \sum_{h=d_{fs}}^{N/r^i} \sum_{w=1}^{Nr^o} \frac{w}{Nr^o} A^{C_s}_{w,h} Q\left(\sqrt{\frac{2hrE_b}{N_0}}\right), \tag{2.51}$$

where $d_{fs}$ is the minimum weight of all nonzero codeword sequence. Since the input of the inner encoder is limited to be the scrambled version of the output of the outer encoder, $d_{fs}$ can be greater than the free distance of the inner code $d^i_f$. For a SCCC with a pseudorandom interleaver, it is valid to assume $\lfloor d_{fs}/d^i_f \rfloor = 1$.

Suppose a uniform interleaver is used between the inner and outer encoders. When the coefficients $A^{C^o}_{w,l}$ and $A^{C^i}_{l,h}$ of the outer and inner codes are known, the coefficients of $A^{C_s}_{w,h}$ can be expressed as [28] [39]

$$A^{C_s}_{w,h} = \sum_{l=d^o_f}^{N} \frac{A^{C^o}_{w,l} A^{C^i}_{l,h}}{\binom{N}{l}}. \tag{2.52}$$

When $N$ is large, the approximation in Equation (2.41) can be adopted for the outer and inner code. Hence,

$$A^{C_s}_{w,h} \approx \sum_{l=d^o_f}^{N} \sum_{n^o=1}^{n^o_{max}} \sum_{n^i=1}^{n^i_{max}} \frac{\binom{Nr^o}{n^o}\binom{N}{n^i}}{\binom{N}{l}} A^o_{w,l,n^o} A^i_{l,h,n^i}, \tag{2.53}$$

where $n^i_{max}$ and $n^o_{max}$ indicate the maximum number of error events concatenated in a codeword sequence. Using the approximation in Equation (2.43), Equation (2.53) can be further approximated by

$$A_{w,h}^{C_s} \approx \sum_{l=d_f^o}^{N} \sum_{n^o=1}^{n_{max}^o} \sum_{n^i=1}^{n_{max}^i} N^{n^o+n^i-l} \frac{(r^o)^{n^o} l!}{n^o! n^i!} A_{w,l,n^o}^o A_{l,h,n^i}^i. \tag{2.54}$$

Substituting Equation (2.54) into Equation (2.51), the upper bound on $P_b(e)$ is approximately

$$P_b(e) \leq \sum_{h=d_{fs}}^{N/r^i} \left( \sum_{w=1}^{Nr^o} \sum_{l=d_f^o}^{N} \sum_{n^o=1}^{n_{max}^o} \sum_{n^i=1}^{n_{max}^i} N^{n^o+n^i-l-1} \frac{w(r^o)^{n^o-1} l!}{n^o! n^i!} A_{w,l,n^o}^o A_{l,h,n^i}^i \right) Q\left( \sqrt{\frac{2hrE_b}{N_0}} \right). \tag{2.55}$$

Similar to the approach for PCCCs, a very important parameter of SCCCs is the exponent of $N$, which determines the interleaver gain. Denote the exponent as $\varphi(w,h) = n^o + n^i - l - 1$. As $N \to \infty$, the dominant term in Equation (2.55) for a given $h$ is the one with the largest exponent of $N$,

$$\varphi(h) = \max_{n^o, n^i, w} \{n^o(h) + n^i(h) - l - 1\}. \tag{2.56}$$

Since investigation of $\varphi(h)$ yields a better understanding of the performance of SCCC, the following two important cases are discussed in detail:

- $\varphi(d_{fs})$, which corresponds to $h = d_{fs}$, the free distance of SCCC. The term with exponent $\varphi(d_{fs})$ in Equation (2.55) dominates the bit error performance at high $E_b/N_0$.
- The largest value $\varphi_{max} = \max_h \varphi(h)$, which determines the dominant term as $N \to \infty$.

1. $\varphi(d_{fs})$

   By definition, we know that

   $$n^i \leq \left\lfloor \frac{d_{fs}}{d_f^i} \right\rfloor, \tag{2.57}$$

   and

   $$n^o \leq \left\lfloor \frac{l}{d_f^o} \right\rfloor, \tag{2.58}$$

   where $d_f^i$ is the free distance of the inner code and $d_f^o$ is the free distance of the outer code. Typically $\lfloor d_{fs}/d_f^i \rfloor = 1$, so we have

   $$\varphi(d_{fs}) \leq \max_l \left\{ \left\lfloor \frac{d_{fs}}{d_f^i} \right\rfloor + \left\lfloor \frac{l}{d_f^o} \right\rfloor - l - 1 \right\}$$

$$\begin{aligned}
&\leq \max_{l}\left\{\left\lfloor\frac{l}{d_f^o}\right\rfloor - l\right\} \\
&= \left.\left(\left\lfloor\frac{l}{d_f^o}\right\rfloor - l\right)\right|_{l=d_f^o} \\
&= 1 - d_f^o. \qquad (2.59)
\end{aligned}$$

Thus, when $E_b/N_0$ is large, Equation (2.55) can be approximated by

$$P_b(e) \leq C_h N^{1-d_f^o} Q\left(\sqrt{\frac{2d_{fs} r E_b}{N_0}}\right), \qquad (2.60)$$

where $C_h$ is a constant independent of $N$. To make $P_b(e)$ decrease as $N$ increases, $d_f^o \geq 2$ is required. This requirement is easily satisfied by most block codes and convolutional codes, recursive or not.

2. $\varphi_{max} = \max_h \varphi(h)$
Nonrecursive and recursive inner codes need to be treated separately in this case.

For a block or nonrecursive convolutional code, every input sequence with weight one generates a finite-weight error event. An input sequence with weight $l$ can generate $l$ error events at most. When block or nonrecursive convolutional codes are used as the inner code, this will certainly happen if the uniform interleaver is used. Recall that the uniform interleaver will generate all possible permutations with equal probability. In this case, $n^i_{max} = l$ and

$$\varphi_{max} = n^o_{max} - 1 \geq 0 \qquad (2.61)$$

This implies that the exponent of $N$ will not be negative, and hence no interleaver gain exists.

For a recursive convolutional code, two is the minimum weight of an input which generates an error event of finite weight. If a recursive code is used as the inner code, an input of weight $l$ can yield at most $n^i_{max} = \lfloor l/2 \rfloor$ error events. Notice that $n^o_{max} \leq \lfloor l/d_f^o \rfloor$ by definition. So

$$\begin{aligned}
\varphi_{max} &\leq \max_{l}\left\{\left\lfloor\frac{l}{d_f^o}\right\rfloor + \left\lfloor\frac{l}{2}\right\rfloor - l - 1\right\} \\
&\leq \max_{l}\left\{\left\lfloor\frac{l}{d_f^o}\right\rfloor - \left\lceil\frac{l+1}{2}\right\rceil - 1\right\}
\end{aligned}$$

$$
\begin{aligned}
&= \left( \left\lfloor \frac{l}{d_f^o} \right\rfloor - \left\lfloor \frac{l+1}{2} \right\rfloor - 1 \right) \Bigg|_{l=d_f^o} \\
&= - \left\lfloor \frac{d_f^o + 1}{2} \right\rfloor \\
&< \ 0, \qquad \text{since} \quad d_f^o > 0.
\end{aligned} \tag{2.62}
$$

This shows that the exponent $\varphi_{max}$ is always a negative integer. Consequently, $\varphi(h) < 0, \forall h$, and interleaver gain exists. Substituting Equation (2.62) into Equation (2.55), we obtain the following approximate upper bound on the bit error probability:

$$
P_b(e) \leq C_h N^{-\lfloor (d_f^o+1)/2 \rfloor} Q\left( \sqrt{\frac{2h(\varphi_{max}) r E_b}{N_0}} \right). \tag{2.63}
$$

An important point is that for PCCC with rate $1/n_0$ RSC codes, $P_b(e)$ can only be proportional to $N^{-1}$, while it can be proportional to $N^{-2}$, or $N^{-3}, \ldots$, for SCCC depending on the value of $d_f^o$ [39]. As a result, the slope of $P_b(e)$ versus $E_b/N_0$ for a SCCC can be steeper than that of a PCCC at high $E_b/N_0$. Thus, the error floor apparent in PCCC disappears in SCCC.

In Equation (2.63), $h(\varphi_{max})$ is the minimum output weight associated with the largest exponent $\varphi_{max}$. It can be estimated as follows. Let $d_{f,eff}^i$ be the minimum weight of the inner code output which is generated by a weight-2 inner code input sequence. When $d_f^o$ is even,

$$
h(\varphi_{max}) = \frac{d_f^o d_{f,eff}^i}{2},
$$

where an overall output is composed of $d_f^o/2$ concatenated error events of weight $d_{f,eff}^i$. When $d_f^o$ is odd,

$$
h(\varphi_{max}) = \frac{(d_f^o - 3) d_{f,eff}^i}{2} + h_{min}^{(3)},
$$

where $h_{min}^{(3)}$ is the minimum weight output of the inner code generated by a weight-3 input.

### 2.5.8 Conclusions on SCCC Design

The discussion above leads to several guidelines for the design of SCCC:

- The outer encoder can be recursive or nonrecursive.
- The inner encoder must be a recursive convolutional encoder. In contrast to block codes and nonrecursive convolutional codes, the recursive encoder structure ensures an interleaver gain.
- The free distance $d_f^o$ of the outer code should be as large as possible. Examining Equation (2.63), we see that this condition is required to minimize the exponent of $N$ for a higher interleaver gain.
- The effective free distance of the inner code $d_{f,eff}^i$ should be maximized. Increasing $d_{f,eff}^i$ will increase the output weight $h(\varphi_{max})$, which diminishes the $Q(\cdot)$ function in Equation (2.63).

## 2.6 Summary

In this chapter, performance bounds were shown for three types of channels, CH-UC, CH-BI, and CH-BIBO. The structures and characteristics of both PCCC and SCCC were introduced. The most significant advantage of these codes in comparison to convolutional codes and block codes is that their error probability decreases as the frame size increases due to the pseudorandom interleaver. Analysis was presented to show that $P_b(e)$ of PCCC decreases at the rate of $N^{-1}$, and $P_b(e)$ of SCCC decreases at the rate of $N^{1-d_f^o}$ (at high $E_b/N_0$ region) or $N^{-\lfloor (d_f^o+1)/2 \rfloor}$ (at low-to-medium $E_b/N_0$ region when $N$ is large) with respect to the increase of frame size $N$. Based on the analysis, design guidelines for both PCCC and SCCC were presented.

The intent of this chapter was to introduce the code design techniques underlying the development of PCCC and SCCC, and to present an analysis justifying their remarkable performance. In Chapter 3, an iterative decoding technique is described for these codes. The development of practical implementations for exploiting the advances described in Chapter 2 is the subject of the remainder of this dissertation.

# Chapter 3

# Iterative Decoding of Concatenated Codes

In addition to the concatenation of two or more convolutional codes and the pseudorandom interleaver, a very important feature of PCCCs (turbo codes) and SCCCs is that their decoding procedure is performed iteratively and the complexity of the decoder increases linearly with the frame size. In fact, the name "turbo" is used because the decoder has feedback, similar to a turbo-charged engine. While the iteration proceeds, the constituent decoders exchange soft information between each other. As proven by Forney [11], the optimal soft output of the decoder should be the *a posteriori* probability (APP), which is the probability that a certain bit is transmitted conditioned on the received signals.

Since the complexity of both PCCCs and SCCCs lies in the iterative decoder, it is important to understand the decoding algorithms and find the most efficient way to realize the decoder without compromising performance. In this chapter, the available decoding algorithms are first introduced. Then the "butterfly" structure of RSC codes is presented in Section 3.2, as a useful aid for understanding the trellis and also for implementing the algorithms. The conventional Log-MAP algorithm for PCCCs is discussed in Section 3.3, and then simplified in Section 3.4. A general soft-input soft-output decoding module suitable for both PCCCs and SCCCs is discussed in Section 3.5, and Section 3.6 explains how to apply it to decode both PCCCs and SCCCs. Afterwards, two methods are presented to reduce the storage required for the decoder. Section 3.7 presents the sliding window approach and Section 3.8 proposes forward computation of the backward path metrics.

## 3.1 Overview of Decoding Algorithms

Developing efficient decoding algorithms is a major concern for practical implementation of PCCCs or SCCCs. Two major classes of decoding algorithms have been proposed [47]. One is the MAP family. This family includes the symbol-by-symbol maximum *a posteriori* (MAP) [23] algorithm, which is the optimal method for producing the APP information; its additive form, the Log-MAP algorithm; and its suboptimal additive form, the Max-Log-MAP algorithm [23] [48]. The other class of decoding algorithms is based on modification of the VA, which is suboptimal since the required APP are not provided by the standard VA. This family includes the well-known soft-output Viterbi algorithm (SOVA) [26] [49] and the serial list output Viterbi algorithm (SLVA) [50] [51]. In addition to these two classes of algorithms, it was also shown that iterative decoding techniques can be viewed as an instance of probability propagation in a graphical model of the code [52] [53] [54].

This dissertation focuses on the MAP family, with SOVA mentioned primarily for comparison. The MAP algorithm [23] is a trellis-based soft-output decoding algorithm. Unlike the VA which is a ML trellis decoding method and minimizes the codeword error probability, the MAP algorithm minimizes the bit error probability. MAP is optimal for estimating the states or outputs of a Markov process observed in white noise. However, MAP is impractical for implementation, mainly because of the numerical difficulties associated with the representation of the probabilities, the nonlinear functions, and the mixed multiplications and additions of these values.

The Log-MAP algorithm is a transformation of MAP, which has equivalent performance but without the problems in practical implementation. Like Max-Log-MAP and SOVA, Log-MAP works exclusively in the logarithmic domain, in which multiplication is converted to addition, and addition is converted to a $\max^*(\cdot)$ operation, which is later defined in Equation (3.86).

Max-Log-MAP and SOVA are the suboptimal algorithms designed for easy implementation [26] [49] [36] [55]. However, in an AWGN channel, their performance suffers, especially in the low $E_b/N_0$ region. Max-Log-MAP uses exactly the same approach as MAP and Log-MAP with forward and backward recursions. It differs from the Log-MAP algorithm in that it reduces the complexity by approximating the $\max^*(\cdot)$ function with a simple $\max(\cdot)$ function. Studies have shown that there can be as much as 0.5 dB loss when Max-Log-MAP is used instead of MAP/Log-MAP in an AWGN channel [56] [57].

The conceptual differences between these algorithms help to explain their performance. At each step $k$ in a trellis, MAP/Log-MAP splits all paths into two sets, one with the information bit $u_k = 1$, and the other with $u_k = 0$ [56] [48]. MAP/Log-MAP computes the log-likelihood ratio (LLR) of these two sets as defined in Equation (3.75). In contrast, Max-Log-MAP searches through all the paths to look for two candidates, the most probable path with $u_k = 1$, and the most probable path with $u_k = 0$. The soft output of the Max-Log-MAP algorithm is the LLR of these two paths.

In comparison, SOVA builds upon the VA with some additional real value additions and storages. This algorithm considers only the surviving path and the competing path which joins it. In essence, SOVA uses the same metric and gives identical hard decisions as Max-Log-MAP [58]. Although SOVA can always find the most likely path, the best competing path may have been eliminated before merging with the ML path. Consequently, the soft output of SOVA can deviate from that of Max-Log-MAP, and the performance of an iterative decoder using SOVA is worse than that using Max-Log-MAP.

Since Log-MAP performs better than SOVA with similar complexity and Log-MAP is more suited to parallel processing [59], the Log-MAP algorithm is presented in detail in this dissertation. The suboptimal Max-Log-MAP version can be readily obtained by replacing all occurrences of $\max^*(\cdot)$ with $\max(\cdot)$.

In the following, the butterfly structure of RSC codes is first introduced, which facilitates the implementation of an algorithm.

## 3.2 Butterfly Structure of RSC Codes

For a binary code alphabet, the branch transitions appear as butterfly pairs when the first and the last stages of the encoder shift register are both connected in the feedback and feed-forward polynomials. This is an important property that is often exploited in implementations.

A rate 1/2 RSC encoder model, which is often used as the constituent code of a PCCC or a SCCC, is shown in Figure 3.13. Denote the information bit to be encoded as $u_k$, the state of the encoder as $(S_{k,0}, ..., S_{k,m-1})$, the substate as $(S_{k,0}, ..., S_{k,m-2})$, and the encoder output as $c_k = (c_k^{(1)}, c_k^{(2)})$, where $c_k^{(1)}$ is the systematic bit, $c_k^{(2)}$ is the parity bit, and $k$ is the time index. For a given substate $(S_{k,0}, ..., S_{k,m-2})$, the

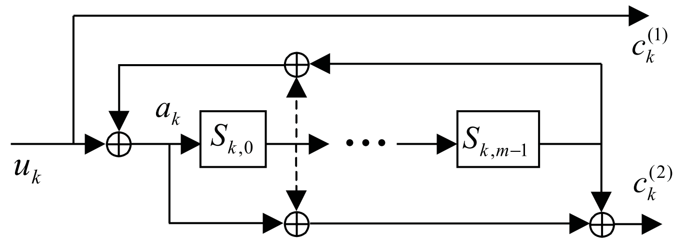



Figure 3.13: A general rate 1/2 RSC encoder.

following two observations have been made.

1. *The relationship between the current state and the next state.* If the feedback polynomial has a connection to both $u_k$ and $S_{k,m-1}$, then the opposite value of $u_k \oplus S_{k,m-1}$ will correspond to the opposite value of $a_k$. Since $a_k$ will be shifted to the first stage of the shift register and become $S_{k+1,0}$ at the next cycle, the next state is determined as a result. This relationship may be expressed as

$$\begin{cases} u_k \oplus S_{k,m-1} = 0 \\ u_k \oplus S_{k,m-1} = 1 \end{cases} \Longrightarrow \begin{cases} a_k = h_a \\ a_k = \bar{h}_a \end{cases} \quad (h_a \in \{0,1\})$$
$$\Longrightarrow \text{next state at time } k+1 : \begin{cases} (h_a, S_{k,0}, ..., S_{k,m-2}) \\ (\bar{h}_a, S_{k,0}, ..., S_{k,m-2}) \end{cases} .$$

In other words, the same $u_k$ will bring the current states to different next states. This is different from a butterfly of a nonrecursive convolutional code, where the same $u_k$ will bring the current states to the same next state. Define the index of the state as

$$s = S_{k,0} \times 2^{m-1} + \cdots + S_{k,m-2} \times 2 + S_{k,m-1},$$

and let the index of the butterfly pair be

$$M = S_{k,0} \times 2^{m-2} + \cdots + S_{k,m-2},$$

where $s \in \{0, 1, ..., 2^m - 1\}$ and $M \in \{0, 1, ..., 2^{m-1} - 1\}$. Then the state pair $2M$ (when $S_{k,m-1} = 0$) and $(2M + 1)$ (when $S_{k,m-1} = 1$) at time $k$ always go to the next state pair $M$ (when $a_k = 0$) and $(2^{m-1} + M)$ (when $a_k = 1$) at time $(k + 1)$.

2. *The codewords in a butterfly pair.* If the feed forward polynomial has connections to both $a_k$ and $S_{k,m-1}$, then the opposite value of $a_k \oplus S_{k,m-1}$ is related

to the opposite value of $c_k^{(2)}$, or,

$$\begin{cases} a_k \bigoplus S_{k,m-1} = 0 \\ a_k \bigoplus S_{k,m-1} = 1 \end{cases} \Longrightarrow \begin{cases} c_k^{(2)} = h_{c2} \\ c_k^{(2)} = \bar{h}_{c2} \end{cases} \quad (h_{c2} \in \{0,1\}).$$

Since $S_{k,m-1}$ is connected to $a_k$ by the backward polynomial and both $a_k$ and $S_{k,m-1}$ are connected to $c_k^{(2)}$ by the forward polynomial, the influence of $S_{k,m-1}$ on $c_k^{(2)}$ is canceled. The substate $(S_{k,0}, ..., S_{k,m-2})$ will uniquely determine whether $c_k^{(2)} = u_k$ or $c_k^{(2)} = \bar{u}_k$. Considering $c_k^{(1)} \equiv u_k$, there are two possible codewords for a given substate:

$$\begin{cases} c_k^{(2)} = u_k \\ c_k^{(2)} = \bar{u}_k \end{cases} \Longrightarrow (c_k^{(1)}, c_k^{(2)}) = \begin{cases} (u_k, u_k) \\ (u_k, \bar{u}_k) \end{cases},$$

where $u_k \in \{0,1\}$. For each butterfly pair of states, one choice is made since the current states share the same substate. In other words, even though there are four branches in a butterfly pair, there can be only two codeword values for a butterfly pair: either (00),(11) for the $(u_k, u_k)$ case, or (01),(10) for the $(u_k, \bar{u}_k)$ case.

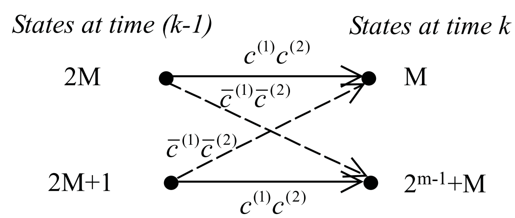



Figure 3.14: The model of butterfly $M$.

To summarize, a RSC code generator which has connections to $u_k$ and $S_{k,m-1}$ for both forward and backward polynomials has a trellis which can be grouped into $2^{m-1}$ butterfly pairs. Each pair is defined by a unique substate. Since good RSC encoders for turbo codes always satisfy this condition, they all have trellises that are composed of butterfly structures. When the rate is 1/2, half of the pairs have codewords (00) and (11), and half of them have codewords (01) and (10). Each pair assumes the form in Figure 3.14. For example, there are two butterfly pairs in a trellis section of $g(D) = [1, 1+D^2/1+D+D^2]$, whose trellis is shown in Figure 3.15. Here the dashed line is for the branches with $u_k = 1$, and the solid line is for the branches with $u_k = 0$.

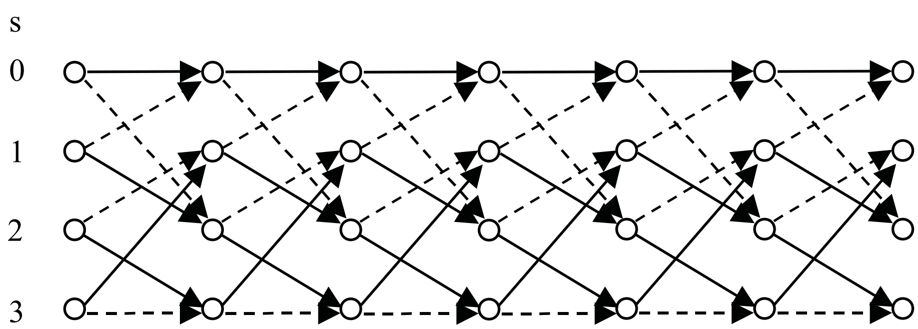



Figure 3.15: Trellis of the RSC code with code generator $g = (7, 5)_{octal}$.

## 3.3 The Conventional Log-MAP Algorithm

For a turbo code with two constituent RSC codes, a decoding algorithm is applied to decode each RSC code. Since Log-MAP has both the optimal performance of MAP and the implementation convenience of Max-Log-MAP and SOVA, it is a very popular algorithm for turbo codes. The following development of Log-MAP begins with the description of the MAP algorithm.

### 3.3.1 MAP Algorithm

To make the notation clear, an edge, sometimes called a "branch", of the encoder trellis is introduced in Figure 3.16. Here $s_k^S(e)$ and $s_k^E(e)$ are the starting and ending states of edge $e$, respectively; $u_k(e)$ is the information word containing $k_0$ bits and $c_k(e)$ is codeword containing $n_0$ bits, respectively. Notice that $s_k^E(e) = s_{k+1}^S(e)$ in this notation. In the following, $Y_i^j$ stands for a sequence of the received signal from $Y_i$ at time $i$ to $Y_j$ at time $j$, inclusive.

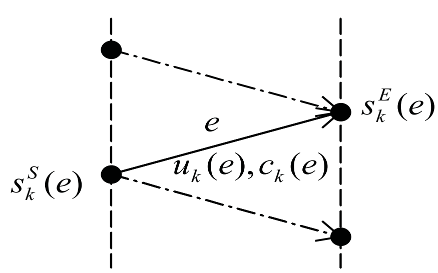



Figure 3.16: An edge of the trellis.

According to the BCJR algorithm [23], which is a forward-backward MAP algorithm, the branch metric at time $k$ is

$$\begin{aligned} M_k(e) &= P(s_k^E(e), Y_k | s_k^S(e)) \\ &= \sum_{X_k} P(s_k^E(e) | s_k^S(e)) \cdot P(X_k|e) \cdot P(Y_k|X_k). \end{aligned} \tag{3.64}$$

Let $A_k(\cdot)$ and $B_k(\cdot)$ be the forward and backward path metrics, respectively. These path metrics are computed recursively as

$$\begin{aligned} A_k(s) &= P(s_k^E(e) = s, Y_1^k) \\ &= \sum_{e:s_k^E(e)=s} A_{k-1}(s_k^S(e)) \cdot M_k(e), \qquad k = 1, ...N-1, \end{aligned} \tag{3.65}$$

and

$$\begin{aligned} B_k(s) &= P(Y_{k+1}^N | s_{k+1}^S(e) = s) \\ &= \sum_{e:s_{k+1}^S(e)=s} B_{k+1}(s_{k+1}^E(e)) \cdot M_{k+1}(e), \quad k = N-1, ..., 1. \end{aligned} \tag{3.66}$$

Suppose the encoder starts in a known state $S_0$. Then the $A_k(s)$ computation will be initialized as

$$A_0(s) = \begin{cases} 1, & s = S_0, \\ 0, & \text{otherwise.} \end{cases} \tag{3.67}$$

If the trellis is terminated to a known state $S_N$, the $B_k(s)$ computation will be initialized as

$$B_N(s) = \begin{cases} 1, & s = S_N, \\ 0, & \text{otherwise.} \end{cases} \tag{3.68}$$

However, if the final state of the trellis is unknown, then each state is equiprobable and

$$B_N(s) = \frac{1}{2^m}, \qquad \forall s. \tag{3.69}$$

Here $m$ is the number of stages in the shift register of the constituent encoder, and $2^m$ is the number of states in the trellis.

The joint probability at time $k$ is

$$\begin{aligned} \sigma_k(e) &= P(e, Y_1^N) \\ &= A_{k-1}(s_k^S(e)) \cdot M_k(e) \cdot B_k(s_k^E(e)), \end{aligned} \tag{3.70}$$

and the *a posteriori* probabilities can be expressed as follows:

$$\begin{aligned} P_k^A(c;O) &= P(c_k = c|Y_1^N) \\ &= \frac{1}{P(Y_1^N)} \sum_{e:c(e)=c} \sigma_k(e) \\ &= \frac{1}{P(Y_1^N)} \sum_{e:c(e)=c} A_{k-1}(s_k^S(e)) \cdot M_k(e) \cdot B_k(s_k^E(e)), \end{aligned} \tag{3.71}$$

and

$$\begin{aligned} P_k^A(u;O) &= P(u_k = u|Y_1^N) \\ &= \frac{1}{P(Y_1^N)} \sum_{e:u(e)=u} \sigma_k(e) \\ &= \frac{1}{P(Y_1^N)} \sum_{e:u(e)=u} A_{k-1}(s_k^S(e)) \cdot M_k(e) \cdot B_k(s_k^E(e)). \end{aligned} \tag{3.72}$$

$P(Y_1^N)$ is a constant for a given frame, so it is ignored in the following descriptions.

### 3.3.2 Log-MAP Algorithm

To construct the Log-MAP algorithm, the variables of MAP are converted into the logarithmic domain so that the computations consist mainly of additions instead of multiplications. Thus the values $A_k(s)$ and $B_k(s)$ within the MAP algorithm are represented as $\alpha_k(s)$ and $\beta_k(s)$ in the Log-MAP algorithm as follows:

$$\alpha_k(s) = \ln A_k(s), \tag{3.73}$$

$$\beta_k(s) = \ln B_k(s). \tag{3.74}$$

Both the input and the output of the decoder module will become LLRs, where LLR is the soft information defined as

$$\lambda_k = \ln \frac{P(u_k = 1|Y_1^N)}{P(u_k = 0|Y_1^N)}. \tag{3.75}$$

There are a series of LLR variables relating Log-MAP to MAP. All of them are defined in the following. Here $P_k^A(\cdot)$ is the complete probability, $\lambda_k^A(\cdot)$ is the complete LLR information, $I$ denotes an input variable, $O$ denotes an output variable, $u$ is the information word $(u^{(1)}, ..., u^{(j)}, ..., u^{(k_0)})$ composed of $k_0$ bits, $c$ is a codeword $(c^{(1)}, ..., c^{(j)}, ..., c^{(n_0)})$ composed of $n_0$ bits, and $k$ is the time index:

$$\lambda_k(c;I) = \ln \frac{P_k(c;I)}{P_k(c = \underline{0};I)} \tag{3.76}$$

$$\lambda_k(u;I) = \ln \frac{P_k(u;I)}{P_k(u=\underline{0};I)} \tag{3.77}$$

$$\lambda_k^A(c;O) = \ln \frac{P_k^A(c;O)}{P_k^A(c=\underline{0};O)} \tag{3.78}$$

$$\lambda_k^A(u;O) = \ln \frac{P_k^A(u;O)}{P_k^A(u=\underline{0};O)} \tag{3.79}$$

$$\lambda_k(c;O) = \ln \frac{P_k(c;O)}{P_k(c=\underline{0};O)} \tag{3.80}$$

$$\lambda_k(u;O) = \ln \frac{P_k(u;O)}{P_k(u=\underline{0};O)} \tag{3.81}$$

$$\lambda_k^A(c^{(j)};O) = \ln \frac{P_k^A(c^{(j)}=1;O)}{P_k^A(c^{(j)}=0;O)}, \quad j=1,...,n_0. \tag{3.82}$$

$$\lambda_k^A(u^{(j)};O) = \ln \frac{P_k^A(u^{(j)}=1;O)}{P_k^A(u^{(j)}=0;O)}, \quad j=1,...,k_0. \tag{3.83}$$

$$\lambda_k(c^{(j)};O) = \ln \frac{P_k(c^{(j)}=1;O)}{P_k(c^{(j)}=0;O)}, \quad j=1,...,n_0. \tag{3.84}$$

$$\lambda_k(u^{(j)};O) = \ln \frac{P_k(u^{(j)}=1;O)}{P_k(u^{(j)}=0;O)}, \quad j=1,...,k_0. \tag{3.85}$$

To evaluate the logarithm of summations, the $\max^*(\cdot)$ operation is used extensively in the Log-MAP algorithm. It is defined as

$$\max_e{}^*(f(e)) = \ln\left(\sum_e \exp(f(e))\right). \tag{3.86}$$

The $\max^*(\cdot)$ function is usually realized by successive pairwise operation when there are more than two eligible terms of $f(e)$. For two variables $x$ and $y$,

$$\max{}^*(x,y) = \ln(e^x+e^y) = \max(x,y)+\ln(1+e^{-|x-y|}) = \max(x,y)+f_c(|x-y|). \tag{3.87}$$

The $\max^*(\cdot)$ function has the following properties which will be used to rewrite or simplify the algorithm:

$$\max{}^*(x,y,z) = \max{}^*(x,\ \max{}^*(y,z)), \tag{3.88}$$

$$\max{}^*(x+y,x+z) = x+\ \max{}^*(y,z), \tag{3.89}$$

and

$$\max{}^*(-\infty,x) = x. \tag{3.90}$$

For a PCCC with two rate 1/2 RSC encoders denoted by RSC1 and RSC2, we have $k_0=1, n_0=2$. The transmitted signal is $X_k = (x_k^{(1)}, x_k^{(2)})$, and the received signal is $Y_k = (y_k^{(1)}, y_k^{(2)})$, with $x_k^{(1)}$ and $y_k^{(1)}$ corresponding to the $k$-th systematic bit. Here antipodal transmission with amplitude $A$ is assumed so that $x_k^{(1)}, x_k^{(2)} \in \{A, -A\}$.

For binary transmission, if 0 and 1 are equiprobable for a code bit, $P(x_k^{(j)} = A) = P(x_k^{(j)} = -A)$. In an AWGN channel, $\lambda_k(c^{(j)}; I)$ can be related to the received signal as follows:

$$
\begin{aligned}
\lambda_k(c^{(j)}; I) &= \ln \frac{P_k(c^{(j)} = 1; I)}{P_k(c^{(j)} = 0; I)} \\
&= \ln \frac{P(x_k^{(j)} = A | Y_1^N)}{P(x_k^{(j)} = -A | Y_1^N)} \\
&= \ln \frac{P(x_k^{(j)} = A | y_k^{(j)})}{P(x_k^{(j)} = -A | y_k^{(j)})} \\
&= \ln \frac{P(y_k^{(j)} | x_k^{(j)} = A) P(x_k^{(j)} = A) / P(y_k^{(j)})}{P(y_k^{(j)} | x_k^{(j)} = -A) P(x_k^{(j)} = -A) / P(y_k^{(j)})} \\
&= \ln \frac{P(y_k^{(j)} | x_k^{(j)} = A)}{P(y_k^{(j)} | x_k^{(j)} = -A)} \\
&= \ln \frac{\frac{1}{\sqrt{2\pi}\sigma} \exp\left(\frac{-1}{2\sigma^2}(y_k^{(j)} - A)^2\right)}{\frac{1}{\sqrt{2\pi}\sigma} \exp\left(\frac{-1}{2\sigma^2}(y_k^{(j)} + A)^2\right)} \\
&= \frac{2A}{\sigma^2} y_k^{(j)}
\end{aligned}
\tag{3.91}
$$

The term $L_c = 2A/\sigma^2$ is usually called the **channel reliability** since it indicates how much confidence can be put on the contaminated received signal in comparison to the *a priori* information about the information bits.

Now we start the derivation of Log-MAP with the branch metric. In Equation (3.64), $P(X_k|e)$ is either 1 or 0 depending on whether $X_k$ is associated with the edge $e$ or not. $P(s_k^E(e)|s_k^S(e))$ is determined by the *a priori* information $\Lambda_a(u_k)$ of the information bit:

$$
P(s_k^E(e)|s_k^S(e)) = \begin{cases} P(u_k = 1) = \frac{e^{\Lambda_a(u_k)}}{1+e^{\Lambda_a(u_k)}}, & u_k = 1. \\ P(u_k = 0) = \frac{1}{1+e^{\Lambda_a(u_k)}}, & u_k = 0. \end{cases} \tag{3.92}
$$

Here $\Lambda_a(u_k)$ is the input LLR of Log-MAP algorithm, and

$$
\Lambda_a(u_k) = \ln \frac{P(u_k = 1)}{P(u_k = 0)}. \tag{3.93}
$$

Thus,

$$
\ln P(s_k^E(e)|s_k^S(e)) = \begin{cases} \Lambda_a(u_k) - \ln\left(1 + e^{\Lambda_a(u_k)}\right), & u_k = 1. \\ -\ln\left(1 + e^{\Lambda_a(u_k)}\right), & u_k = 0. \end{cases} \tag{3.94}
$$

For an AWGN channel with noise variance $\sigma^2$,

$$\begin{aligned} P(Y_k|X_k) &= P(y_k^{(1)}|u_k) \cdot P(y_k^{(2)}|u_k, s_k^S(e), s_k^E(e)) \\ &= \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(\frac{-(y_k^{(1)} - x_k^{(1)})^2}{2\sigma^2}\right) \cdot \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(\frac{-(y_k^{(2)} - x_k^{(2)})^2}{2\sigma^2}\right). \end{aligned} \tag{3.95}$$

Omitting the constant which will be canceled and considering that $x_k^{(j)} = A(2c_k^{(j)} - 1), j = 1, 2$, we have

$$\begin{aligned} \ln P(Y_k|X_k) &= \frac{y_k^{(1)} x_k^{(1)}}{\sigma^2} + \frac{y_k^{(2)} x_k^{(2)}}{\sigma^2} \\ &= \frac{A}{\sigma^2}\left[y_k^{(1)}(2c_k^{(1)} - 1) + y_k^{(2)}(2c_k^{(2)} - 1)\right]. \end{aligned} \tag{3.96}$$

With $\lambda_k(c^{(j)}; I)$ expressed in Equation (3.91), Equation (3.96) can be also written as

$$\ln P(Y_k|X_k) = 0.5\lambda_k(c^{(1)}; I)(2c_k^{(1)} - 1) + 0.5\lambda_k(c^{(2)}; I)(2c_k^{(2)} - 1). \tag{3.97}$$

Using Equation (3.94) and (3.97), the path metric in the logarithmic domain is obtained as follows:

$$\begin{aligned} &\gamma_k(e) = \ln M_k(e) \\ = \; &\ln P(s_k^E(e)|s_k^S(e)) + \ln P(Y_k|X_k) \\ = \; &\begin{cases} \Lambda_a(u_k) - \ln(1 + e^{\Lambda_a(u_k)}) + 0.5\lambda_k(c^{(1)}; I) + 0.5\lambda_k(c^{(2)}; I)(2c_k^{(2)} - 1), & u_k = 1. \\ -\ln(1 + e^{\Lambda_a(u_k)}) - 0.5\lambda_k(c^{(1)}; I) + 0.5\lambda_k(c^{(2)}; I)(2c_k^{(2)} - 1), & u_k = 0. \end{cases} \end{aligned} \tag{3.98}$$

Note that this equation holds only when a transition exists between $s_k^S(e)$ and $s_k^E(e)$; otherwise $\gamma_k(e)$ is $-\infty$. Although the systematic bit for RSC2 is not transmitted, it is known to be identical to the systematic bit of RSC1, reordered by the interleaver. Since the input LLR $\lambda_k(c^{(1)}; I)$ is obtained by scaling the received signal of the systematic bit (see Equation (3.91)), we can interleave $\lambda_{1k}(c^{(1)}; I)$ of RSC1 to obtain $\lambda_{2k}(c^{(1)}; I)$ of RSC2; therefore $\lambda_{2k}(c^{(1)}; I) = \lambda_{1\alpha(k)}(c^{(1)}; I)$, where $\alpha(k)$ is the interleaver mapping. Thus, both RSC1 and RSC2 have $\lambda_k(c^{(1)}; I)$ and $\lambda_k(c^{(2)}; I)$ for an information bit, and both encoders have the same computation formula. The terms $\alpha_k(s)$ and $\beta_k(s)$ are computed as follows:

$$\alpha_k(s) = \ln \sum_{e: s_k^E(e) = s} A_{k-1}(s_k^S(e)) \cdot M_k(e)$$

$$= \max_{e:s_k^E(e)=s}{}^{*} \left[\alpha_{k-1}(s_k^S(e)) + \gamma_k(e)\right], \quad k = 1, ..., N-1, \tag{3.99}$$

$$\begin{aligned} \beta_k(s) &= \ln \sum_{e:s_{k+1}^S(e)=s} B_{k+1}(s_{k+1}^E(e)) \cdot M_{k+1}(e) \\ &= \max_{e:s_{k+1}^S(e)=s}{}^{*} \left[\beta_{k+1}(s_{k+1}^E(e)) + \gamma_{k+1}(e)\right], \quad k = N-1, ..., 1, \end{aligned} \tag{3.100}$$

where the $\max^*(\cdot)$ operation is defined in Equation (3.86). Since the values of $\alpha_k(s)$ and $\beta_k(s)$ increase almost monotonically as their computation proceeds through the trellis, it is a common practice to normalize them by subtracting a number which is constant with respect to all $s$ at time $k$. A good choice is to normalize by the maximum value,

$$\tilde{\alpha}_k(s) = \alpha_k(s) - \max_s \alpha_k(s), \tag{3.101}$$

$$\tilde{\beta}_k(s) = \beta_k(s) - \max_s \beta_k(s), \tag{3.102}$$

and use $\tilde{\alpha}_k(s)$, $\tilde{\beta}_k(s)$ for the subsequent computations. This practice will ensure that $\alpha_k(s)$ and $\beta_k(s)$ do not overflow in a real implementation. It has been proven that adding a constant to all $\alpha_k(s)$ or $\beta_k(s)$ for a given $k$ has no influence on the soft output since the constant will be canceled eventually. Although this is required for regular computation, Section 5.2.3 will show that this subtractive normalization is not necessary when integer representation and two's complement arithmetic are used for the representation and computation of $\alpha$ and $\beta$.

Analogous to Equations (3.67) to (3.69), the $\alpha$ and $\beta$ computations are initialized. Suppose the encoder starts with a known state $S_0$. Then the $\alpha_k(s)$ computation will then be initialized as

$$\alpha_0(s) = \begin{cases} 0, & s = S_0, \\ -\infty, & \text{otherwise.} \end{cases} \tag{3.103}$$

If the trellis is terminated to a known state $S_N$, then the $\beta_k(s)$ computation will be initialized as

$$\beta_N(s) = \begin{cases} 0, & s = S_N, \\ -\infty, & \text{otherwise.} \end{cases} \tag{3.104}$$

If the final state of the trellis is unknown, then

$$\beta_N(s) = 0 \quad \text{(or any other constant)}, \qquad \forall s. \tag{3.105}$$

For numerical computation, a very large value can be used in place of $\infty$.

The complete LLR information of the information bit is

$$
\begin{aligned}
\lambda_k^A(u;O) &= \ln \sum_{e:u(e)=1} A_{k-1}(s_k^S(e)) \cdot M_k(e) \cdot B_k(s_k^E(e)) \\
&\quad - \ln \sum_{e:u(e)=0} A_{k-1}(s_k^S(e)) \cdot M_k(e) \cdot B_k(s_k^E(e)) \\
&= \max_{e:u(e)=1}{}^{*} \left[\alpha_{k-1}(s_k^S(e)) + \gamma_k(e) + \beta_k(s_k^E(e))\right] \\
&\quad - \max_{e:u(e)=0}{}^{*} \left[\alpha_{k-1}(s_k^S(e)) + \gamma_k(e) + \beta_k(s_k^E(e))\right]. \qquad (3.106)
\end{aligned}
$$

Substituting Equation (3.98) for $\gamma_k(e)$ and pulling out the common factor, we have

$$
\begin{aligned}
\lambda_k^A(u;O) &= \max_{e:u(e)=1}{}^{*} \Big[\Lambda_a(u_k) - \ln(1+e^{\Lambda_a(u_k)}) + 0.5\lambda_k(c^{(1)};I) + 0.5\lambda_k(c^{(2)};I)(2c_k^{(2)}-1) \\
&\qquad +\alpha_{k-1}(s_k^S(e)) + \beta_k(s_k^E(e))\Big] \\
&\quad - \max_{e:u(e)=0}{}^{*} \Big[-\ln(1+e^{\Lambda_a(u_k)}) - 0.5\lambda_k(c^{(1)};I) + 0.5\lambda_k(c^{(2)};I)(2c_k^{(2)}-1) \\
&\qquad +\alpha_{k-1}(s_k^S(e)) + \beta_k(s_k^E(e))\Big] \\
&= (\Lambda_a(u_k) - \ln(1+e^{\Lambda_a(u_k)}) + 0.5\lambda_k(c^{(1)};I)) \\
&\quad + \max_{e:u(e)=1}{}^{*} \left[\alpha_{k-1}(s_k^S(e)) + 0.5\lambda_k(c^{(2)};I)(2c_k^{(2)}-1) + \beta_k(s_k^E(e))\right] \\
&\quad -(-\ln(1+e^{\Lambda_a(u_k)}) - 0.5\lambda_k(c^{(1)};I)) \\
&\quad - \max_{e:u(e)=0}{}^{*} \left[\alpha_{k-1}(s_k^S(e)) + 0.5\lambda_k(c^{(2)};I)(2c_k^{(2)}-1) + \beta_k(s_k^E(e))\right] \\
&= \Lambda_a(u_k) + \lambda_k(c^{(1)};I) \\
&\quad + \max_{e:u(e)=1}{}^{*} \left[\alpha_{k-1}(s_k^S(e)) + 0.5\lambda_k(c^{(2)};I)(2c_k^{(2)}-1) + \beta_k(s_k^E(e))\right] \\
&\quad - \max_{e:u(e)=0}{}^{*} \left[\alpha_{k-1}(s_k^S(e)) + 0.5\lambda_k(c^{(2)};I)(2c_k^{(2)}-1) + \beta_k(s_k^E(e))\right]. \qquad (3.107)
\end{aligned}
$$

The complete LLR information ($\lambda_k^A(u;O)$) is the summation of the following three terms: the *a priori* information ($\Lambda_a(u_k)$), the systematic information ($\lambda_k(c^{(1)};I)$), and the extrinsic information (the remainder).

Since both constituent decoders share the same systematic information, $\lambda_k(c^{(1)};I)$, together with $\Lambda_a(u_k)$, has to be removed from the complete information to provide the proper extrinsic information [55]. The extrinsic information will be sent to the next constituent decoder and used as its *a priori* information. The extrinsic information is

$$
\Lambda_e(u_k) = \lambda_k^A(u;O) - \Lambda_a(u_k) - \lambda_k(c^{(1)};I) \qquad (3.108)
$$

$$= \max_{e:u(e)=1}^{*} \left[\alpha_{k-1}(s_k^S(e)) + 0.5\lambda_k(c^{(2)};I)(2c_k^{(2)}-1) + \beta_k(s_k^E(e))\right]$$
$$- \max_{e:u(e)=0}^{*} \left[\alpha_{k-1}(s_k^S(e)) + 0.5\lambda_k(c^{(2)};I)(2c_k^{(2)}-1) + \beta_k(s_k^E(e))\right].$$

Equations (3.98), (3.99), (3.100), (3.106), and (3.108) make up the conventional Log-MAP algorithm for a constituent decoder, where $\lambda_k(c;I)$ is computed using Equation (3.91).

The Log-MAP algorithm can be embedded in the iterative turbo decoder. Let the first constituent decoder be DEC1, and the second constituent decoder be DEC2. On the first iteration, the *a priori* information $\Lambda_{1a}(u_k)$ of DEC1 is initialized to zero, $\forall k$, thus assuming that 1 and 0 are equiprobable for an information bit.

After DEC1 produces its output, its extrinsic information $\Lambda_{1e}(u_k)$ will be interleaved and then passed as the *a priori* information for DEC2: $\Lambda_{2a}(u_k) = \Lambda_{1e}(u_{\alpha(k)})$. After DEC2 generates its output, one iteration is completed. The extrinsic information $\Lambda_{2e}(u_k)$ from DEC2 can be deinterleaved and fed back as the *a priori* information of DEC1 in the next iteration: $\Lambda_{1a}(u_{\alpha(k)}) = \Lambda_{2e}(u_k)$. All subsequent iterations are done by repeating the above procedure. The whole process is illustrated in Figure 3.17.

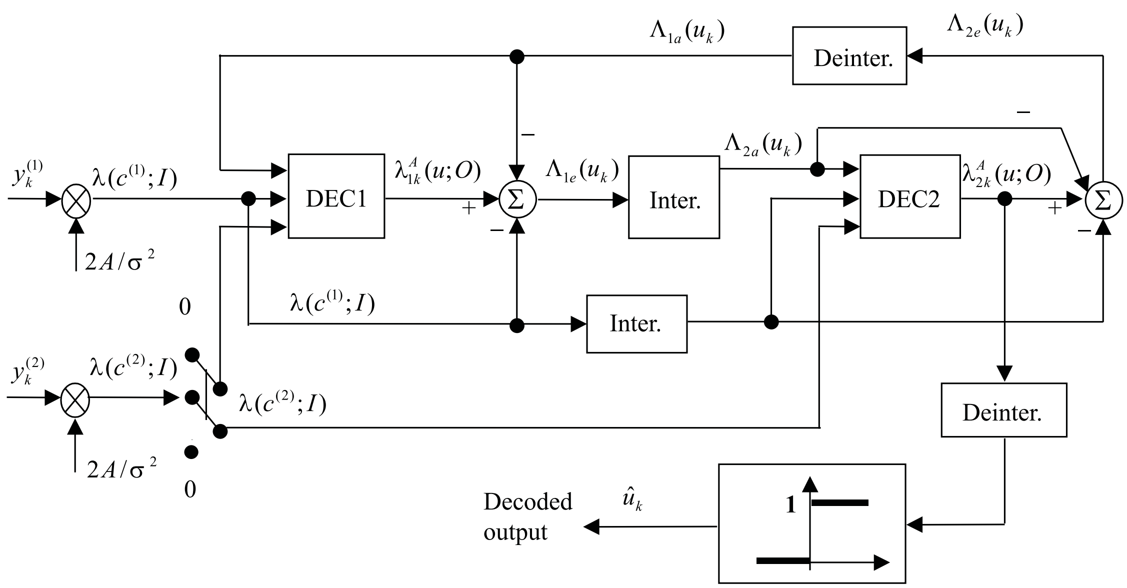



Figure 3.17: PCCC decoder with conventional Log-MAP algorithm.

## 3.4 A Simple SISO Module for PCCCs

Based on the conventional Log-MAP algorithm, a generalized soft-input soft-output (SISO) decoding module can be obtained with a few modifications. This section shows that the SISO module has lower complexity and is easier to extend to the case of SCCCs. In Section 3.5, a general SISO module will be fully developed for decoding both PCCCs and SCCCs.

Similar to Section 3.3, consider the PCCC with two rate 1/2 RSC encoders. The SISO module has two inputs, $\lambda(u;I)$ and $\lambda(c;I)(=[\lambda(c^{(1)};I),\lambda(c^{(2)};I)])$. One output, $\lambda(u;O)$, is generated and passed to the next decoding module as $\lambda(u;I)$. These variables are defined in Equations (3.76), (3.77), and (3.81). Three major modifications of the conventional Log-MAP algorithm are necessary to obtain the SISO module.

1. Notice that any constant common to $u_k=0$ and $u_k=1$ for all $\gamma_k(e)$ at a given $k$ will be canceled during the computation of the soft output; thus, it can be ignored from the computation. Using $\lambda_k(u;I)$ in place of $\Lambda_a(u_k)$ in Equation (3.98), $\gamma_k(e)$ can be rewritten as follows:

$$\begin{aligned}\gamma_k(e) &= \begin{cases} \left[\lambda_k(u;I)+c_k^{(1)}\lambda_k(c^{(1)};I)+c_k^{(2)}\lambda_k(c^{(2)};I)\right] \\ \quad +\left[-\ln(1+e^{\lambda_k(u;I)})-0.5\lambda_k(c^{(1)};I)-0.5\lambda_k(c^{(2)};I)\right], \\ \qquad\qquad c_k^{(1)}=u_k=1. \\ \left[c_k^{(1)}\lambda_k(c^{(1)};I)+c_k^{(2)}\lambda_k(c^{(2)};I)\right] \\ \quad +\left[-\ln(1+e^{\lambda_k(u;I)})-0.5\lambda_k(c^{(1)};I)-0.5\lambda_k(c^{(2)};I)\right], \\ \qquad\qquad c_k^{(1)}=u_k=0. \end{cases} \\ &= u_k\lambda_k(u;I)+c_k^{(1)}\lambda_k(c^{(1)};I)+c_k^{(2)}\lambda_k(c^{(2)};I)+h_\gamma, \qquad c_k^{(1)}=u_k\in\{0,1\}.\end{aligned} \tag{3.109}$$

   The constant

$$h_\gamma=-\ln(1+e^{\lambda_k(u;I)})-0.5\lambda_k(c^{(1)};I)-0.5\lambda_k(c^{(2)};I)$$

   can be ignored within the computation. In this way the computation for the branch metric is simplified. Also, notice that $u_k, c_k^{(j)}\in\{0,1\}, j=1,2$. Whenever $u_k$ or $c_k^{(j)}$ equals 0, the corresponding terms can be dropped. This further removes half of the terms in $\gamma_k(\cdot)$.

2. While the conventional Log-MAP algorithm treats both RSC1 and RSC2 as rate 1/2 codes, SISO treats only RSC1 as a rate 1/2 code, but RSC2 as a rate

1/1 code. In other words, the codeword of RSC1 is $(c_{1k}^{(1)}, c_{1k}^{(2)}) = (u_k, c_{1k}^{(2)})$, and the codeword of RSC2 is $(c_{2k}^{(2)})$. Thus, the branch metric is

$$\begin{cases} \gamma_{1k}(e) = u_k\lambda_{1k}(u;I) + c_{1k}^{(1)}\lambda_{1k}(c^{(1)};I) & +c_{1k}^{(2)}\lambda_{1k}(c^{(2)};I), \qquad \text{for RSC1.} \\ \gamma_{2k}(e) = u_k\lambda_{2k}(u;I) & +c_{2k}^{(2)}\lambda_{2k}(c^{(2)};I), \qquad \text{for RSC2.} \end{cases} \tag{3.110}$$

The computation for $\gamma_{2k}(e)$ is simpler because the systematic bit $c_{2k}^{(1)}$ is not considered. As a result, interleaving systematic information $\lambda_{1k}(c^{(1)};I)$ for DEC2 is unnecessary.

3. Since $\lambda_k(c^{(1)};I)$ is included in the branch metrics for DEC1, but not for DEC2, the SISO output $\lambda(u;O)$ is different from the extrinsic information $\Lambda_e(u_k)$. Replacing $\Lambda_a(u_k)$ by $\lambda_k(u;I)$ in Equation (3.107), the complete information for DEC1 is

$$\begin{aligned} \lambda_{1k}^{A}(u;O) &= \lambda_{1k}(u;I) + \lambda_{1k}(c^{(1)};I) \\ &\quad + \max_{e:u(e)=1}{}^{*} \left[\alpha_{1,k-1}(s_k^S(e)) + 0.5\lambda_{1k}(c^{(2)};I)(2c_{1k}^{(2)} - 1) + \beta_{1k}(s_k^E(e))\right] \\ &\quad - \max_{e:u(e)=0}{}^{*} \left[\alpha_{1,k-1}(s_k^S(e)) + 0.5\lambda_{1k}(c^{(2)};I)(2c_{1k}^{(2)} - 1) + \beta_{1k}(s_k^E(e))\right]. \\ &= \lambda_{1k}(u;I) + \lambda_{1k}(c^{(1)};I) + \Lambda_{1e}(u_k). \end{aligned} \tag{3.111}$$

Since $\lambda_{2k}(c^{(1)};I)$ is not included in $\gamma_{2k}(e)$ for DEC2, $\lambda_{1k}(c^{(1)};I)$ should be included in the output of DEC1. So,

$$\begin{aligned} \lambda_{1k}(u;O) &= \lambda_{1k}^{A}(u;O) - \lambda_{1k}(u;I) \\ &= \lambda_{1k}(c^{(1)};I) + \Lambda_{1e}(u_k) \end{aligned} \tag{3.112}$$

is the output for DEC1, which is interleaved to generate the input $\lambda_{2k}(u;I)$ for DEC2. Therefore, the systematic information is passed to DEC2 as a part of input $\lambda_{2k}(u;I)$, which is

$$\begin{aligned} \lambda_{2k}(u;I) &= \lambda_{1\alpha(k)}(u;O) \\ &= \Lambda_{1e}(u_{\alpha(k)}) + \lambda_{1\alpha(k)}(c^{(1)};I) \\ &= \Lambda_{2a}(u_k) + \lambda_{1\alpha(k)}(c^{(1)};I). \end{aligned} \tag{3.113}$$

The complete information for DEC2 is similar to Equation (3.107), but without the $\lambda_k(c^{(1)};I)$ term:

$$\lambda_{2k}^{A}(u;O) = \lambda_{2k}(u;I)$$

$$
\begin{aligned}
& + \max^*_{e:u(e)=1} \left[\alpha_{2,k-1}(s_k^S(e)) + 0.5\lambda_{2k}(c^{(2)};I)(2c_{2k}^{(2)}-1) + \beta_{2k}(s_k^E(e))\right] \\
& - \max^*_{e:u(e)=0} \left[\alpha_{2,k-1}(s_k^S(e)) + 0.5\lambda_{2k}(c^{(2)};I)(2c_{2k}^{(2)}-1) + \beta_{2k}(s_k^E(e))\right] \\
= \; & \left[\Lambda_{2a}(u_k) + \lambda_{1\alpha(k)}(c^{(1)};I)\right] + \Lambda_{2e}(u_k) \qquad (3.114)
\end{aligned}
$$

The output of DEC2 is

$$
\begin{aligned}
\lambda_{2k}(u;O) &= \lambda_{2k}^A(u;O) - \lambda_{2k}(u;I) \\
&= \Lambda_{2e}(u_k). \qquad (3.115)
\end{aligned}
$$

In consequence,

$$
\begin{aligned}
\lambda_{1\alpha(k)}(u;I) &= \lambda_{2k}(u;O) = \Lambda_{2e}(u_k) \\
&= \Lambda_{1a}(u_{\alpha(k)}). \qquad (3.116)
\end{aligned}
$$

Substituting Equation (3.116) into Equation (3.111), the complete information $\lambda_{1k}^A(u;O)$ is composed of *a priori* information $\Lambda_{1a}(u_k)$, systematic information $\lambda_{1k}(c^{(1)};I)$, and extrinsic information $\Lambda_{1e}(u_k)$, just like $\lambda_{2k}^A(u;O)$ in Equation (3.114). Both outputs can be expressed as

$$
\lambda_k(u;O) = \lambda_k^A(u;O) - \lambda_k(u;I). \qquad (3.117)
$$

They can be obtained directly using

$$
\begin{aligned}
\lambda_k(u;O) = \; & \max^*_{e:u(e)=1} \left[\alpha_{k-1}(s_k^S(e)) + c_k^{(2)}(e)\lambda_k(c^{(2)};I) + \beta_k(s_k^E(e))\right] \\
& - \max^*_{e:u(e)=0} \left[\alpha_{k-1}(s_k^S(e)) + c_k^{(2)}(e)\lambda_k(c^{(2)};I) + \beta_k(s_k^E(e))\right] \\
& + \begin{cases} \lambda_k(c^{(1)};I), & \text{for SISO1,} \\ 0, & \text{for SISO2,} \end{cases} \qquad (3.118)
\end{aligned}
$$

without computing $\lambda_k^A(u;O)$. Here SISO1 and SISO2 are used to represent the first and the second constituent SISO decoding modules.

In summary, the computation in SISO module is composed of Equations (3.110), (3.99), (3.100), and (3.118) (or using (3.106) and (3.117) to get $\lambda_k(u;O)$). The iterative decoding procedure is the same as that of conventional Log-MAP. The decoding process is illustrated later in Figure 3.18.

Essentially the PCCC decoder using SISO modules is equivalent to that using the conventional Log-MAP. The following relationships exist:

$$\begin{cases} \lambda_{1k}(u;I) & = \Lambda_{1a}(u_k), & \text{for SISO1.} \\ \lambda_{2k}(u;I) & = \Lambda_{2a}(u_k) + \lambda_{1\alpha(k)}(c^{(1)};I), & \text{for SISO2.} \\ \lambda_{1k}(u;O) & = \Lambda_{1e}(u_k) + \lambda_{1k}(c^{(1)};I), & \text{for SISO1.} \\ \lambda_{2k}(u;O) & = \Lambda_{2e}(u_k), & \text{for SISO2.} \end{cases} \tag{3.119}$$

However, SISO modules are more efficient for implementation. Subtraction of systematic information from the complete information is unnecessary. The SISO2 module does not need $\lambda_{2k}(c^{(1)};I)$. Interleaving of systematic information $\lambda_{2k}(c^{(1)};I)$ is eliminated and the computation of the branch metric is reduced.

## 3.5 A General SISO Module for PCCCs and SCCCs

In this section, the derivation of the SISO module will be generalized so that it can be used to decode both PCCCs and SCCCs [57]. First, the algorithm is derived thoroughly, starting with a probability analysis. Two versions of SISO are presented, multiplicative SISO and additive SISO. Multiplicative SISO is the extension of MAP algorithm which deals with the probabilities; additive SISO is the extension of Log-MAP which deals with the log-likelihood ratios. Multiplicative SISO is the direct result of the derivation, while additive SISO is a transformation of multiplicative SISO. Additive SISO is usually employed in practice and is the one referred to when the term “SISO” is used throughout the remainder of this dissertation.

In general, a SISO module has two input vectors, $\left[\lambda(u^{(1)};I),...,\lambda(u^{(k_0)};I)\right]$ (input LLR of information bits) and $\left[\lambda(c^{(1)};I),...,\lambda(c^{(n_0)};I)\right]$ (input LLR of codeword bits). After the processing, two output vectors are produced, $\left[\lambda(u^{(1)};O),...,\lambda(u^{(k_0)};O)\right]$ (output LLR of information bits) and $\left[\lambda(c^{(1)};O),\ ...,\ \lambda(c^{(n_0)};O)\right]$ (output LLR of codeword bits). $\lambda(c^{(i)};I)$ is usually determined by the soft value of the received signal (see Equation (3.91)). In particular, for a systematic constituent code, $\lambda(c^{(1)};I)$ is called the systematic information, and $\left[\lambda(u^{(1)};I),...,\lambda(u^{(k_0)};I)\right]$ is the LLR of *a priori* probability if any. Otherwise each element of this vector is initialized to zero.

### 3.5.1 Output Probabilities for Information Symbols and Codeword Symbols

Recall the branch metric in Equation (3.64):

$$M_k(e) = \sum_{X_k} P(s_k^E(e)|s_k^S(e)) \cdot P(X_k|e) \cdot P(Y_k|X_k). \tag{3.120}$$

The first term can be expressed as

$$\begin{aligned} P(s_k^E(e)|s_k^S(e)) &= P(u_k) \\ &= P_k(u(e); I). \end{aligned} \tag{3.121}$$

The second term $P(X_k|e)$ can only be 0 or 1 depending on whether $s_k^S(e)$ and $s_k^E(e)$ are connected with the output $X_k$. If all possible $X_k$ are equiprobable, the third term is

$$\begin{aligned} P(Y_k|X_k) &= \frac{P(X_k|Y_k)P(Y_k)}{P(X_k)} \\ &= h_{M'} P(X_k|Y_k) \\ &= h_{M'} P(c_k|Y_k) \\ &= h_{M'} P_k(c(e); I), \end{aligned} \tag{3.122}$$

where $h_{M'} = P(Y_k)/P(X_k)$ is a constant for a given frame and will be ignored in the following descriptions. In a trellis without multiple connections between two states, if $s_k^S(e)$ and $s_k^E(e)$ are connected,

$$M_k(e) = P_k(u(e); I) \cdot P_k(c(e); I). \tag{3.123}$$

If there is no connection between $s_k^S(e)$ and $s_k^E(e)$, $M_k(e) = 0$. The APP for the codeword symbol, $P_k^A(c; O)$, and the information word, $P_k^A(u; O)$, are

$$\begin{aligned} P_k^A(c; O) &= P(c_k = c|Y_1^N) \\ &= \frac{1}{P(Y_1^N)} \sum_{e:c(e)=c} \sigma_k(e) \\ &= \frac{1}{P(Y_1^N)} \sum_{e:c(e)=c} A_{k-1}(s_k^S(e)) \cdot M_k(e) \cdot B_k(s_k^E(e)) \\ &= h_c \sum_{e:c(e)=c} A_{k-1}(s_k^S(e)) \cdot P_k(u(e); I) \cdot P_k(c(e); I) \cdot B_k(s_k^E(e)) \\ &= h_c P_k(c(e); I) \sum_{e:c(e)=c} A_{k-1}(s_k^S(e)) \cdot P_k(u(e); I) \cdot B_k(s_k^E(e)), \end{aligned} \tag{3.124}$$

and

$$
\begin{aligned}
P_k^A(u;O) &= P(u_k = u | Y_1^N) \\
&= \frac{1}{P(Y_1^N)} \sum_{e:u(e)=u} \sigma_k(e) \\
&= \frac{1}{P(Y_1^N)} \sum_{e:u(e)=u} A_{k-1}(s_k^S(e)) \cdot M_k(e) \cdot B_k(s_k^E(e)) \\
&= h_c \sum_{e:u(e)=u} A_{k-1}(s_k^S(e)) \cdot P_k(u(e);I) \cdot P_k(c(e);I) \cdot B_k(s_k^E(e)) \\
&= h_c P_k(u(e);I) \sum_{e:u(e)=u} A_{k-1}(s_k^S(e)) \cdot P_k(c(e);I) \cdot B_k(s_k^E(e)). \quad (3.125)
\end{aligned}
$$

Here $h_c = 1/P(Y_1^N)$ is a constant and will be ignored. Computing the ratio of the output APP and the input conditional probability, we obtain the output probability for the codeword symbol, $P_k(c;O)$, and the information word, $P_k(u;O)$, as follows:

$$
P_k(c;O) = \frac{P_k^A(c;O)}{P_k(c;I)} = \sum_{e:c(e)=c} A_{k-1}(s_k^S(e)) \cdot P_k(u(e);I) \cdot B_k(s_k^E(e)), \quad (3.126)
$$

and

$$
P_k(u;O) = \frac{P_k^A(u;O)}{P_k(u;I)} = \sum_{e:u(e)=u} A_{k-1}(s_k^S(e)) \cdot P_k(c(e);I) \cdot B_k(s_k^E(e)). \quad (3.127)
$$

The output probability represents the new information obtained by the current decoding module, and it comes from the probability distributions of all symbols of the sequence except the $k$-th ones, $P_k(c;I)$ and $P_k(u;I)$. The output probabilities can be used as the input probabilities for the next decoding module.

### 3.5.2 Output Probabilities for Information Bits and Codeword Bits

Consider a trellis encoder whose input symbol has $k_0$ bits and whose output symbol has $n_0$ bits. Assuming independency between the bits in a symbol,

$$
P_k(c;I) = \prod_{j=1}^{n_0} P_k(c^{(j)};I), \quad (3.128)
$$

and

$$
P_k(u;I) = \prod_{j=1}^{k_0} P_k(u^{(j)};I), \quad (3.129)
$$

where $c^{(j)} \in \{0,1\}$ represents the value of the $j$-th bit of the output symbol $c$ at time $k$, and $u^{(j)}$ represents the value of the $j$-th bit of the input symbol $u$ at time $k$. Equations (3.128) and (3.129) are valid when a bit interleaver is used ahead of the channel instead of a symbol interleaver, or the channel is AWGN. Using Equation (3.128) along with Equation (3.126), the complete information for bit $c^{(j)}$ is

$$\begin{aligned} P_k^A(c^{(j)};O) &= \sum_{c:c^{(j)}(e)=c^{(j)}} P_k^A(c(e);O) \\ &= \sum_{c:c^{(j)}(e)=c^{(j)}} P_k(c(e);I)\cdot P_k(c(e);O) \\ &= P_k(c^{(j)};I) \sum_{c:c^{(j)}(e)=c^{(j)}} P_k(c(e);O) \prod_{i=1,i\neq j}^{n_0} P_k(c^{(i)}(e);I), \quad j=1,...,n_0. \end{aligned} \tag{3.130}$$

The output probability for bit $c^{(j)}$ is

$$\begin{aligned} P_k(c^{(j)};O) &= \frac{P_k^A(c^{(j)};O)}{P_k(c^{(j)};I)} \\ &= \sum_{c:c^{(j)}(e)=c^{(j)}} P_k(c(e);O) \prod_{i=1,i\neq j}^{n_0} P_k(c^{(i)}(e);I), \quad j=1,...,n_0. \end{aligned} \tag{3.131}$$

Similarly, the complete probability and the output probability can be derived for an information bit as follows:

$$P_k^A(u^{(j)};O) = P_k(u^{(j)};I) \sum_{u:u^{(j)}(e)=u^{(j)}} P_k(u(e);O) \prod_{i=1,i\neq j}^{k_0} P_k(u^{(i)}(e);I), \quad j=1,...,k_0. \tag{3.132}$$

$$\begin{aligned} P_k(u^{(j)};O) &= \frac{P_k^A(u^{(j)};O)}{P_k(u^{(j)};I)} \\ &= \sum_{u:u^{(j)}(e)=u^{(j)}} P_k(u(e);O) \prod_{i=1,i\neq j}^{k_0} P_k(u^{(i)}(e);I), \quad j=1,...,k_0. \end{aligned} \tag{3.133}$$

### 3.5.3 Binary Transmission

Suppose the relationship between the codeword bit and the transmitted symbol is as follows:

$$x_k^{(j)} = \begin{cases} A, & \text{when } c_k^{(j)}=1, \\ -A, & \text{when } c_k^{(j)}=0, \end{cases} \qquad k=1,...,N;\ j=1,...,n_0. \tag{3.134}$$

Define $R^c_{kj}$ and $R^u_{kj}$ as follows:

$$R^c_{kj} \triangleq \frac{P_k(c^{(j)}=1;I)}{P_k(c^{(j)}=0;I)} = \frac{P(c^{(j)}_k=1|Y^N_1)}{P(c^{(j)}_k=0|Y^N_1)}, \tag{3.135}$$

and

$$R^u_{kj} \triangleq \frac{P_k(u^{(j)}=1;I)}{P_k(u^{(j)}=0;I)} = \frac{P(u^{(j)}_k=1)}{P(u^{(j)}_k=0)}. \tag{3.136}$$

Considering Equation (3.135) and that $P(c^{(j)}_k=1|Y^N_1)+P(c^{(j)}_k=0|Y^N_1)=1$, we have

$$P_k(c^{(j)}=1;I) = \frac{R^c_{kj}}{1+R^c_{kj}}, \tag{3.137}$$

and

$$P_k(c^{(j)}=0;I) = \frac{1}{1+R^c_{kj}}, \tag{3.138}$$

or in general

$$P_k(c^{(j)};I) = \frac{(R^c_{kj})^{c^{(j)}}}{1+R^c_{kj}}. \tag{3.139}$$

Combining Equation (3.139) with (3.128), we obtain

$$\begin{aligned} P_k(c;I) &= \prod_{j=1}^{n_0} P_k(c^{(j)};I) \\ &= h_{pc}\prod_{j=1}^{n_0}(R^c_{kj})^{c^{(j)}}. \end{aligned} \tag{3.140}$$

Similarly, for the information bit we have

$$P_k(u^{(j)};I) = \frac{(R^u_{kj})^{u^{(j)}}}{1+R^u_{kj}},$$

which, when substituted into Equation (3.129), yields

$$\begin{aligned} P_k(u;I) &= \prod_{j=1}^{k_0} P_k(u^{(j)};I) \\ &= h_{pu}\prod_{j=1}^{k_0}(R^u_{kj})^{u^{(j)}}. \end{aligned} \tag{3.141}$$

Here $h_{pc} = \prod_{j=1}^{n_0}(1+R^c_{kj})^{-1}$ and $h_{pu} = \prod_{j=1}^{k_0}(1+R^u_{kj})^{-1}$ are constants that are independent of the codeword $c$ and the information word $u$, respectively.

### 3.5.4 Multiplicative SISO

Using the previous results, we obtain the following expressions for multiplicative SISO.

1. Considering Equations (3.123), (3.140), and (3.141), the branch metric is

$$M_k(e) = \prod_{j=1}^{n_0} (R_{kj}^c)^{c^{(j)}} \cdot \prod_{j=1}^{k_0} (R_{kj}^u)^{u^{(j)}}, \tag{3.142}$$

   where the constants are omitted.

2. From Equations (3.65), (3.66), and (3.142), the forward and backward recursions are obtained as:

$$\begin{aligned} A_k(s) &= \sum_{e:s_k^E(e)=s} A_{k-1}(s_k^S(e)) \cdot M_k(e), \quad k = 1, ..., N-1. \\ &= \sum_{e:s_k^E(e)=s} A_{k-1}(s_k^S(e)) \cdot \prod_{j=1}^{n_0} (R_{kj}^c)^{c^{(j)}} \cdot \prod_{j=1}^{k_0} (R_{kj}^u)^{u^{(j)}} \qquad (3.143) \\ B_k(s) &= \sum_{e:s_{k+1}^S(e)=s} B_{k+1}(s_{k+1}^E(e)) \cdot M_{k+1}(e), \quad k = N-1, ..., 1. \\ &= \sum_{e:s_{k+1}^S(e)=s} B_{k+1}(s_{k+1}^E(e)) \cdot \prod_{j=1}^{n_0} (R_{k+1,j}^c)^{c^{(j)}} \cdot \prod_{j=1}^{k_0} (R_{k+1,j}^u)^{u^{(j)}} \qquad (3.144) \end{aligned}$$

   The $A_k(s)$ computation is initialized by Equation (3.67), and the $B_k(s)$ computation is initialized by Equation (3.68) or (3.69) depending on the knowledge of the final state.

3. When the constants are omitted from Equations (3.124), (3.125), and (3.142), the complete information for the codeword symbol and for the information symbol are obtained as:

$$\begin{aligned} P_k^A(c;O) &= \sum_{e:c(e)=c} A_{k-1}(s_k^S(e)) \cdot M_k(e) \cdot B_k(s_k^E(e)) \qquad (3.145) \\ &= \left( \prod_{j=1}^{n_0} (R_{kj}^c)^{c^{(j)}} \right) \sum_{e:c(e)=c} A_{k-1}(s_k^S(e)) \cdot \prod_{j=1}^{k_0} (R_{kj}^u)^{u^{(j)}} \cdot B_k(s_k^E(e)) \\ P_k^A(u;O) &= \sum_{e:u(e)=u} A_{k-1}(s_k^S(e)) \cdot M_k(e) \cdot B_k(s_k^E(e)) \qquad (3.146) \\ &= \left( \prod_{j=1}^{k_0} (R_{kj}^u)^{u^{(j)}} \right) \sum_{e:u(e)=u} A_{k-1}(s_k^S(e)) \cdot \prod_{j=1}^{n_0} (R_{kj}^c)^{c^{(j)}} \cdot B_k(s_k^E(e)) \end{aligned}$$

4. From Equations (3.126), (3.127), (3.145), and (3.146), the output probabilities for the codeword symbol and for the information symbol are given as follows:

$$\begin{aligned} P_k(c;O) &= \frac{P_k^A(c;O)}{\prod_{j=1}^{n_0}(R_{kj}^c)^{c^{(j)}}} \\ &= \sum_{e:c(e)=c} A_{k-1}(s_k^S(e)) \cdot \prod_{j=1}^{k_0}(R_{kj}^u)^{u^{(j)}} \cdot B_k(s_k^E(e)) \qquad (3.147) \\ P_k(u;O) &= \frac{P_k^A(u;O)}{\prod_{j=1}^{k_0}(R_{kj}^u)^{u^{(j)}}} \\ &= \sum_{e:u(e)=u} A_{k-1}(s_k^S(e)) \cdot \prod_{j=1}^{n_0}(R_{kj}^c)^{c^{(j)}} \cdot B_k(s_k^E(e)) \qquad (3.148) \end{aligned}$$

5. The complete information for the codeword bits and for the information bits are as follows:

$$\begin{aligned} P_k^A(c^{(j)};O) &= \sum_{e:c^{(j)}(e)=c^{(j)}} A_{k-1}(s_k^S(e)) \cdot M_k(e) \cdot B_k(s_k^E(e)), \\ & \qquad j = 1, ..., n_0. \qquad (3.149) \\ P_k^A(u^{(j)};O) &= \sum_{e:u^{(j)}(e)=u^{(j)}} A_{k-1}(s_k^S(e)) \cdot M_k(e) \cdot B_k(s_k^E(e)), \\ & \qquad j = 1, ..., k_0. \qquad (3.150) \end{aligned}$$

6. From Equations (3.131) and (3.133), the output probabilities for the codeword bits and for the information bits are as follows:

$$\begin{aligned} P_k(c^{(j)};O) &= \frac{P_k^A(c^{(j)};O)}{P_k(c^{(j)};I)}, \qquad j = 1, ..., n_0. \qquad (3.151) \\ &= \sum_{e:c^{(j)}(e)=c^{(j)}} A_{k-1}(s_k^S(e)) \cdot \prod_{i=1,i\neq j}^{n_0} (R_{ki}^c)^{c^{(i)}} \cdot \prod_{i=1}^{i=k_0} (R_{ki}^u)^{u^{(i)}} \cdot B_k(s_k^E(e)) \\ P_k(u^{(j)};O) &= \frac{P_k^A(u^{(j)};O)}{P_k(u^{(j)};I)}, \qquad j = 1, ..., k_0. \qquad (3.152) \\ &= \sum_{e:u^{(j)}(e)=u^{(j)}} A_{k-1}(s_k^S(e)) \cdot \prod_{i=1}^{n_0}(R_{ki}^c)^{c^{(i)}} \cdot \prod_{i=1,i\neq j}^{k_0} (R_{ki}^u)^{u^{(i)}} \cdot B_k(s_k^E(e)) \end{aligned}$$

### 3.5.5 Additive SISO

Using Equations (3.73) to (3.85), all computations of the multiplicative SISO can be converted to the logarithmic domain to obtain the additive SISO [57]. First consider the computation of the input LLRs $\lambda_k(c;I)$ and $\lambda_k(u;I)$. Substituting Equation (3.140) for $P_k(c;I)$ and considering the definition of $R^c_{kj}$ in Equation (3.135), we have

$$\begin{aligned}\lambda_k(c;I) &= \ln \frac{h_{pc}\prod_{j=1}^{n_0}(R^c_{kj})^{c^{(j)}}}{h_{pc}\prod_{j=1}^{n_0}(R^c_{kj})^0} \\ &= \sum_{j=1}^{n_0} c^{(j)} \ln R^c_{kj} \\ &= \sum_{j=1}^{n_0} c^{(j)} \lambda_k(c^{(j)};I), \qquad (3.153)\end{aligned}$$

where $\lambda_k(c^{(j)};I)$ is expressed in Equation (3.91). Similarly, with Equation (3.141) for $P_k(u;I)$ and Equation (3.136) for $R^u_{kj}$,

$$\lambda_k(u;I) = \sum_{j=1}^{k_0} u^{(j)} \lambda_k(u^{(j)};I). \qquad (3.154)$$

This result shows that $\lambda_k(c;I)$ can be expressed as the weighted summation of the input LLR for each codeword bit $\lambda_k(c^{(j)};I)$, and $\lambda_k(u;I)$ can be expressed as the weighted summation of the input LLR $\lambda_k(u^{(j)};I)$. The value of $\lambda_k(u^{(j)};I)$ is initialized to zero for the first SISO decoding stage. Afterwards, it assumes the interleaved/deinterleaved $\lambda_k(u^{(j)};O)$ from the previous SISO decoder.

In summary, Equations (3.142) to (3.152) are converted to the following equations in logarithmic domain. Recall that Equations (3.153), (3.154) and (3.91) are used to obtain $\lambda_k(c;I)$ and $\lambda_k(u;I)$.

1. The forward and backward recursions are computed as follows:

$$\begin{aligned}\alpha_k(s) &= \max^*_{e:s^E_k(e)=s}\left[\alpha_{k-1}(s^S_k(e)) + \lambda_k(u(e);I) + \lambda_k(c(e);I)\right], \\ &\qquad k = 1, ..., N-1. \qquad (3.155)\\ \beta_k(s) &= \max^*_{e:s^S_{k+1}(e)=s}\left[\beta_{k+1}(s^E_{k+1}(e)) + \lambda_{k+1}(u(e);I) + \lambda_{k+1}(c(e);I)\right], \\ &\qquad k = N-1, ..., 1. \qquad (3.156)\end{aligned}$$

   The $\alpha_k(s)$ computation is initialized by Equation (3.103), and the $\beta_k(s)$ computation is initialized by Equation (3.104) or (3.105) depending on the knowledge of the final state. As explained for the conventional Log-MAP, Equations (3.101) and (3.102) can be used to normalize $\alpha_k(s)$ and $\beta_k(s)$ for computation stability.

2. The complete information for the codeword symbol and for the information symbol are as follows:

$$\begin{aligned}\lambda_k^A(c;O) &= \max^*_{e:c(e)=c}\left[\alpha_{k-1}(s_k^S(e)) + \lambda_k(u(e);I) + \lambda_k(c(e);I) + \beta_k(s_k^E(e))\right]\\ &\quad - \max^*_{e:c(e)=\underline{0}}\left[\alpha_{k-1}(s_k^S(e)) + \lambda_k(u(e);I) + \lambda_k(c(e);I) + \beta_k(s_k^E(e))\right]\end{aligned} \tag{3.157}$$

$$\begin{aligned}\lambda_k^A(u;O) &= \max^*_{e:u(e)=u}\left[\alpha_{k-1}(s_k^S(e)) + \lambda_k(u(e);I) + \lambda_k(c(e);I) + \beta_k(s_k^E(e))\right]\\ &\quad - \max^*_{e:u(e)=\underline{0}}\left[\alpha_{k-1}(s_k^S(e)) + \lambda_k(u(e);I) + \lambda_k(c(e);I) + \beta_k(s_k^E(e))\right]\end{aligned} \tag{3.158}$$

3. The output LLRs for the codeword symbol and for the information symbol are as follows:

$$\lambda_k(c;O) = \lambda_k^A(c;O) - \lambda_k(c;I) \tag{3.159}$$

$$\begin{aligned}&= \max^*_{e:c(e)=c}\left[\alpha_{k-1}(s_k^S(e)) + \lambda_k(u(e);I) + \beta_k(s_k^E(e))\right]\\ &\quad - \max^*_{e:c(e)=\underline{0}}\left[\alpha_{k-1}(s_k^S(e)) + \lambda_k(u(e);I) + \beta_k(s_k^E(e))\right]\end{aligned} \tag{3.160}$$

$$\lambda_k(u;O) = \lambda_k^A(u;O) - \lambda_k(u;I) \tag{3.161}$$

$$\begin{aligned}&= \max^*_{e:u(e)=u}\left[\alpha_{k-1}(s_k^S(e)) + \lambda_k(c(e);I) + \beta_k(s_k^E(e))\right]\\ &\quad - \max^*_{e:u(e)=\underline{0}}\left[\alpha_{k-1}(s_k^S(e)) + \lambda_k(c(e);I) + \beta_k(s_k^E(e))\right]\end{aligned} \tag{3.162}$$

4. The complete information for the codeword bits and for the information bits are as follows:

$$\begin{aligned}\lambda_k^A(c^{(j)};O) &= \max^*_{e:c^{(j)}(e)=1}\left[\alpha_{k-1}(s_k^S(e)) + \lambda_k(u(e);I) + \lambda_k(c(e);I) + \beta_k(s_k^E(e))\right]\\ &\quad - \max^*_{e:c^{(j)}(e)=0}\left[\alpha_{k-1}(s_k^S(e)) + \lambda_k(u(e);I) + \lambda_k(c(e);I) + \beta_k(s_k^E(e))\right]\\ &\qquad j = 1, ..., n_0.\end{aligned} \tag{3.163}$$

$$\begin{aligned}\lambda_k^A(u^{(j)};O) &= \max^*_{e:u^{(j)}(e)=1}\left[\alpha_{k-1}(s_k^S(e)) + \lambda_k(u(e);I) + \lambda_k(c(e);I) + \beta_k(s_k^E(e))\right]\\ &\quad - \max^*_{e:u^{(j)}(e)=0}\left[\alpha_{k-1}(s_k^S(e)) + \lambda_k(u(e);I) + \lambda_k(c(e);I) + \beta_k(s_k^E(e))\right]\\ &\qquad j = 1, ..., k_0.\end{aligned} \tag{3.164}$$

5. The output LLR for the codeword bits and for the information bits are as follows:

$$\lambda_k(c^{(j)};O) = \lambda_k^A(c^{(j)};O) - \lambda_k(c^{(j)};I), \qquad j = 1, ..., n_0. \tag{3.165}$$

$$\begin{aligned} = & \max_{e:c^{(j)}(e)=1}{}^{*} \left[ \alpha_{k-1}(s_k^S(e)) + \lambda_k(u(e);I) + \sum_{i=1;i\neq j}^{n_0} c^{(i)}(e)\lambda_k(c^{(i)};I) + \beta_k(s_k^E(e)) \right] \\ & - \max_{e:c^{(j)}(e)=0}{}^{*} \left[ \alpha_{k-1}(s_k^S(e)) + \lambda_k(u(e);I) + \sum_{i=1;i\neq j}^{n_0} c^{(i)}(e)\lambda_k(c^{(i)};I) + \beta_k(s_k^E(e)) \right] \end{aligned} \tag{3.166}$$

$$\lambda_k(u^{(j)};O) = \lambda_k^A(u^{(j)};O) - \lambda_k(u^{(j)};I), \qquad j = 1, ..., k_0. \tag{3.167}$$

$$\begin{aligned} = & \max_{e:u^{(j)}(e)=1}{}^{*} \left[ \alpha_{k-1}(s_k^S(e)) + \sum_{i=1;i\neq j}^{k_0} u^{(i)}\lambda_k(u^{(i)};I) + \lambda_k(c(e);I) + \beta_k(s_k^E(e)) \right] \\ & - \max_{e:u^{(j)}(e)=0}{}^{*} \left[ \alpha_{k-1}(s_k^S(e)) + \sum_{i=1;i\neq j}^{k_0} u^{(i)}\lambda_k(u^{(i)};I) + \lambda_k(c(e);I) + \beta_k(s_k^E(e)) \right] \end{aligned} \tag{3.168}$$

## 3.6 Decoding Procedures

It has been shown that SISO is a general algorithm which can produce soft outputs for the codeword bits and the information bits. SISO can be used to decode both PCCCs and SCCCs. In the following sections, decoding procedures are presented for both structures where the information bits are transmitted in frames.

### 3.6.1 Decoding Procedure for PCCCs

The procedure to decode a PCCC using SISO is described with an example of two RSC codes of rate 1/2. Thus, $k_0 = 1$, $n_0 = 2$. For PCCCs comprising more constituent codes or codes of different rates, the same procedure can be readily extended. If $c_2^{(1)}$ of RSC2 is a systematic bit and not transmitted, then set $c_{2k}^{(1)} = 0$ and $y_{2k}^{(1)} = 0$, $\forall k$. In the following, subscripts 1 and 2 indicate the first and the second constituent decoder, and $\tau_{max}$ represents the maximum number of iterations.

1. Demultiplex the received signal into two streams, one for DEC1 and one for DEC2. Scale the received signal using Equation (3.91) to get $\lambda_{1k}(c^{(j)};I)$ for DEC1 and $\lambda_{2k}(c^{(j)};I)$ for DEC2, $j = 1, 2$.

2. Iteration $\tau = 1$. For DEC1, initialize $\lambda_{1k}(u; I) = 0, \forall k$.

3. **DEC1**. Compute path metrics $\alpha_{1k}(s)$ and $\beta_{1k}(s)$ with Equations (3.155) and (3.156). Then use Equations (3.164) and (3.167) to calculate the output LLR $\lambda_{1k}(u; O)$ for the information bits.

4. Interleave $\lambda_{1k}(u; O)$ to provide the input LLRs for DEC2: $\lambda_{2k}(u; I) = \lambda_{1\alpha(k)}(u; O)$.

5. **DEC2**. First use Equations (3.155) and (3.156) to obtain path metrics $\alpha_{2k}(s)$ and $\beta_{2k}(s)$. Then,

   (a) if $\tau < \tau_{max}$, use Equation (3.164) in conjuction with (3.167) to compute the output LLR $\lambda_{2k}(u; O)$ for the interleaved information bits. Deinterleave $\lambda_{2k}(u; O)$ to be $\lambda_{1k}(u; I)$ of the next iteration: $\lambda_{1\alpha(k)}(u; I) = \lambda_{2k}(u; O)$. Increment $\tau$, go back to Step 3 and start next iteration.

   (b) if $\tau = \tau_{max}$, use Equation (3.164) to get the complete information $\lambda^A_{2k}(u; O)$ for the interleaved bits. Deinterleave and make decisions on the transmitted bits as follows:

$$\hat{u}_{\alpha(k)} = \begin{cases} 1, & \text{when} \quad \lambda^A_{2k}(u; O) > 0. \\ 0, & \text{when} \quad \lambda^A_{2k}(u; O) < 0. \end{cases}$$

   Go back to Step 1 to decode next frame.

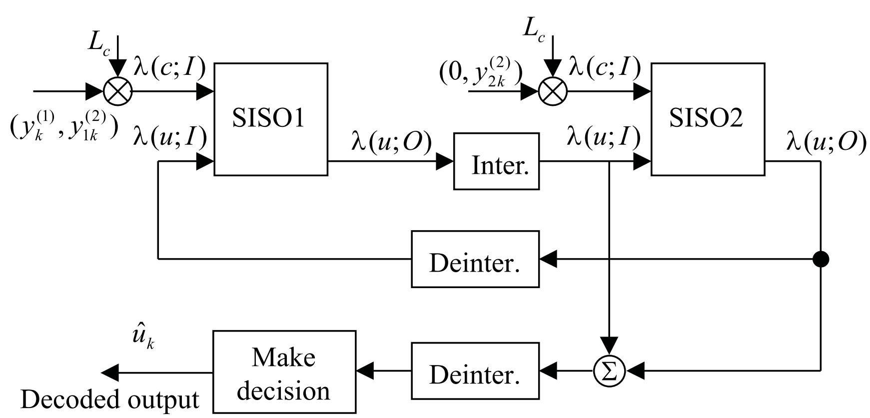



Figure 3.18: PCCC decoder with the general SISO module.

This procedure is illustrated in Figure 3.18. Note that the output LLRs in Steps 3 and 5a can also be obtained using Equation (3.168) directly, instead of subtracting

$\lambda_k(u; I)$ from $\lambda_k^A(u; I)$. On the other hand, the complete information at Step 5b can be obtained using $\lambda_{2k}^A(u; O) = \lambda_{2k}(u; O) + \lambda_{2k}(u; I)$, if $\lambda_{2k}(u; O)$ has been computed.

It is observed that among the four inputs of DEC1 and DEC2, both $\lambda_{1k}(c; I)$ and $\lambda_{2k}(c; I)$ remain constant, while both $\lambda_{1k}(u; I)$ and $\lambda_{2k}(u; I)$ are updated throughout the iterations.

### 3.6.2 Decoding Procedure for SCCCs

To decode a SCCC with SISO, the following procedure is employed. Here the superscripts $i$ and $o$ indicate the inner and outer code, respectively.

1. Scale the received signal using Equation (3.91) to obtain the input LLRs $\lambda_k^i(c^{(j)}; I)$ for the codeword bits of the inner code.

2. Iteration $\tau = 1$. For the inner decoder, initialize $\lambda_k^i(u^{(j)}; I) = 0, \forall k, j$.

3. **Inner decoder**. Calculate path metrics $\alpha_k^i(s)$ and $\beta_k^i(s)$ with Equations (3.155) and (3.156). Then use Equations (3.164) and (3.167) to compute the output LLRs $\lambda_k^i(u^{(j)}; O)$ for the information bits of the inner code.

4. Deinterleave the stream of $\lambda_k^i(u^{(j)}; O)$ to be the input LLR $\lambda_k^o(c^{(j)}; I)$ for the codeword bits of the outer code.

5. **Outer decoder**. First calculate path metrics $\alpha_k^o(s)$ and $\beta_k^o(s)$ using Equations (3.155) and (3.156). Set $\lambda_k^o(u; I) \equiv 0$, thus removing it from the computation. Then,

   (a) If $\tau < \tau_{max}$, use Equation (3.163) and (3.165) to calculate the output LLR $\lambda_k^o(c^{(j)}; O)$ for the codeword bits of the outer code. Interleave the stream of $\lambda_k^o(c^{(j)}; O)$ to obtain the input LLRs $\lambda_k^i(u^{(j)}; I)$ of the inner code. Increment $\tau$, go back to Step 3 and start next iteration.

   (b) If $\tau = \tau_{max}$, use Equation (3.164) (with $\lambda_k^o(u; I) \equiv 0$) to obtain the complete information $\lambda_k^{oA}(u^{(j)}; O)$ of the outer code. Make decisions on the transmitted bits as follows:

$$\hat{u}_k^{(j)} = \begin{cases} 1, & \text{when} \quad \lambda_k^{oA}(u^{(j)}, O) > 0. \\ 0, & \text{when} \quad \lambda_k^{oA}(u^{(j)}, O) < 0. \end{cases}$$

   Go back to Step 1 to decode next frame.

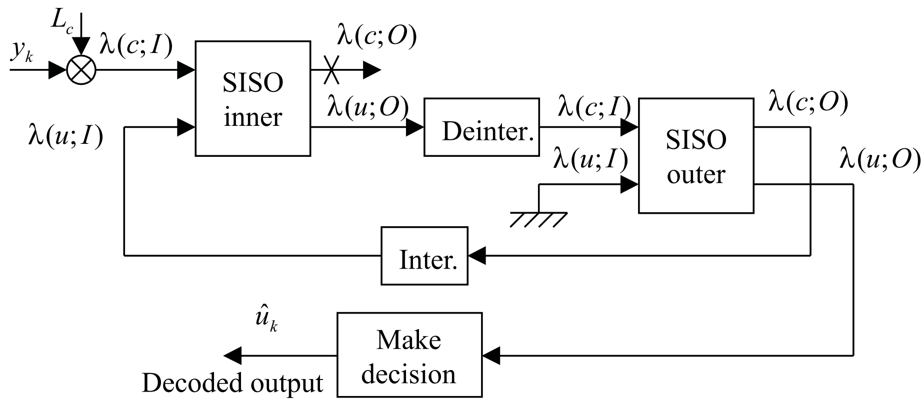



Figure 3.19: SCCC decoder with the general SISO module.

This procedure is illustrated in Figure 3.19. The output LLR in Steps 3 and 5a for the codeword bits and the information bits can be computed directly using Equations (3.166) and (3.168), respectively, instead of subtracting the input LLR from the complete information.

It is noticed that among the four inputs of the inner and outer SISO, $\lambda_k^i(c;I)$ and $\lambda_k^o(u;I)(\equiv 0)$ remain constant for all iterations, while $\lambda_k^i(u;I)$ and $\lambda_k^o(c;I)$ are updated at each iteration.

## 3.7 A Sliding Window Version of SISO

The SISO module described above requires that the whole sequence be received before the decoding procedure starts. This restriction, which results from the backward recursion which starts from the final trellis state, results in two inconveniences. First, the information bits have to be transmitted in frames, and tail bits, which reduce bandwidth efficiency, are added at the end of each frame to terminate the trellis. Continuous transmission is impossible. Secondly, the delay and memory requirements can become intolerably large as the frame size increases. This is contradictory to the requirement that the frame size needs to be fairly large to achieve a good interleaver gain.

To tackle these problems, the sliding window technique can be employed [57] [60]. The result is a sliding window soft-input soft-output (SW-SISO) module, which is a SISO module with an embedded sliding window. The SW-SISO segments the received sequence into small windows of size $N_u$. To produce the soft output for $N_u$ trellis

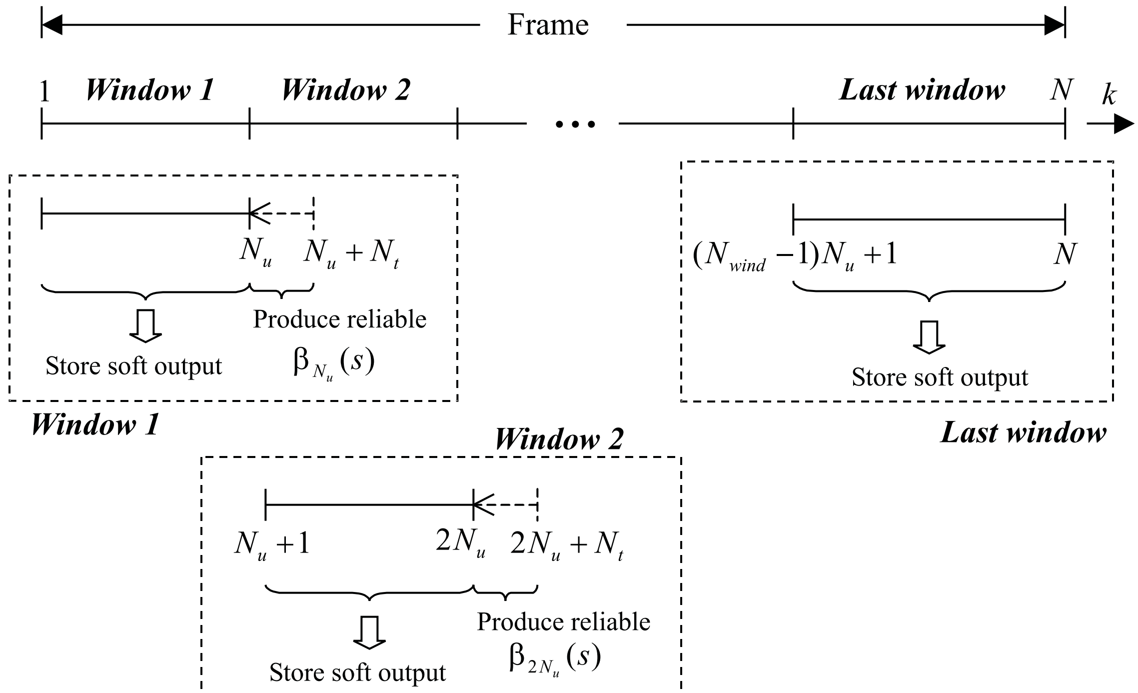



Figure 3.20: Sliding window SISO for a frame.

steps, a span of $N_w = N_u + N_t$ trellis steps will be processed together. Here $N_u$ is the size of an updating window, and $N_t$ is the size of a training window which follows the updating window. The SW-SISO processes each window of size $N_w$ in the same way SISO processes a frame. The only difference lies in the initialization of the backward path metrics. At the end of each window processing, $N_u$ soft outputs will be generated, and SW-SISO jumps over $N_u$ trellis steps to work on the next window of size $N_w$. The procedure is illustrated in Figure 3.20.

The SW-SISO for a frame of size $N$ is detailed as follows. When the bits are transmitted by frames, a frame of size $N$ will be first divided into $N_{wind} = \lceil (N - N_t)/N_u \rceil$ windows. Then SW-SISO operates following these steps.

1. Window $i = 1$. Initialize $\alpha_0(s)$ as in Equation (3.103).

2. 
   - If $i < N_{wind}$, the current window size is $N_u + N_t$. Initialize $\beta_{i(N_u+N_t)}(s) = 0, \forall s$. Compute the backward recursion from $k = i(N_u + N_t) - 1$ to $k = (i-1)(N_u + N_t) + 1$, using Equation (3.156). Store $\beta_{iN_u}(s)$ through $\beta_{(i-1)N_u+1}(s)$.
   - If $i = N_{wind}$, the current window size is $N - (N_{wind} - 1)N_u$. Initialize

$\beta_N(s)$ by Equation (3.104) or (3.105) depending on the knowledge of the final state of the trellis. Compute the backward recursion from $k = N - 1$ to $k = (N_{wind} - 1)N_u + 1$, using Equation (3.156). Store $\beta_{N-1}(s)$ through $\beta_{(N_{wind}-1)N_u+1}(s)$.

3. - When $i < N_{wind}$, perform the forward recursion from $k = (i-1)N_u + 1$ to $k = iN_u$.
   - When $i = N_{wind}$, perform the forward recursion from $k = (i-1)N_u + 1$ to $k = N$.

   As the forward recursion is computed, also find the complete information or the output LLRs, depending on the requirement. Increase $i$ by 1.

4. - If $i \leq N_{wind}$, go back to Step 2 to process next window.
   - Otherwise, shift to the next frame.

It is evident that in comparison to regular SISO, SW-SISO reduces the storage requirement of $\beta_k(s)$ from $(N-1)2^m$ to $N_u 2^m$ real numbers. When the processor is sufficiently fast so that the only delay comes from waiting to receive the signal from the channel, the delay of SW-SISO is equal to the time required to receive $N_w n_0$ signals, instead of $Nn_0$ as in SISO. In addition, the storage and delay are independent of the frame size, since $N_u$ and $N_t$ are independent of $N$.

On the other hand, an extra computation of $N_t$ steps of $\beta_k(s)$ is required for each training window. In other words, the $\beta_k(s)$ computation of SW-SISO is $[1+(N_t/N_u)]$ times that of SISO. $N_t \approx 5(m+1)$ is usually large enough to provide a reliable $\beta_{iN_u}(s)$ vector for the $i$-th updating window. $N_u$ should be chosen carefully to get the best compromise between complexity and latency.

## 3.8 Forward Computation of Backward Path Metrics for MAP Decoder

As shown in Section 3.3.1, the MAP algorithm usually involves a computation of the backward path metric $B_k(s)$ and the forward path metric $A_k(s)$. Either one of them can be computed first and stored in memory. In order to produce a soft output with increasing time index, the backward recursion can be performed first to obtain all $B_k(s)$, where $k \in \{1, ..., N\}$ is the time index, $s \in \{0, ..., 2^m - 1\}$ is the state index, $N$

is the frame size, and $m$ is the memory size of the component RSC encoders. Next, the forward recursion is performed, and each soft output is computed after $A_k(s)$ is known. In this dissertation, this backward-forward version of the MAP algorithm is adopted for analysis.

In the backward-forward version, all computations of $B_k(s)$ are finished before the computation of $A_k(s)$ starts. Not all $A_k(s)$ need to be stored, and only the most recent vector $A_{k-1}(s)$ needs to be kept to obtain the output LLR. However, all values of $B_k(s)$ need to be stored for the computation of the output LLR. This can be a problem when the frame size is large since $2^m \times N$ values of $B_k(s)$ need to be stored. A sliding window method has been presented in Section 3.7 to save memory. However, it does not make use of the information in the whole frame. In this section, a method is developed to reduce storage requirements even when the entire frame is processed at a time [61]. We propose to compute $B_k(s)$ in the forward direction, and consequently reduce the memory requirement for $B_k(s)$. In addition, this method can be combined with the sliding window technique to achieve even greater savings.

### 3.8.1 Forward Evaluation of Backward Path Metrics $B_k(s)$

Using the butterfly structure shown in Section 3.2, the MAP algorithm can be split into a sequence of $2 \times 2$ matrix operations, one for a butterfly pair. Here a rate 1/2 RSC encoder with code generator $g(D) = [1, 1 + D^2/1 + D + D^2]$ is used as an example. A trellis section for this code generator is shown in Figure 3.21. Clearly, there are two butterfly pairs having the form given in Figure 3.14.

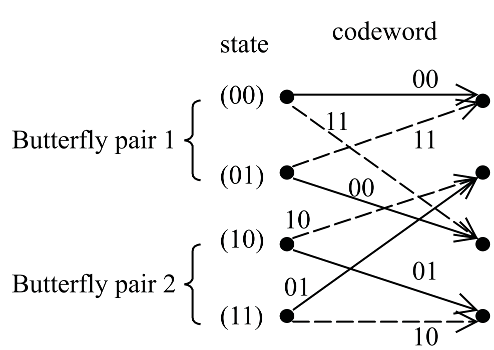



Figure 3.21: A trellis section of the RSC code with $g = (7, 5)_{octal}$.

There are four different branch metric values corresponding to four different codewords. Two branch metrics are for butterfly pair 1:

$$\begin{aligned} M_k(00) &= P_k(u_k = 0; I)P_k(c_k = 00; I) \\ M_k(11) &= P_k(u_k = 1; I)P_k(c_k = 11; I) \end{aligned}$$

There are also two branch metrics for butterfly pair 2:

$$\begin{aligned} M_k(01) &= P_k(u_k = 0; I)P_k(c_k = 01; I) \\ M_k(10) &= P_k(u_k = 1; I)P_k(c_k = 10; I) \end{aligned}$$

The backward path metrics $B_k(s)$ are updated by the following equations.

- For butterfly pair 1:

$$\begin{aligned} B_k(0) &= M_{k+1}(00)B_{k+1}(0) + M_{k+1}(11)B_{k+1}(2) \\ B_k(1) &= M_{k+1}(11)B_{k+1}(0) + M_{k+1}(00)B_{k+1}(2) \end{aligned}$$

  These equations can be expressed in matrix form,

$$\begin{aligned} \begin{bmatrix} B_k(0) \\ B_k(1) \end{bmatrix} &= \begin{bmatrix} M_{k+1}(00) & M_{k+1}(11) \\ M_{k+1}(11) & M_{k+1}(00) \end{bmatrix} \begin{bmatrix} B_{k+1}(0) \\ B_{k+1}(2) \end{bmatrix} \\ &= M^B_{1,k+1} \begin{bmatrix} B_{k+1}(0) \\ B_{k+1}(2) \end{bmatrix} \end{aligned} \tag{3.169}$$

- And for butterfly pair 2:

$$\begin{aligned} B_k(2) &= M_{k+1}(10)B_{k+1}(1) + M_{k+1}(01)B_{k+1}(3) \\ B_k(3) &= M_{k+1}(01)B_{k+1}(1) + M_{k+1}(10)B_{k+1}(3) \end{aligned}$$

  The equations above can be represented in matrix form,

$$\begin{aligned} \begin{bmatrix} B_k(2) \\ B_k(3) \end{bmatrix} &= \begin{bmatrix} M_{k+1}(10) & M_{k+1}(01) \\ M_{k+1}(01) & M_{k+1}(10) \end{bmatrix} \begin{bmatrix} B_{k+1}(1) \\ B_{k+1}(3) \end{bmatrix} \\ &= M^B_{2,k+1} \begin{bmatrix} B_{k+1}(1) \\ B_{k+1}(3) \end{bmatrix} \end{aligned} \tag{3.170}$$

If $(M^B_{1,k+1})^{-1}$ and $(M^B_{2,k+1})^{-1}$ exist, then $B_{k+1}(\cdot)$ can be obtained from $B_k(\cdot)$.

- For butterfly pair 1:

$$\begin{aligned}
\begin{bmatrix} B_{k+1}(0) \\ B_{k+1}(2) \end{bmatrix} &= (M_{1,k+1}^{B})^{-1} \begin{bmatrix} B_k(0) \\ B_k(1) \end{bmatrix} \\
&= \frac{1}{M_{k+1}^2(00) - M_{k+1}^2(11)} \begin{bmatrix} M_{k+1}(00) & -M_{k+1}(11) \\ -M_{k+1}(11) & M_{k+1}(00) \end{bmatrix} \begin{bmatrix} B_k(0) \\ B_k(1) \end{bmatrix} \\
&= h_{M'} \times (M_{k+1}^2(10) - M_{k+1}^2(01)) \times \\
&\quad \begin{bmatrix} M_{k+1}(00) & -M_{k+1}(11) \\ -M_{k+1}(11) & M_{k+1}(00) \end{bmatrix} \begin{bmatrix} B_k(0) \\ B_k(1) \end{bmatrix}
\end{aligned} \tag{3.171}$$

- For butterfly pair 2:

$$\begin{aligned}
\begin{bmatrix} B_{k+1}(1) \\ B_{k+1}(3) \end{bmatrix} &= (M_{2,k+1}^{B})^{-1} \begin{bmatrix} B_k(2) \\ B_k(3) \end{bmatrix} \\
&= \frac{1}{M_{k+1}^2(10) - M_{k+1}^2(01)} \begin{bmatrix} M_{k+1}(10) & -M_{k+1}(01) \\ -M_{k+1}(01) & M_{k+1}(10) \end{bmatrix} \begin{bmatrix} B_k(2) \\ B_k(3) \end{bmatrix} \\
&= h_{M'} \times (M_{k+1}^2(00) - M_{k+1}^2(11)) \times \\
&\quad \begin{bmatrix} M_{k+1}(10) & -M_{k+1}(01) \\ -M_{k+1}(01) & M_{k+1}(10) \end{bmatrix} \begin{bmatrix} B_k(2) \\ B_k(3) \end{bmatrix}
\end{aligned} \tag{3.172}$$

In the above equations, $h_{M'} = \frac{1}{(M_{k+1}^2(10)-M_{k+1}^2(01))(M_{k+1}^2(00)-M_{k+1}^2(11))}$. When the output LLR is computed, $h_{M'}$ will cancel; thus, it can be omitted.

Equations (3.171) and (3.172) show that $B_{k+1}(s)$ can actually be determined from $B_k(s)$. This suggests that the backward recursion can be performed, without storing any $B_k(s)$, to obtain $B_1(s)$. Then Equations (3.171) and (3.172) can be employed to compute $B_k(s)$ simultaneously with $A_k(s)$ to obtain the soft output.

Unfortunately, simulation showed that this direct method suffered from numerical stability problems. However, further investigation showed that this instability could be surmounted by dividing the frame into blocks of $N_b$ bits, storing only the $B_k(s)$ at the start of each block, and using Equations (3.171) and (3.172) to compute the $B_k(s)$ within each block. The choice of $N_b$ will be discussed in Section 3.8.2. The resulting technique is expressed as the follows.

1. Perform the backward recursion using Equations (3.169) and (3.170), but only store $B_1(s)$ and $B_k(s), k = iN_b, 0 < i < N_{blk}$, where $N_{blk} = \lceil N/N_b \rceil$.

2. Perform the forward recursion of the conventional MAP algorithm to obtain $A_k(s)$ for $k \in \{1, ..., N-1\}$. Use the stored values of $B_k(s)$ for $k = 1$ and $k = iN_b, 0 < i < N_{blk}$. Otherwise, calculate $B_k(s)$ from the previous vector $B_{k-1}(s)$ using Equations (3.171) and (3.172). Only the most recent vector $B_{k-1}(s)$ is kept for the computation of the current vector $B_k(s)$, as in the computation of the forward path metric $A_k(s)$.

3. Use $B_k(s)$ in conjunction with $A_{k-1}(s)$ to evaluate the soft output.

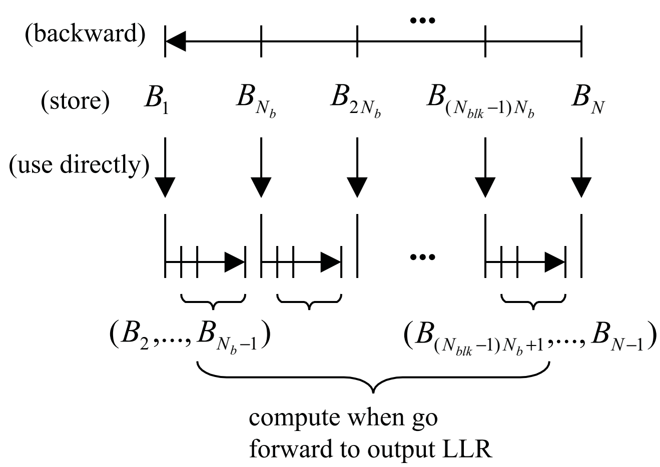



Figure 3.22: Procedure to compute $B_k(s)$.

The procedure to compute $B_k(s)$ is illustrated in Figure 3.22. Notice that when $N_b = 1$, this method degenerates to the conventional MAP algorithm. A factor of $(N_b - 1)/N_b$ storage space for $B_k(s)$ will be saved with this method. However, the computation of $B_k(s)$ will be increased to $(1 + (N_b - 1)/N_b)$ times that of the conventional method.

### 3.8.2 Choice of Block Size $N_b$

As mentioned above, numerical stability problems place constraints upon the choice of the value of $N_b$. Considering the previous example in which $g(D) = [1, 1 + D^2/1 + D + D^2]$, if the Equations (3.169) and (3.170) are used iteratively to compute $B_i(s)$ from $B_{i+l}(s), 1 \leq l \leq N - i$, the following expression linking the path metrics at time index $k = i$ and $k = i + l$ can be obtained:

$$\begin{bmatrix} B_i(0) \\ B_i(1) \\ B_i(2) \\ B_i(3) \end{bmatrix} = M^B_{i+1;i+l} \begin{bmatrix} B_{i+l}(0) \\ B_{i+l}(1) \\ B_{i+l}(2) \\ B_{i+l}(3) \end{bmatrix}, \tag{3.173}$$

where $M^B_{i+1;i+l}$ is a $4 \times 4$ matrix. Therefore, to compute $B_{i+l}(s)$ from $B_i(s)$, the following is used:

$$\begin{bmatrix} B_{i+l}(0) \\ B_{i+l}(1) \\ B_{i+l}(2) \\ B_{i+l}(3) \end{bmatrix} = (M^B_{i+1;i+l})^{-1} \begin{bmatrix} B_i(0) \\ B_i(1) \\ B_i(2) \\ B_i(3) \end{bmatrix}. \tag{3.174}$$

The element $(p, q), 0 \leq p, q \leq 3$, of $M^B_{i+1;i+l}$ is proportional to the probability of transition from state $p$ at time $i$ to state $q$ at time $i + l$, or

$$M^B_{i+1;i+l}(p, q) = P(s_{i+l} = q, Y^{i+l}_{i+1} | s_i = p).$$

Consequently, if $l$ is larger than the time the trellis needs to converge, the matrix $M^B_{i+1;i+l}$ will have an entry which is much larger than the other entries. This entry corresponds to the most likely path in the trellis section between $k = i$ and $k = i + l$. Mathematically, $M^B_{i+1;i+l}$ becomes nearly singular. As a result, the inverse matrix $(M^B_{i+1;i+l})^{-1}$ becomes ill-conditioned. In another view, as the backward recursion is performed, the information from stage to stage is aggregated together. When $l$ increases, it becomes harder and harder to retrieve the detailed information for each transition.

Simulation has been carried out to study the phenomenon. A typical case of $g(D) = [1, 1 + D^2/1 + D + D^2]$ in an AWGN channel at $E_b/N_0 = 1$ dB is used throughout the following simulations. A plot of the logarithm of the condition number of $(M^B_{i+1;i+l})^{-1}$ is shown in Figure 3.23 for one MAP operation. The depth $l$ ranges from 1 to $7(m + 1)$, where $m + 1 = 3$ is the constraint length. As seen in the plot, the condition number of the forward matrix $(M^B_{i+1;i+l})^{-1}$ increases exponentially with the distance $l$.

Let $\lambda^A_k(u; O)$ be the $k$-th soft output obtained with the conventional method, and let $\lambda^{A,N_b}_k(u; O)$ be the one obtained for the method proposed here with block size $N_b$. Define the relative error as

$$\Delta^{N_b}_r = \frac{|\lambda^A_k(u; O) - \lambda^{A,N_b}_k(u; O)|}{E(|\lambda^A_k(u; O)|)},$$

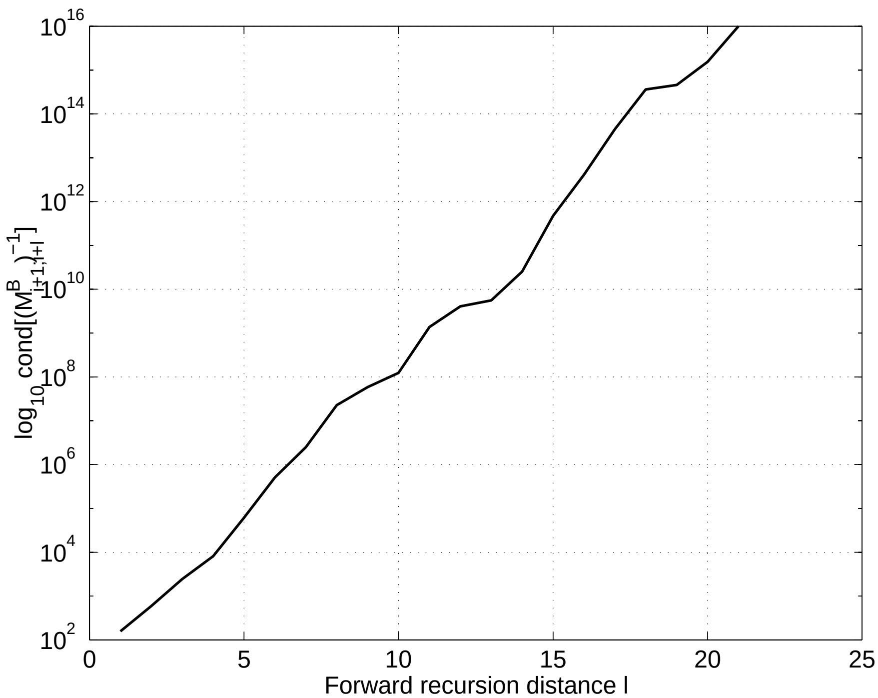



Figure 3.23: Logarithm of the condition number of $(M^B_{i+1;i+l})^{-1}$ ($g = (7,5)_{octal}$, rate 1/3, $E_b/N_0$=1 dB, one iteration).

where $E(x)$ is the mean value of $x$. In Figure 3.24, the maximum $\Delta_r^{N_b}$ for a frame of size 600 is shown with $N_b$ ranging from $3(m+1)$ to $10(m+1)$. This graph shows that the relative error is negligible when $N_b \leq 5(m+1)$. When $N_b \geq 6(m+1)$, the computation error accumulates to an unacceptable value.

In Figure 3.25, curves are shown for BER versus the iteration number. When $E_b/N_0 = 0$ dB, all $N_b \in \{1, ..., 12\}$ gives the same BER. When $E_b/N_0 = 1$ dB, the BER decreases steadily at the first few iterations for any $N_b \in \{1, ..., 12\}$, and then it increases as the iteration proceeds for $N_b \in \{3, ..., 12\}$. As explained above, this is because of the convergence speed of the trellis. As the number of iterations increases, the decoder makes more and more confident estimations, and the trellis converges with fewer stages. At medium to high $E_b/N_0$, $N_b$ must be set to a smaller value when more MAP operations are required. For a decoder working with medium to high $E_b/N_0$ and/or a large number of iterations, it may become necessary to set $N_b = 2$, which results in a 50% reduction in the storage requirements for $B_k(s)$.

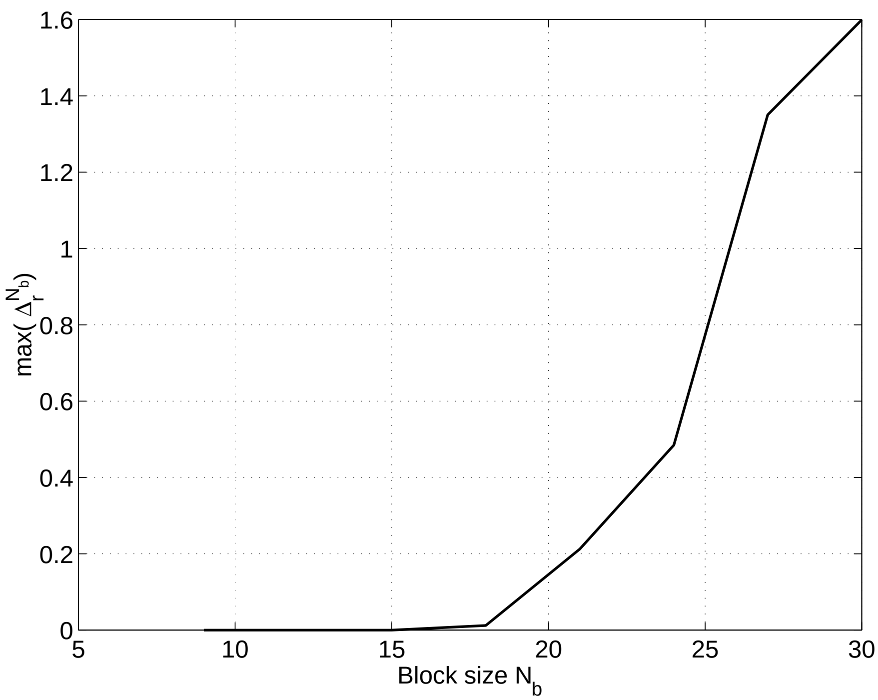



Figure 3.24: Maximum relative error of output LLR ($g = (7,5)_{octal}$, rate 1/3, $N = 600$, $E_b/N_0$=1 dB, one iteration).

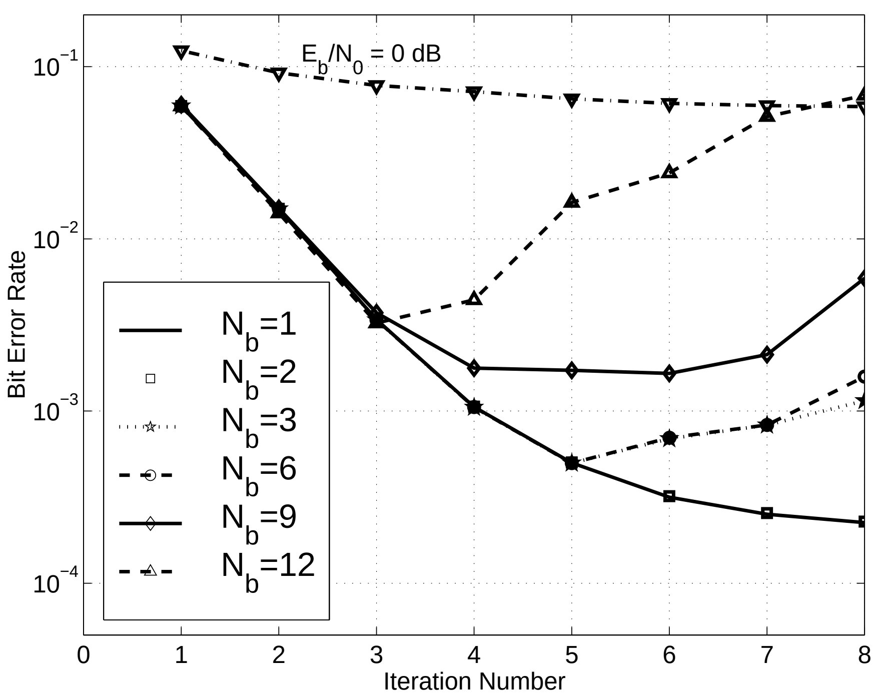



Figure 3.25: $N_b$ choice for multiple iterations ($g = (7,5)_{octal}$, rate 1/3, $N = 1020$, $E_b/N_0$=1 dB).

### 3.8.3 Conclusion

Based on the butterfly structure of the RSC code, a new method was presented in this section to compute the backward path metrics $B_k(s)$ for the forward-backward MAP algorithm. This method promises to save at least 50% memory size of the backward path metrics $B_k(s)$ without performance degradation. This reduction is advantageous for the implementation of a MAP decoder for the turbo codes, especially when the frame size is large, and may provide the possibility to store $B_k(s)$ on-chip. In comparison to the implementation with a large amount of off-chip storage, an implementation with most or all on-chip storage can eliminate the clock cycles required to fetch data, resulting in increased processing speed perhaps more than compensating for increased computational requirements in a practical implementation.

## 3.9 Summary

In this chapter, decoding algorithms for PCCCs and SCCCs were discussed. The MAP algorithm and its additive form were presented. By analyzing the decoding procedure, several strategies were found and applied to simplify the calculation without changing the function following the derivation of [48] [57]. In comparison to the original algorithm, the input size of DEC2 was smaller, and the branch metric computation and the output LLR computation were reduced. Then a general soft-input soft-output decoding algorithm was presented for decoding both PCCCs and SCCCs, and the decoding procedures with the SISO module were shown in detail for both PCCCs and SCCCs. Afterwards, the sliding window SISO was introduced to reduce the decoder memory requirement. Finally, a new method was proposed to compute the backward metrics of the MAP algorithm in the forward direction so as to reduce storage requirement at the expense of an increased computational complexity.

This chapter has presented the procedures and notation associated with the iterative decoding of PCCCs and SCCCs. In subsequent chapters, the practical implementation of these procedures will be addressed.

# Chapter 4

# Performance of Concatenated Codes

With the structures and the decoding algorithms of both PCCC and SCCC systems explained in previous chapters, this chapter presents a comprehensive simulation study of the characteristics of their bit error probability performance. This knowledge enables us to choose proper design parameters in accordance to performance requirements. For both PCCCs and SCCCs, the simulation procedure is first introduced. Then curves of BER versus $E_b/N_0$ are plotted to show the influence of various parameters, including the number of iterations, the frame size, the code rate, and the code generator. Also the performance of SCCCs was compared with PCCCs in Section 4.2. Based on the extensive simulations, the characteristics of PCCC and SCCC performance are summarized.

## 4.1 Simulation Study of PCCCs

In this section, the simulation procedure for modeling PCCC systems is presented. Then simulation curves are presented show the influence of iteration number, frame size, code rate, and code generator. The properties of PCCC performance are discussed based on the simulations.

### 4.1.1 Simulation Scheme

#### Interleaver

In [62], it was shown that interleaver design is a minor issue when BER is concerned. The BER of PCCCs using random interleavers continues to decrease even when the FER increases again. A random interleaver is sufficient for achieving BER close to capacity. In contrast, the FER of PCCCs using random interleavers decreases as the frame size $N$ increases when $N$ is small, but the FER increases again after reaching a minimum around $N = 2000$. Thus careful interleaver design (e.g., the spread interleaver of [63]) is required for long block lengths to ensure FER performance within a fraction of a decibel of the theoretical limit.

Since this chapter focuses on the BER performance, random interleavers are used in the simulations of PCCCs. In addition, new random interleavers are generated for each frame to reflect the average performance among different interleavers.

#### Trellis Termination

For each frame, the encoder is reset before encoding so that the initial state $S_0$ is always the all-zero state, and the $\alpha$ recursion is initialized by Equation (3.103).

Since the constituent encoders are recursive, the trellis can not be returned to all-zero state by simply appending $m$ zeros to the end of the data stream. For a given RSC code and a given information frame, the $m$ tail bits are determined by the state after encoding the information frame. Because of the existence of the pseudorandom interleaver between the constituent RSC encoders, the information frames are sent to RSC1 and RSC2 in different order. As a result the vector of tail bits for each encoder is different. In the simulation, the first trellis is terminated with $m$ tail bits. The expanded frame, (information bits + $m$ tail bits), is fed to RSC2 after being permuted by the interleaver. The second trellis is left open. As a result, the $\beta$ recursion of RSC1 is initialized by Equation (3.104), while that of RSC2 is initialized by (3.105). The performance of this terminating scheme is similar to that of terminating both trellises since the tail bits are shared between RSC1 and RSC2, and are decoded in the same way as the information bits.

**Channel Model**

In the simulation, BPSK modulation and an AWGN channel are assumed. The transmitted symbols are +1/-1, corresponding to the code bits of 1/0. By scaling the random number of distribution $\mathcal{N}(0,1)$ with the standard deviation $\sigma$, an AWGN noise of distribution $\mathcal{N}(0,\sigma^2)$ is obtained, which is added to the symbol to emulate the noisy channel effect.

### 4.1.2 Influence of Iteration Number

In Figures 4.26 through 4.31, BER versus $E_b/N_0$ curves are shown parameterized by the number of decoding iterations. The simulation parameters are provided in Table 4.1. Rate 1/2 codes are obtained from their rate 1/3 counterparts by alternately puncturing the parity bits of the constituent encoders.

Table 4.1: Simulation parameters for Figures 4.26 to 4.31.

| **Figure** | **Code Generator** | **Rate** | **Frame Sizes** |
|---|---|---|---|
| 4.26 | $g(D) = [1, \frac{1+D^2}{1+D+D^2}]$ | 1/2 | 200, 400, 1000, 2000, 4000, 16000 |
| 4.27 | $g(D) = [1, \frac{1+D^2}{1+D+D^2}]$ | 1/3 | 200, 400, 1000, 2000, 4000, 16000 |
| 4.28 | $g(D) = [1, \frac{1+D+D^2+D^3}{1+D+D^3}]$ | 1/2 | 200, 400, 1000, 2000, 4000, 16000 |
| 4.29 | $g(D) = [1, \frac{1+D+D^2+D^3}{1+D+D^3}]$ | 1/3 | 200, 400, 1000, 2000, 4000, 16000 |
| 4.30 | $g(D) = [1, \frac{1+D+D^2+D^3+D^4}{1+D+D^4}]$ | 1/2 | 200, 400, 1000, 2000, 4000, 16000 |
| 4.31 | $g(D) = [1, \frac{1+D+D^2+D^3+D^4}{1+D+D^4}]$ | 1/3 | 200, 400, 1000 |

Figures 4.26 and 4.27 show that the BER decreases as the number of iterations increases and tends to converge. When $N$ is small, three to five iterations are enough for the BER to converge and more iterations bring little gain. However, more iterations can continue to bring significant improvement when $N$ is large, e.g., $N = 16000$. Also, when $N$ is large, the error floor, where the BER versus $E_b/N_0$ curve flattens, shows up clearly. Comparing Figure 4.27 to Figure 4.26, it can be observed that a larger coding gain is achieved by decreasing the code rate from 1/2 to 1/3.

Comparing Figures 4.28, 4.29, 4.30, and 4.31 to Figures 4.26 through 4.27, it can be observed that, for the same frame size, code rate, and number of iterations, increasing the constraint length decreases BER within the region where BER$< 10^{-3}$. In Figure 4.31, BER versus $E_b/N_0$ curves were not generated for higher $N$ because the BER was too low to simulate within a reasonable period of time.

### 4.1.3 Influence of Frame Size

In Figures 4.32 to 4.37, BER versus $E_b/N_0$ curves are shown parameterized by the frame size. The simulation parameters are provided in Table 4.2. Rate 1/2 codes are obtained from the rate 1/3 counterparts by alternately puncturing the parity bits of the constituent encoders. These figures show that frame size is an important parameter in the design of turbo codes. For a fixed $E_b/N_0$, a larger frame size corresponds to a lower BER. For a fixed BER, the amount of improvement attained by increasing $N$ decreases as $N$ increases.

Table 4.2: Simulation parameters for Figures 4.32 to 4.37.

| **Figure** | **Code Generator** | **Rate** | **Frame Sizes** | **Iteration** |
|---|---|---|---|---|
| 4.32 | $g(D) = [1, \frac{1+D^2}{1+D+D^2}]$ | 1/2 | 200 to 16000 | 10 |
| 4.33 | $g(D) = [1, \frac{1+D^2}{1+D+D^2}]$ | 1/3 | 200 to 16000 | 10 |
| 4.34 | $g(D) = [1, \frac{1+D+D^2+D^3}{1+D+D^3}]$ | 1/2 | 200 to 16000 | 10 |
| 4.35 | $g(D) = [1, \frac{1+D+D^2+D^3}{1+D+D^3}]$ | 1/3 | 200 to 16000 | 10 |
| 4.36 | $g(D) = [1, \frac{1+D+D^2+D^3+D^4}{1+D+D^4}]$ | 1/2 | 200 to 16000 | 10 |
| 4.37 | $g(D) = [1, \frac{1+D+D^2+D^3+D^4}{1+D+D^4}]$ | 1/3 | 200, 400, 1000 | 10 |

### 4.1.4 Influence of Code Rate

In Figures 4.38 to 4.43, the BER versus $E_b/N_0$ curves are shown parameterized by the code rate. The simulation parameters are provided in Table 4.3. Rate 1/2 codes are obtained from their rate 1/3 counterparts by alternately puncturing the parity bits of the constituent encoders. In Figure 2.6, we see that to achieve errorless transmission, the minimum $E_b/N_0$ is $E_b/N_0 = 0.2$ dB for rate 1/2 codes and $E_b/N_0 = -0.48$ dB for rate 1/3 codes. In other words, 0.68 dB extra coding gain is achieved by decreasing the code rate from 1/2 to 1/3. This difference is approximately matched by the simulation results in Figures 4.38 to 4.43.

Table 4.3: Simulation parameters for Figures 4.38 to 4.43.

| Figure | Constraint Length | Rate | Frame Size | Iteration |
|---|---|---|---|---|
| 4.38 | 3, 4, 5 | 1/2, 1/3 | 200 | 10 |
| 4.39 | 3, 4, 5 | 1/2, 1/3 | 400 | 10 |
| 4.40 | 3, 4, 5 | 1/2, 1/3 | 1000 | 10 |
| 4.41 | 3, 4 | 1/2, 1/3 | 2000 | 10 |
| 4.42 | 3, 4 | 1/2, 1/3 | 4000 | 10 |
| 4.43 | 3, 4 | 1/2, 1/3 | 16000 | 18 |

### 4.1.5 Influence of Code Generator

In Figures 4.44 and 4.45, BER versus $E_b/N_0$ curves are shown for constituent codes of constraint length three, four, and five. Figure 4.44 is for code rate 1/2, and Figure 4.45 is for code rate 1/3. Ten decoding iterations were performed for frame sizes of 200 and 1000, and eighteen decoding iterations were performed for the frame size of 16000.

From these figures, we see that for a given frame size and code rate, the BER curves for different constraint lengths diverges gradually as $E_b/N_0$ increases. The divergence is more pronounced when the frame size is large. Also, the improvement achieved when the constraint length is increased from three to four is larger than the improvement achieved when the constraint length is increased from four to five.

### 4.1.6 Conclusion

The following conclusions can be drawn from the simulation campaign:

- The BER decreases as the number of iterations increases. The improvement gradually diminishes as the number of iterations increases. After a certain number of iterations, the BER converges. For a given code rate, it is obvious that the larger the frame size, the more iterations the decoder needs to converge.

- Code rate has a noticeable yet small influence on the speed of convergence. The lower the code rate, the more iterations the BER takes to converge.

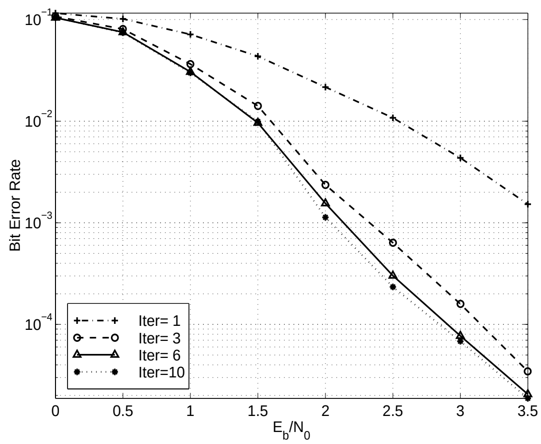
- The BER curves can be divided into three regions, which are especially evident when the frame size is large, e.g., 16000.

  - The first region is BER$> 10^{-2}$, where BER decreases slowly as $E_b/N_0$ increases. Codes with different parameters do not show much difference.
  - The second region is approximately $10^{-2} >$BER$> 10^{-6}$, where BER drops abruptly as $E_b/N_0$ increases. This is also called the "waterfall" region. This is the region where the PCCCs perform better in comparison to other codes. The gain comes from the characteristic that the multiplicities of low weight code sequences are small.
  - The third region is BER$< 10^{-6}$, where BER decreases very slow with respect to $E_b/N_0$. This is called the "error floor" of PCCCs, and it exists because of the small free distance of PCCCs.

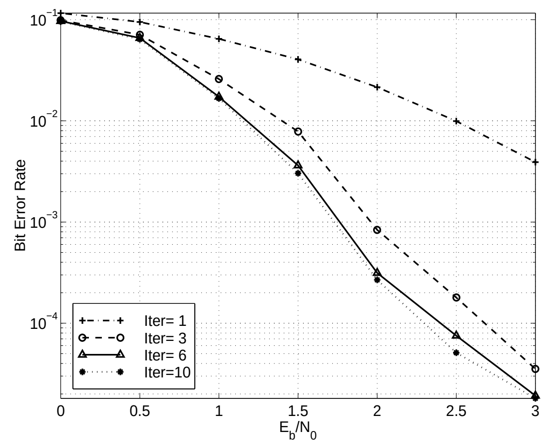
- The BER decreases approximately at the rate of $N^{-1}$ as the frame size increases in the waterfall region. This manifests the coding gain of PCCCs.

- The BER is lower for smaller code rate. Decreasing the code rate from 1/2 to 1/3 provides 0.5∼0.8 dB extra coding gain around BER$= 10^{-3} \sim 10^{-5}$. The extra coding gain coming from the decreased code rate increases with the increase of the frame size and with the decrease of BER.

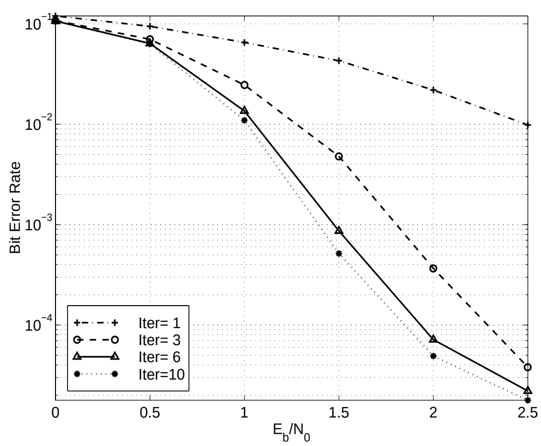
- Constraint length of the code generator does not play a significant role in the performance for BER$> 10^{-3}$. However, when BER$< 10^{-4}$, a difference becomes evident. The lower BER is, the more significant the influence of the code generator. The error floor is lower for the schemes with larger constraint length. A code generator with a larger constraint length provides a better performance at the price of complexity. The improvement coming from increasing constraint length $K = 3$ to $K = 4$ is larger than that coming from $K = 4$ to $K = 5$. Since the complexity of the decoder is proportional to $2^{K-1}$, it is a better tradeoff to use a $K = 4$ code than a $K = 5$ code.



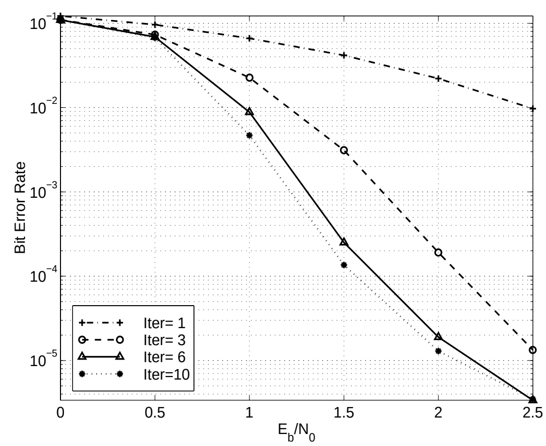

(a) N=200



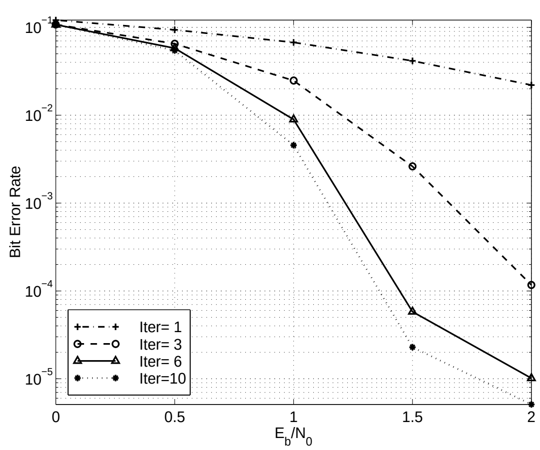

(b) N=400



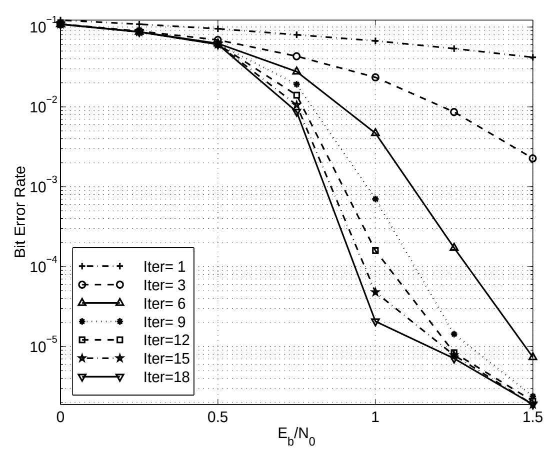

(c) N=1000




(d) N=2000

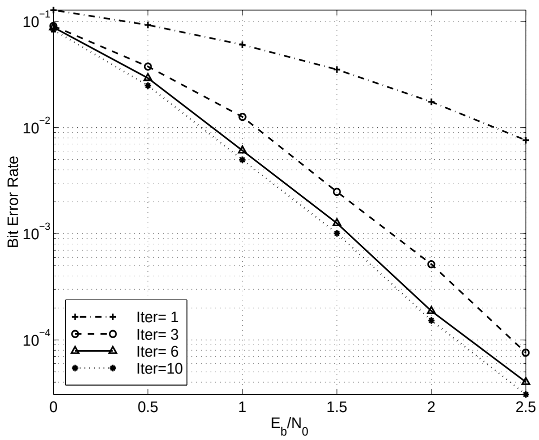



(e) N=4000

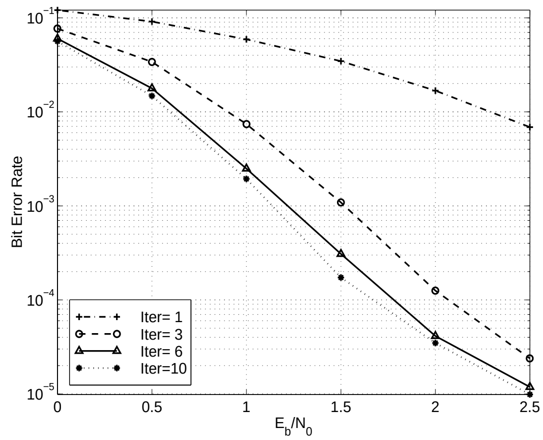



(f) N=16000

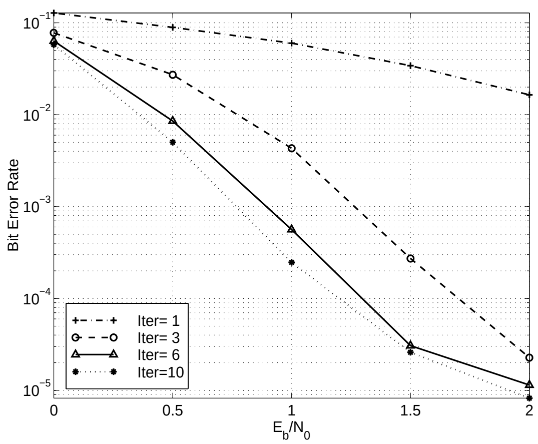
Figure 4.26: BER versus $E_b/N_0$ as parameterized by the number of decoding iterations ($g = (7,5)_{octal}$, rate 1/2).



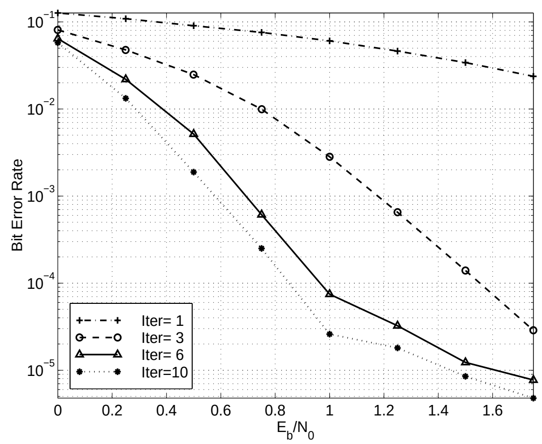

(a) N=200



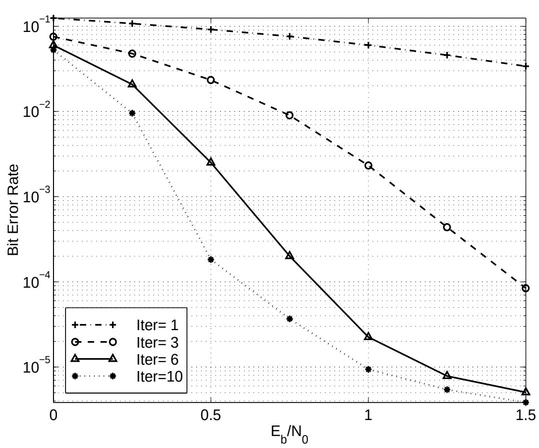

(b) N=400



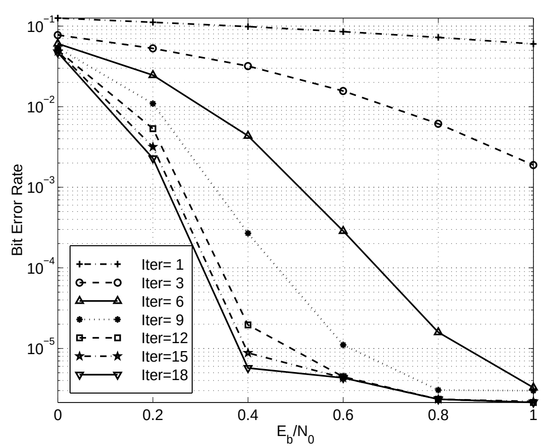

(c) N=1000




(d) N=2000

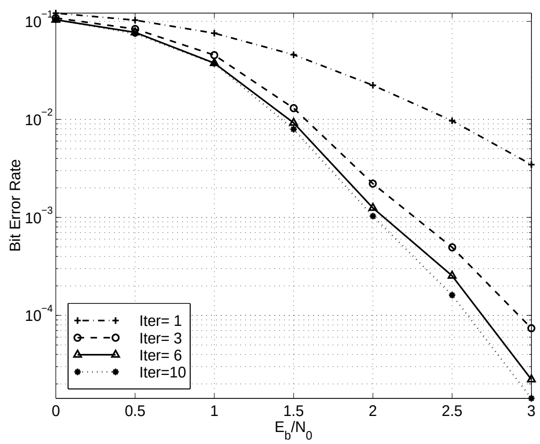



(e) N=4000

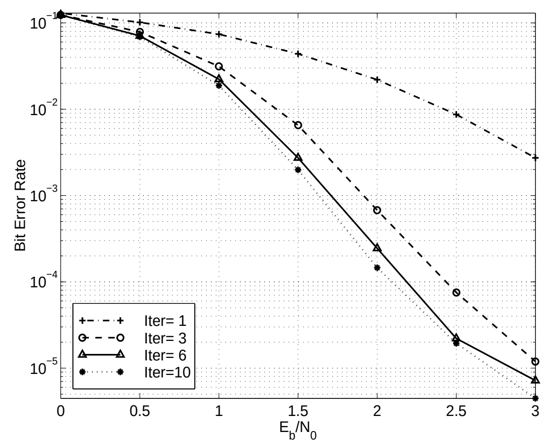



(f) N=16000

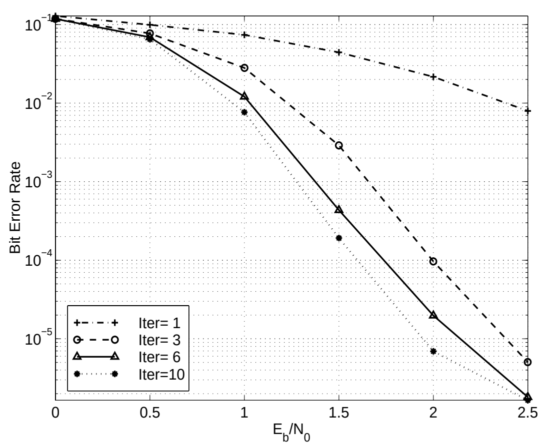
Figure 4.27: BER versus $E_b/N_0$ as parameterized by the number of decoding iterations ($g = (7, 5)_{octal}$, rate 1/3).



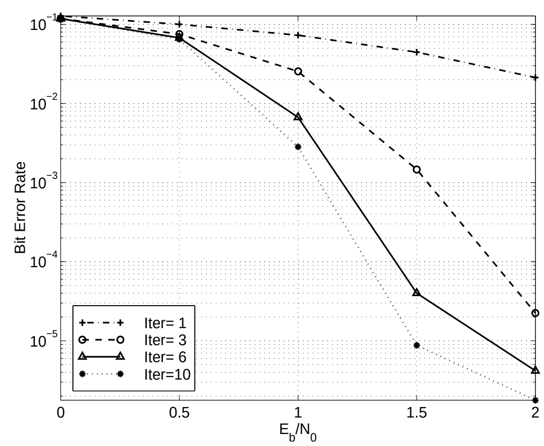

(a) N=200



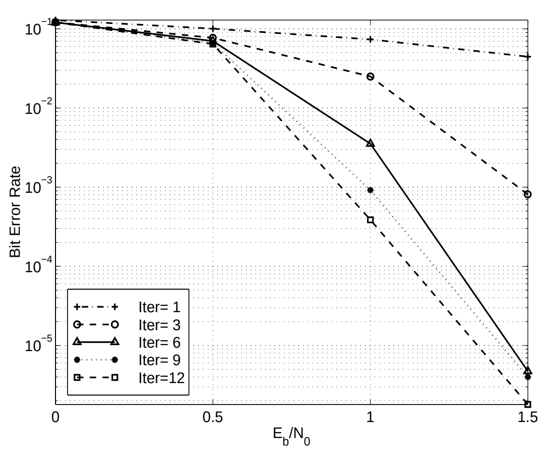

(b) N=400



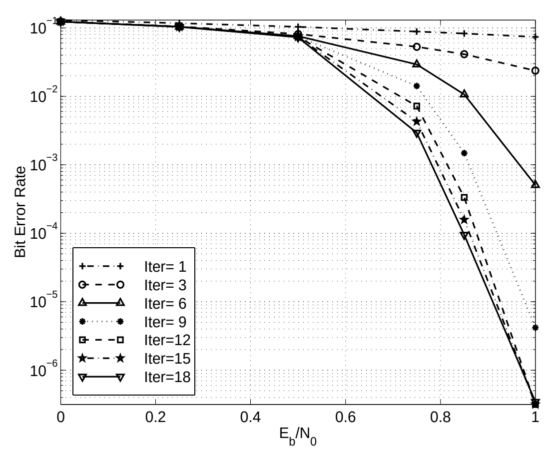

(c) N=1000




(d) N=2000




(e) N=4000




(f) N=16000

Figure 4.28: BER versus $E_b/N_0$ as parameterized by the number of decoding iterations ($g = (15, 17)_{octal}$, rate 1/2).

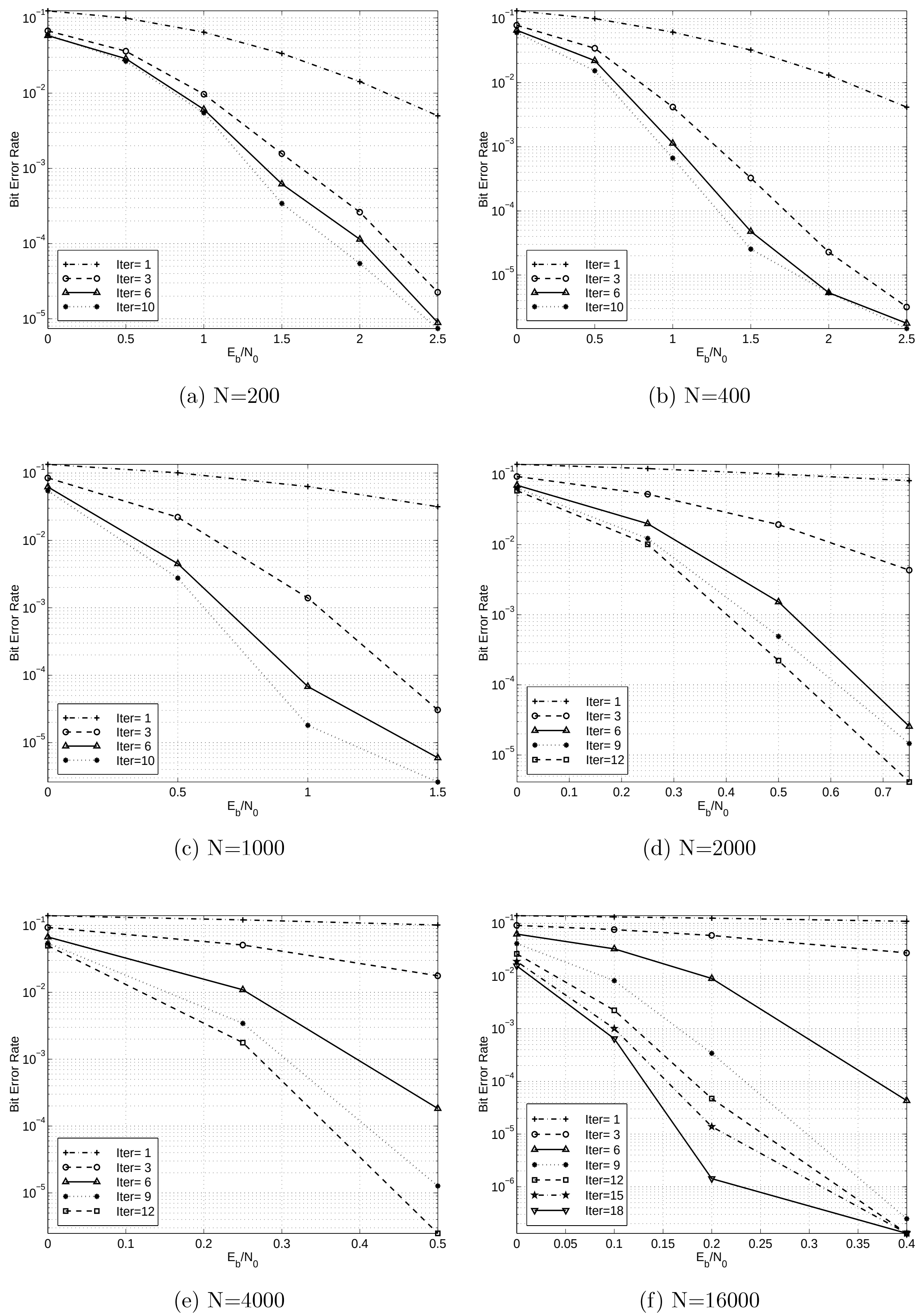
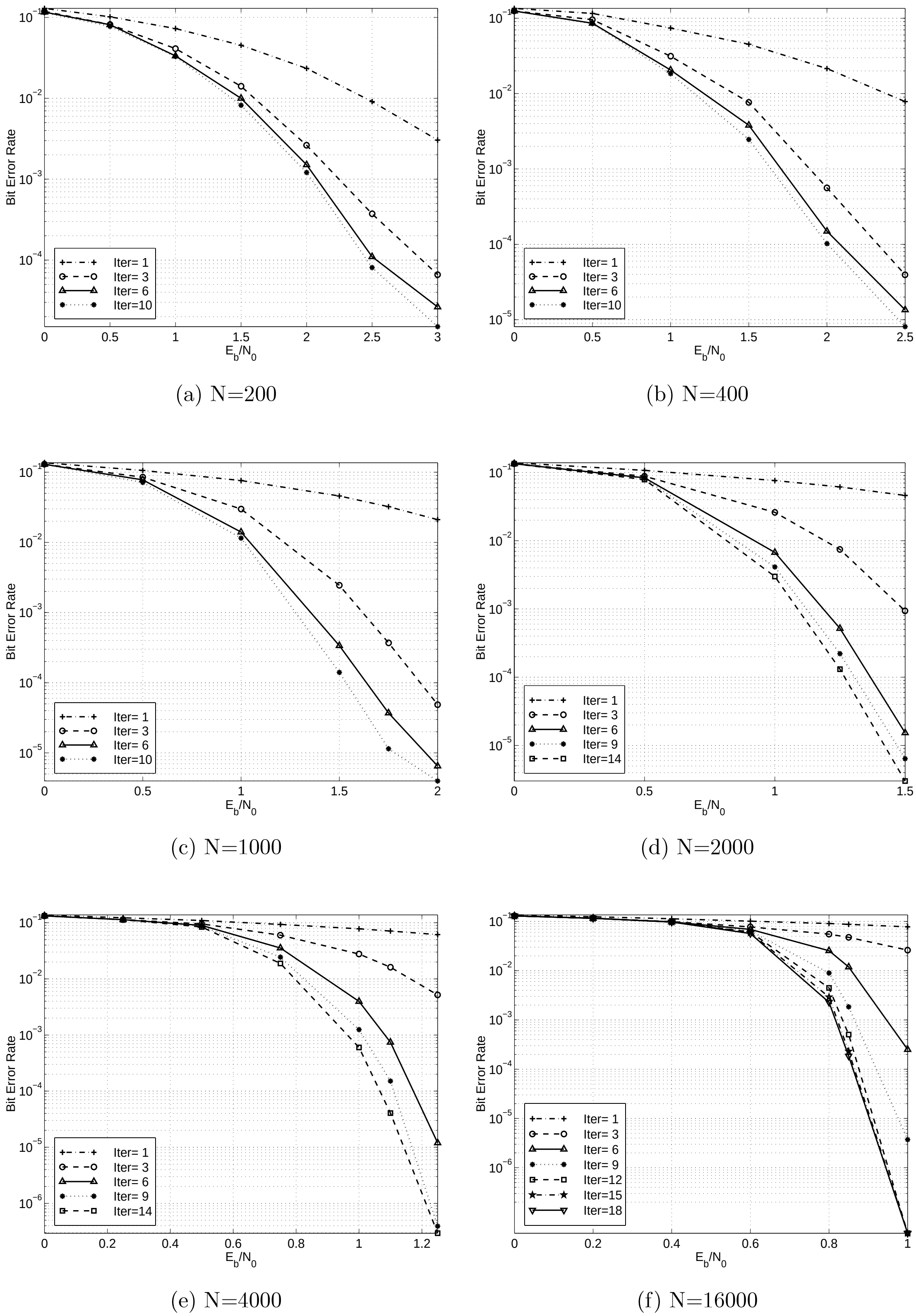



Figure 4.29: BER versus $E_b/N_0$ as parameterized by the number of decoding iterations ($g = (15, 17)_{octal}$, rate 1/3).




(a) N=200

(b) N=400

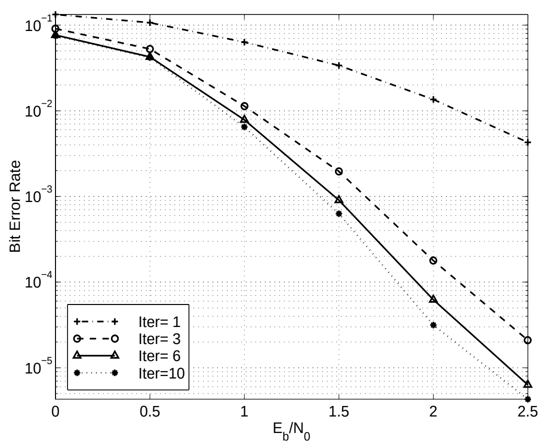
(c) N=1000

(d) N=2000

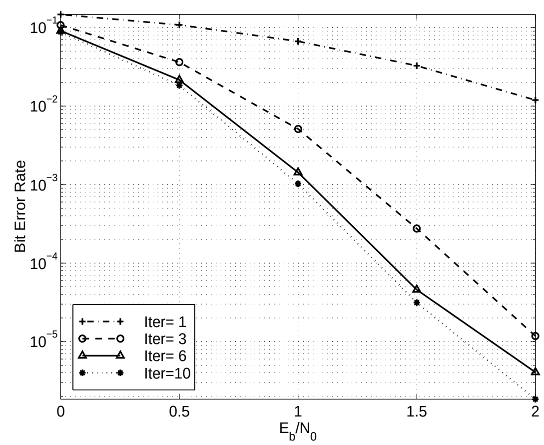
(e) N=4000

(f) N=16000

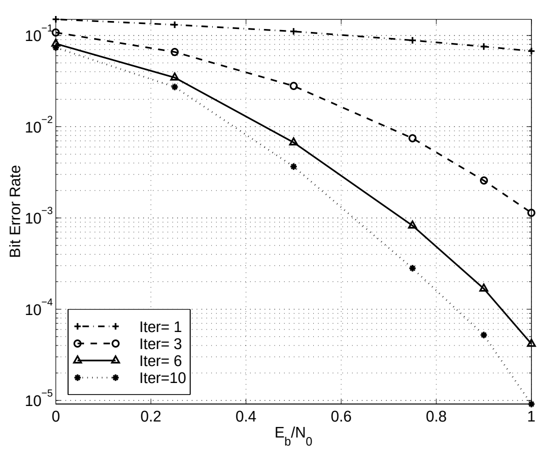
Figure 4.30: BER versus $E_b/N_0$ as parameterized by the number of decoding iterations ($g = (31, 37)_{octal}$, rate 1/2).




(a) N=200




(b) N=400




(c) N=1000

Figure 4.31: BER versus $E_b/N_0$ as parameterized by the number of decoding iterations $(g = (31, 37)_{octal}$, rate 1/3).

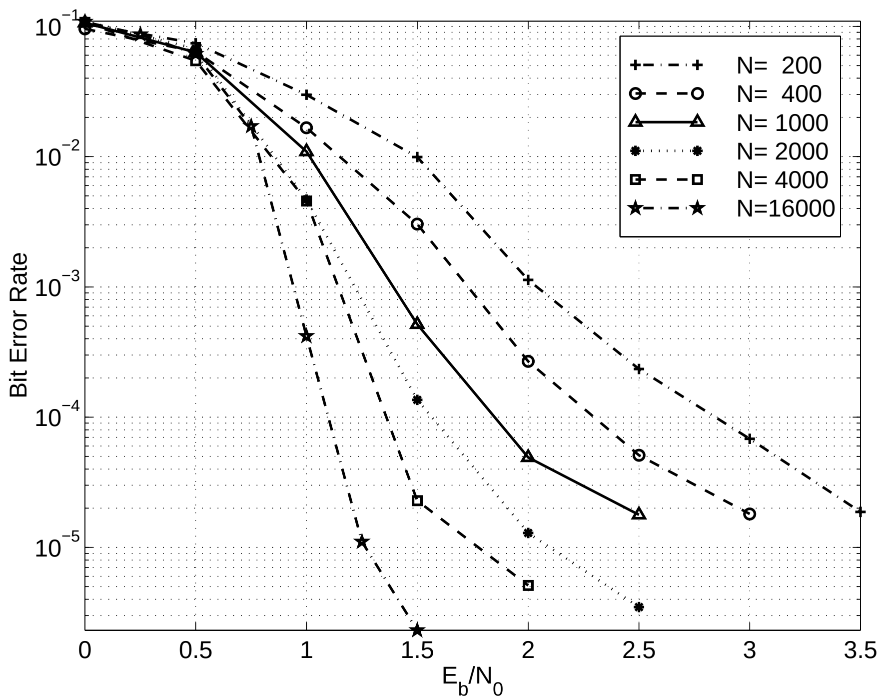



Figure 4.32: BER versus $E_b/N_0$ as parameterized by frame size $N$ ($g = (7,5)_{octal}$, rate 1/2, 10 iterations).

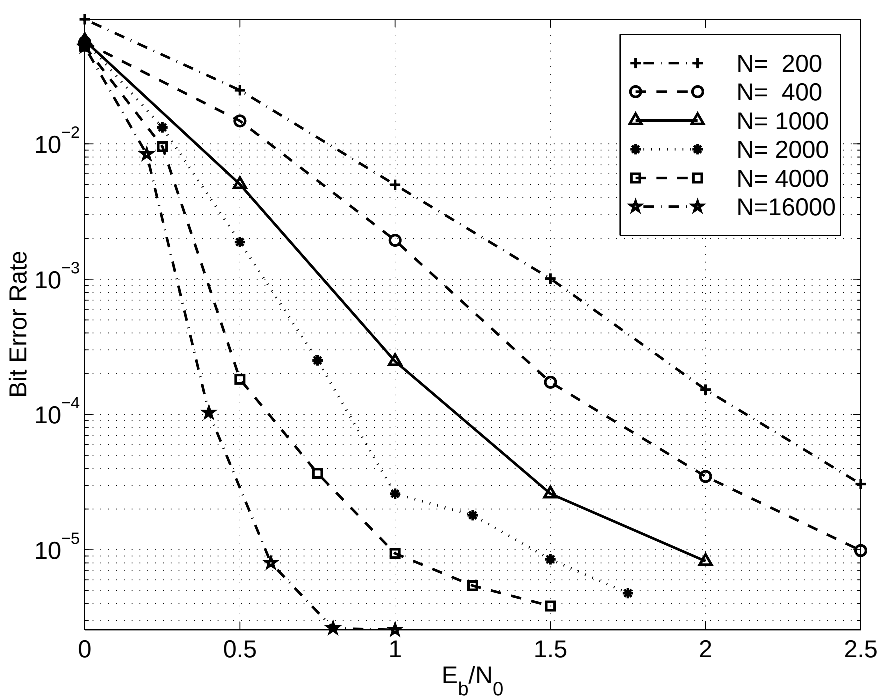



Figure 4.33: BER versus $E_b/N_0$ as parameterized by frame size $N$ ($g = (7,5)_{octal}$, rate 1/3, 10 iterations).

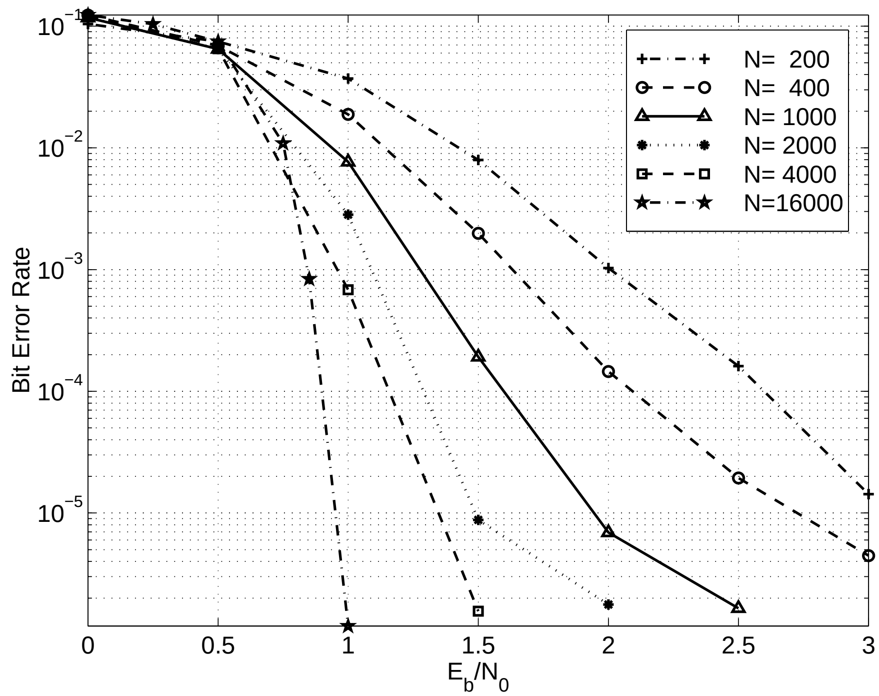



Figure 4.34: BER versus $E_b/N_0$ as parameterized by frame size $N$ ($g = (15, 17)_{octal}$, rate 1/2, 10 iterations).

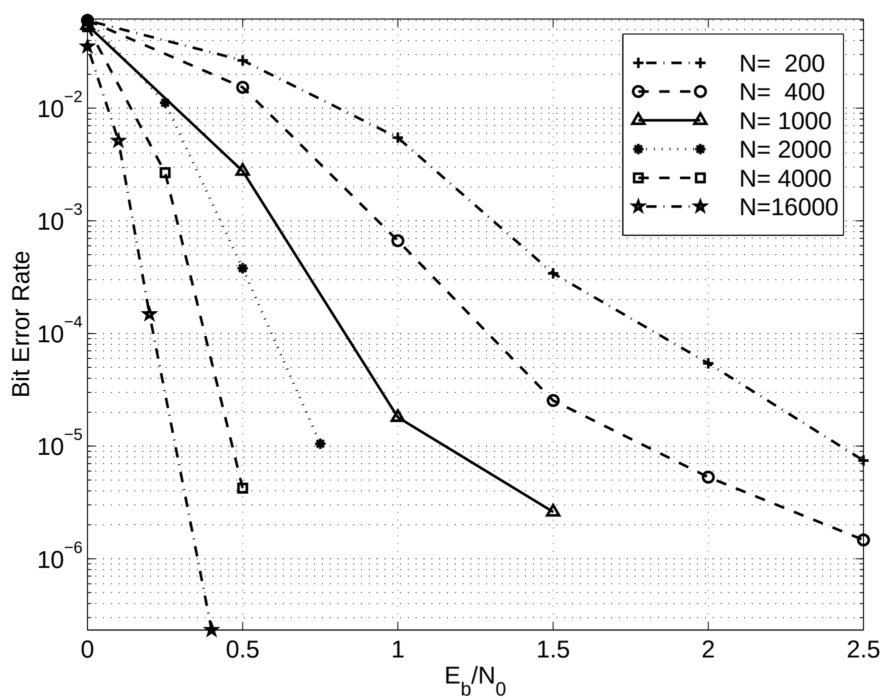



Figure 4.35: BER versus $E_b/N_0$ as parameterized by frame size $N$ ($g = (15, 17)_{octal}$, rate 1/3, 10 iterations).

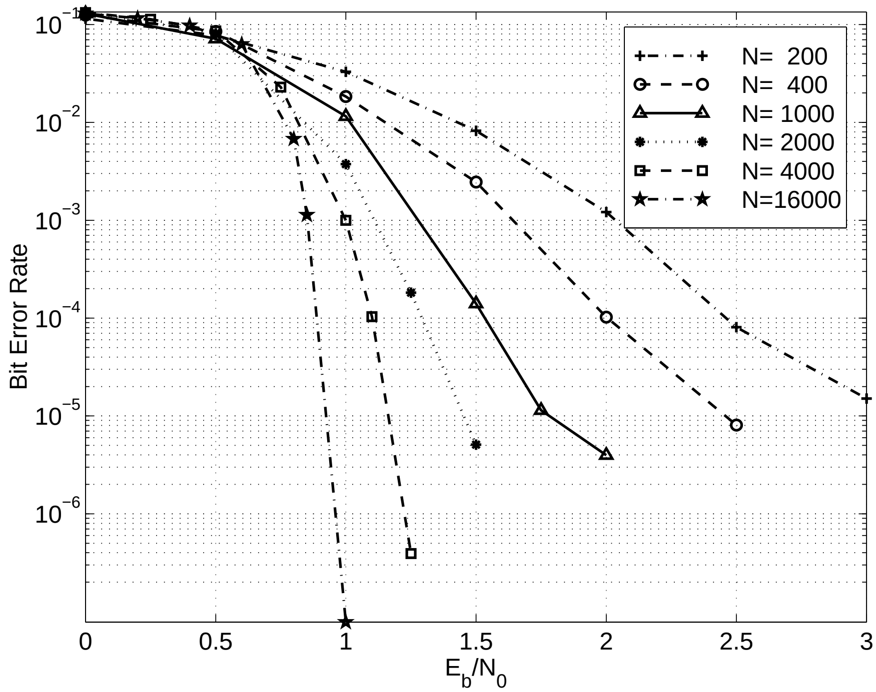



Figure 4.36: BER versus $E_b/N_0$ as parameterized by frame size $N$ ($g = (31, 37)_{octal}$, rate 1/2, 10 iterations).

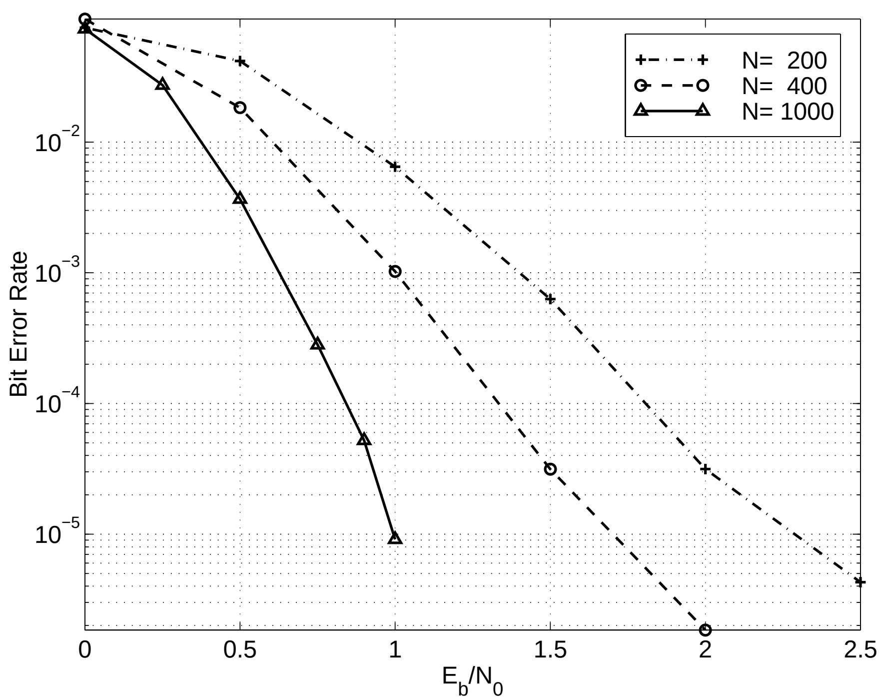



Figure 4.37: BER versus $E_b/N_0$ as parameterized by frame size $N$ ($g = (31, 37)_{octal}$, rate 1/3, 10 iterations).

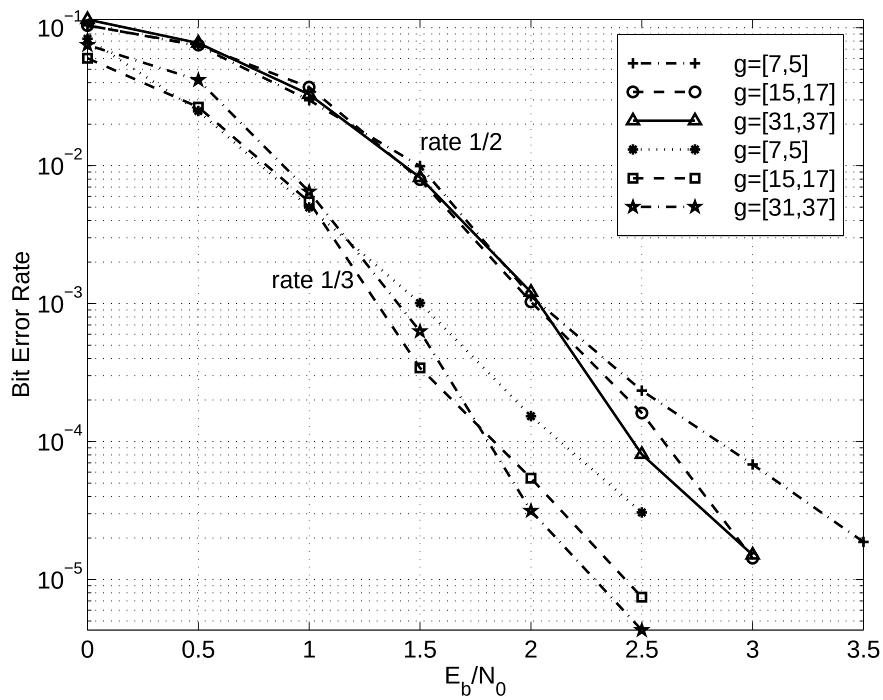



Figure 4.38: BER versus $E_b/N_0$ as parameterized by code rate ($N = 200$, 10 iterations).

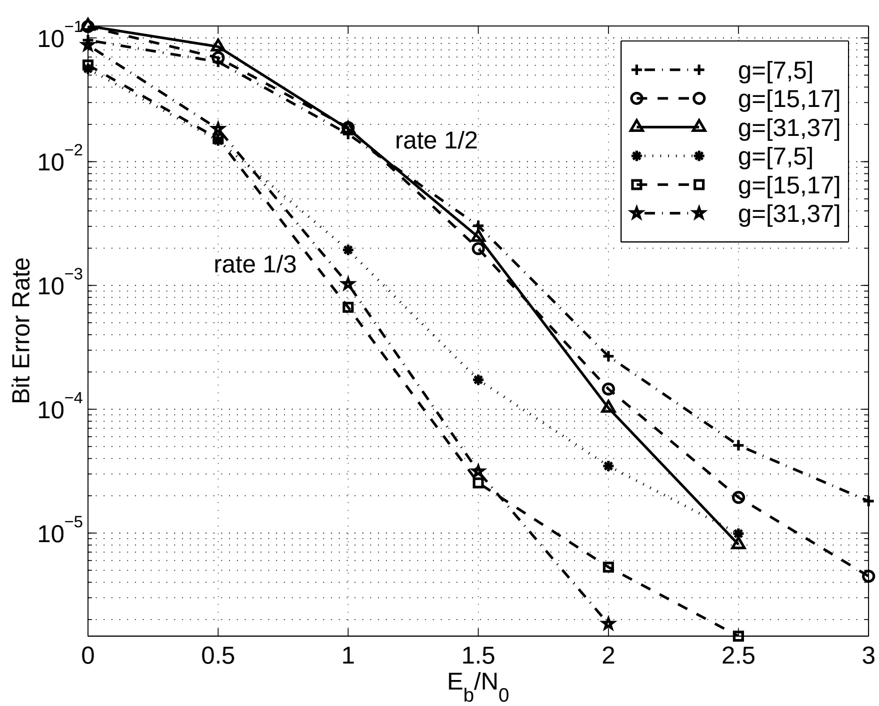



Figure 4.39: BER versus $E_b/N_0$ as parameterized by code rate ($N = 400$, 10 iterations).

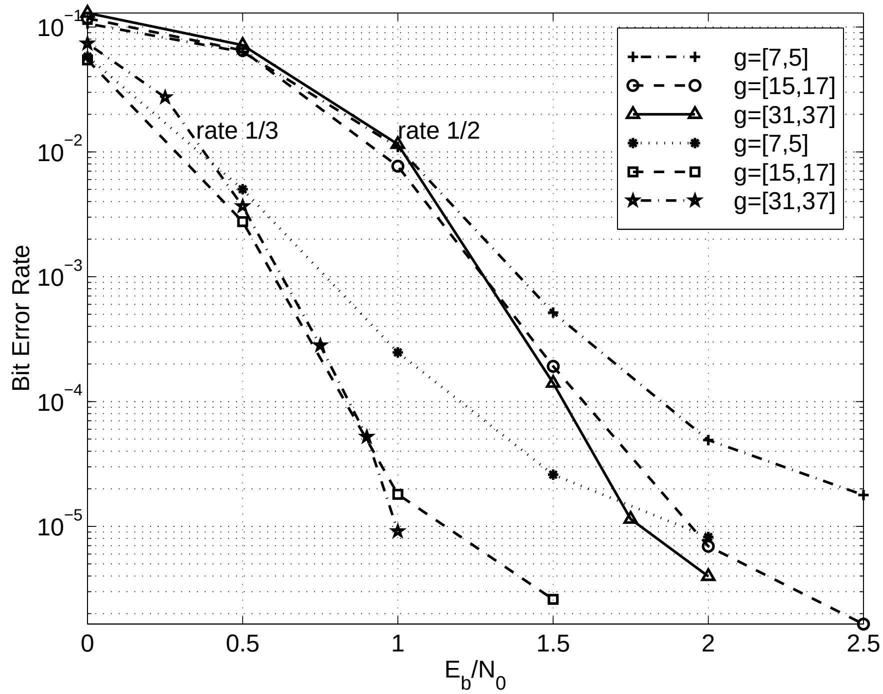



Figure 4.40: BER versus $E_b/N_0$ as parameterized by code rate ($N = 1000$, 10 iterations).

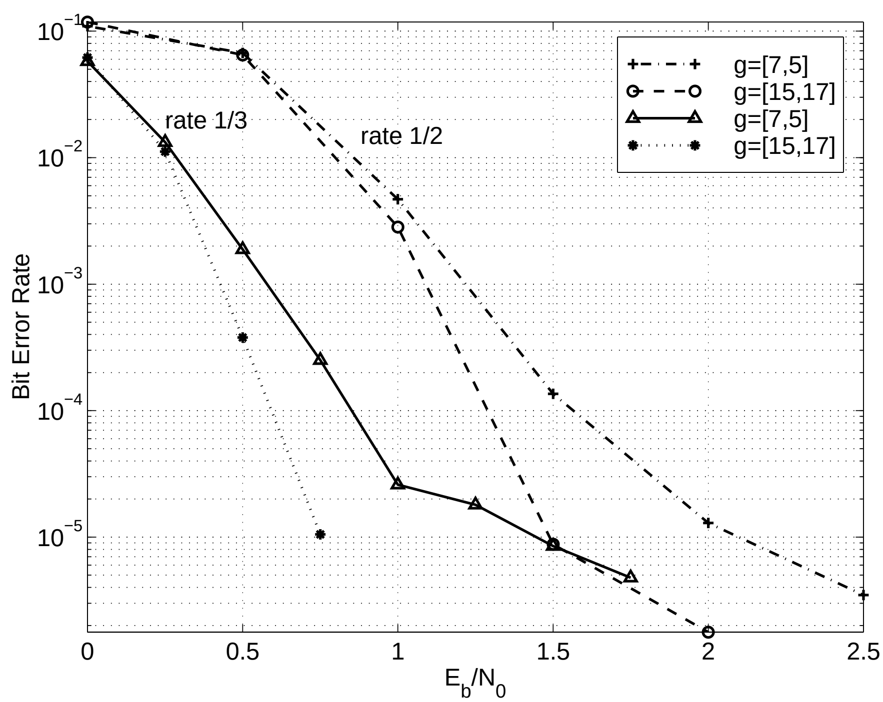



Figure 4.41: BER versus $E_b/N_0$ as parameterized by code rate ($N = 2000$, 10 iterations).

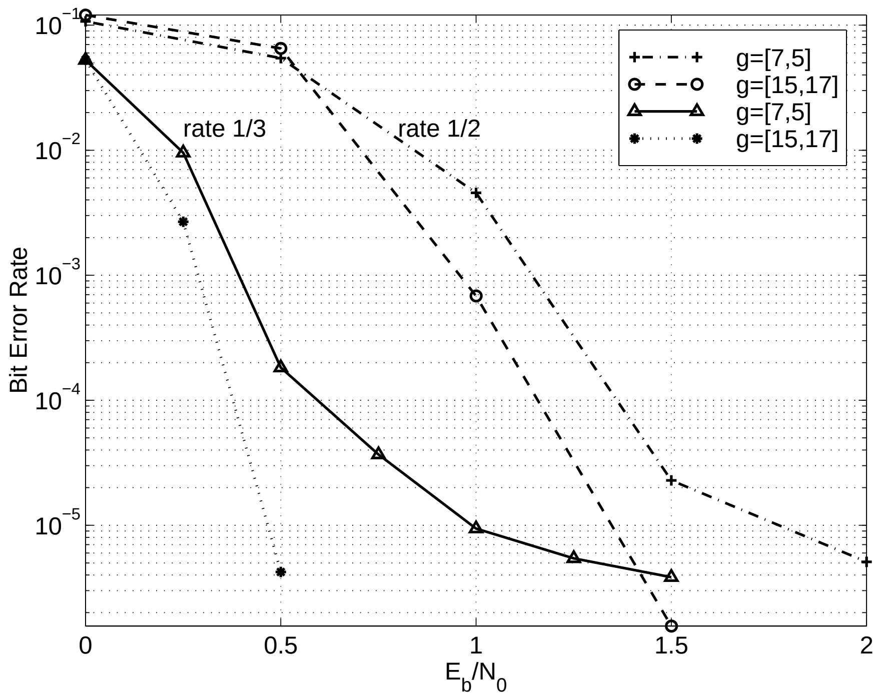



Figure 4.42: BER versus $E_b/N_0$ as parameterized by code rate ($N = 4000$, 10 iterations).

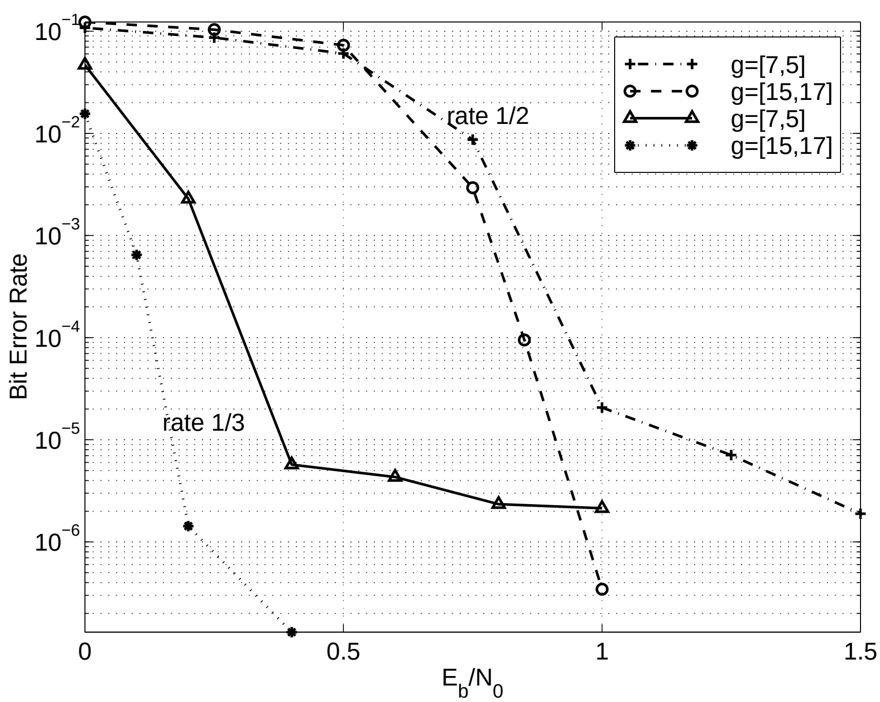



Figure 4.43: BER versus $E_b/N_0$ as parameterized by code rate ($N = 1.6 \times 10^4$, 18 iterations).

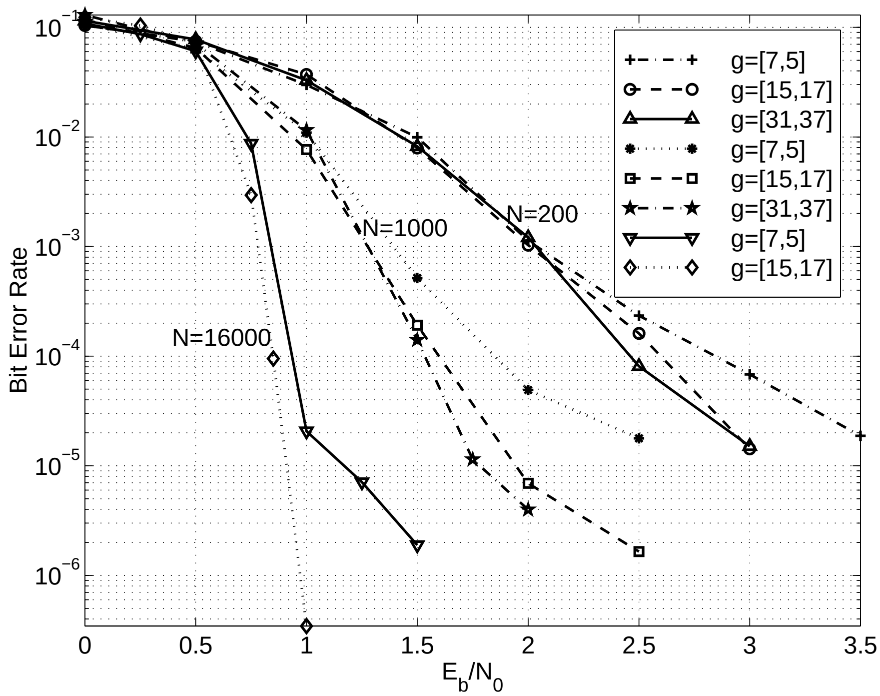



Figure 4.44: BER versus $E_b/N_0$ as parameterized by code generator (code rate 1/2, 10 iterations for $N = 200$ and 1000, 18 iterations for $N = 1.6 \times 10^4$).

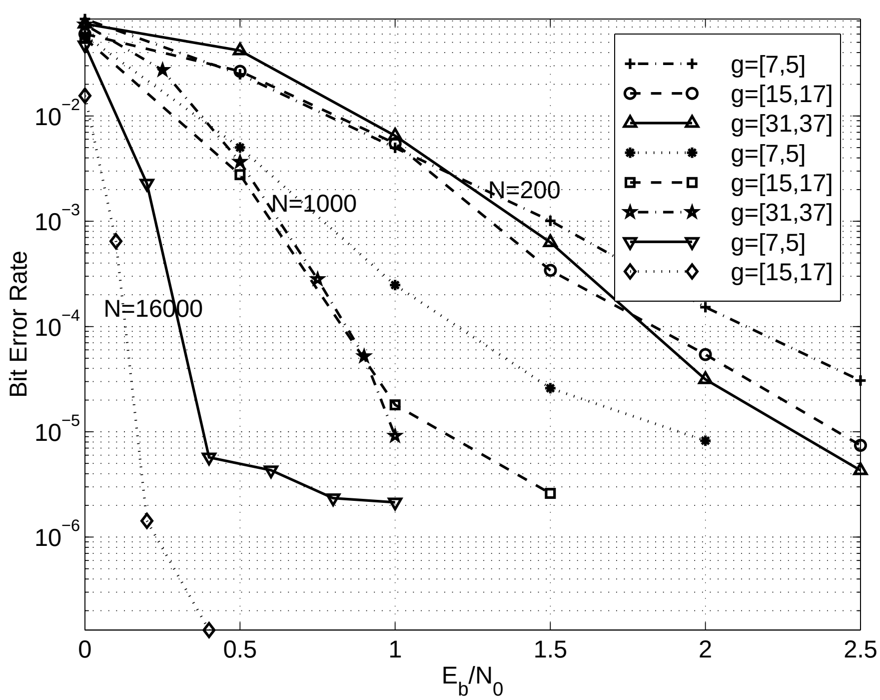



Figure 4.45: BER versus $E_b/N_0$ as parameterized by code generator (code rate 1/3, 10 iterations for $N = 200$ and 1000, 18 iterations for $N = 1.6 \times 10^4$).

## 4.2 Simulation Study of SCCCs

### 4.2.1 Simulation Scheme

Spread interleavers [63], instead of random interleavers, are used for SCCCs. It was found that, although interleaver design has almost no influence on the SCCC performance when BER$> 10^{-3}$, the choice of interleaver makes great difference for BER$< 10^{-4}$. The coding gain improvement resulting from the interleaver choice can be as much as 1 dB at BER$= 10^{-5}$. Since spread interleavers were shown to perform best [64], a spread interleaver is designed for each frame size in the SCCC simulations.

In the SCCC encoder, both the inner code and outer code are RSC codes. Thus $m$ tail bits are added at the end of the frame in order to terminate them. The outer code is punctured and has rate $r^o = 1/2$. The parity bits of the inner code are alternately punctured so that the inner code rate is increased to $r^i = 2/3$. In consequence, the overall code rate is: $1/2 \times 2/3 = 1/3$.

Just like PCCCs, BPSK modulation and an AWGN channel model are assumed for SCCCs.

### 4.2.2 Simulation Results

In Figure 4.46, the curves of BER versus $E_b/N_0$ are shown for a SCCC with varing number of decoding iterations. The information frame size is 5120. Both constituent codes are $K = 3$ RSC codes with $g(D) = [1, 1 + D^2/1 + D + D^2]$.

Figure 4.47 shows the curves for a PCCC of rate 1/3 as a comparison. The constituent codes are $K = 4$ RSC codes with $g(D) = [1, 1+D+D^2+D^3/1+D+D^3]$. These codes are chosen because the complexity of a $K = 3$ SCCC is closer to a $K = 4$ PCCC than a $K = 3$ PCCC.

In Figure 4.48, simulation curves for BER versus $E_b/N_0$ are drawn for a SCCC with the same structure as that of Figure 4.46. Four different frame sizes are simulated: $N$=160, 320, 640, and 5120. Figure 4.49 shows the BER curves of the same frame sizes for a PCCC of rate 1/3 as a comparison.

In Figure 4.50, $\Delta E_b/N_0$ is shown, which is the difference between the $E_b/N_0$ required by a SCCC and that required by a PCCC to reach a certain bit error probability. In the region where $\Delta E_b/N_0 > 0$, the SCCC needs higher $E_b/N_0$ to achieve the same performance, while at $\Delta E_b/N_0 < 0$, the SCCC requires less power to achieve the same quality of performance.

The following conclusions can be drawn for SCCCs in comparison to PCCCs based on the simulation results.

- Just as for PCCCs, the BER of SCCCs decreases as the number of iteration increases, and the improvement slows down gradually with respect to the number of iterations.
- The BER can be reduced significantly by increasing the frame size.
- SCCCs perform better than PCCCs at very low BER region.
  - When BER$> 10^{-5}$, PCCCs have a larger coding gain. Based on our simulations, the difference is about $0.3 \sim 0.4$ dB. The slowly decreasing region of SCCCs extends into the waterfall region of PCCCs.
  - At around BER$= 10^{-5} \sim 10^{-6}$, PCCCs and SCCCs have similar coding gain.
  - At BER$< 10^{-6}$, the BER of SCCCs decreases sharply, while PCCCs reach their error floor region. SCCCs do not have a flat error floor region as PCCCs do.

  Hence, PCCC is a better scheme than SCCC at BER$> 10^{-5}$, while SCCC is a better choice than PCCC if the specification is BER$< 10^{-6}$. Note that the complexity of a constraint length-$K$ SCCC is about half way between that of a constraint length-$K$ PCCC and that of a constraint length-$(K+1)$ PCCC.

## 4.3 Summary

In this chapter, the results of an extensive simulation campaign were presented to illustrate the performance of PCCC and SCCC systems with the SISO decoding module introduced in Chapter 3. The waterfall and error floor regions of PCCCs were displayed, while SCCCs were shown to have a steep BER curve at medium-to-high $E_b/N_0$ region. In consequence, with comparable complexity, PCCCs perform better for BER$> 10^{-5}$, while SCCCs perform better for BER$< 10^{-6}$.

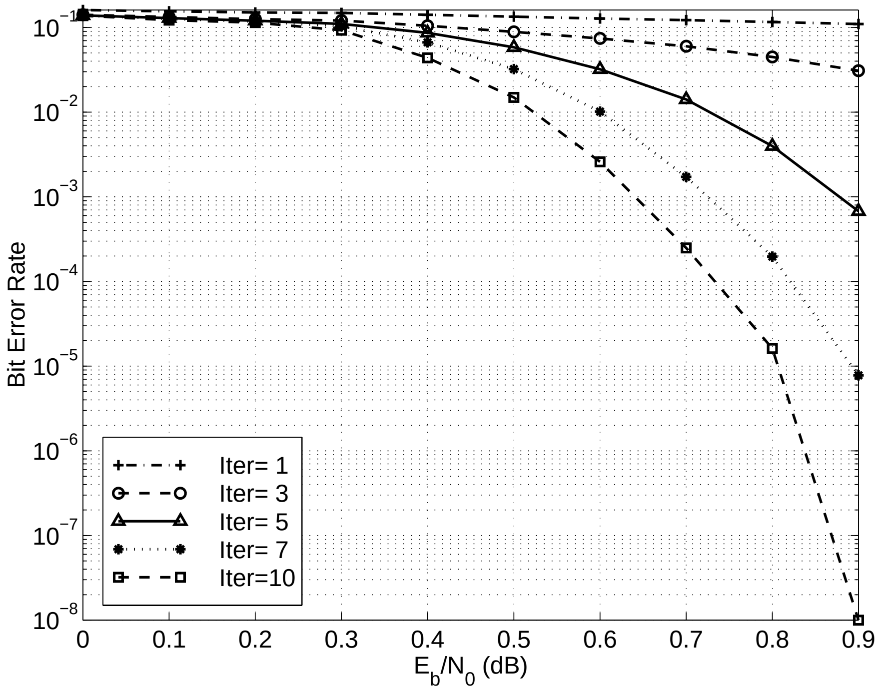



Figure 4.46: BER versus $E_b/N_0$ as parameterized by number of decoding iterations (SCCC, $g = (7, 5)_{octal}$, rate 1/3 ($r^o = 1/2, r^i = 2/3$), $N = 5120$).

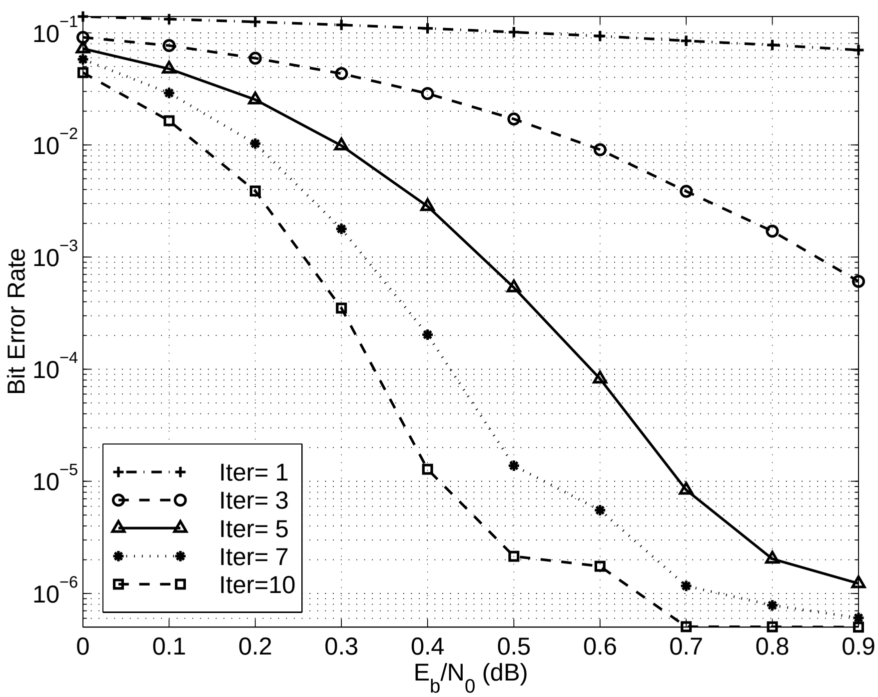



Figure 4.47: BER versus $E_b/N_0$ as parameterized by number of decoding iterations (PCCC, $g = (15, 17)_{octal}$, rate 1/3, $N = 5120$).

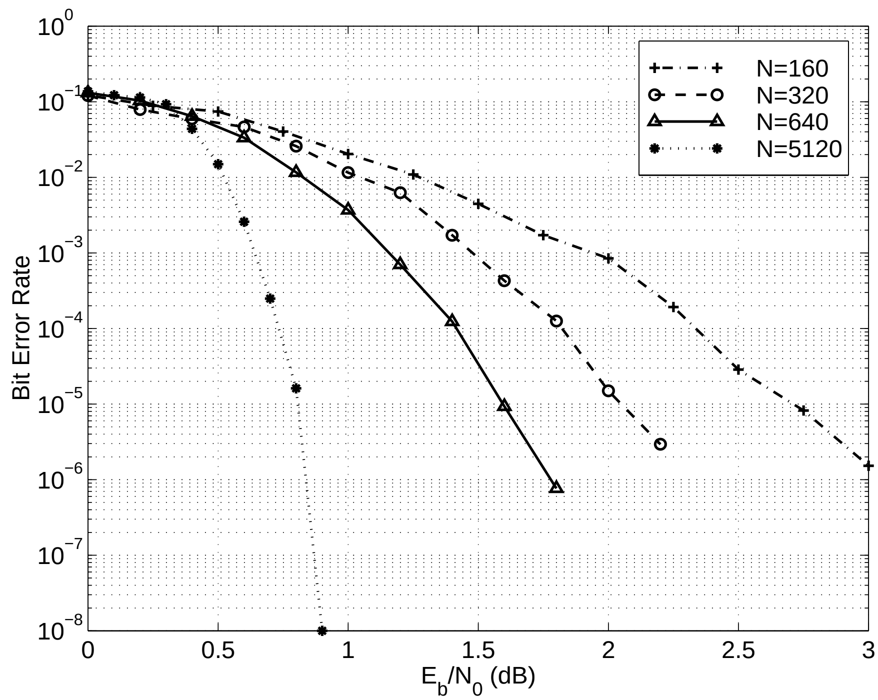



Figure 4.48: BER versus $E_b/N_0$ as parameterized by frame size $N$ (SCCC, $g = (7,5)_{octal}$, rate 1/3 ($r^o = 1/2, r^i = 2/3$), 10 iterations, $N$=160, 320, 640, 5120).

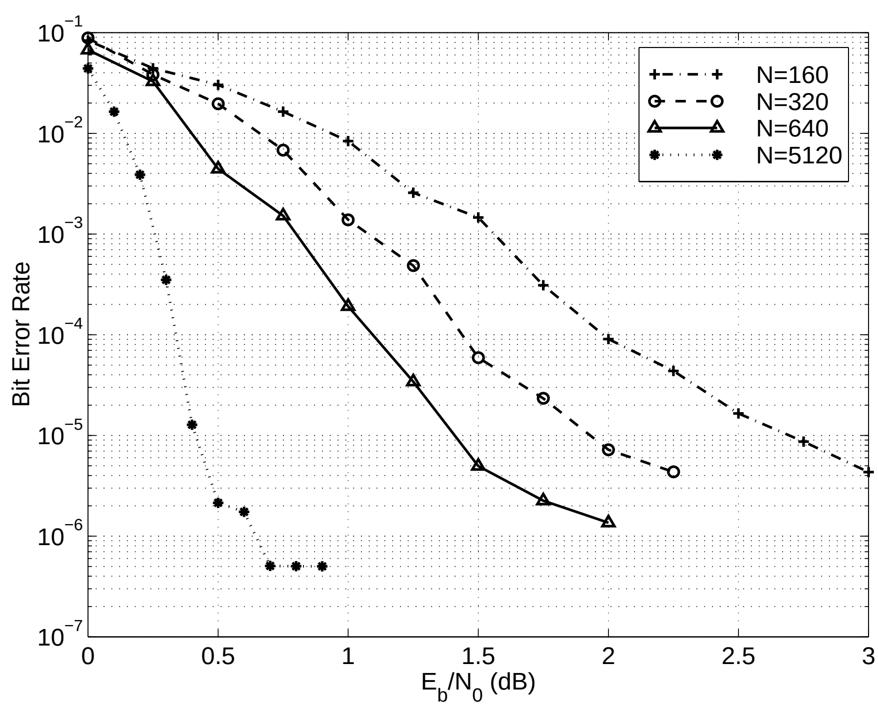



Figure 4.49: BER versus $E_b/N_0$ as parameterized by frame size $N$ (PCCC, $g = (15,17)_{octal}$, rate 1/3, 10 iterations, $N$=160, 320, 640, 5120).

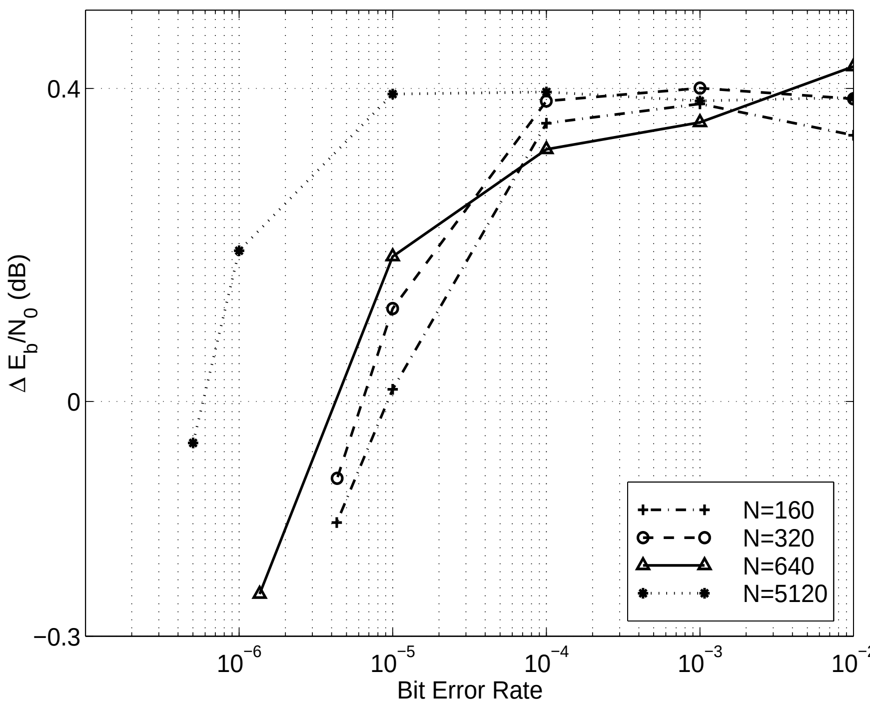



Figure 4.50: $\Delta E_b/N_0$ versus BER as parameterized by frame size $N$ (rate 1/3, 10 iterations, $\Delta E_b/N_0 = E_b/N_{0,\mathrm{SCCC}} - E_b/N_{0,\mathrm{PCCC}}$).

# Chapter 5

# Fixed Point Implementation of Turbo Coded Systems

In the preceding chapter, the superior performance of turbo codes was shown. However, the simulations assumed the availability of high precision floating point arithmetic. In a real-time decoder, it is likely that the decoding algorithms would need to be implemented using fixed point arithmetic, such as would be employed by a field programmable gate array (FPGA) or fixed point digital signal processing (DSP) chip. As a result, the amplitude of both signals and coefficients in the decoder is discrete.

This chapter examines the issues when a turbo decoder is implemented on a real-time hardware. Integer representation and fixed point arithmetic are considered. While the self-scaling of floating point arithmetic largely eliminates the quantization and rounding noise problem, numerical precision is an important issue for fixed point arithmetic. Section 5.1 focuses on the problem of quantizing input to the turbo decoder and the influence of the associated fixed point arithmetic where 32 bits are assumed available for representation and calculation inside the decoder.

Section 5.2 concentrates on the computation inside the SISO module. This section starts by examining the dynamic ranges of internal data, which leads to an analytical expression for the minimum data width. A modular renormalization technique, which eliminates the subtractive normalization, is shown applicable to the forward and backward path metric computation of SISO.

## 5.1 Influence of Quantization and Integer Arithmetic on Decoding Performance

Quantization and fixed point arithmetic inevitably add noise to the system. Adequate signal-to-noise ratio at the quantizer and throughout the whole process are essential for decoder implementation. A simple solution is to increase the signal level since the rounding noise level is fixed for a given structure. However, the signal level cannot be increased too much, otherwise the dynamic range of the quantizer and the fixed point arithmetic will be exceeded, and overflow follows. The possibility of overflow should be kept low since it causes severe nonlinear distortion. Thus, a balanced scaling factor, or optimal gain, needs to be found for the signal before it is fed into the decoding processor.

Motivated by the above considerations, this section investigates the dynamic range adjustment, and the influence of quantization and fixed point arithmetic for the implementation of the turbo decoder [65]. Two typical decoding algorithms were examined: Log-MAP and SOVA.

### 5.1.1 System Model

The system has a turbo encoder which consists of two identical parallel RSC encoders with rate 1/2 and code generator $g(D) = [1, 1 + D^2/1 + D + D^2]$. Information bits, $u_k$, are fed into the turbo encoder, where $u_k = 0$ or $u_k = 1$ with equal probability. The information bits are grouped into frames of 1022 bits for encoding. Two tail bits are added at the end of each frame to return RSC1 to the all-zero state, while leaving the state of RSC2 open. A random interleaver is used between the two constituent encoders. The two parity bit streams are alternately punctured [27] to increase the overall code rate to 1/2 before BPSK modulation.

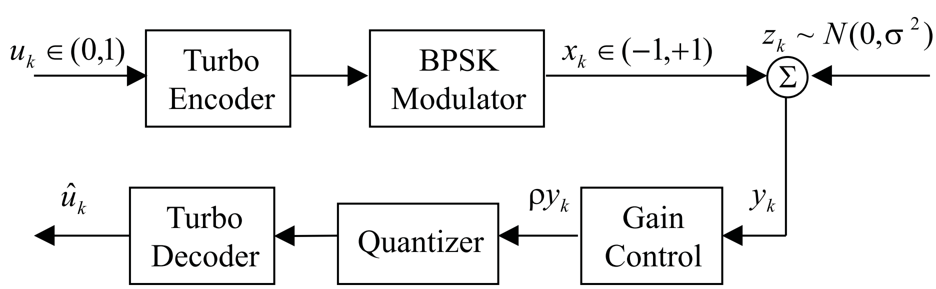



Figure 5.51: Turbo coding system model with quantization.

The coded bits are BPSK modulated into antipodal signals $(-1, +1)$ and transmitted through the channel. AWGN noise $z_k \sim \mathcal{N}(0, \sigma^2)$ is added to the transmitted signal by the channel. After the BPSK demodulation, the received symbols are scaled by a factor $\rho$ before they are digitized by the quantizer.

In the simulation of the quantization, the continuous signal within the voltage range ($V_{low}$, $V_{hi}$) is mapped to integer numbers between $(-2^{n_q-1}, 2^{n_q-1}-1)$, where $n_q$ is the number of bits of the quantizer's resolution [66] [67]. Usually $V_{low} = -V_{hi}$, and 0 is the center of a bin. The quantization is realized by dividing the region between the voltages $(V_{low}, V_{hi})$ into $2^{n_q}$ evenly spaced bins. These bins are numbered between $-2^{n_q-1}$ and $2^{n_q-1}-1$, inclusive. The bin width is $\delta = (V_{hi} - V_{low})/2^{n_q}$, and the bin boundaries are at

$$\{-\infty; V_{low} + \delta/2; \cdots; V_{low} + (2i-1)\delta/2; \cdots; V_{hi} - 3\delta/2; +\infty\}, \quad i = 1, ..., 2^{n_q} - 1.$$

For each continuous input sample, a search is performed to identify the bin in which the sample lies, and the corresponding integer number will be used as the quantized value of the input.

This section focuses on the number of quantization bits available. Thus, there was no attempt to optimize the number of bits available in the internal data path, which is studied later in Section 5.2. All the internal calculation was performed with signed integers which occupy at most 32 bits.

Two turbo decoding algorithms, Log-MAP and SOVA, are used to decode the quantized signal. In the Log-MAP algorithm, the computations are performed in the logarithmic domain, and the Jacobian logarithm is used as in Equation (3.86) [48].

$$\begin{aligned} \max{}^*(x_1, x_2) &= \ln(e^{x_1} + e^{x_2}) \\ &= \max(x_1, x_2) + f_c(|x_1 - x_2|) \end{aligned} \tag{5.175}$$

where $f_c(x) = \ln(1 + e^{-x})$ is a nonlinear correction function. All the other computations with Log-MAP are linear and can be performed in the same manner as with the floating point arithmetic, except $f_c(\cdot)$. Because of the use of fixed point arithmetic, the quantities $x_1$ and $x_2$ in the Log-MAP algorithm can be thought of as having been scaled from their floating point counterparts by a factor of approximately $2^{n_q}/(V_{hi} - V_{low})$. In our simulation, the floating point $f_c(\cdot)$ was implemented using the fixed point counterpart $f_c^q(\cdot)$. This necessitated the quantity $|x_1 - x_2|$ to be first scaled to $|x_1^q - x_2^q|$ as follows:

$$|x_1^q - x_2^q| = \frac{|x_1 - x_2|(V_{hi} - V_{low})}{2^{n_q}\,\rho}, \tag{5.176}$$

where $\rho$ is the scaling factor. The look-up table for $f_c^q(\cdot)$ was constructed by computing the nonlinear correction function using the value of $|x_1^q - x_2^q|$ as follows:

$$f_c^q(|x_1 - x_2|) = \left[\frac{2^{n_q}\,\rho\, f_c(|x_1^q - x_2^q|)}{V_{hi} - V_{low}}\right], \tag{5.177}$$

where $[\cdot]$ stands for the operation of rounding to the nearest integer.

In SOVA, all the computations are linear. Therefore, there is no need to adjust the operations of the algorithm to accommodate the employment of the fixed point quantities.

### 5.1.2 Optimal Gain

For the system in Figure 5.51, to achieve the best signal-to-distortion ratio at the quantizer, the amplitude of the signal is adjusted by the gain control. After being scaled by the gain, the input signal distribution to the quantizer will be either spread or compressed. For a given input distribution, there exists an optimal scaling factor which minimizes the distortion, i.e., maximizes the signal-to-distortion ratio, of the quantizer.

Here the optimal gain is derived for BPSK signals passing through an AWGN channel. Assuming the BPSK signal constellation is $(+1, -1)$ with equal probability, the pdf of the signal $x$ is

$$p_x(x) = 0.5[\delta(x+1) + \delta(x-1)]. \tag{5.178}$$

The Gaussian noise $z$ has zero mean, $\sigma^2$ variance and pdf

$$p_z(z) = \frac{1}{\sqrt{2\pi}\sigma}\exp\left(-\frac{z^2}{2\sigma^2}\right). \tag{5.179}$$

The received signal $y$ is the summation of the pure signal and the additive channel noise: $y = x + z$. Assuming that $x$ and $z$ are independent, the pdf of $y$ is the convolution of $p_x(x)$ and $p_z(z)$, or

$$\begin{aligned} p_y(y) &= \int_{-\infty}^{\infty} p_z(y-t)p_x(t)dt \\ &= \frac{1}{2\sqrt{2\pi}\sigma}\left(\exp\left(-\frac{(y-1)^2}{2\sigma^2}\right) + \exp\left(-\frac{(y+1)^2}{2\sigma^2}\right)\right). \end{aligned} \tag{5.180}$$

After being scaled by a factor $\rho$, the signal becomes $\nu = \rho y$. $\rho$ is assumed to be constant during transmission of a frame. The scaled signal $\nu$ has pdf

$$\begin{aligned} p_\nu(\nu) &= \frac{1}{|\rho|} p_y\left(\frac{\nu}{\rho}\right) \qquad\qquad (\rho > 0) \qquad (5.181) \\ &= \frac{1}{2\sqrt{2\pi}\rho\sigma}\left(\exp\left(-\frac{(\nu-\rho)^2}{2\rho^2\sigma^2}\right) + \exp\left(-\frac{(\nu+\rho)^2}{2\rho^2\sigma^2}\right)\right). \end{aligned}$$

Or, $\nu$'s distribution is equivalent to the summation of two Gaussian distributions: $N(\rho, \rho\sigma)$ and $N(-\rho, \rho\sigma)$.

Assume the quantization levels are $\tilde{\nu}_k$, and the quantization boundaries are $(\nu_{k-1}, \nu_k)$, where $k \in \{1, ..., L\}$, $L = 2^{n_q}$ is the number of quantization levels. The distortion function is thus

$$D = \sum_{k=1}^{L} \int_{\nu_{k-1}}^{\nu_k} (\nu - \tilde{\nu}_k)^2 p_\nu(\nu) d\nu = \mathcal{A} + \sum_{k=1}^{L} \left[-2\tilde{\nu}_k \mathcal{B}_k + \tilde{\nu}_k^2 \mathcal{C}_k\right], \qquad (5.182)$$

where $\mathcal{A} = \int_{-\infty}^{\infty} \nu^2 p_\nu(\nu) d\nu$, $\mathcal{B}_k = \int_{\nu_{k-1}}^{\nu_k} \nu p_\nu(\nu) d\nu$, $\mathcal{C}_k = \int_{\nu_{k-1}}^{\nu_k} p_\nu(\nu) d\nu$, and $p_\nu(\nu)$ is expressed in Equation (5.181). The following relations are derived, where $Q(\cdot)$ is the commonly used $Q$ function defined in Equation (2.26):

$$\begin{aligned} \mathcal{A} &= \rho^2(\sigma^2 + 1) \qquad (5.183) \\ \mathcal{B}_k &= \frac{\rho\sigma}{2\sqrt{2\pi}}\left[\exp\left(\frac{-(\nu_{k-1}-\rho)^2}{2\rho^2\sigma^2}\right) - \exp\left(\frac{-(\nu_k-\rho)^2}{2\rho^2\sigma^2}\right)\right. \\ &\quad \left. + \exp\left(\frac{-(\nu_{k-1}+\rho)^2}{2\rho^2\sigma^2}\right) - \exp\left(\frac{-(\nu_k+\rho)^2}{2\rho^2\sigma^2}\right)\right] \\ &\quad + \frac{\rho}{2}\left[Q\left(\frac{\nu_{k-1}-\rho}{\rho\sigma}\right) - Q\left(\frac{\nu_k-\rho}{\rho\sigma}\right) - Q\left(\frac{\nu_{k-1}+\rho}{\rho\sigma}\right) + Q\left(\frac{\nu_k+\rho}{\rho\sigma}\right)\right] \\ &\qquad (5.184) \\ \mathcal{C}_k &= \frac{1}{2}\left[Q\left(\frac{\nu_{k-1}-\rho}{\rho\sigma}\right) - Q\left(\frac{\nu_k-\rho}{\rho\sigma}\right) + Q\left(\frac{\nu_{k-1}+\rho}{\rho\sigma}\right) - Q\left(\frac{\nu_k+\rho}{\rho\sigma}\right)\right] (5.185) \end{aligned}$$

From the above equations it is obvious that the signal power is equal to $\mathcal{A}$,

$$P = \int_{-\infty}^{\infty} \nu^2 p_\nu(\nu) d\nu = \mathcal{A} = \rho^2(\sigma^2 + 1). \qquad (5.186)$$

And the signal-to-distortion ratio is

$$\frac{P}{D} = \frac{\mathcal{A}}{\mathcal{A} + \sum_{k=1}^{L}\left[-2\tilde{\nu}_k \mathcal{B}_k + \tilde{\nu}_k^2 \mathcal{C}_k\right]}. \qquad (5.187)$$

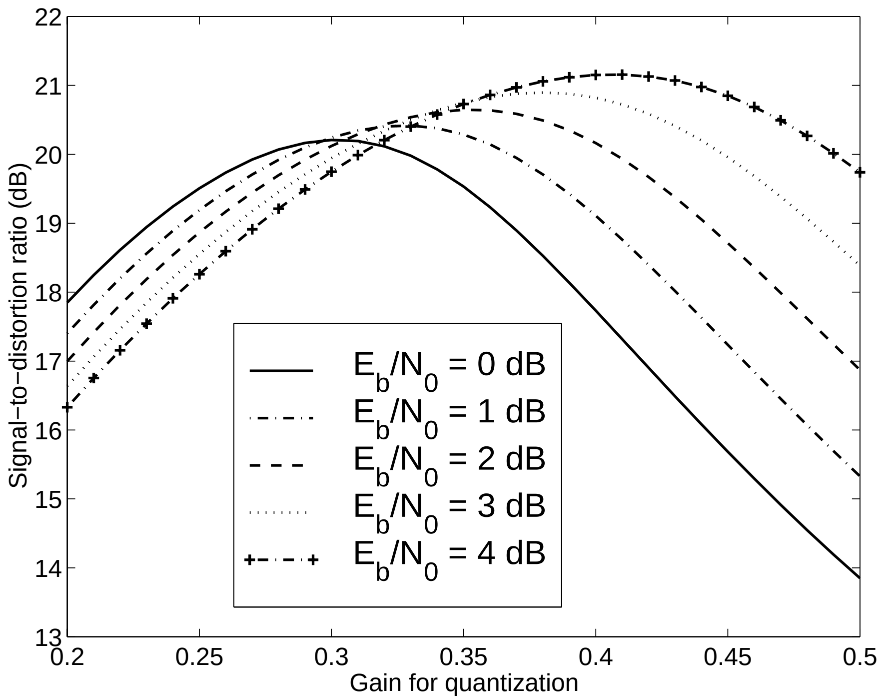



Figure 5.52: The signal-to-distortion ratio for a four-bit, range $(-1, 1)$ quantizer.

To find the optimal scaling factor, first the curve of signal-to-distortion ratio versus $\rho$ is plotted. Then the optimal $\rho$ is the value corresponding to the maximum signal-to-distortion ratio.

The signal-to-distortion ratio plot for a four-bit, range $(-1, 1)$ quantizer is shown in Figure 5.52. The curves of the optimal gain when the quantizer range is $(-1, 1)$ are plotted in Figure 5.53. Let the optimal gain for quantizer range $(-1, 1)$ be $\rho_1$ at a given $E_b/N_0$. When the dynamic range of the quantizer is $(-v, v)$, the optimal gain is $\rho_v = v\rho_1$.

Based on the above studies, the following conclusions are made for the scaling factor of an $n_q$-bit quantizer:

- There is an optimal gain by which to scale the received signal before it is fed into the quantizer. When the gain is too small, quantization noise will distort the signal severely; when the gain is too large, saturation will occur. Both cases result in the undesirable degradation of the signal-to-distortion ratio.

- Figure 5.52 shows that larger values of $n_q$ allow for higher signal-to-distortion ratio. However, with proper scaling, a lower resolution quantizer can yield a

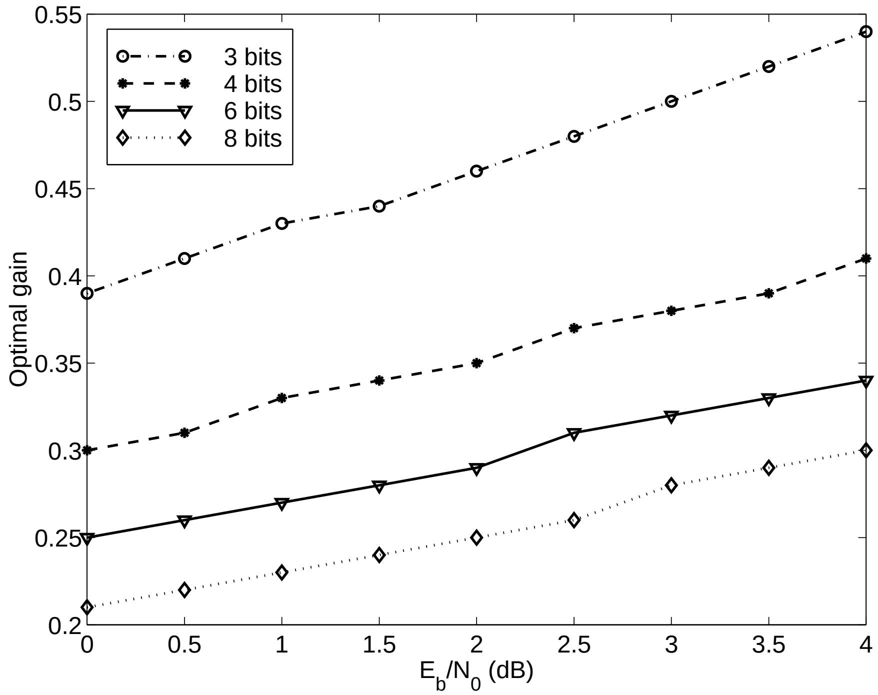



Figure 5.53: Optimal scaling factor for $n_q$-bit, range $(-1, 1)$ quantizer.

higher signal-to-distortion ratio than that of a higher resolution quantizer, if the input of the latter is not properly scaled.

- Figure 5.53 shows that, for a given $n_q$, the optimal gain increases approximately linearly with the $E_b/N_0$ (dB) of the channel.
- When the $E_b/N_0$ is given, a higher resolution (larger $n_q$) quantizer needs a smaller gain to maximize the signal-to-distortion ratio, as shown in Figure 5.53.

### 5.1.3 Simulation Results

The noise affecting the fixed point arithmetic decoding procedure is composed of two parts: the quantization error of the quantizer ahead of the decoder, and the accumulated errors resulting from the rounding or truncation of multiplication products inside the decoder. A higher signal-to-noise ratio is achieved with more quantization bits. However, to save power and memory and to increase processing speed, a smaller number of bits is desirable. Rounding error at each step propagates through the whole decoding process. Since the decoder is a complicated system with feedback, analysis of rounding error effect is hard to derive analytically. Instead, simulation was used to study the effect.

In the simulations, three different quantizer resolutions, $n_q = 3, 4$ and 8-bits, were examined. Perfect knowledge of the channel was assumed at the decoder. Two quantizer ranges were considered: $(-8, 8)$ and $(-0.5, 0.5)$. The range $(-0.5, 0.5)$ was chosen to illustrate the overflow problem with the quantizer, and the range $(-8, 8)$ was chosen to show the rounding error problem. The optimal gain obtained in Section 5.1.2 was used in contrast to both the case without scaling and the case using floating point arithmetic. After eight decoding iterations, the decoder estimates $\hat{u}_k$ were compared with the information bits $u_k$ to determine BER.

In Figures 5.54 through 5.57, the curves of BER versus $E_b/N_0$ are plotted. Figures 5.54 and 5.55 are for a quantizer of range $(-0.5, 0.5)$. Figures 5.56 and 5.57 are for a quantizer of range $(-8, 8)$.

The following observations are made from these plots:

- The effects of quantization are more evident at higher $E_b/N_0$, and the effects of AWGN tend to dominate at lower $E_b/N_0$. This is concluded because all curves with optimal gain tend to converge at lower $E_b/N_0$.

- As expected, higher $n_q$ provides better performance for both decoding algorithms when the distribution of the signal is fixed. The most significant performance improvement was observed when the fixed point error was mainly contributed by rounding error (no scaling with quantizer having a range of $(-8, 8)$). However, in all cases, the improvement followed a law of diminishing returns, since the difference between four-bit quantization and eight-bit quantization was small, and the difference between eight-bit quantization and floating point was negligible. Thus, no more than eight-bit quantization is required for accurate decoding of turbo codes.

- In cases of severe overflow (no scaling with quantizer range $(-0.5, 0.5)$), both decoding algorithms exhibited marked inability to correct errors. However, the performance of SOVA did improve by increasing either $n_q$ or $E_b/N_0$, in contrast to the Log-MAP algorithm. This phenomenon suggests that the SOVA algorithm is more computationally stable than the Log-MAP algorithm.

- With range $(-8, 8)$, the BER for three-bit quantization with optimal gain control was lower than that of four-bit quantization without gain control. This indicates that for a low resolution quantizer, it was crucial to adjust the gain so that the received signals fit in the dynamic range of the quantizer properly.

- For all cases where the received signals were optimally scaled, the coding gain of the Log-MAP algorithm is about 0.5 dB greater than that of the SOVA algorithm for the same value of $n_q$ in the $E_b/N_0 > 1$ dB region.

In summary, scaling the received signal by an optimal gain leads to excellent turbo decoder performance, even with very few bits ($n_q = 4$). SOVA involves no extra computation when using fixed point arithmetic because of its linearity. SOVA is also more computationally stable than Log-MAP. However, Log-MAP generally performs better than SOVA except when the overflow problem is severe. Overflow impairs the performance more than the rounding noise, and it cannot be solved by increasing the number of quantization bits. Adjusting the received signal to fit it into the full scale range of the quantizer is important.

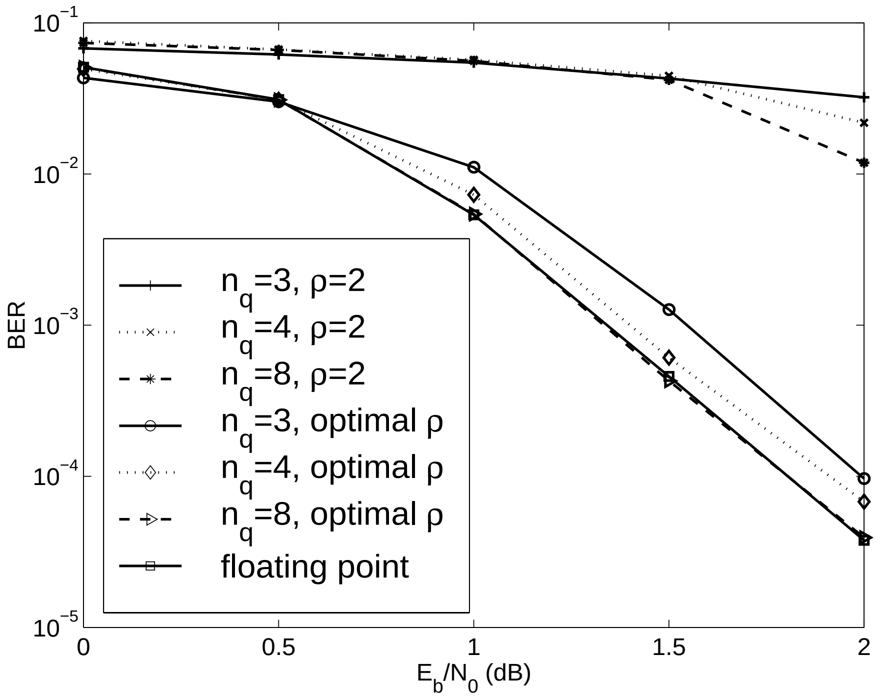



Figure 5.54: BER using Log-MAP for various resolution quantizers with range $(-0.5, 0.5)$.

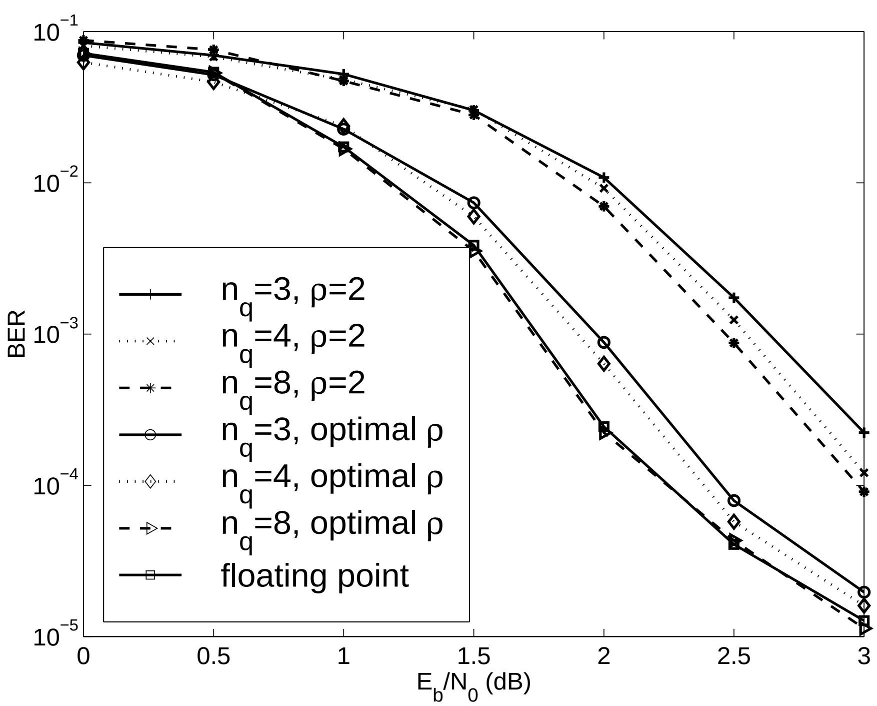



Figure 5.55: BER using SOVA for various resolution quantizers with range $(-0.5, 0.5)$.

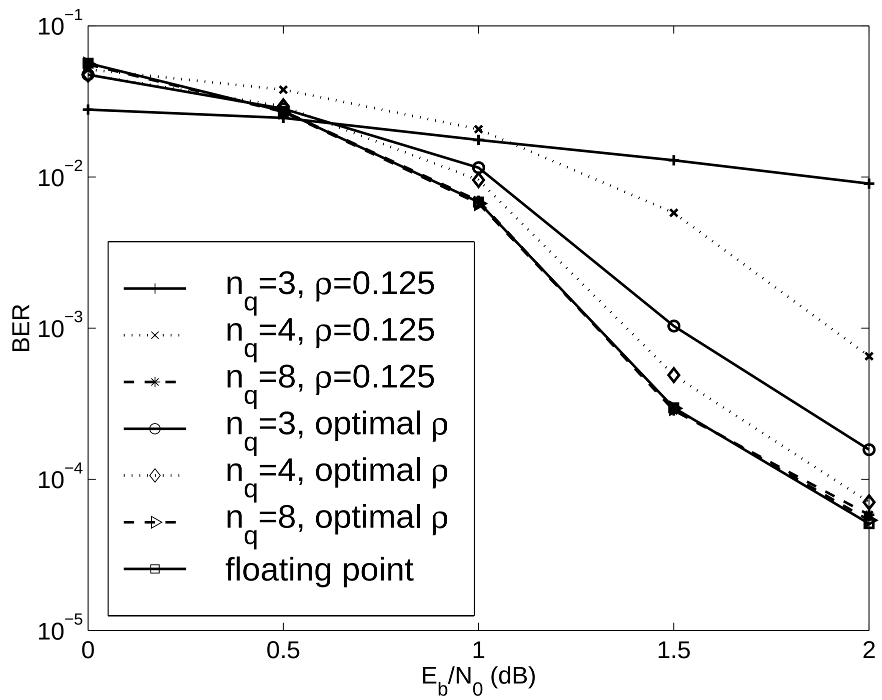



Figure 5.56: BER using Log-MAP for various resolution quantizers with range $(-8, 8)$.

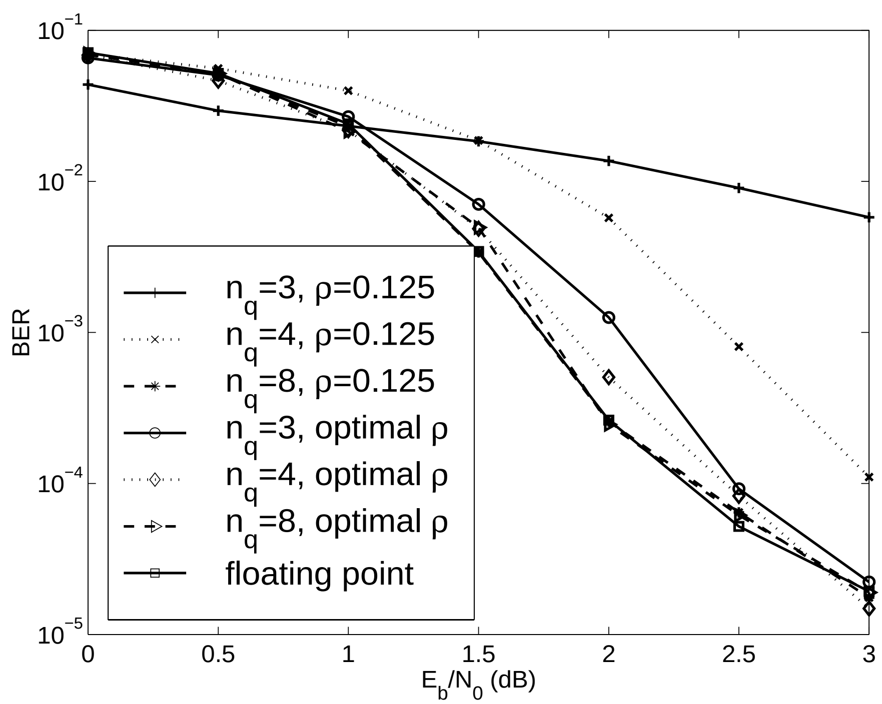



Figure 5.57: BER using SOVA for various resolution quantizers with range $(-8, 8)$.

# 5.2 Internal Data Width Requirements and Modular Renormalization for SISO Modules

For the implementation of a SISO module using fixed point arithmetic, it is important to reduce the computational complexity and minimize the data width for representation and calculation. In Section 5.1, it was shown that for most cases of interest, a precision of eight bits was sufficient for quantization of the received signals. This section examines the internal computation requirements of a SISO decoding module. The minimum internal data width is determined following the derivation of upper bounds on path metric differences and the output LLR of a SISO module. In addition, the modular renormalization method is shown to be applicable to the SISO module, as alluded to in [68]. This method, which has been applied to the implementation of Viterbi decoders [69], takes advantage of the properties of two's complement arithmetic to accommodate the overflow of the path metrics. In comparison to the subtractive normalization technique, this method reduces the hardware complexity and improves the efficiency of the metric update.

## 5.2.1 Review of Operations in a SISO Module

The structure of an iterative PCCC decoder has been illustrated in Figure 3.18. Two SISO modules, SISO1 and SISO2, decode two constituent RSC codes respectively, and exchange soft information between each other [70]. For a constituent code with one input and $n_0$ outputs, the SISO module has two input vectors: $[\lambda(u; I)]$, which are the LLRs for the information bits $u$, and $\left[\lambda(c^{(1)}; I), ..., \lambda(c^{(n_0)}; I)\right]$, which are the LLRs measured from the channel for the codeword bits $c^{(1)}$ through $c^{(n_0)}$, respectively. After the decoding process, an output vector $[\lambda(u; O)]$ is produced. For an AWGN channel with noise distribution $\mathcal{N}(0, \sigma^2)$, $\lambda_k(c^{(i)}; I) = \frac{2A}{\sigma^2} y_k^{(i)}$, for $1 \leq i \leq n_0$ and $1 \leq k \leq N$, where $A$ is the signal amplitude. For the first iteration of SISO1, $\lambda(u; I)$ are initialized to zero. For subsequent calculations, $\lambda(u; I)$ take the values of $\lambda(u; O)$ out of the previous decoding stage.

The computation inside the SISO module is composed of the following three parts [70] [48] as discussed in Section 3.5.5:

- The forward recursion to obtain forward path metrics $\alpha_k(s)$:

$$\alpha_k(s) = \max_{e: s_k^E(e) = s}^{*} \left[\alpha_{k-1}(s_k^S(e)) + \gamma_k(e)\right], \tag{5.188}$$

where $1 \leq k \leq N-1$, $0 \leq s \leq 2^m - 1$, $s_k^S(e)$ and $s_k^E(e)$ are the starting and ending states of edge $e$ at time $k$, respectively. Here $\max^*(\cdot)$ is defined as in Equation (3.86). The branch metric at time $k$ is

$$\gamma_k(e) = u(e)\lambda_k(u; I) + \sum_{i=1}^{n_0} c^{(i)}(e)\lambda_k(c^{(i)}; I), \tag{5.189}$$

where $u(e)$ and $(c^{(1)}(e), ..., c^{(n_0)}(e))$ are the information bit and the codeword of edge $e$, respectively, $u(e) \in \{0,1\}$, and $c^{(i)}(e) \in \{0,1\}$.

- The backward recursion to obtain backward path metrics $\beta_k(s)$:

$$\beta_k(s) = \max_{e:s_{k+1}^S(e)=s}{}^{*} \left[\beta_{k+1}(s_{k+1}^E(e)) + \gamma_{k+1}(e)\right]. \tag{5.190}$$

where $1 \leq k \leq N-1$ and $0 \leq s \leq 2^m - 1$.

- The calculation of output LLR $\lambda_k(u; O)$ using $\alpha_{k-1}(s)$ and $\beta_k(s)$:

$$\lambda_k(u; O) = \max_{e:u(e)=1}{}^{*} \left[\Gamma_k(e)\right] - \max_{e:u(e)=0}{}^{*} \left[\Gamma_k(e)\right], \tag{5.191}$$

where $1 \leq k \leq N$ and

$$\Gamma_k(e) = \alpha_{k-1}(s_k^S(e)) + \sum_{i=1}^{n_0} c^{(i)}(e)\lambda_k(c^{(i)}; I) + \beta_k(s_k^E(e)). \tag{5.192}$$

### 5.2.2 Bounds on $\Delta\alpha_k$, $\Delta\beta_k$, and $|\lambda_k(u; O)|$

In order to determine the internal data width requirement for implementation of the SISO module, this section derives bounds for the dynamic range of the difference between the forward path metrics, the difference between the backward path metrics, and the output LLR $\lambda_k(u; O)$.

#### Bounds on $\Delta\alpha_k$

First the dynamic range of the difference between the forward path metrics is considered. Let $\Delta\alpha_k$ be the maximum absolute difference between forward path metrics $\alpha_k(s)$ at time $k$, i.e., $\Delta\alpha_k = \max_{s_1 \neq s_2} |\alpha_k(s_1) - \alpha_k(s_2)|$, where $s_1$ and $s_2$ are indices of two arbitrary eligible states, $s_1, s_2 \in \{0, ..., 2^m - 1\}, k \in \{1, ..., N-1\}$, and $N$ is the frame size.

**Proposition 1** $\Delta\alpha_k$ *is bounded by* $B_{\alpha,k}$*, or*

$$\Delta\alpha_k \leq B_{\alpha,k}, \tag{5.193}$$

*where*

$$B_{\alpha,k} = \sum_{j=\max(1,k-m+1)}^{k} \left( |\lambda_j(u;I)| + \sum_{i=1}^{n_0} |\lambda_j(c^{(i)};I)| \right).$$

*A looser bound common to all* $k \in \{1, ..., N-1\}$ *is*

$$\Delta\alpha_k \leq B_\alpha, \tag{5.194}$$

*where*

$$B_\alpha = m \left( \max_k |\lambda_k(u;I)| + n_0 \times \max_{k,i} |\lambda_k(c^{(i)};I)| \right). \tag{5.195}$$

*Proof*: At each time $k$, the forward path metric $\alpha_k(s)$ is updated with Equation (5.188), and the branch metric $\gamma_k(e)$ is computed with Equation (5.189). Let the maximum absolute difference between branch metrics at time $k$ be $\Delta\gamma_k = \max_{e_1,e_2} |\gamma_k(e_1) - \gamma_k(e_2)|$, where $e_1$ and $e_2$ are two arbitrary edges. Based on Equation (5.189), $\Delta\gamma_k$ satisfies

$$\Delta\gamma_k \leq |\lambda_k(u;I)| + \sum_{i=1}^{n_0} |\lambda_k(c^{(i)};I)|. \tag{5.196}$$

Here, for convenient derivation, the logarithmic domain is temporarily abandoned. Corresponding to $\alpha_k(s)$ and $\gamma_k(e)$ in logarithmic domain, $A_k(s)$ and $M_k(e)$ are the metrics in the linear domain as discussed in Section 3.3.2, where

$$A_k(s) \quad = \exp(\alpha_k(s)), \tag{5.197}$$

$$M_k(e) \quad = \exp(\gamma_k(e)). \tag{5.198}$$

Thus Equation (5.188) can be rewritten as

$$A_k(s) = \sum_{e:s_k^E(e)=s} A_{k-1}(s_k^S(e)) \cdot M_k(e). \tag{5.199}$$

For $k \geq 2m$, when $A_j(s)$ is recursively expressed as a function of $A_{j-1}(s)$, $j = k, k-1, ..., k-m+1$, each $A_k(s)$ is a function of all $A_{k-m}(s')$, $s' \in \{0, ..., 2^m - 1\}$:

$$A_k(s) = \sum_{s'} A_{k-m}(s') \cdot M_{k-m+1}(e_{k-m}^{k-m+1}) M_{k-m+2}(e_{k-m+1}^{k-m+2}) \cdots M_k(e_{k-1}^{k}), \tag{5.200}$$

where $e_{k_1}^{k_1+1}$ denotes an edge that links a state at time $k_1$ to a state at time $k_1 + 1$. In Equation (5.200), for every $s' \in \{0, ..., 2^m - 1\}$, the consecutive edges associated with $A_{k-m}(s')$ and $A_k(s)$ are those that link state $s'$ at time $(k-m)$ to state $s$ at time $k$.

Equation (5.200) is true because all states at time $k$ have paths to all states at time $(k-m)$ for a memory-$m$ RSC encoder, as illustrated in Figure 5.58. This can be proven using the butterfly structure discussed in Section 3.2.

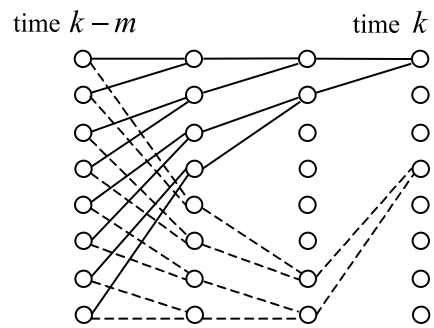



Figure 5.58: A trellis section of a memory-3 RSC encoder.

By replacing $M_j(e_{k_1}^{k_1+1})$ by $\max_e M_j(e)$ or $\min_e M_j(e)$ in Equation (5.200), $\forall j \in \{k-m+1,...,k\}$, the following bound on $A_k(s)$ is obtained:

$$\sum_{s'} A_{k-m}(s') \prod_{j=k-m+1}^{k} \min_e(M_j(e)) \leq A_k(s) \leq \sum_{s'} A_{k-m}(s') \prod_{j=k-m+1}^{k} \max_e(M_j(e)). \tag{5.201}$$

Considering Equations (5.196) and (5.198), when $M_j(e_1) = \max_e M_j(e)$, $M_j(e_2) = \min_e M_j(e)$, we have

$$\begin{aligned} \frac{\max_e M_j(e)}{\min_e M_j(e)} &= \exp(\gamma_j(e_1) - \gamma_j(e_2)) \\ &\leq \exp\left(|\lambda_j(u;I)| + \sum_{i=1}^{n_0} |\lambda_j(c^{(i)};I)|\right). \qquad (5.202) \end{aligned}$$

With Equations (5.198), (5.201), and (5.202), for any two states $s_1$ and $s_2$ at time $k$,

$$\begin{aligned} \alpha_k(s_1) - \alpha_k(s_2) &= \ln \frac{A_k(s_1)}{A_k(s_2)} \qquad (5.203) \\ &\leq \ln \frac{\prod_{j=k-m+1}^{k} \max_e(M_j(e))}{\prod_{j=k-m+1}^{k} \min_e(M_j(e))} \\ &\leq \sum_{j=k-m+1}^{k} \left(|\lambda_j(u;I)| + \sum_{i=1}^{n_0} |\lambda_j(c^{(i)};I)|\right). \qquad (5.204) \end{aligned}$$

For $1 \leq k < 2m$, eligible states $s_1$ and $s_2$ at time $k$ have paths only to a subset of all the states at time $\max(1, k-m+1)$. Nevertheless, the bound in Equation (5.204) still holds with a modification on the lower index of the summation as follows:

$$\alpha_k(s_1) - \alpha_k(s_2) \leq \sum_{j=\max(1,k-m+1)}^{k} \left( |\lambda_j(u;I)| + \sum_{i=1}^{n_0} |\lambda_j(c^{(i)};I)| \right). \tag{5.205}$$

Combining Equations (5.204) and (5.205), and considering that $s_1$ and $s_2$ are both arbitrary states, the bound expressed in Equation (5.193) follows. By replacing $|\lambda_k(u;I)|$ and $|\lambda_k(c^{(i)};I)|$ in Equation (5.193) with $\max_k |\lambda_k(u;I)|$ and $\max_{k,i} |\lambda_k(c^{(i)};I)|$, respectively, Equation (5.194) is obtained. □

**Bounds on $\Delta\beta_k$**

A similar argument leads to a bound on the maximum difference between backward path metrics. Here the key difference between this bound and the bound on $\Delta\alpha_k$ in Section 5.2.2 is highlighted. Let $\Delta\beta_k$ be the maximum absolute difference between backward path metrics $\beta_k(s)$ at time $k$, i.e., $\Delta\beta_k = \max_{s_1 \neq s_2} |\beta_k(s_1) - \beta_k(s_2)|$, where $s_1$ and $s_2$ are indices of two arbitrary eligible states, $s_1, s_2 \in \{0, ..., 2^m - 1\}, k \in \{1, .., N-1\}$, and $N$ is the frame size.

**Proposition 2** *$\Delta\beta_k$ is bounded by $B_{\beta,k}$, or*

$$\Delta\beta_k \leq B_{\beta,k}, \tag{5.206}$$

*where*

$$B_{\beta,k} = \sum_{j=k+1}^{\min(N,k+m)} \left( |\lambda_j(u;I) + \sum_{i=0}^{n_0} |\lambda_j(c^{(i)};I)| \right).$$

*A looser bound common to all $k \in \{1, ..., N-1\}$ is*

$$\Delta\beta_k \leq B_\alpha, \tag{5.207}$$

*where $B_\alpha$ is the bound on $\Delta\alpha_k$ expressed in Equation (5.195).*

*Proof*: The recursion of the backward path metric $\beta_k(s)$ is expressed in Equation (5.190). The computation of $\beta_k(s)$ proceeds in the same way as that of $\alpha_k(s)$ in Equation (5.188), with only the direction reversed. Thus Equations (5.206) and (5.207) are obtained analogously to Equations (5.193) and (5.194). □

**Bounds on** $|\lambda_k(u;O)|$

Combining the bounds on $\Delta\alpha_k$ and $\Delta\beta_k$ in Sections 5.2.2 and 5.2.2, respectively, leads to a bound on the dynamic range of the output LLR $\lambda_k(u;O)$.

**Proposition 3** *The output $\lambda_k(u;O)$ is bounded by $B_{\lambda,k}$, or*

$$|\lambda_k(u;O)| \leq B_{\lambda,k}, \tag{5.208}$$

*where*

$$B_{\lambda,k} = \min\left[\Delta\alpha_{k-1}, \Delta\beta_k\right] + \sum_{i=1}^{n_0} |\lambda_k(c^{(i)};I)|.$$

*A looser bound applicable to all $k \in \{1, ..., N\}$ is*

$$|\lambda_k(u;O)| \leq B_{\lambda}, \tag{5.209}$$

*where*

$$B_{\lambda} = m \times \max_k |\lambda_k(u;I)| + (m+1)n_0 \times \max_{k,i} |\lambda_k(c^{(i)};I)|. \tag{5.210}$$

*Proof*: The output LLR $\lambda_k(u;O)$ for the information bit at time $k$ is calculated from Equation (5.191). Corresponding to $\alpha_k(s)$, $\beta_k(s)$, and $\sum_{i=1}^{n_0} c^{(i)}(e)\lambda_k(c^{(i)};I)$ in logarithmic domain, $A_k(s)$, $B_k(s)$, and $\bar{M}_k(e)$ are defined in the linear domain, where

$$\begin{aligned} A_k(s) &= \exp(\alpha_k(s)), \\ B_k(s) &= \exp(\beta_k(s)), \\ \bar{M}_k(e) &= \exp\left(\sum_{i=1}^{n_0} c^{(i)}(e)\lambda_k(c^{(i)};I)\right). \end{aligned} \tag{5.211}$$

Since $c^{(i)}(e) \in \{0,1\}$,

$$\ln \frac{\max_e \bar{M}_k(e)}{\min_e \bar{M}_k(e)} \leq \sum_{i=1}^{n_0} |\lambda_k(c^{(i)};I)|. \tag{5.212}$$

Rewriting Equation (5.191) using $A_k(s)$, $B_k(s)$, and $\bar{M}_k(e)$, we have

$$\lambda_k(u;O) = \ln \frac{\sum_{e:u(e)=1} A_{k-1}(s_k^S(e))\bar{M}_k(e)B_k(s_k^E(e))}{\sum_{e:u(e)=0} A_{k-1}(s_k^S(e))\bar{M}_k(e)B_k(s_k^E(e))}. \tag{5.213}$$

Replacing $A_{k-1}(s_k^S(e))$ and $\bar{M}_k(e)$ by $\max_s A_{k-1}(s)$ and $\max_e \bar{M}_k(e)$, respectively, in the numerator of Equation (5.213), and by $\min_s A_{k-1}(s)$ and $\min_e \bar{M}_k(e)$, respectively, in the denominator, and using Equation (5.212), we obtain

$$\begin{aligned} \lambda_k(u;O) &\leq \ln \frac{\max_s A_{k-1}(s) \max_e \bar{M}_k(e) \sum_s B_k(s)}{\min_s A_{k-1}(s) \min_e \bar{M}_k(e) \sum_s B_k(s)} \\ &\leq \Delta\alpha_{k-1} + \sum_{i=1}^{n_0} |\lambda_k(c^{(i)};I)|. \end{aligned} \tag{5.214}$$

Similarly, by replacing $A_{k-1}(s_k^S(e))$ and $\bar{M}_k(e)$ by $\min_s A_{k-1}(s)$ and $\min_e \bar{M}_k(e)$, respectively, in the numerator of Equation (5.213), and by $\max_s A_{k-1}(s)$ and $\max_e \bar{M}_k(e)$, respectively, in the denominator, we have

$$\begin{aligned}\lambda_k(u;O) &\geq \ln \frac{\min_s A_{k-1}(s) \min_e \bar{M}_k(e) \sum_s B_k(s)}{\max_s A_{k-1}(s) \max_e \bar{M}_k(e) \sum_s B_k(s)} \\ &\geq -\Delta\alpha_{k-1} - \sum_{i=1}^{n_0} |\lambda_k(c^{(i)};I)| \qquad (5.215)\end{aligned}$$

Combining Equations (5.214) and (5.215), we have

$$|\lambda_k(u;O)| \leq \Delta\alpha_{k-1} + \sum_{i=1}^{n_0} |\lambda_k(c^{(i)};I)| \qquad (5.216)$$

By the same reasoning,

$$|\lambda_k(u;O)| \leq \Delta\beta_k + \sum_{i=1}^{n_0} |\lambda_k(c^{(i)};I)| \qquad (5.217)$$

Since both Equations (5.216) and (5.217) are true, the bound in Equation (5.208) is obtained. With Equations (5.194) and (5.207), the bound in Equation (5.208) can be loosened, leading to Equation (5.209),

$$\begin{aligned}|\lambda_k(u;O)| &\leq B_\alpha + \sum_{i=1}^{n_0} |\lambda_k(c^{(i)};I)| \\ &\leq B_\alpha + n_0 \times \max_{k,i} |\lambda_k(c^{(i)};I)| \\ &= B_\lambda \qquad (5.218)\end{aligned}$$

□

### 5.2.3 Modular Renormalization and Determination of Internal Data Width

Based on the bounds of $\Delta\alpha_k$, $\Delta\beta_k$, and $|\lambda_k(u;O)|$ found in the previous section, the minimum internal data width for a SISO module can be derived. This section first proves that modular renormalization of the path metrics produces identical results as subtractive normalization of the path metrics when two's complement arithmetic is used. Then the resulting internal data width requirement is determined.

### Modular Renormalization

In a SISO decoding module, both $\alpha_k(s)$ and $\beta_k(s)$ grow without bound as the recursion proceeds. Without normalization, detrimental overflow would occur in a hardware implementation. Since, as later shown in the proof of Proposition 4, the soft output is only affected by the difference between path metrics, but not their absolute values, a common practice is to normalize $\alpha_k(s)$ and $\beta_k(s)$ by subtracting a constant from all the metrics for a given $k$. Usually, the maximum metric at time $k$ is determined, and the subtractive normalization is done according to Equations (3.101) and (3.102),

$$\left.\begin{aligned}\tilde{\alpha}_k(s) &= (\alpha_k(s) - \max\nolimits_{s'} \alpha_k(s')) \mod 2^n,\\ \tilde{\beta}_k(s) &= (\beta_k(s) - \max\nolimits_{s'} \beta_k(s')) \mod 2^n,\end{aligned}\right\} \quad \forall s, \tag{5.219}$$

where $\tilde{\alpha}_k(s)$ and $\tilde{\beta}_k(s)$ are the subtractively normalized metrics. In the following it is shown that subtractive normalization is unnecessary when two's complement arithmetic is used.

In a common realization of two's complement arithmetic with $n$ bits, both addition and subtraction are defined modulo $2^n$. The "mod $2^n$" operation reduces any integer to an element of the set

$$F_n = \left\{-2^{n-1}, -2^{n-1}+1, \cdots, 2^{n-1}-1\right\} \tag{5.220}$$

by a periodic extension of $F_n$ over the set of integers. In practice, this mapping is accomplished by only considering the signed value of the $n$ least significant bits of any integer. The following property holds true for two's complement arithmetic.

**Property 1** *With two's complement representation and arithmetic, the difference between any two integers $I_x$ and $I_y$ stays intact when both are shifted by $h_C$, where $h_C$ is an arbitrary integer, or*

$$\begin{aligned}&(I_x \bmod 2^n - I_y \bmod 2^n) \bmod 2^n \qquad (5.221)\\ =\;& ((I_x - h_C) \bmod 2^n - (I_y - h_C) \bmod 2^n) \bmod 2^n.\end{aligned}$$

According to this property,

$$\begin{aligned}(\bar{\alpha}_k(s_1) - \bar{\alpha}_k(s_2)) \bmod 2^n &= (\tilde{\alpha}_k(s_1) - \tilde{\alpha}_k(s_2)) \bmod 2^n\\ (\bar{\beta}_k(s_1) - \bar{\beta}_k(s_2)) \bmod 2^n &= (\tilde{\beta}_k(s_1) - \tilde{\beta}_k(s_2)) \bmod 2^n\end{aligned}$$

for any two states $s_1$ and $s_2$, where $\bar{\alpha}_k(s)$ and $\bar{\beta}_k(s)$ are automatically converted from $\alpha_k(s)$ and $\beta_k(s)$ by the overflow mechanism of two's complement arithmetic:

$$\left.\begin{aligned}\bar{\alpha}_k(s) &= \alpha_k(s) \mod 2^n, \\ \bar{\beta}_k(s) &= \beta_k(s) \mod 2^n,\end{aligned}\right. \quad \forall k, \forall s. \tag{5.222}$$

Since the soft output is related to only the difference between path metrics, $\bar{\alpha}_k(s)$ and $\bar{\beta}_k(s)$ produce the same soft output as $\tilde{\alpha}_k(s)$ and $\tilde{\beta}_k(s)$ for any given $n$. Therefore the subtractive normalization of path metrics can be replaced by modular renormalization.

### Minimum $n$

To determine the internal data width $n$ for representation of fixed point numbers in a SISO module, the following property of two's complement arithmetic is exploited.

**Property 2** *With $n$-bit two's complement representation and arithmetic, the difference between any two integers $I_x$ and $I_y$ remains correct, as long as [69]*

$$2^{n-1} - 1 \geq |I_x - I_y|. \tag{5.223}$$

Thus,

$$I_x - I_y = ((I_x \bmod 2^n) - (I_y \bmod 2^n)) \bmod 2^n$$

is valid as long as Equation (5.223) is satisfied.

**Proposition 4** *With two's complement representation and arithmetic, the data width $n$ inside a SISO module satisfies the following inequality:*

$$2^{n-1} - 1 \quad \geq \quad 2m \times \max_k |\lambda_k(u; I)| + (2m+1)n_0 \times \max_{k,i} |\lambda_k(c^{(i)}; I)|. \tag{5.224}$$

*Proof:* From the SISO computations, which are composed of Equations (5.188), (5.190), and (5.191), it is clear that $\alpha_k(s)$, $\beta_k(s)$, and $\lambda_k(u; O)$ are all dependent on the function $\max^*(\cdot)$ expressed in Equation (3.86). This function can be realized with

$$\begin{aligned}\max{}^*(x,y) &= \ln(e^x + e^y) \\ &= \ln(1 + \exp(-|x-y|)) + \max(x,y) \\ &= f_c(|x-y|) + \begin{cases} x, & \text{if } x - y \geq 0 \\ y, & \text{if } x - y < 0 \end{cases},\end{aligned} \tag{5.225}$$

where $f_c(|x-y|) = \ln(1+\exp(-|x-y|))$ is the correction function. The values of $\alpha_k(s)$ and $\beta_k(s)$ are thus determined by the difference between entries of the $\max^*(\cdot)$ function. The value of $\lambda_k(u;O)$ is determined by the difference between entries of $\max^*(\cdot)$ and the difference between the values of two $\max^*(\cdot)$ functions. Based on Equation (5.223), the differences can always be obtained correctly, as long as $(2^{n-1}-1)$ is greater than the absolute value of the difference between any minuend/subtrahend pair throughout SISO.

1. *Requirement from the recursion computation.* According to Equations (5.194) and (5.196), at any time $k$, the difference between any two entries of the $\max^*(\cdot)$ function in Equation (5.188) satisfies

$$\begin{aligned} & |(\alpha_{k-1}(s_k^S(e_1)) + \gamma_k(e_1)) - (\alpha_{k-1}(s_k^S(e_2)) + \gamma_k(e_2))| \\ \leq\ & \Delta\alpha_{k-1} + \sum_{i=1}^{n_0} |\lambda_k(c^{(i)}; I)| \\ \leq\ & B_\alpha + n_0 \times \max_{k,i} |\lambda_k(c^{(i)}; I)| \\ =\ & B_\lambda, \end{aligned}$$

   where $e_1$ and $e_2$ are two distinct edges that reach a common state at time $k$. Therefore, according to Equation (5.223), the $\alpha_k(s)$ computation requires

$$2^{n-1} - 1 \geq B_\lambda. \tag{5.226}$$

   Since $B_\lambda > \Delta\alpha_k(s)$, Equation (5.226) also guarantees the correctness of the difference between any two $\alpha_k(s)$ for a given $k$. Based on Equation (5.190), the same conclusion can be drawn concerning the computation of $\beta_k(s)$.

2. *Requirements from the computation of the soft output $\lambda_k(u;O)$.* In Equation (5.191), $\max^*(x,y)$ operates over $2^m$ branches of a fixed $u(e)$ for both $u(e)=0$ and $u(e)=1$. In both cases, the difference between any two entries defined in Equation (5.192) is bounded by the following:

$$
\begin{aligned}
& \max_{k,e_1,e_2,u(e_1)=u(e_2)} |\Gamma_k(e_1) - \Gamma_k(e_2)| \\
\leq \; & \max_k \Delta\alpha_k + \max_k \Delta\beta_k + \max_{k,e_1,e_2} \sum_{i=1}^{n_0} (c^{(i)}(e_1) - c^{(i)}(e_2))\lambda_k(c^{(i)}; I) \\
\leq \; & 2B_\alpha + n_0 \times \max_{k,i} |\lambda_k(c^{(i)}; I)|.
\end{aligned}
$$

Based on Equation (5.223), representation of $\max_{k,e_1,e_2,u(e_1)=u(e_2)} |\Gamma_k(e_1)-\Gamma_k(e_2)|$ thus requires

$$2^{n-1} - 1 \geq 2B_\alpha + n_0 \times \max_{k,i} |\lambda_k(c^{(i)}; I)|. \tag{5.227}$$

For the final subtraction of two $\max^*(\cdot)$ functions to obtain $\lambda_k(u; O)$, $(2^{n-1} - 1) \geq |\lambda_k(u; O)|, \forall k$, is required. Since $|\lambda_k(u; O)|$ is bounded by $B_\lambda$ as expressed in Equation (5.209),

$$2^{n-1} - 1 \geq B_\lambda. \tag{5.228}$$

must be satisfied.

3. *Requirement from the representation of the complete information.* At the output of the last decoding stage, the complete information $\lambda_k^A(u; O)$ for $u_k$ is calculated and hard decisions are made based on the sign of $\lambda_k^A(u; O)$. The formula for $\lambda_k^A(u; O)$ is

$$\lambda_k^A(u; O) = \lambda_k(u; O) + \lambda_k(u; I), \tag{5.229}$$

as in Equation (3.161). It is necessary that the complete information lies within the dynamic range of an $n$-bit representation. With Equations (5.229) and (5.209),

$$|\lambda_k^A| \leq B_\lambda + \max_k |\lambda_k(u; I)|, \quad \forall k.$$

Thus, to provide correct complete information at the final SISO stage,

$$2^{n-1} - 1 \geq B_\lambda + \max_k |\lambda_k(u; I)| \tag{5.230}$$

is required.

Ordering the requirements listed in Equations (5.226), (5.227), (5.228), and (5.230), we have

$$2B_\alpha + n_0 \times \max_{k,i} |\lambda_k(c^{(i)}; I)| > B_\lambda + \max_k |\lambda_k(u; I)| > B_\lambda,$$

where $B_\alpha$ and $B_\lambda$ are defined in Equations (5.195) and (5.210), respectively. Since the largest dynamic range has to be accommodated, it is necessary to choose an $n$ that satisfies

$$\begin{aligned} & 2^{n-1} - 1 \\ \geq\ & 2B_\alpha + n_0 \times \max_{k,i} |\lambda_k(c^{(i)}; I)| \\ =\ & 2m \times \max_k |\lambda_k(u; I)| + (2m+1)n_0 \times \max_{k,i} |\lambda_k(c^{(i)}; I)|. \end{aligned} \tag{5.231}$$

□

Using Equation (5.224), the relationship between $n$ and $n_q$ can be found, where $n_q$ is the number of bits used to quantize the received signal. Since $\lambda_k(c^{(i)}; I) = 4r\frac{E_b}{N_0}y_k^{(i)}$ and $|y_k^{(i)}| \leq 2^{n_q-1}$,

$$\max_{k,i} |\lambda_k(c^{(i)}; I)| \leq 4r\frac{E_b}{N_0} \times 2^{n_q-1}. \tag{5.232}$$

Since simulations show that clipping $|\lambda_k(u; I)|$ at a reasonable threshold does not affect the performance, a constraint can be added on $\max_k |\lambda_k(u; I)|$. A conservative threshold is empirically found to be

$$\max_k |\lambda_k(u; I)| \leq 2^5 \times \max_{k,i} |y_k^{(i)}| = 2^{n_q+4}. \tag{5.233}$$

Substituting Equations (5.232) and (5.233) into Equation (5.224), the minimum internal data width $n_{min}$ is found to be a function of $n_q$ as following:

$$n_{min} = \left\lceil n_q + \log_2 \left( 2^6 m + 4r\frac{E_b}{N_0}(2m+1)n_0 \right) \right\rceil \tag{5.234}$$

Here a small $2^{-n_q+1}$ term is neglected inside the logarithm.

### 5.2.4 Simulation Results

To validate the bounds and the modular renormalization technique presented above, simulations were performed with the PCCC specified in [71], where the code generator was $g(D) = [1, 1 + D + D^3/1 + D^2 + D^3]$ $(m = 3, n_0 = 2)$ and the code rate was 1/3. Both constituent RSC encoders were terminated with 3 tail bits. A spread interleaver [63] was used in the simulation. An non-fading AWGN channel was assumed.

#### Test bounds on $\Delta\alpha$, $\Delta\beta$, and $|\lambda_k(u; O)|$

Simulation results validated the bounds on the path metrics and the output LLR $\lambda_k(u; O)$ presented in Section 5.2.2. In Figure 5.59, the numerical estimates of pdfs of $\Delta\alpha_k/B_{\alpha,k}$, $\Delta\beta_k/B_{\beta,k}$ and $|\lambda_k(u;O)|/B_{\lambda,k}$ were plotted for a PCCC of frame size 1024, code rate $r$ =1/3, and $E_b/N_0 = 0$ dB. Each pdf was calculated based on $2.45 \times 10^5$ simulation data points. For higher $E_b/N_0$ and other frame sizes, the pdfs exhibited similar characteristics. The pdfs never extended beyond unity on the $X$ axis; therefore, $\Delta\alpha_k, \Delta\beta_k$, and $|\lambda_k(u;O)|$ are indeed bounded by $B_{\alpha,k}, B_{\beta,k}$, and $B_{\lambda,k}$ respectively.

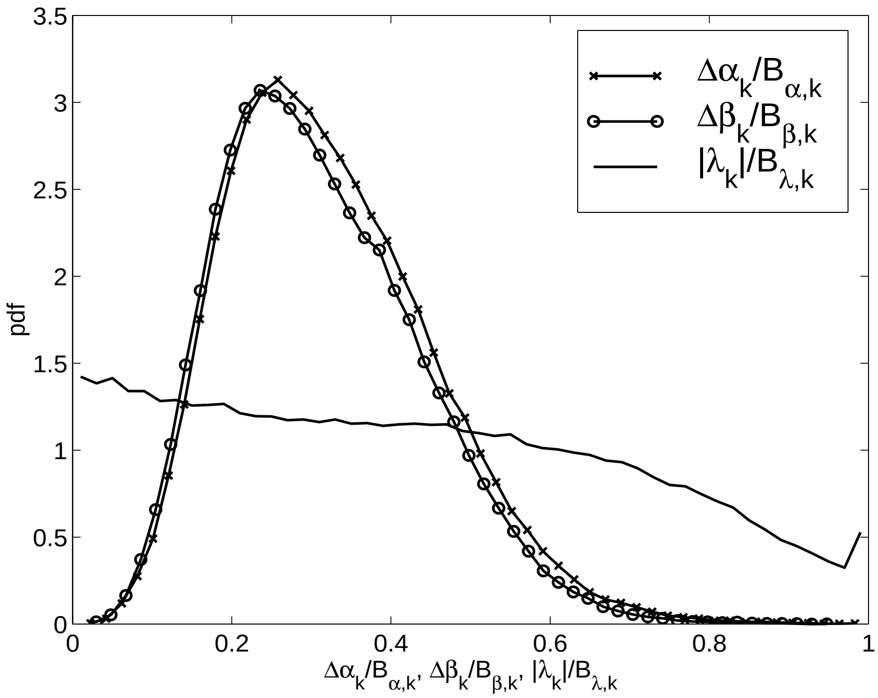



Figure 5.59: Numerical estimations of pdfs of $\Delta\alpha_k/B_{\alpha,k}$, $\Delta\beta_k/B_{\beta,k}$ and $|\lambda_k(u;O)|/B_{\lambda,k}$ (frame size 1024, $g = (13, 15)_{octal}$, code rate 1/3, 1 through 8 iterations, $E_b/N_0 = 0$ dB, $2.45 \times 10^5$ data points).

The small spike within the pdf of $|\lambda_k(u;O)|/B_{\lambda,k}$ near unity became more prominent as $E_b/N_0$ grows. When $E_b/N_0 \to \infty$, the pdf of $|\lambda_k(u;O)|/B_{\lambda,k}$ approached a single impulse at unity and was flat elsewhere. This indicates that when the transition at time $k$ is estimated with great confidence, $|\lambda_k(u;O)| \approx B_{\lambda,k}$ with high probability, while still satisfying $|\lambda_k(u;O)| \leq B_{\lambda,k}$.

**Test bound on $n$**

Integer representation and two's complement arithmetic were examined for an iterative decoder comprising two interconnected SISO modules. Two frame sizes were tested: $N = 320$ and 5120. Eight bit quantization was used for the received signal $y_k^{(i)}$ [65]. The correction function $f_c(x)$ was linearly approximated.

Simulation data showed that modular renormalization gave exactly the same soft output as subtractive normalization, even when $n$ was not sufficient. Therefore, normalization of the path metrics can be automatically implemented in two's complement arithmetic without additional computation.

To determine the minimum internal data width, the analytical result can be obtained according to Equation (5.224) or (5.234). For $0 \leq E_b/N_0(\text{dB}) \leq 2.5$ (the range used for $N = 320$), assuming $\max_k |\lambda_k(u; I)| \leq 2^5 \times \max_{k,i} |y_k^{(i)}|$, $n_{min}$ is obtained using Equation (5.234):

$$\begin{aligned} n_{min} &= \left\lceil 8 + \log_2 \left( 2^6 \times 3 + 4/3 \times 10^{2.5/10} \times (2 \times 3 + 1) \times 2 \right) \right\rceil \\ &= 16 \end{aligned} \tag{5.235}$$

For $0 \leq E_b/N_0(\text{dB}) \leq 0.5$ (the range used for $N = 5120$), a similar calculation shows that the analytical minimum is also $n_{min} = 16$.

Monte Carlo simulations were carried out for the above schemes with various $n$. These simulations confirm that when $n$ is determined by the above procedure, overflow that corrupts the computation never occurs, although overflow that does not affect correctness occurs during the $\alpha$ and $\beta$ recursions due to the modular renormalization technique. In Figure 5.60, curves of BER versus $E_b/N_0$ are plotted for both $N = 320$ and $N = 5120$. This figure shows that when $N = 320$, $n_{min} = 15$ is required for $E_b/N_0 \leq 2.5$ dB, while $n_{min} = 14$ is required for $N = 5120$ and $E_b/N_0 \leq 0.5$ dB.

Since $\max_k |\lambda_k(u; I)|$ increases with the iteration number, $n_{min}$ also increases with the iteration number. This is illustrated in Figure 5.61 with two cases. BER versus iteration curves for $N = 320$ at $E_b/N_0 = 2.5$ dB and $N = 5120$ at $E_b/N_0 = 0.5$ dB are shown. When $N = 320$ and $E_b/N_0 = 2.5$ dB, $n_{min} = 14$ is enough for four or fewer iterations, but not for five or more iterations. When $N = 5120$ and $E_b/N_0 = 0.5$ dB, $n_{min} = 13$ is enough for four or fewer iterations, but not for five or more iterations. To reduce the internal data width without appreciably compromising the performance, a tighter limit on $\max_k |\lambda_k(u; I)|$ can be used.

When compared to the experimental minimum, the analytical $n_{min}$ is one or two bits greater than the experimental answer. Thus, under the conditions examined here, the bound in Equation (5.224) is slightly pessimistic, resulting in a reasonable engineering design rule.

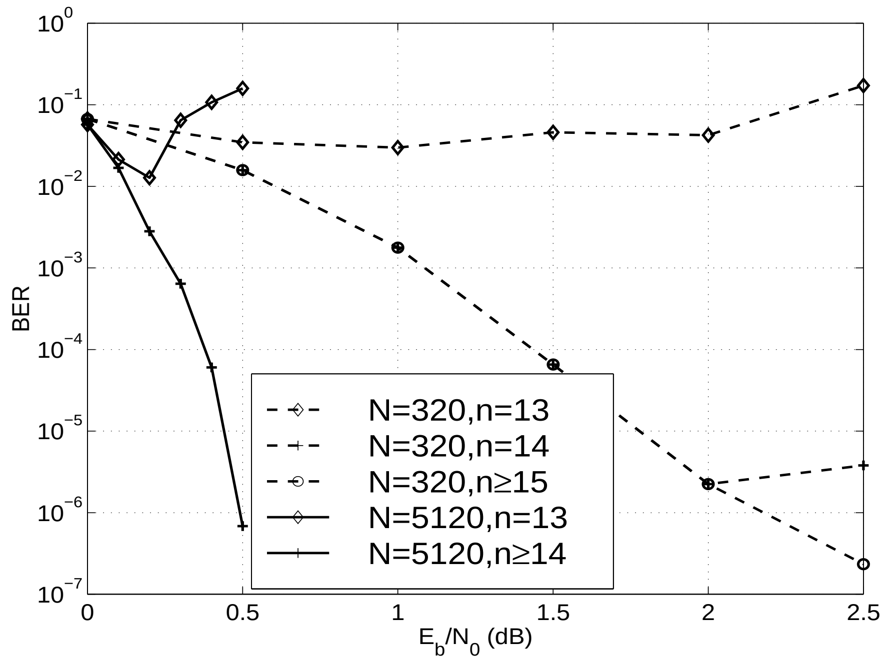



Figure 5.60: BER versus $E_b/N_0$ to search for $n_{min}$ of PCCC ($g = (13, 15)_{octal}$, rate 1/3, 8 iterations for $N = 320$, 10 iterations for $N = 5120$, and 8 bit quantization of received signals $y_k^{(i)}$).

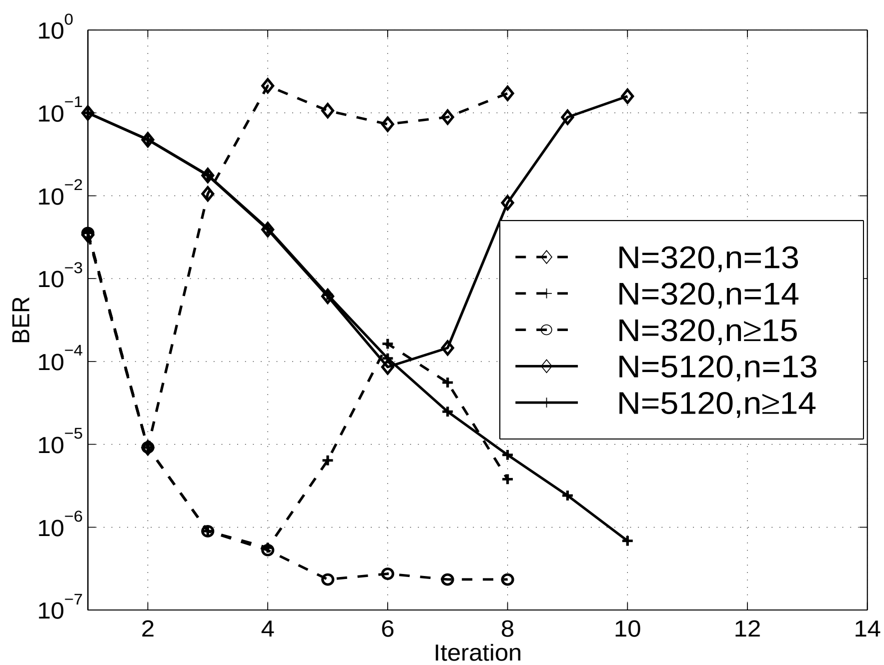



Figure 5.61: BER versus Iteration to search for $n_{min}$ of PCCC ($g = (13, 15)_{octal}$, rate 1/3, $E_b/N_0 = 2.5$ dB for $N = 320$, $E_b/N_0 = 0.5$ dB for $N = 5120$, and 8 bit quantization of received signals $y_k^{(i)}$).

## 5.3 Summary

In this chapter, the influence of quantization and fixed point arithmetic upon the BER performance of turbo decoders was discussed. It was shown how to find the optimal scaling factor the received signal so that it fits the full scale range and the resolution of the quantizer.

Then upper bounds on the absolute differences between forward metrics, absolute differences between backward metrics, and the absolute value of the output LLR were obtained. With these bounds, an analytic expression for the minimum internal data width was determined. In addition, the applicability of a modular renormalization technique was proven. All the analysis was validated by computer simulations for a range of typical examples.

Simulations showed that $n_{min}$ determined by Equation (5.224) was accurate. With $n_{min}$-bit two's complement arithmetic, no detrimental overflow occurred. Furthermore, the modular renormalization technique works with neither hardware cost nor data width penalty. There are significant advantages in applying these techniques to practical implementations such as the FPGA design presented in Chapter 9.

# Chapter 6

# Stopping Criteria for Iterative Turbo Decoding

Previous chapters have shown that iterative decoding is a key feature of turbo codes [27]. As the number of iterations increases, the bit error rate (BER) and frame error rate (FER) of the decoder decrease and the incremental improvement gradually diminishes. As illustrated in Section 3.6, a fixed number $\tau_{max}$ is often chosen and each frame is decoded for $\tau_{max}$ iterations (called "FIXED" scheme in the following). Usually $\tau_{max}$ is set with the worst corrupted frames in mind. Most frames need fewer iterations to converge. Thus the amount of processing required for turbo decoding can be reduced without sacrificing performance if the decoder terminates the iterations for each individual frame immediately after the bits are correctly estimated.

While this is unrealistic when the transmitted bits are unknown, several schemes have been proposed to control the termination. For purpose of comparison, three existing termination schemes are briefly reviewed in Section 6.1. Then a new stopping criterion is proposed in Section 6.2 which requires no storage and minimal calculation. Simulation results are presented in Section 6.3 to compare the new criterion with the existing schemes.

## 6.1 Review of Existing Stopping Criteria

Three known stopping criteria are reviewed in this section. A maximum of $\tau_{max}$ iterations are performed even if the dynamic stopping criteria are not satisfied within $\tau_{max}$ iterations.

1. **Cyclic Redundancy Check (CRC)** [72]: Under this approach, $n_{crc}$ CRC bits are appended to the end of each information frame by the CRC encoder before the expanded frame is sent to the turbo encoder. In the decoder, following each iteration, the decoder makes hard decisions and the CRC bits are used to check for frame errors. The iterative decoding is stopped when the CRC detects no error. This method is attractive because most communication standards use some form of CRC for error detection. Note that an outer code different than CRC, e.g., BCH code [73], could also be used.

2. **Cross Entropy (CE)** [55]: After each iteration $\tau$, the CE technique computes the approximate cross entropy $T(\tau)$ between LLRs of the component decoders, where
$$T(\tau) = \sum_k \frac{|\Lambda_{2e}^{(\tau)}(u_k) - \Lambda_{2e}^{(\tau-1)}(u_k)|^2}{\exp(|\lambda_{1k}^{A(\tau)}(u;O)|)},$$
$\Lambda_{2e}^{(\tau)}(u_k)$ is the extrinsic information of information bit $u_k$ produced by the second constituent decoder at iteration $\tau$, and $\lambda_{1k}^{A(\tau)}(u;O)$ is the complete information of $u_k$ produced by the first constituent decoder. The iterative decoding is stopped if $T(\tau) < (10^{-2} \sim 10^{-4})T(1)$.

3. **Sign Change Ratio (SCR)** [74]: This technique is related to the CE technique [74]. It computes $C(\tau)$, which is the number of sign changes of the extrinsic information between iteration $(\tau - 1)$ and $\tau$. Decoding is terminated when $C(\tau) \leq q_c N$, where $q_c$ is a constant usually chosen to be $0.005 \leq q_c \leq 0.03$ and $N$ is the frame size.

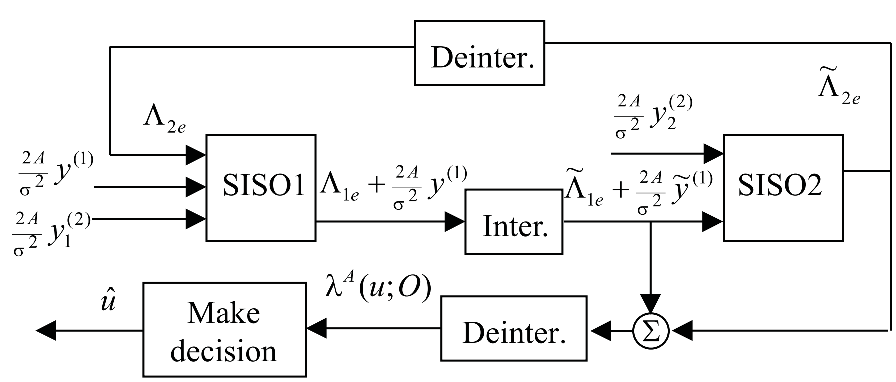



Figure 6.62: The reproduced simplified turbo decoder.

Consider the turbo decoder structure in Figure 6.62, which is redrawn from Figure 3.18 with $\tilde{\Lambda}$ representing $\Lambda$ in an interleaved order. As discussed in Section 3.4, this form of the decoder minimizes the required storage of SISO input vectors, and reduces the computation inside SISO. The CRC method requires the transmission of $n_{crc}$ extra bits for error detection. For each iteration, the CRC method requires that the LLRs be computed ($N$ real number additions to get complete information $\lambda_k^A(u;O)$), hard decisions based on the sign of $\lambda_k^A(u;O)$ to be made, and finally CRC decoding using the hard decisions (at least $3N$ binary additions for commonly used CRC generator polynomials). The CE method requires $(6N-1)$ real number operations (including $N$ additions to get $\lambda_{1k}^{A(\tau)}(u;O)$) and $(N+2)$ real number memory units for storage [74]. The SCR technique requires only $N$ binary additions, a counter no greater than $N$, and $N$ bits to store the sign bits of the extrinsic information if one uses a minimal-memory implementation which overwrites SISO output of previous iteration.

Although the SCR method is very simple, a modified scheme is proposed to further obviate the need for storage of values from the previous iteration. For notational convenience, this new technique is called SDR.

## 6.2 A New Stopping Criterion

Consider a turbo code with two identical recursive systematic convolutional (RSC) codes. Let $u_k$, $k \in \{1, ..., N\}$, be the information bits which are BPSK modulated and transmitted through an $\mathcal{N}(0, \sigma^2)$ AWGN channel. At the receiver, $(y_{1k}^{(1)}, y_{1k}^{(2)}, y_{2k}^{(2)})$ are signals corresponding to $u_k$, where $y_{1k}^{(1)}$ is the systematic signal, $y_{1k}^{(2)}$ and $y_{2k}^{(2)}$ are parity signals for RSC1 and RSC2 respectively. They are sent to the SISO [70] decoding modules SISO1 and SISO2 to produce estimates $\hat{u}_k$.

At the decoder input $\lambda_{1k}(c^{(1)}; I) = (2A/\sigma^2)y_{1k}^{(1)}$ is the LLR representation of the measured information corresponding to $u_k$. In the $\tau$-th iteration, let $\Lambda_{1a}^{(\tau)}(u_k)$ and $\Lambda_{1e}^{(\tau)}(u_k)$ be the *a priori* information and the extrinsic information of SISO1, respectively, and let $\Lambda_{2a}^{(\tau)}(u_k)$ and $\Lambda_{2e}^{(\tau)}(u_k)$ be those of SISO2. Then the complete information at the output of SISO1 and SISO2 are $\lambda_{1k}^{A(\tau)}(u;O)$ and $\lambda_{2k}^{A(\tau)}(u;O)$:

$$\begin{cases} \lambda_{1k}^{A(\tau)}(u;O) &= \lambda_{1k}(c^{(1)}; I) + \Lambda_{1a}^{(\tau)}(u_k) + \Lambda_{1e}^{(\tau)}(u_k) \\ \lambda_{2k}^{A(\tau)}(u;O) &= \lambda_{1\alpha(k)}(c^{(1)}; I) + \Lambda_{2a}^{(\tau)}(u_k) + \Lambda_{2e}^{(\tau)}(u_k) \end{cases} . \tag{6.236}$$

The iterative process is implemented by making the following assignments:

$$\begin{cases} \Lambda_{1a}^{(\tau)}(u_{\alpha(k)}) & = \Lambda_{2e}^{(\tau-1)}(u_k) \\ \Lambda_{2a}^{(\tau)}(u_k) & = \Lambda_{1e}^{(\tau)}(u_{\alpha(k)}) \end{cases}, \tag{6.237}$$

where $\alpha(k)$ stands for the interleaver mapping. According to Equation (6.236), for $j = 1, 2$, the $j$-th complete information of $u_k$ is composed of three estimates: $\lambda_{1k}(c^{(1)}; I)$ from the channel directly, the *a priori* value $\Lambda_{ja}^{(\tau)}(u_k)$, and $\Lambda_{je}^{(\tau)}(u_k)$ obtained based on the trellis structure. Among the three estimates, $\lambda_{1k}(c^{(1)}; I)$ is fixed for every iteration, while $\Lambda_{ja}^{(\tau)}(u_k)$ and $\Lambda_{je}^{(\tau)}(u_k)$ are updated from iteration to iteration. Since the extrinsic information is used as the *a priori* information for the next decoder as specified by (6.237), $\Lambda_{ja}^{(\tau)}(u_k)$ and $\Lambda_{je}^{(\tau)}(u_k)$ are correlated for $\tau \geq 2$.

It is reasonable to expect that for a "good" (easy to decode) frame, $\Lambda_{je}^{(\tau)}(u_k)$ will agree with $\Lambda_{ja}^{(\tau)}(u_k)$ on the hard estimation $\hat{u}_k$ as the iterations converge. In other words, $\text{sign}(\Lambda_{je}^{(\tau)}(u_k))$ converges to $\text{sign}(\Lambda_{ja}^{(\tau)}(u_k))$ as the decoding proceeds, since the sign of the soft value results in a hard estimate of the desired bit. If $\text{sign}(\Lambda_{je}^{(\tau)}(u_k)) = \text{sign}(\Lambda_{ja}^{(\tau)}(u_k))$, then they are correlated. The *a priori* information will be a positive excitation to the extrinsic information and as a result $|\Lambda_{ja}^{(\tau)}(u_k)|$ increases with the sign intact. This implies that the *a priori* information of the next decoder is enhanced, and the extrinsic information of the next SISO will tend to have the same sign.

Let $N_{ber}$ be the number of bit errors in a frame and let $D_j(\tau)$ be the number of sign differences between $\Lambda_{ja}^{(\tau)}(u_k)$ and $\Lambda_{je}^{(\tau)}(u_k)$. The above speculations are supported by the following observations made from repeated simulations:

1. For a "bad" (hard to decode) frame, both $E[|\Lambda_{ja}^{(\tau)}(u_k)|]$ and $E[|\Lambda_{je}^{(\tau)}(u_k)|]$ do not increase significantly, but stay close to or lower than $E[\lambda_{1k}(c^{(1)}; I)]$.

2. For a "good" frame, both $E[|\Lambda_{ja}^{(\tau)}(u_k)|]$ and $E[|\Lambda_{je}^{(\tau)}(u_k)|]$ increase as $\tau$ increases. When $N_{ber}$ gets close to 0, $E[|\Lambda_{ja}^{(\tau)}(u_k)|]$ and $E[|\Lambda_{je}^{(\tau)}(u_k)|]$ are significantly (5-10 times) larger than $E[|\lambda_{1k}(c^{(1)}; I)|]$. Consequently, $\lambda_{1k}^{A}(u; O)$ is determined primarily by $\Lambda_{ja}^{(\tau)}(u_k) + \Lambda_{je}^{(\tau)}(u_k)$.

3. For a "bad" frame, $D_j(\tau)$ stays high as $\tau$ increases.

4. For a "good" frame, $D_j(\tau)$ tends towards 0 as $\tau$ increases, similar to $N_{ber}$. Usually, $N_{ber}$ reaches 0 about $(0.5 \sim 1)$ iteration earlier than $D_j(\tau)$.

5. Even after $N_{ber}$ drops to 0, $|\Lambda_{ja}^{(\tau)}(u_k)|$, $|\Lambda_{je}^{(\tau)}(u_k)|$, and $|\lambda_{jk}^{A(\tau)}(u;O)|$ keep increasing with $\tau$ until $E[|\Lambda_{ja}^{(\tau)}(u_k)|] \approx E[|\Lambda_{je}^{(\tau)}(u_k)|]$, $\forall j$.

In addition to simulations, similar conclusions were also drawn by analysis in [75]. Observation 5 tells us that it is a waste to iterate until the decoder stabilizes if the goal is to make $N_{ber} = 0$. Observation 4 suggests that $D_j(\tau)$ is an indicator for stopping the iteration without degrading the performance. In consequence, the new terminating scheme is as follows:

$$D_j(\tau) \begin{cases} \geq q_d \times N, & \text{continue the iteration;} \\ \\ < q_d \times N, & \text{stop the iteration.} \end{cases} \tag{6.238}$$

where $q_d$ is the sign difference ratio, $N$ is the frame size, thus leading to the designation of the **sign difference ratio (SDR)** criterion. Because of Equation (6.237), $D_j(\tau)$ is also the number of sign differences between the extrinsic information of both SISOs. Thus the SDR criterion requires sign consistency between the extrinsic information of the two component SISOs. As a consequence of observation 2, it is also valid to use $D'_j(\tau)$ in place of $D_j(\tau)$, where $D'_1(\tau)$ is the number of sign differences between $\Lambda_{1a}^{(\tau)}(u_k)$ and $(\Lambda_{1e}^{(\tau)}(u_k) + \lambda_{1k}(c^{(1)};I))$, and $D'_2(\tau)$ is that between $(\Lambda_{2a}^{(\tau)}(u_k) + \lambda_{1\alpha(k)}(c^{(1)};I))$ and $\Lambda_{2e}^{(\tau)}(u_k)$. It is desirable to use $D'_j(\tau)$ instead of $D_j(\tau)$ for the decoder structure shown in Figure 6.62, where the summation $(\Lambda_{1e}^{(\tau)}(u_{\alpha(k)})+\lambda_{1\alpha(k)}(c^{(1)};I)) = (\Lambda_{2a}^{(\tau)}(u_k)+\lambda_{1k}(c^{(1)};I))$ is passed from SISO1 to SISO2. Simulation results show no noticeable difference between these two schemes, and thus they are not differentiated in this paper.

Just as with the SCR, the SDR scheme requires $N$ binary additions of sign bits and a counter no greater than $N$ to check the criterion. However, SDR eliminates the storage of sign bits from the previous iteration. The computation complexity and storage comparison between the discussed stopping criteria are shown in Table 6.4

Simulations show that the proper range of $q_d$ is $0.001 \leq q_d \leq 0.01$. Generally speaking, the smaller $q_d$ is, the smaller the BER/FER degradation is, but the larger the average number of iterations is. The maximum number of sign inconsistencies $\lfloor q_d N \rfloor$ increases as $N$ grows. For a given $N$, $q_d$ should be smaller for higher $E_b/N_0$. In the BER$< 10^{-6}$ region, the performance is not degraded when $\lfloor q_d N \rfloor$ is less than or equal to one.

Table 6.4: Complexity comparison of four stopping criteria.

| **Criteria** | **Extra Bits per Frame** | **Non-Binary Computations** | **Binary Additions** | **Storage** |
|---|---|---|---|---|
| **CRC** | $4 \sim 32$ | $N$ | $> 3N$ | 0 |
| **CE** | 0 | $6N - 1$ | 0 | $(N+2)$ non-binary numbers |
| **SCR** | 0 | $N$ | $N$ | $N$ bits |
| **SDR** | 0 | $N$ | $N$ | 0 |

## 6.3 Simulation Results

Simulations were performed with constituent RSC code $g(D) = [1, 1 + D + D^3/1 + D^2 + D^3]$. Five terminating schemes were studied: FIXED, CRC, CE, SCR, and SDR. The "GENIE" case, where the information bits are known and the iteration is stopped immediately after the frame is correctly decoded, is shown as the limit of all possible schemes. The CRC generator polynomial used was $g(x) = x^{16} + x^{15} + x^2 + 1$ (see Appendix C).

The performance for $N = 200$ with code rate $r = 1/3$ is shown in Figure 6.63 and 6.64, while that for $N = 5120$ and $r = 1/2$ is shown in Figure 6.65 and 6.66. In both cases, all six schemes exhibit similar BER and FER performance. The simple SDR technique is as efficient as CE and SCR methods in terms of BER, FER, and the average number of iterations.

It is observed that the CRC method uses almost the same average number of iterations as the GENIE scheme, while the CE, SCR, and SDR methods all require about one more iteration on average. However, the CRC method transmits extra bits (16 in this case) which compromises the bandwidth efficiency. The CRC technique is also more computationally expensive than the SCR and SDR schemes. In addition, it is observed that the BER of CRC at $E_b/N_0 = 1.2$ dB in Figure 6.65 is significantly higher than that of the other techniques, which implies the occurrence of detection failure.

Figure 6.66 shows that for the high BER region (BER$> 5\times10^{-2}$), the CE and SCR methods require fewer iterations than the others. This happens because at low BER,

$\Lambda_{ja}^{(\tau)}$ is usually small, thus $\lambda_{1k}(c^{(1)}; I)$ and $\lambda_{jk}(c^{(2)}; I)$ dominate in the computation. Since $\lambda_{1k}(c^{(1)}; I)$ and $\lambda_{jk}(c^{(2)}; I)$ are invariant from iteration to iteration, the extrinsic information stabilizes quickly. This leads to premature termination, although the frame may be still in error. Since turbo codes are usually used in the low BER region (BER$< 10^{-3}$), this effect can be ignored.

The performance of the SDR method is sensitive to $q_d$ in the low BER region. As an example, Figure 6.67 shows that the FER performance at $E_b/N_0 = 2.5$ dB is compromised when $q_d N = 2$. Figure 6.69 illustrates that the BER and FER performance degrade significantly at $E_b/N_0 = 1.2$ dB when $q_d$ increases from $10^{-4}$ to $5 \times 10^{-3}$ ($\lfloor q_d N \rfloor$ increases from 0 to 5). However, the performance is not affected by $q_d$ in the BER$> 10^{-4}$ region. Thus $q_d$ should be chosen carefully in the low BER region in order not to compromise the performance. Figures 6.68 and 6.70 illustrate that the average number of iterations increases when $q_d$ decreases. It is also shown that the SDR criterion can be used after both SISOs to save roughly 0.2 iteration on average without BER or FER degradation.

## 6.4 Summary

In this chapter, a new stopping criterion, the sign difference ratio, is presented and compared with three existing stopping criteria. It has been shown by simulations that the SDR method performs with negligible difference from CE and SCR methods. Like the previously reported SCR method, the new SDR method requires significantly less computation than the CE method, with the additional advantage of reduced storage requirement. Although the CRC method requires the fewest iterations, it results in much more computation than the SDR and SCR methods, and its potential failures at low BER (BER$\leq 10^{-6}$) result in degraded performance.

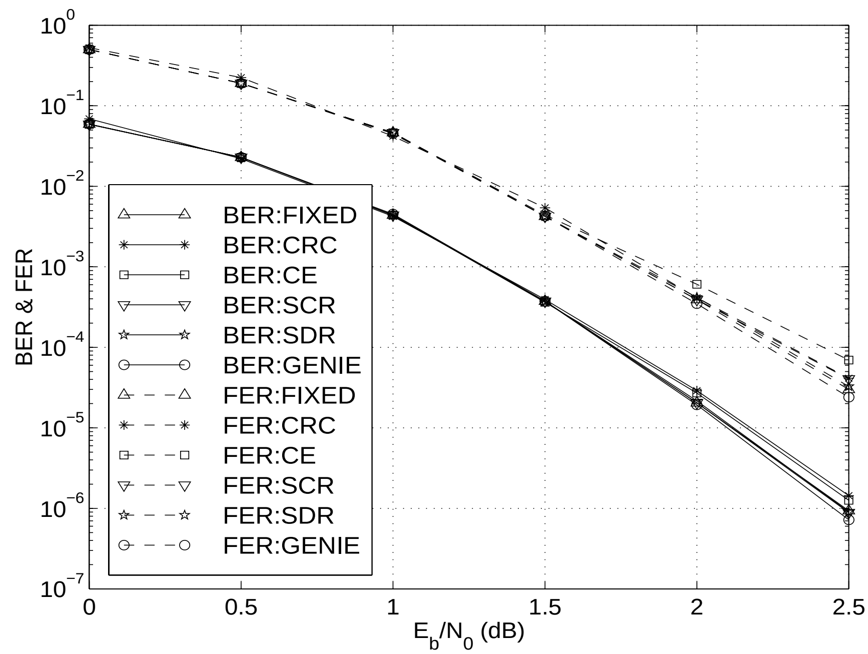



Figure 6.63: BER and FER versus $E_b/N_0$ for six stopping schemes: FIXED, CRC, CE, SCR($q_c = 10^{-2}$), SDR($q_d = 10^{-3}$) and GENIE (rate 1/3, $g = (13, 15)_{octal}$, $N = 200$, 8 maximum iterations).

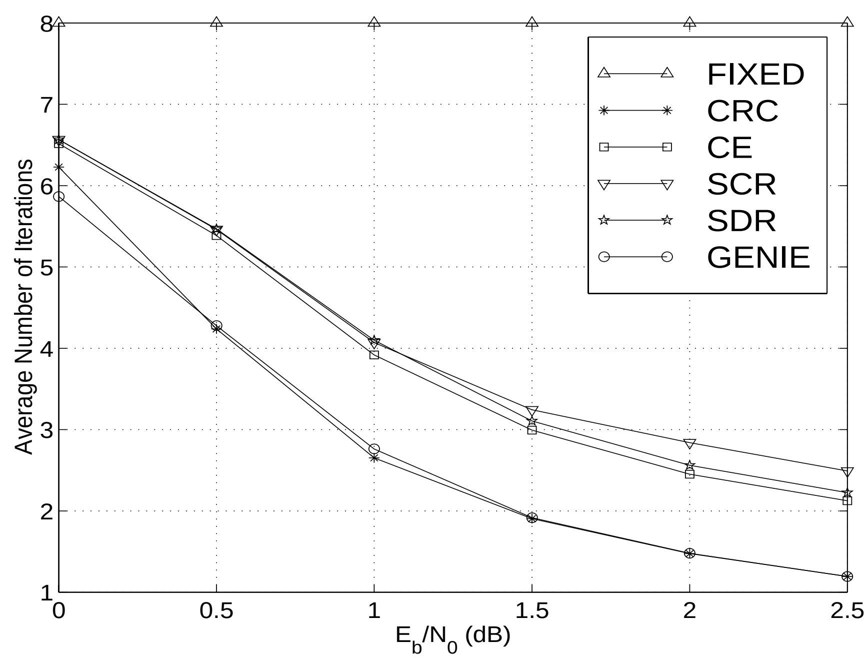



Figure 6.64: Average number of iterations versus $E_b/N_0$ for six stopping schemes: FIXED, CRC, CE, SCR($q_c = 10^{-2}$), SDR($q_d = 10^{-3}$) and GENIE (rate 1/3, $g = (13, 15)_{octal}$, $N = 200$, 8 maximum iterations).

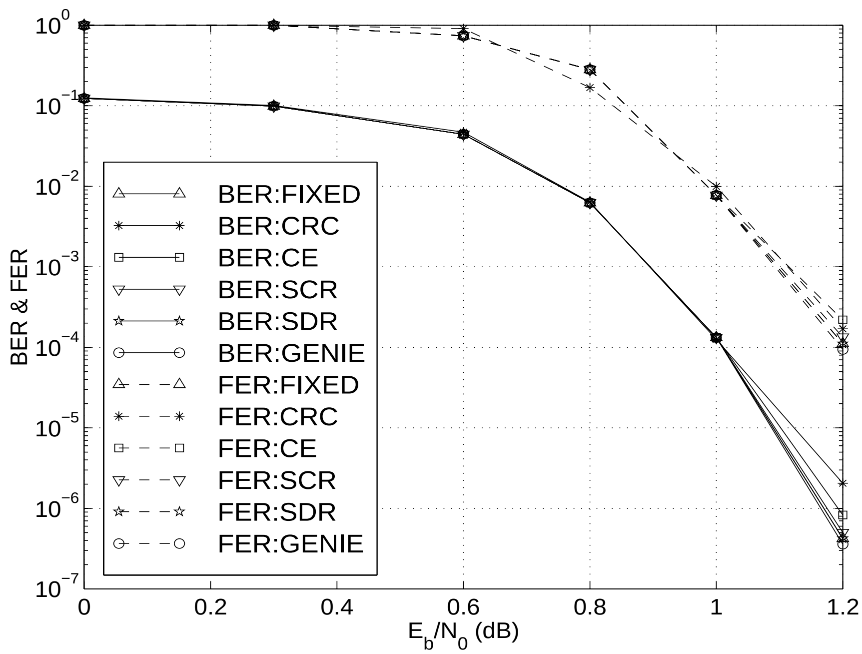



Figure 6.65: BER and FER versus $E_b/N_0$ for six stopping schemes: FIXED, CRC, CE, SCR($q_c = 10^{-3}$), SDR($q_d = 10^{-4}$) and GENIE (rate 1/2, $g = (13, 15)_{octal}$, $N = 5120$, 10 maximum iterations).

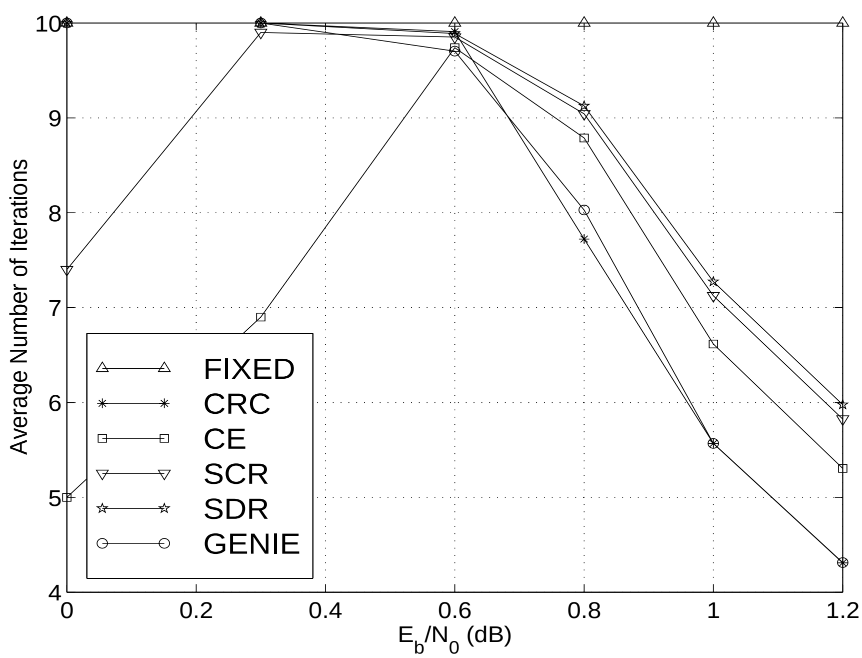



Figure 6.66: Average number of iterations versus $E_b/N_0$ for six stopping schemes: FIXED, CRC, CE, SCR($q_c = 10^{-3}$), SDR($q_d = 10^{-4}$) and GENIE (rate 1/2, $g = (13, 15)_{octal}$, $N = 5120$, 10 maximum iterations).

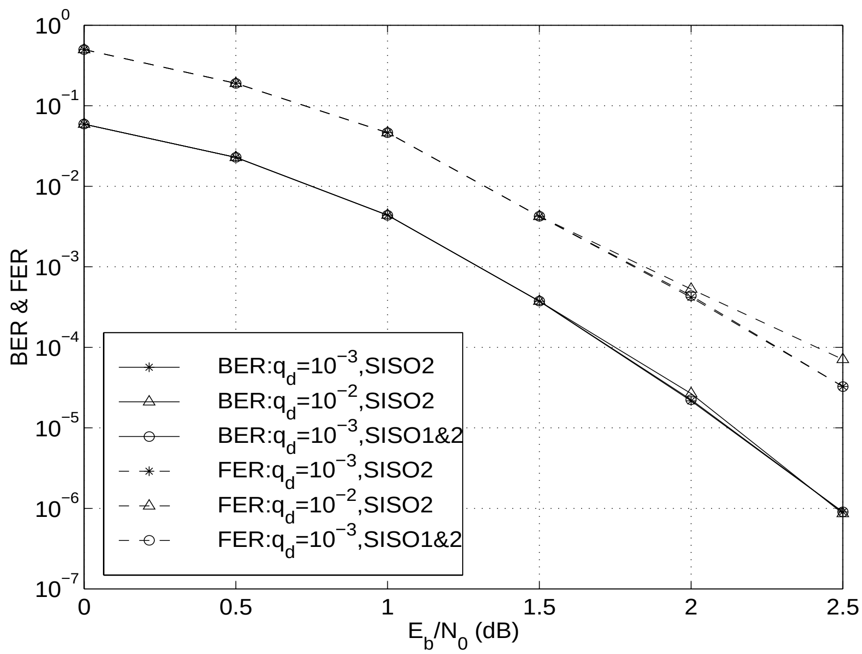



Figure 6.67: BER and FER versus $E_b/N_0$ for four SDR cases (rate 1/3, $g = (13, 15)_{octal}$, $N = 200$, 8 maximum iterations). "SISO2" indicates that SDR criterion is used only on SISO2, while "SISO1&2" implies both SISO1 and SISO2.

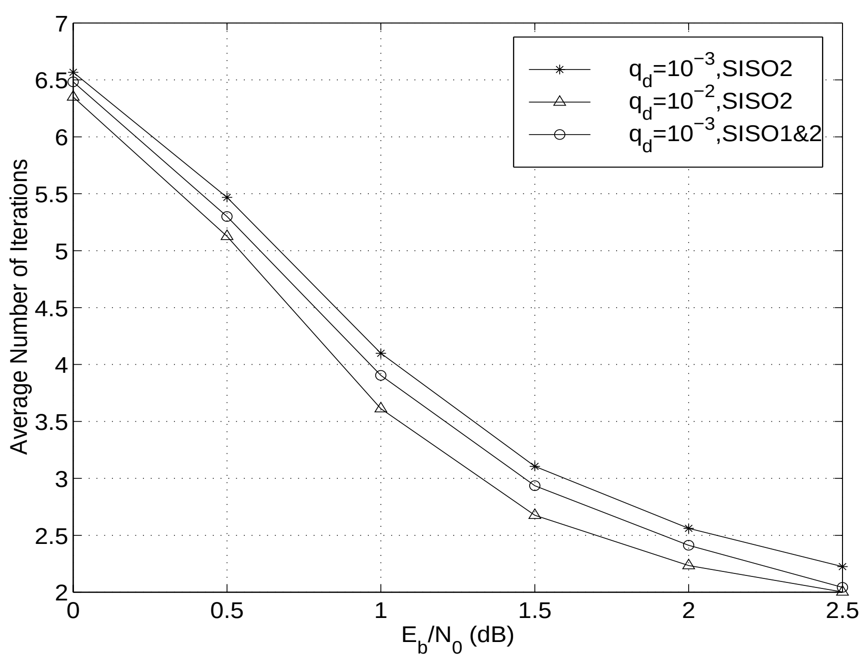



Figure 6.68: Average number of iterations versus $E_b/N_0$ for four SDR cases (rate 1/3, $g = (13, 15)_{octal}$, $N = 200$, 8 maximum iterations). "SISO2" and "SISO1&2" are used as in Figure 6.67.

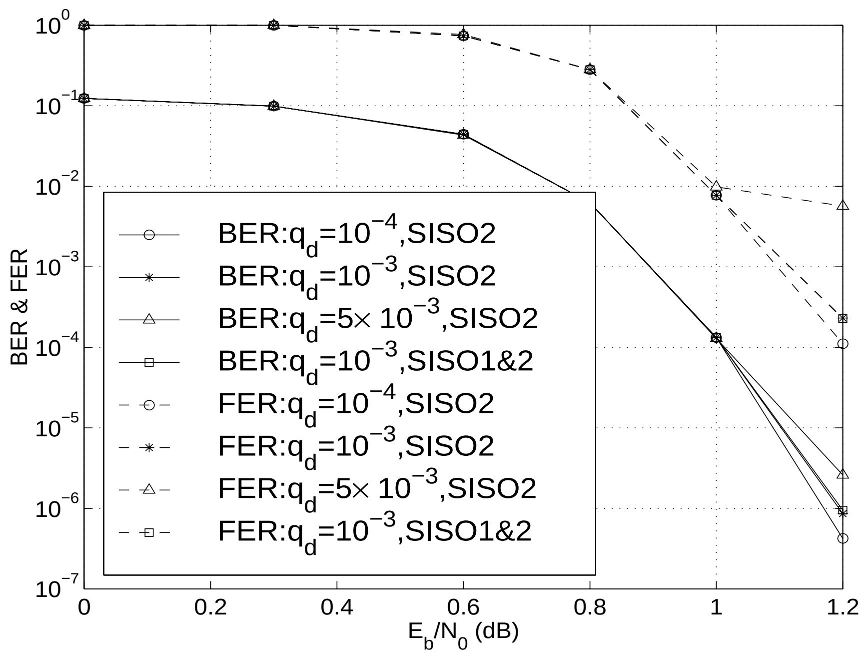



Figure 6.69: BER and FER versus $E_b/N_0$ for four SDR cases (rate 1/2, $g = (13, 15)_{octal}$, $N = 5120$, 10 maximum iterations). "SISO2" and "SISO1&2" are used as in Figure 6.67.

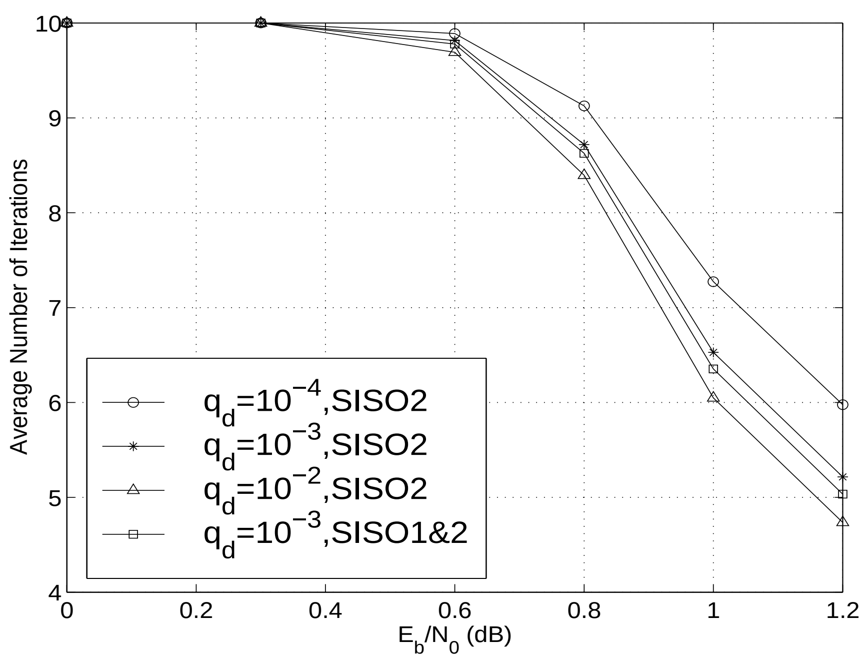



Figure 6.70: Average number of iterations versus $E_b/N_0$ for four SDR cases (rate 1/2, $g = (13, 15)_{octal}$, $N = 5120$, 10 maximum iterations). "SISO2" and "SISO1&2" are used as in Figure 6.67.

# Chapter 7

# An ARQ Technique Using Related PCCC and SCCC

While all the other chapters consider FEC only, this chapter proposes a new type-II code-combining hybrid ARQ strategy that uses both PCCCs and SCCCs. The new ARQ technique is termed "P/S ARQ" scheme and is based on the observation that a PCCC can be represented by an equivalent punctured SCCC.

First, the P/S ARQ scheme is outlined in Section 7.1. Then it is shown that a PCCC can be represented as a punctured SCCC with a restriction on the interleaver design and the puncturing pattern. Next, the proposed ARQ strategy is described in detail in Section 7.3. Finally, simulation results are shown in Section 7.4 to compare the performance of the proposed technique to a simple type-I hybrid PCCC ARQ system and a PCCC FEC system. The proposed technique shows remarkable performance in terms of throughput efficiency, BER, and FER.

## 7.1 P/S ARQ Scheme

PCCCs have been shown in Chapter 4 to achieve remarkable power efficiencies for BER down to about $10^{-5}$ or $10^{-6}$ [27]. However, for BER below $10^{-6}$, the performance is much less impressive due to a "flattening" or shallowing of the BER curve, which is the result of a small number of low weight codeword sequences [76]. The BER-shallowing problem can be improved by using SCCCs, which offer performance superior to PCCCs for moderate and low BER regions [39]. However, PCCCs still offer better performance than SCCCs for $E_b/N_o$ very close to the capacity limit. The contrast between PCCCs and SCCCs is illustrated by the simulation curves in Figure 7.71. Apparently, communication systems operating near the capacity limit should use PCCCs in favor of SCCCs, while systems operating at higher $E_b/N_o$ should use SCCCs.

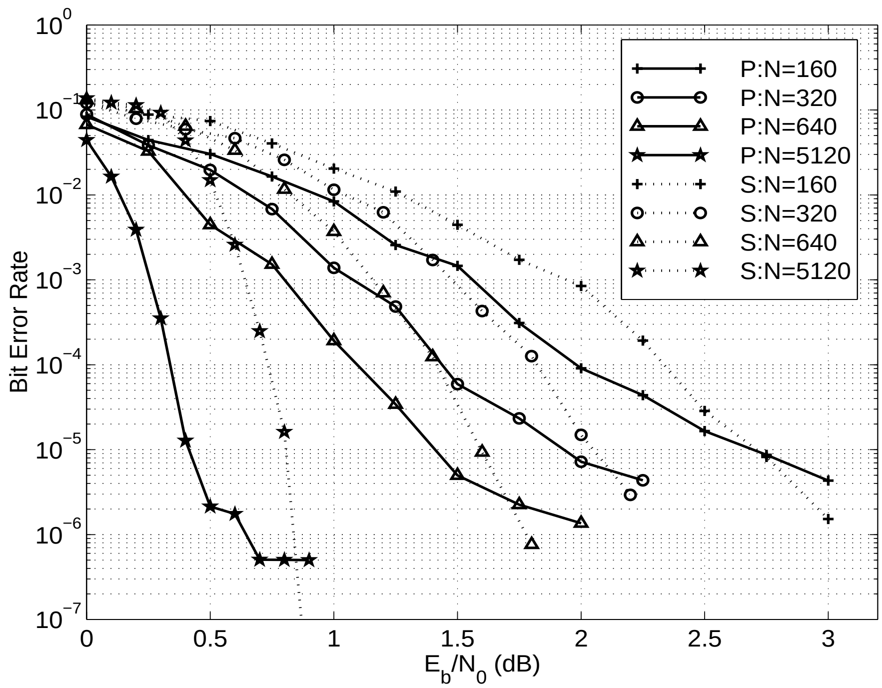



Figure 7.71: Simulation comparison between PCCCs and SCCCs with rate 1/3 and 10 iterations. "P" refers to PCCCs and "S" refers to SCCCs. PCCCs have $g = (15, 17)_{octal}$. SCCCs have $g = (7, 5)_{octal}$, $r^o = 1/2$, $r^i = 2/3$.

The concept of hybrid ARQ can be used to design an adaptive system that automatically selects either a PCCC or a related SCCC depending on the channel conditions. The ARQ system proposed here relies on the observation that any PCCC can be represented by an equivalent punctured SCCC, with an implied restriction on the interleaver design and the puncture pattern. In the proposed ARQ system, the data is encoded using a SCCC with an embedded outer error detecting code. The output of the SCCC encoder is punctured to form a PCCC, and the punctured bits are stored in a buffer. Initially, only the PCCC code bits are transmitted over the channel. If an error is detected, then the punctured bits stored in the buffer are transmitted. These bits are then combined with the PCCC code bits already received to form a received SCCC codeword sequence. Finally, the SCCC codeword sequence is decoded and the frame is either accepted or rejected after error detection. The scheme is illustrated in

Figure 7.72.

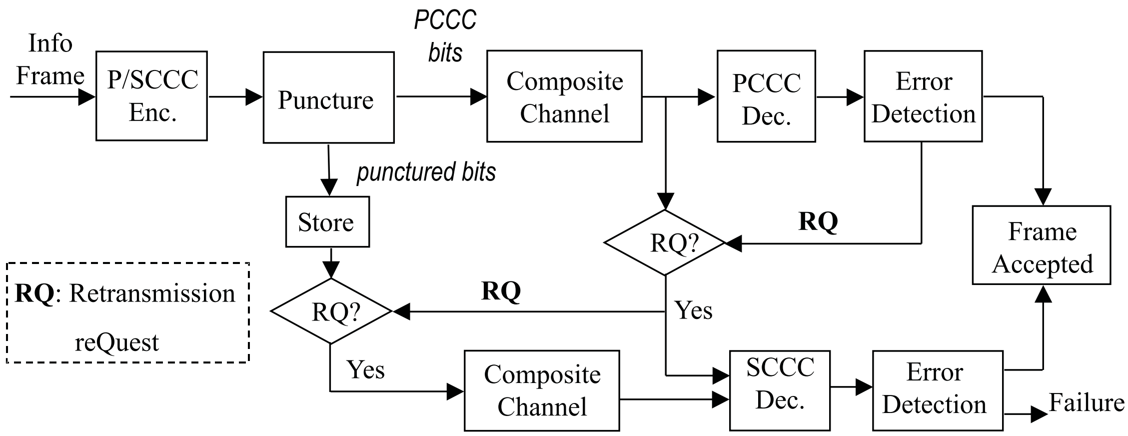



Figure 7.72: P/S ARQ scheme.

This proposed system can be categorized as a type-II code-combining hybrid ARQ system with incremental redundancy [77]. Turbo codes have previously been considered for ARQ in [78]. However, to our knowledge, this is the first discussion of an ARQ scheme that uses both PCCCs and SCCCs.

## 7.2 Generalized P/SCCC Encoder

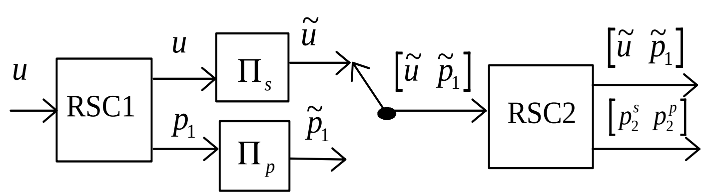



Figure 7.73: P/SCCC encoder structure.

In this section, a generalized encoder termed a **parallel/serial concatenated convolutional code (P/SCCC)** encoder is presented. The encoder produces a $(4N, N)$ SCCC, which can be punctured to form a $(3N, N)$ PCCC. Let the encoder input be a vector $\mathbf{u}$ of $N$ information bits. As shown in Figure 7.73, the data vector passes serially through a pair of RSC encoders, with interleaving between the two encoders. Let $\mathbf{G}_1 = [\mathbf{I}_N\ \mathbf{R}_1]$ and $\mathbf{G}_2 = [\mathbf{I}_{2N}\ \mathbf{R}_2]$ be the systematic-form code generator matrices of RSC1 and RSC2 respectively, where $\mathbf{I}_n$ is a $n \times n$ identity matrix and $\mathbf{R}_i$ generates the parity bits, $i = 1, 2$. Following the notation of [79], let the $N \times N$

interleaver matrix $\mathbf{\Pi}$ be denoted as $\mathbf{\Pi} = [\alpha(1), ..., \alpha(N)]$, where the integer $\alpha(i)$ indicates the position of a "1" in the $i$-th column of $\mathbf{\Pi}$, $1 \leq \alpha(i) \leq N, \alpha(i) \neq \alpha(j), \forall i \neq j$. Each column of $\mathbf{\Pi}$ is composed of a single "1" and $(N-1)$ "0"s. For a row vector $\mathbf{y}$ of size $N$, $\tilde{\mathbf{y}} = \mathbf{y}\mathbf{\Pi}$ is the interleaved vector, and $\mathbf{y} = \tilde{\mathbf{y}}\mathbf{\Pi}^T$ is the deinterleaved vector, where $\mathbf{\Pi}^T$ is the transpose of $\mathbf{\Pi}$.

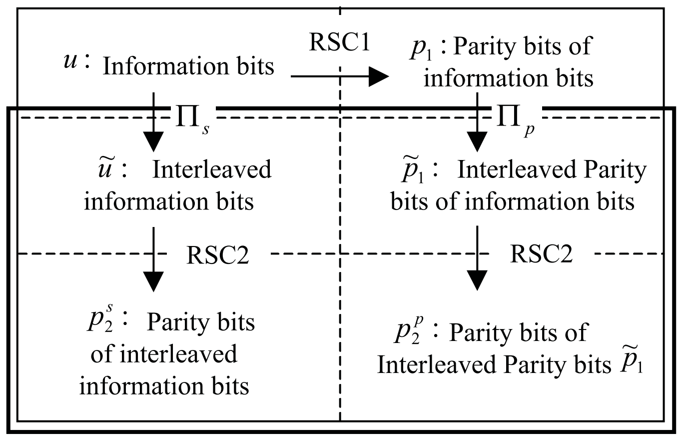



Figure 7.74: P/SCCC encoder interpreted as a product encoder.

The encoding proceeds as follows. First, $\mathbf{u}$ is encoded by the outer RSC encoder (RSC1), producing the output $\mathbf{u}\mathbf{G}_1 = [\mathbf{u}\ \mathbf{p}_1]$, where $\mathbf{p}_1 = \mathbf{u}\mathbf{R}_1$ is the vector of outer parity bits from RSC1. Two $N \times N$ interleaver matrices, $\mathbf{\Pi}_s$ and $\mathbf{\Pi}_p$, are used to interleave the systematic and parity outputs, respectively. The interleaved vectors $\tilde{\mathbf{u}} = \mathbf{u}\mathbf{\Pi}_s$ and $\tilde{\mathbf{p}}_1 = \mathbf{p}_1\mathbf{\Pi}_p$ are concatenated to form the input $[\tilde{\mathbf{u}}\ \tilde{\mathbf{p}}_1]$ to the inner RSC encoder (RSC2). The output of RSC2 is thus $[\tilde{\mathbf{u}}\ \tilde{\mathbf{p}}_1]\mathbf{G}_2 = [\tilde{\mathbf{u}}\ \tilde{\mathbf{p}}_1\ \mathbf{p}_2^s\ \mathbf{p}_2^p]$. The systematic output of RSC2 is $[\tilde{\mathbf{u}}\ \tilde{\mathbf{p}}_1]$, while the parity output $[\mathbf{p}_2^s\ \mathbf{p}_2^p]$ can be partitioned into the parity bits due to the outer encoder's systematic bits ($\mathbf{p}_2^s$) and the parity bits due to the outer encoder's parity bits ($\mathbf{p}_2^p$). The resulting code can be viewed as a special type of product code, since the same information sequence is encoded twice in different orders. This is illustrated from Figure 7.74, where the highlighted part is the output of the overall code.

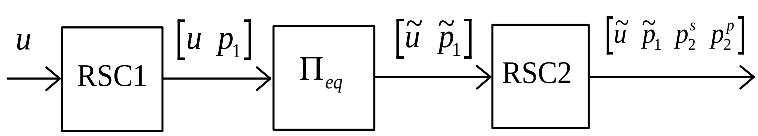



Figure 7.75: Equivalent rate 1/4 SCCC encoder.

When the entire output of RSC2 is transmitted, the code is a rate 1/4 SCCC. As shown in Figure 7.75, the outer encoder is RSC1, while the inner encoder is RSC2. The interleaver $\mathbf{\Pi}_{eq}$ is specified by the matrix

$$\mathbf{\Pi}_{eq} = \begin{bmatrix} \mathbf{\Pi}_s & 0 \\ 0 & \mathbf{\Pi}_p \end{bmatrix}. \tag{7.239}$$

Note that this form is slightly different than that of a conventional SCCC, since the inner encoder encodes all of the outer encoder's systematic bits before it encodes the outer encoder's parity bits. Although this is equivalent to a constraint on the interleaver, simulations show that the degradation is negligible.

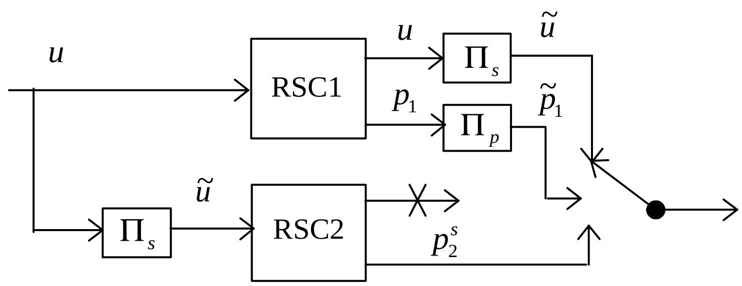



Figure 7.76: Equivalent rate 1/3 PCCC encoder when $p_2^p$ is punctured.

When only $\tilde{\mathbf{u}}$, $\tilde{\mathbf{p}}_1$, and $\mathbf{p}_2^s$ are transmitted, the code is a rate 1/3 PCCC. The equivalent encoder is shown in Figure 7.76. The generator matrix of RSC2 is now $\mathbf{G}_{2,eq} = [\mathbf{I}_N \ \mathbf{R}_{2,eq}]$, where $\mathbf{R}_{2,eq}$ is the upper $N \times N$ partition of $\mathbf{R}_2$. Note that this form is also slightly different than that of a standard PCCC, because the output of RSC1 is interleaved.

# 7.3 New ARQ Strategy with Code Combining

## 7.3.1 Description

The P/SCCC encoder described in the previous section can be used to generate either a rate 1/4 SCCC or a rate 1/3 PCCC, where the PCCC codeword sequence is a subset of the SCCC codeword sequence. Making use of this property, a type-II hybrid ARQ system with code combining has been proposed [77]. The ARQ strategy, which is denoted here as "PS-ARQ", is as follows.

1. Pass $(N - n_{crc})$ information bits through a $(N, N - n_{crc})$ CRC encoder (used for error detection) to obtain a $N$ bit frame $\mathbf{u}$.

2. Encode $\mathbf{u}$ with the P/SCCC encoder shown in Figure 7.73.

3. Transmit the code bits of the rate 1/3 PCCC: $[\tilde{\mathbf{u}}\ \tilde{\mathbf{p}}_1\ \mathbf{p}_2^s]$. Save $\mathbf{p}_2^p$ in a buffer.

4. Let $\tilde{\mathbf{y}}$, $\tilde{\mathbf{y}}_1$, and $\mathbf{y}_2^s$ be the received versions of $\tilde{\mathbf{u}}$, $\tilde{\mathbf{p}}_1$, and $\mathbf{p}_2^s$, respectively. Deinterleave $\tilde{\mathbf{y}}$ and $\tilde{\mathbf{y}}_1$, and pass $[\mathbf{y}\ \mathbf{y}_1\ \mathbf{y}_2^s]$ through a standard PCCC decoder [27].

5. Pass the output $\hat{\mathbf{u}}$ of the PCCC decoder through a CRC decoder. If a frame error is not detected, accept the frame, and return to Step 1 to transmit next frame.

6. If a frame error is detected, the receiver saves $[\tilde{\mathbf{y}}\ \tilde{\mathbf{y}}_1^s\ \mathbf{y}_2^s]$ in memory and sends a negative acknowledgment (NAK) back to the transmitter. Upon receiving the NAK, the transmitter sends the parity sequence $\mathbf{p}_2^p$ through the channel.

7. Combine $\mathbf{y}_2^p$, which is the received version of $\mathbf{p}_2^p$, with the previously received data to form an expanded sequence $[\tilde{\mathbf{y}}\ \tilde{\mathbf{y}}_1^s\ \mathbf{y}_2^s\ \mathbf{y}_2^p]$. Pass this sequence through a SCCC decoder [39] to obtain an estimate $\hat{\mathbf{u}}$ of the data.

8. Pass $\hat{\mathbf{u}}$ to CRC decoder. If a frame error is detected, declare a "failure" and discard all the signals related to the current frame.

9. Go back to Step 1 and transmit the next frame.

### 7.3.2 Throughput Efficiency

The **throughput efficiency**, $\eta$, is defined as the average number of accepted information bits per transmitted channel symbol. Let $P_{p,e}$ be the frame error probability for the rate 1/3 PCCC and $P_{s,e}$ be the frame error probability for the rate 1/4 SCCC. Assume the CRC performs perfect error detection and that the feedback channel is error-free and has negligible delay [1]. Then, the throughput efficiency of the PS-ARQ scheme is

$$\eta_{ps} = \frac{(N - n_{crc})(1 - P_{p,e}P_{s,e})}{N(3 + P_{p,e})}, \tag{7.240}$$

where the fractional rate loss due to the CRC is included. In contrast, the throughput efficiency of a type-I hybrid ARQ scheme[2] [77] with a rate 1/3 PCCC and $n_{crc}$-bit

[1] The negligible delay assumption means that stop-and-wait, go-back-N, and selective-repeat ARQ will all have the same performance.

[2] Normally, type-I hybrid ARQ schemes continue to retransmit until the packet is accepted. Here, a simpler scheme is assumed which allows a maximum of one retransmission. This scheme provides a better comparison to the proposed system, which also only supports one NAK.

CRC parity check (called “PC-ARQ” in the following) is

$$\eta_p = \frac{(N - n_{crc})(1 - P_{p,e}^2)}{3N(1 + P_{p,e})} = \frac{(N - n_{crc})(1 - P_{p,e})}{3N}, \tag{7.241}$$

which is the same as the throughput efficiency of the rate 1/3 PCCC with forward error correction only (called “PC-FEC” in the following).

Because the rate 1/4 SCCC is a more powerful code than the rate 1/3 PCCC, $P_{s,e} < P_{p,e}, \forall E_s/N_0$, and thus $\eta_{ps}$ is always larger than $\eta_p$. When $E_s/N_0$ is large, $P_{p,e} \to 0$ and $P_{s,e} \to 0$, thus $\eta_{ps} \approx \eta_p \approx 1/3$. However for the low $E_s/N_0$ region where $P_{p,e} \to 1$, but $P_{s,e} \approx 0$, thus $\eta_p \approx 0$ and $\eta_{ps} \approx 0.25$. This is verified by the simulation results shown in Section 7.4.

### 7.3.3 Computational Complexity of the Decoder

Since the complexity of a hybrid ARQ system lies mainly in the decoder, it is useful to compare the different ARQ techniques on the basis of decoder complexity. Define the **complexity load**, $\chi$, to be the ratio of the average number of decoding operations per accepted frame to the number of operations for decoding one PCCC frame. Then the complexity load of PS-ARQ is

$$\chi_{ps} = \frac{1 + 1.5P_{p,e}}{1 - P_{p,e}P_{s,e}}, \tag{7.242}$$

since the decoding complexity of SCCC is approximately 1.5 times of that of PCCC for a given frame size. The complexity load of PC-ARQ is

$$\chi_p = \frac{1 + P_{p,e}}{1 - P_{p,e}^2} = \frac{1}{1 - P_{p,e}}, \tag{7.243}$$

which is the same as the complexity load of PC-FEC.

## 7.4 Simulation Results

Simulations were performed to test the proposed ARQ scheme. In the simulation, a 16-bit CRC code with generator polynomial $g(x) = x^{16} + x^{15} + x^2 + 1$ (see Appendix C) was employed to detect the frame errors.

Interleavers $\mathbf{\Pi}_s$ and $\mathbf{\Pi}_p$ were both spread interleavers [63]. The code generator $g(D) = [1, 1 + D + D^2/1 + D^2]$ was used for both constituent encoders. Tail bits were added to terminate both constituent encoders. The frame size is $N = 1024$, including 16 CRC bits. Ten iterations using the Log-MAP algorithm [48] were run for each frame in each scheme.

The simulation results for BPSK modulation over an AWGN channel are plotted in Figures 7.77 through 7.80. $E_s/N_o$ is the ratio of the energy per code symbol to the one-sided noise spectral density of the AWGN channel[3]. Each figure shows curves for the PS-ARQ, PC-ARQ, and PC-FEC schemes.

Figure 7.77 shows the BER of the three systems. For BER$= 10^{-4}$ there is a 0.3 dB gain going from PC-FEC to PC-ARQ, while there is 0.8 dB gain when going from PC-ARQ to PS-ARQ. Similar performance is shown for FER$= 10^{-2}$ in Figure 7.78, except that the gain is 1.0 dB going from PC-ARQ to PS-ARQ. Figure 7.79 demonstrates the throughput efficiency for the three schemes. As expected, PC-ARQ and PC-FEC have the same throughput efficiency, ranging from 0 at low $E_s/N_0$ to 0.33 at high $E_s/N_0$. In contrast, in the same $E_s/N_0$ region, PS-ARQ exhibits throughput efficiency ranging from 0.24 to 0.33. Figure 7.80 confirms that the complexity load of PC-ARQ and PC-FEC are the same, and that the complexity load of PS-ARQ is the same as PC-FEC/ARQ for high $E_s/N_o$ and actually lower for $E_s/N_o < -4.5$ dB.

Thus, the PS-ARQ scheme not only has the best BER/FER performance, but also the best throughput efficiency and complexity load, as compared to PC-ARQ and PC-FEC.

[3]Energy per bit $E_b$ is not used because its definition for hybrid ARQ systems is vague.

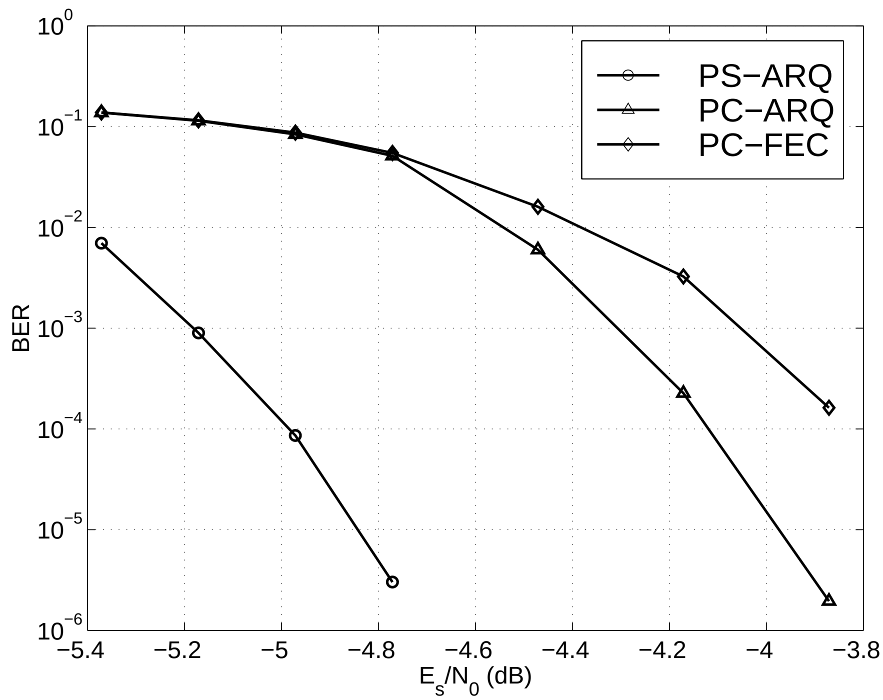



Figure 7.77: BER versus $E_s/N_0$ for AWGN channel.

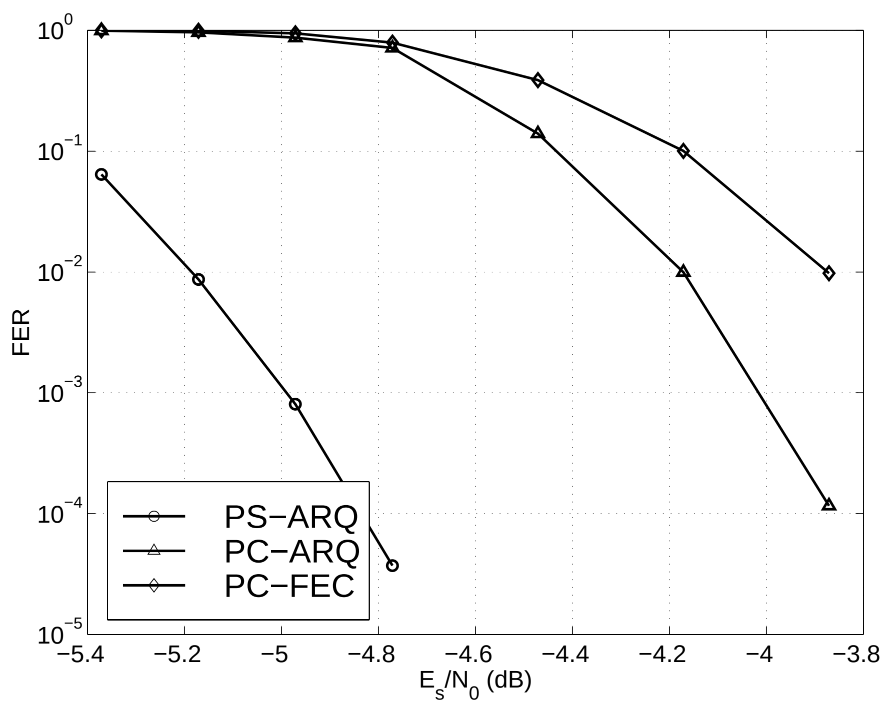



Figure 7.78: FER versus $E_s/N_0$ for AWGN channel.

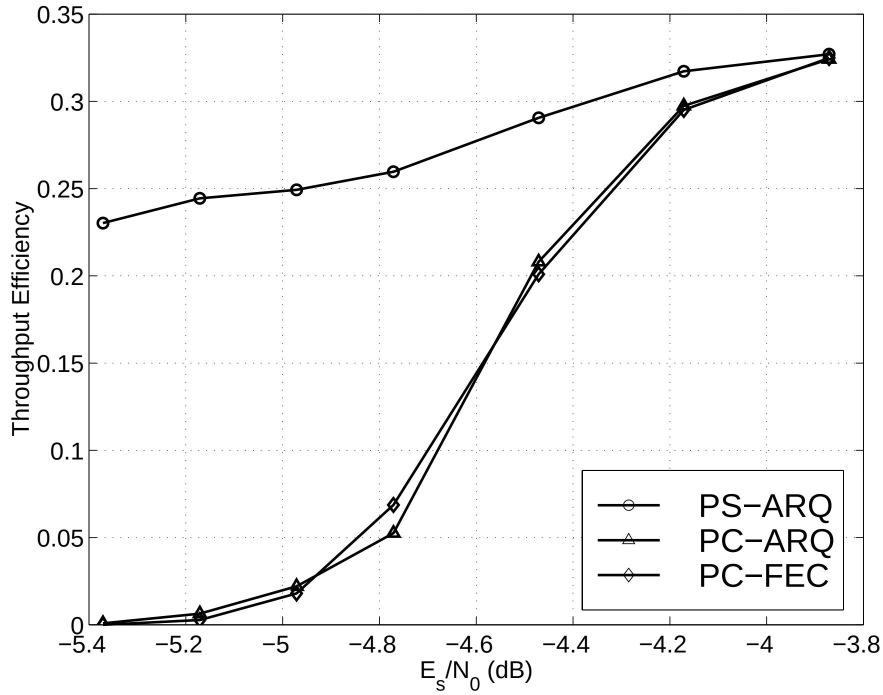



Figure 7.79: Throughput efficiency versus $E_s/N_0$ for AWGN channel.

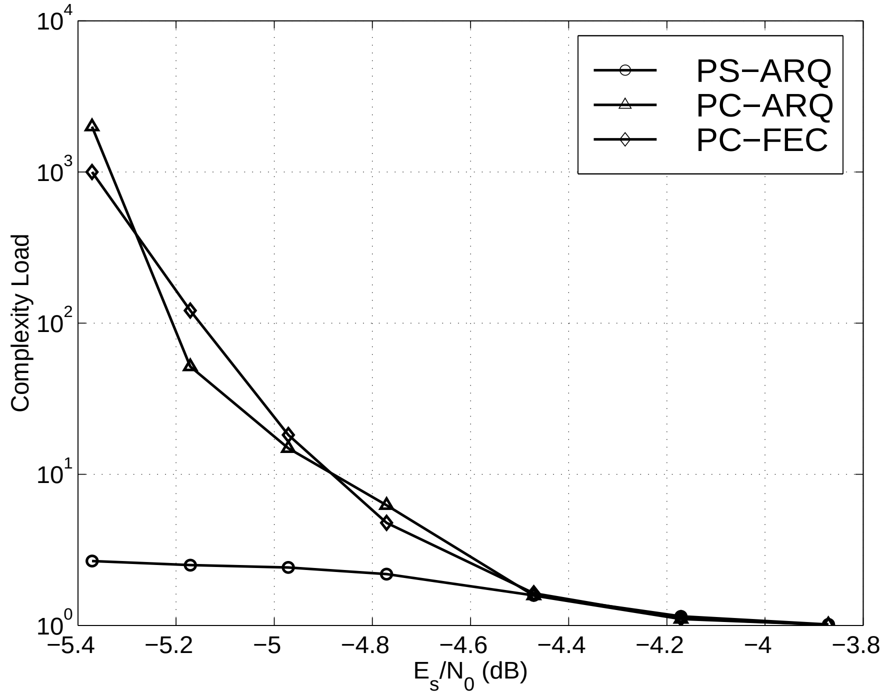



Figure 7.80: Complexity load versus $E_s/N_0$ for AWGN channel.

## 7.5 Summary

In this chapter, a type-II hybrid ARQ technique has been presented which uses a generalized parallel/serial concatenated convolutional encoder. Simulation results indicate better BER, FER, and throughput efficiency performance compared to a conventional PCCC or even a PCCC with type-I hybrid ARQ. The proposed technique also benefits from a lower complexity load at low $E_s/N_0$.

A disadvantage of the proposed technique, as with code-combining in general, is the requirement for the receiver to store the initially received codeword sequence whenever a retransmission is requested.

A more thorough comparison should include comparisons with other hybrid ARQ techniques. For ease of exposition, only unpunctured rate 1/4 SCCCs and rate 1/3 PCCCs have been used, but higher-rate, punctured systems should also be considered in future work.

# Chapter 8

# Turbo Code Assisted Synchronization

The analysis and simulations of previous chapters were performed under the implicit assumption of perfect receiver synchronization. It was shown that, under this assumption, turbo codes meet typical practical performance targets (e.g., BERs of $10^{-3}$) at values of $E_b/N_0$ relatively close to the Shannon limit.

However, prior to demodulation and subsequent channel decoding a practical receiver usually has to synchronize to the received signal at several different levels. Unfortunately, at $E_b/N_0$ values close to the Shannon limit, efficient synchronization becomes a serious issue. Straightforward methods usually compromise bandwidth efficiency or result in longer synchronization time constants due to the need of decreasing loop bandwidths to reject noise.

Thus, realization of the dramatic performance gains using turbo codes at low $E_b/N_0$ necessitates the development of improved synchronization techniques which also operate effectively at low $E_b/N_0$.

Following a brief discussion of the types of synchronization in Section 8.1, this chapter presents two methods of frame synchronization in Section 8.3 which take advantage of the soft output of the SISO decoders and the iterative decoding technique. The complexity of these two methods are compared and their performance is evaluated through simulation in Section 8.4.

## 8.1 Types of Synchronization

In digital communication systems, the term "synchronization" can refer to carrier synchronization, symbol synchronization, or frame synchronization. Carrier synchronization occurs when the receiver can regenerate a local carrier which is identical in frequency and phase to the carrier of the received signal, and is required in systems employing coherent demodulation. Symbol synchronization occurs when the receiver can regenerate a symbol clock signal which is identical in frequency and phase to the symbol timing of the received signal, and is almost always required. Frame synchronization occurs when the receiver can regenerate a frame clock signal which is identical in frequency and phase to the frame timing of the received signal, and is required in systems in which the information is organized in frames.

In [80], carrier and symbol synchronization of turbo coding systems were considered with a decision-directed structure.The topic of frame synchronization under conditions of perfect symbol synchronization is addressed in this chapter.

## 8.2 Correlation Synchronization

Correlation synchronization is a common method for achieving frame synchronization. In this method [81] a **sync word** composed of $N_\theta$ symbols, $(\theta_1, \theta_2, ..., \theta_{N_\theta})$, and known to the receiver is transmitted immediately preceding the frame of data, as shown in Figure 8.81. The receiver uses a filter matched to the sync word to detect occurrences of the sync word in the received symbol stream [32]. The symbol stream offset producing the greatest energy out of the matched filter is selected as the maximum likelihood frame boundary. Sync words with autocorrelation functions such that

$$R_\theta(k) \begin{cases} = 1, & \forall k = 0 \\ \approx 0, & k \neq 0 \end{cases} \tag{8.244}$$

should be chosen to enhance the matched filter performance. Barker sequences [81] and PN sequences [32] are examples of sequences with desirable autocorrelation properties.

The advantages of correlation synchronization include its simplicity and speed. Unfortunately, as $E_b/N_0$ decreases, if the power of the sync word is kept constant, then the length of the sync word must increase to keep the probability of false detection small, thus compromising bandwidth efficiency. An alternative, which was not

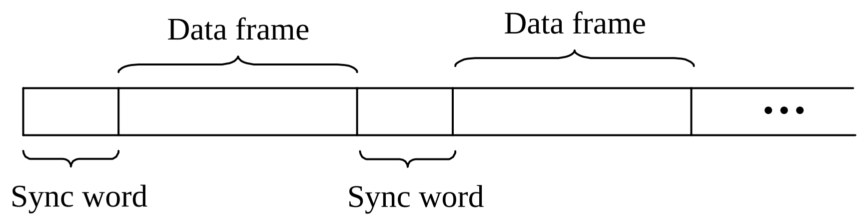



Figure 8.81: Format of the data stream with frame sync word.

explored in the following, is to increase the power of the sync word at the expense of decreasing the power of the data frame in such a way as to maintain the same total power. However, this alternative can create in-band and out-of-band interference due to the gating effect and may be undesirable in secure communication systems.

## 8.3 List Synchronization

Two synchronization techniques for the low $E_b/N_0$ region are explored in this section. These techniques supplement correlation synchronization by exploiting information produced by the turbo decoder. Both techniques begin by generating a list of potential frame boundaries developed by correlation, hence the rubric "list synchronization." The turbo decoder is then used to choose the most probable frame boundary from this list. In the following the general procedure is presented first, followed by presentations of the two list synchronization procedures.

### 8.3.1 General Procedure

Assume that composite frames composed of $N_\theta$-symbol sync words followed by an $N_f$-symbol data frames are transmitted, and let $y_i, i \in \{0, \ldots, N_\theta + N_f - 1\}$, denote a frame of sequentially received soft values in which the true frame boundary is unknown. In a list synchronization procedure, the sequence $y_i$ is convolved with the sync word to obtain a full set of $(N_\theta + N_f)$ correlation outputs corresponding to the $(N_\theta + N_f)$ possible offsets between the sequence of received soft values and the sync word. The offsets of the $N_\xi$ highest correlation outputs are located and sorted to form the set $\xi = \{\xi_1, ..., \xi_{N_\xi}\}$, where $\xi_1$ denotes the offset of the largest correlation output. The turbo decoder then decodes a frame of data at each offset in the set $\xi$ as if it were the proper starting symbol of the frame. The soft outputs of the turbo decoder are then used to determine which offset produces the "best" results. Note that correlation synchronization can be considered as a special case of list synchronization where the set $\xi$ degenerates to a single element.

To make the list synchronization method work well, the parameters $N_\theta$ and $N_\xi$ have to be chosen carefully. $N_\theta$ has to be large enough so that the true frame boundary $\xi_{true}$ is included in the set $\xi$ with high probability, yet small enough so that bandwidth efficiency is not compromised. Since turbo decoding is much more complex than correlation, the average complexity of list synchronization is directly related to $N_\xi$. Therefore, $N_\xi$ has to be chosen small to keep the complexity small, yet large enough so that $\xi_{true}$ is included in $\xi$ with high probability.

The complexity of the list method is higher than that of the correlation method, but lower than using soft information only to synchronize, as presented in [82] or [83], thus providing a good compromise between complexity and performance. Two variations of the list methods which were developed and tested in this research are discussed in the following.

### 8.3.2 LLR Synchronization Procedure

The "**LLR sync**" frame synchronization procedure is illustrated in Figure 8.82. At the transmitter, $N_\omega$ bits $(\omega_1, ..., \omega_{N_\omega})$ known to the receiver are added in front of each length-$N$ information frame, and the extended frame of length $(N_\omega + N)$ is turbo encoded. Note that the $N_\omega$ header bits are permuted by the turbo encoder interleaver and encoded by RSC2. As a result, the parity bits from RSC2 corresponding to $(\omega_1, ..., \omega_{N_\omega})$ are unpredictable. Hence, these bits are punctured from the output stream since they cannot be used by the receiver for the correlation step. Let $N_\theta$ denote the number of bits in the sync word and $N_f$ denote the number of code bits from the turbo encoder not corresponding to the $N_\omega$ header bits.

The receiver uses the systematic and parity bits from RSC1 corresponding to $(\omega_1, ..., \omega_{N_\omega})$ as the sync word. A list $\xi$ of the $N_\xi$ offsets having the largest correlation with the sync word is generated, as discussed in Section 8.3.1. For each $\xi_i \in \xi$, a turbo decoder is applied to the soft values $(y_{\xi_i}, ..., y_{\xi_i+N_f+N_\theta-1})$.

For each application of the turbo decoder, the logarithm of the *a posteriori* probability $P((\omega_1, ..., \omega_{N_\omega})|Y_1^N)$ is computed as follows. Let $(\lambda_1^A(\omega; O), ...., \lambda_{N_\omega}^A(\omega; O))$ denote the output log-likelihood ratios from the turbo decoder corresponding to

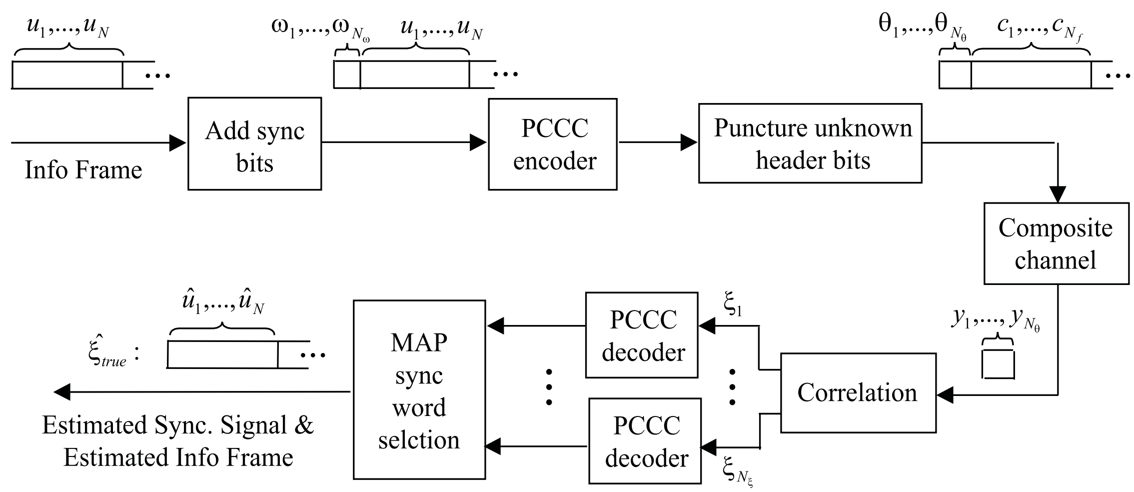



Figure 8.82: The LLR synchronization procedure.

$(\omega_1, ..., \omega_{N_\omega})$. Since

$$\lambda^A(\omega; O) = \ln \frac{P(\omega = 1|Y_1^N)}{P(\omega = 0|Y_1^N)}, \tag{8.245}$$

from Equation (3.75),

$$\begin{cases} P(\omega = 1|Y_1^N) = \frac{\exp(\lambda^A(\omega;O))}{1+\exp(\lambda^A(\omega;O))} \\ P(\omega = 0|Y_1^N) = \frac{1}{1+\exp(\lambda^A(\omega;O))} \end{cases}. \tag{8.246}$$

Assuming mutual independence of $\omega_i$ and taking the logarithm, it follows that

$$\ln P((\omega_1, ..., \omega_{N_\omega})|Y_1^N) = \sum_{i=1}^{N_\omega} \left[\omega_i \lambda_i^A(\omega; O) - \ln(1 + \exp(\lambda_i^A(\omega; O)))\right]. \tag{8.247}$$

Since the logarithm is monotonically increasing, the offset yielding the largest value of $\ln P((\omega_1, ..., \omega_{N_\omega})|Y_1^N)$ represents the maximum *a posteriori* offset and is thus selected as the frame boundary.

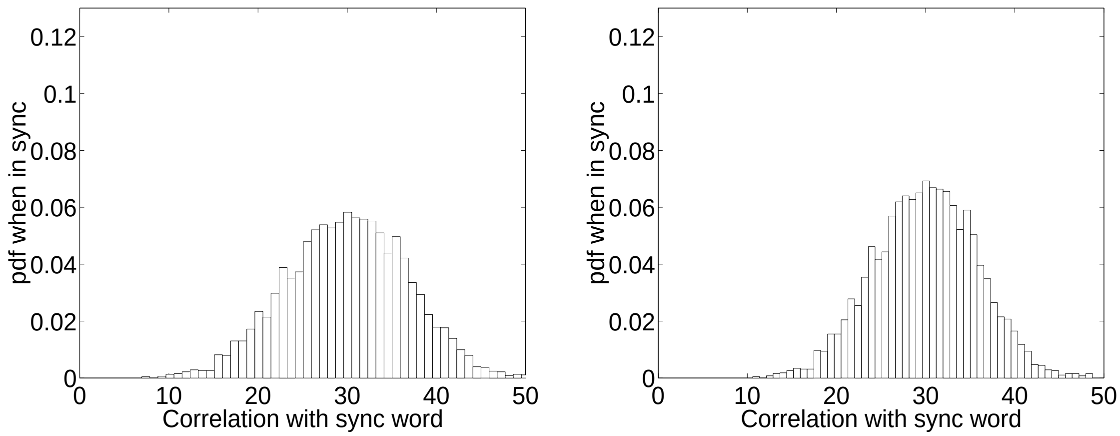
LLR sync requires $N_\xi$ turbo decoding operations and $N_\xi$ computations of the metric $\ln P((\omega_1, ..., \omega_{N_\omega})|Y_1^N)$, which represent a significant increase in the computational complexity in comparison to the simple correlation method.

### 8.3.3 SDR-Based Synchronization Procedure

Examination of the histograms in Figures 8.83 through 8.86 is used to motivate the "**SDR sync**" procedure. Figure 8.83 shows histograms of the correlation between the sync word and the received frame header when the frame is in-sync at $E_b/N_0 = 0.0$

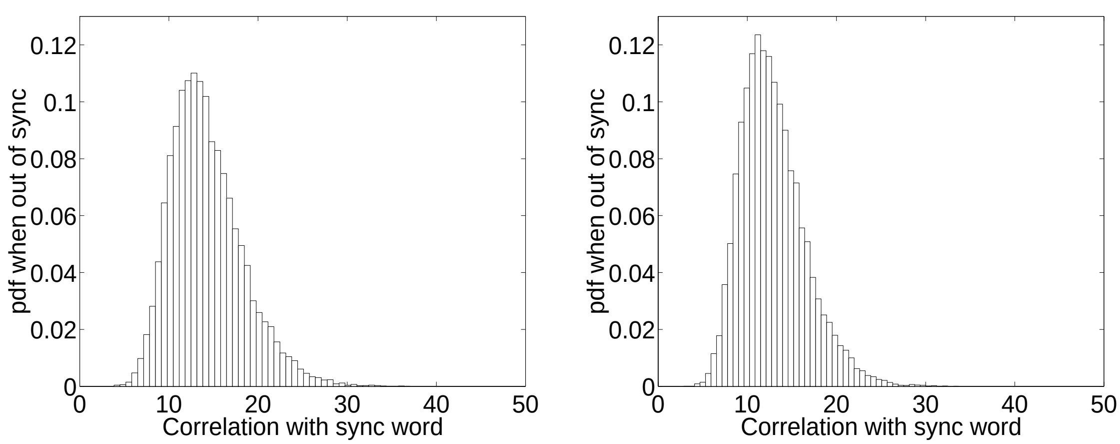
dB and $E_b/N_0 = 1.5$ dB. Likewise, Figure 8.84 shows histograms of the correlation between the sync word and the received frame header when the frame is out-of-sync at $E_b/N_0 = 0.0$ dB and $E_b/N_0 = 1.5$ dB. Significant overlap between in-sync and out-of-sync situations is observed even at $E_b/N_0 = 1.5$ dB. Thus, using correlation alone to pick the sync position leads to a high probability of false alarm.

In contrast, Figure 8.85 shows histograms of the sign differences between the input *a priori* information and the output extrinsic information, $D_2(6)$, of SISO2 on the sixth decoding iteration when the frame is in-sync at $E_b/N_0 = 0.0$ dB and $E_b/N_0 = 1.5$ dB. Likewise, Figure 8.85 shows histograms of the sign differences between the input *a priori* information and the output extrinsic information, $D_2(6)$, of SISO2 on the sixth decoding iteration when the frame is out-of-sync at $E_b/N_0 = 0.0$ dB and $E_b/N_0 = 1.5$ dB. At $E_b/N_0 = 0.0$ dB, $D_2(6)$ spans from $D_2(6) = 0$ to $D_2(6) \approx N/2 = 160$, yet the in-sync case has a much greater distribution at $D_2(6) \approx 0$. At $E_b/N_0 = 1.5$ dB, the distribution of $D_2(6)$ shrinks to $D_2(6) \approx 0$ when in-sync, while $D_2(6)$ is clustered around $N/2$ regardless of $E_b/N_0$. The distinctive nature of the in-sync and out-of-sync histograms of the SDR criterion $D_j(\tau)$ in comparison to the in-sync and out-of-sync histograms of the correlation criterion shows that the SDR criterion should be better for making synchronization judgments than the correlation value.



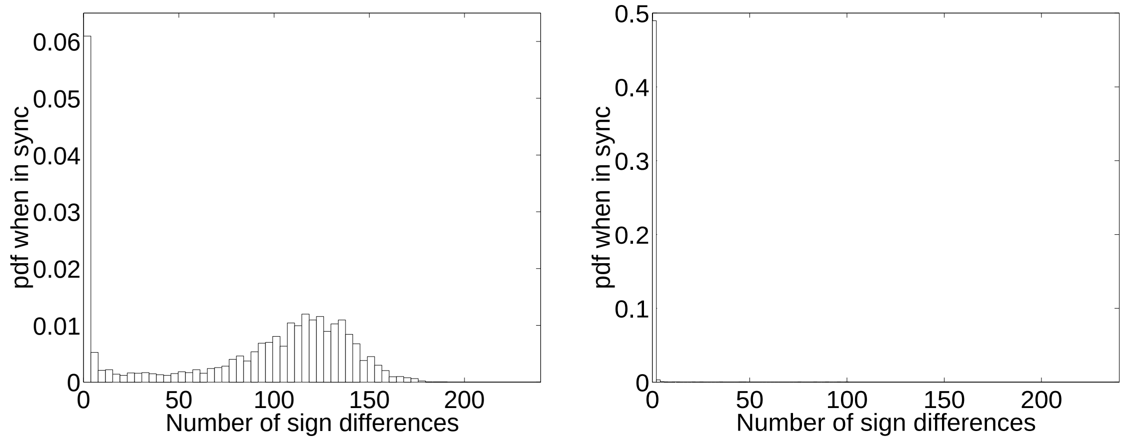

(a) $E_b/N_0 = 0.0$ dB (b) $E_b/N_0 = 1.5$ dB

Figure 8.83: Histogram of the correlation between the sync word and the header of a frame when in-sync. The RSC code generator is $g = (13, 15)_{octal}$, and the size of sync word is $N_\theta = 30$.



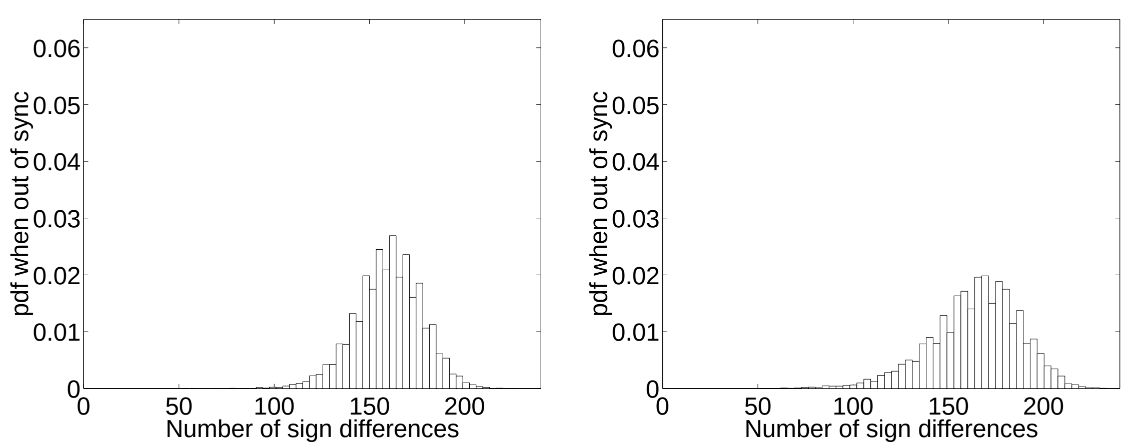

(a) $E_b/N_0 = 0.0$ dB (b) $E_b/N_0 = 1.5$ dB

Figure 8.84: Histogram of the correlation between the sync word and the header of a frame when out of sync. Here the same scheme is used as in Figure 8.83.




(a) $E_b/N_0 = 0.0$ dB (b) $E_b/N_0 = 1.5$ dB

Figure 8.85: Histogram of $D_2(6)$ when PCCC decoder is in sync. The RSC code generator is $g = (13, 15)_{octal}$, the information frame size is $N = 320$, the size of sync word is $N_\theta = 30$, the list size is $N_\xi = 5$, and 5000 frames are transmitted.




(a) $E_b/N_0 = 0.0$ dB (b) $E_b/N_0 = 1.5$ dB

Figure 8.86: Histogram of $D_2(6)$ when PCCC decoder is out of sync with the same structure in Figure 8.85. Here only the positions in list $\xi$ are considered.

Using these histograms as a basis, the SDR sync procedure, illustrated in Figure 8.87, was developed. The transmitter adds an $N_\theta$-symbol sync word to the $N_f$-symbol turbo encoded frame and the receiver obtains list $\xi$ by correlation as stated in Section 8.3.1. The turbo decoder begins at $i = 1$ to sequentially decode each frame of soft symbols $(y_{\xi_i}, ..., y_{\xi_i+N_f+N_\theta-1})$, and after each decoding attempt the number of sign differences $D_2(\tau)$ is determined, where $\tau$ is the number of turbo decoding iterations. The first $\xi_i$ for which $D_2(\tau) < q_d N$ is satisfied is declared to be the frame boundary, and if no such $\xi_i$ is found, then $\xi_1$ is declared to be the frame boundary. In Chapter 6, it was found that when $D_j(\tau) < q_d N$, $10^{-3} < q_d < 10^{-2}$, the decoder has most probably decoded the frame correctly and the decoder can stop. For synchronization purposes, where a few estimated bits may be in error even when the frame is in sync, $q_d$ should be set larger. Experimental results indicated that $q_d = 0.2$ is satisfactory.

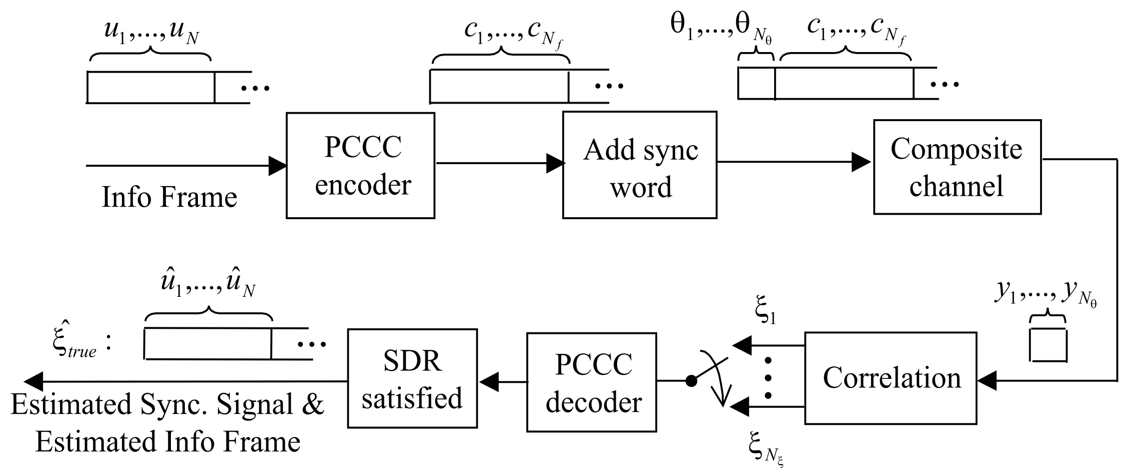



Figure 8.87: The SDR synchronization procedure.

Note that instead of applying the turbo decoder at every $\xi_i$ as in LLR sync, the SDR sync algorithm will often halt before scanning the entire set $\xi$. Consequently the computation load and delay for SDR sync are smaller than those of LLR sync. In particular, when SDR sync stops at $\xi_1$, no decoding operation is wasted.

A more sophisticated method of SDR sync might adapt $N_\xi$ to the channel conditions in order to achieve a better compromise between performance and complexity. To understand this, let $P_{b,sync}(e)$ denote the decoded bit error probability when both perfect and imperfect synchronization are taken into account. Furthermore, let $P_b(e)$ denote the decoded bit error probability under conditions of perfect synchronization and $P_{\overline{sync}}$ denote the probability of false sync. Then

$$\begin{aligned} P_{b,sync}(e) &\simeq P_b(e)(1 - P_{\overline{sync}}) + 0.5P_{\overline{sync}} \\ &\simeq P_b(e) + 0.5P_{\overline{sync}}, \end{aligned} \tag{8.248}$$

when $P_b(e) \ll 0.5$.

When $P_b(e)$ is high, $P_{\overline{sync}}$ will also be high and $D_2(\tau) < q_d N$ will occur only rarely. Under this condition there is a high probability that decoding attempts will be made at all $N_\xi$ offsets in $\xi$, thus leading to high computational complexity. In this case, $N_\xi$ should be set relatively small to save computation. It was found empirically that this concept is valid provided that $P_{\overline{sync}} < 0.5P_b(e)$ so that the composite BER $P_{b,sync}(e)$ is not dominated by $P_{\overline{sync}}$.

On the other hand, when $P_b(e)$ is low, Equation (8.248) indicates that $P_{\overline{sync}}$ must also be low to maintain adequate performance. Thus $N_\xi$ needs to be larger in order to ensure that $\xi$ contains the offset of the true frame boundary. In general, this will not incur a significant increase in the average complexity since the entire list $\xi$ will only need to be scanned on rare occasions. This assertion is corroborated in Figure 8.95, where it is shown that at high $E_b/N_0$ the average number of decoding attempts per frame is approximately one.

## 8.4 Simulation Results

Simulations were carried out to test the performance of the synchronization techniques introduced above. In the following figures, "Corr." refers to the correlation method, "LLR" refers to the LLR sync scheme, "SDR" refers to the SDR sync scheme, and "GENIE" refers to perfect synchronization scenario. Synchronization of PCCC with code generator $g(D) = [1, 1 + D + D^3/1 + D^2 + D^3]$ was studied in AWGN channel. Randomly generated sync word was used in the simulation, although a well-designed sequence may bring in better performance. Without loss of generality, an ambiguity interval of $N_\Delta = 20$ symbols was assumed to speed up the simulation, thus limiting the possible sync positions to the region $[\xi_{true} - N_\Delta, \xi_{true} + N_\Delta - 1]$.

Figures 8.88 and 8.89 are for rate 1/3, $N_\omega + N = 320$ PCCC with a list of length 10. The sync word size is $N_\theta = 30$, which is $30/957(= 3.13\%)$ of the entire frame. Checking the curves of false sync rate in Figure 8.88, both LLR sync and SDR sync methods show steeper slope than the conventional correlation method. Since

$P_{\overline{sync}} < 0.5P_b(e)$ is maintained, BER curves of both LLR sync and SDR sync show little degradation from GENIE, as shown in Figure 8.89. Furthermore, there is no noticeable degradation of FER performance. With the same size of sync word, the correlation method has much worse BER/FER performance except at very high BER (BER$= 0.1$).

Similar conclusions can be drawn from Figures 8.90 and 8.91 for rate 1/2 PCCC. When the frame size is larger, e.g., $N_\omega + N = 2048$ in Figures 8.92 and 8.93, superior performance of SDR sync method in comparison to the correlation method is observed with sync word size equal to $40/6136 (= 0.65\%)$ the entire frame. For $N_\omega + N = 2048$, LLR sync method was not tested due to the extremely long simulation time.

Figure 8.94 demonstrates the influence of parameters on the false sync rate. Figure 8.95 compares the complexity among different schemes where the average number of decoding operations is used as the indicator. Since $\tau = 6$ was simulated, one decoding operation was composed of six iterations.

Based on Figure 8.88 to 8.95, the following conclusions are drawn:

- Both LLR sync and SDR sync steepen the curve of false sync rate and consequently conserve the good BER/FER performance of turbo codes, in comparison to the correlation method.

- SDR sync makes slightly better synchronization decisions than LLR sync. This is because SDR sync makes use of the soft information from the whole frame, while LLR sync uses only soft information related to the sync word.

- Increase of $N_\theta$ shifts down the curve of false sync rate without changing the slope, as shown in Figure 8.94.

- Increase of list length $N_\xi$ steepens the false sync rate slope for SDR sync, as shown in Figure 8.94.

- Variable list length, which increases with the increase of $E_b/N_0$, reduces the average number of decoding operations at high BER without degrading the performance, as shown in Figures 8.94 and 8.95.

As shown in Figure 8.95, the average number of decoding operations in BER$\leq 10^{-3}$ region is less than 1.2 when using SDR sync. However, LLR sync always performs $N_\xi$ (10 in Figure 8.95) decoding operations before it can make a decision. Since SDR sync requires lower computation load to achieve a better performance, it is recommended as the frame synchronization technique for turbo coding systems.

## 8.5 Summary

In this chapter, the frame synchronization techniques for turbo coding systems have been studied. The list method was developed with a fine synchronization stage in addition to the conventional correlation synchronization. Two variations of the list method, LLR sync and SDR sync, were introduced. The performance of turbo coding systems using LLR sync and SDR sync was examined with simulation in comparison to the correlation synchronization and the perfect synchronization. It was shown that at the expense of computational complexity, both LLR sync and SDR sync performed order of magnitude better than the correlation method with the same sync word size. Furthermore, SDR sync exhibited better performance with lower complexity than LLR sync.

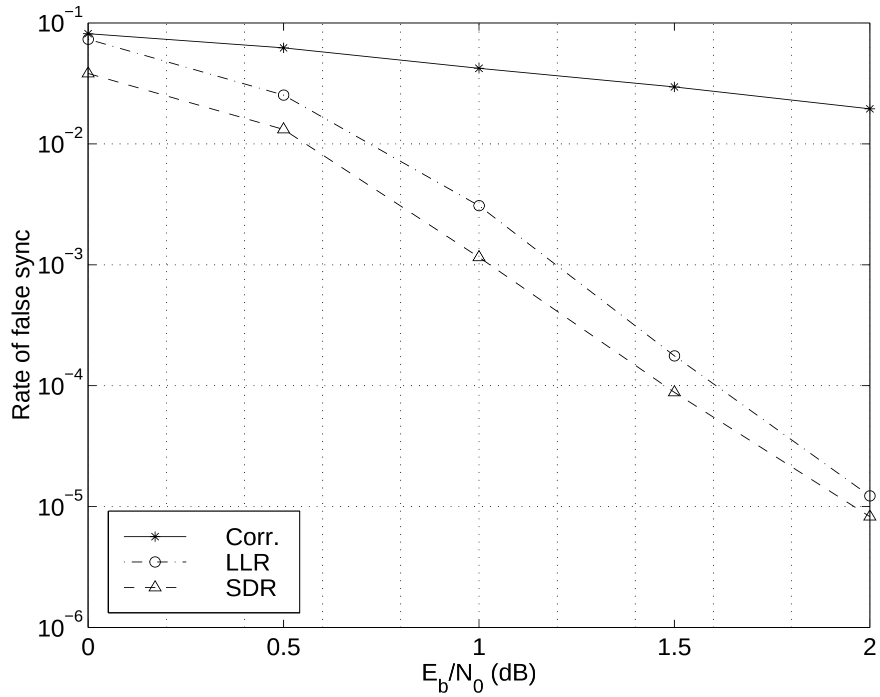



Figure 8.88: False alarm rate with $N_\omega = 15$ ($N_\theta = 30$), $N = 305$, $N_\Delta = 20$, $N_\xi = 10$, rate 1/3, code generator $g = (13, 15)_{octal}$, and 6 iterations per decoding operation.

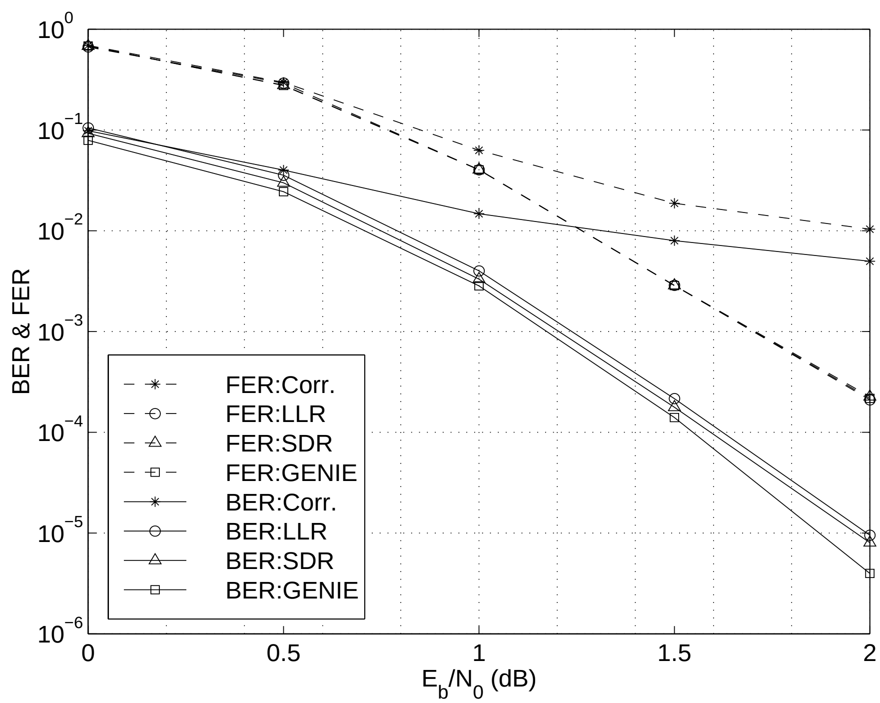



Figure 8.89: BER and FER with the same setting as Figure 8.88.

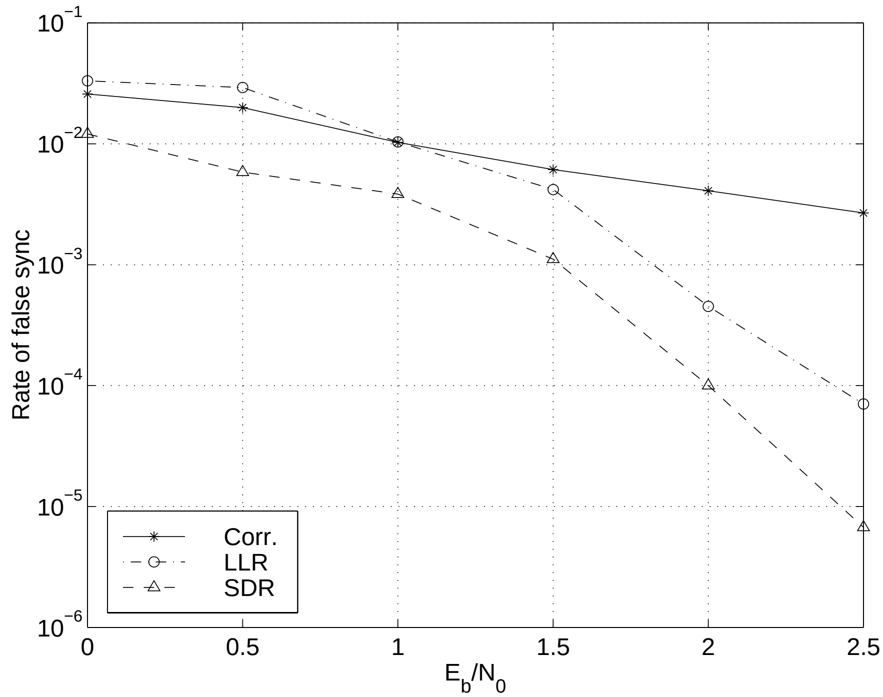



Figure 8.90: False alarm rate with $N_\omega = 20$ ($N_\theta = 30$), $N = 300$, $N_\Delta = 20$, $N_\xi = 10$, rate 1/2, code generator $g = (13, 15)_{octal}$, and 6 iterations per decoding operation.

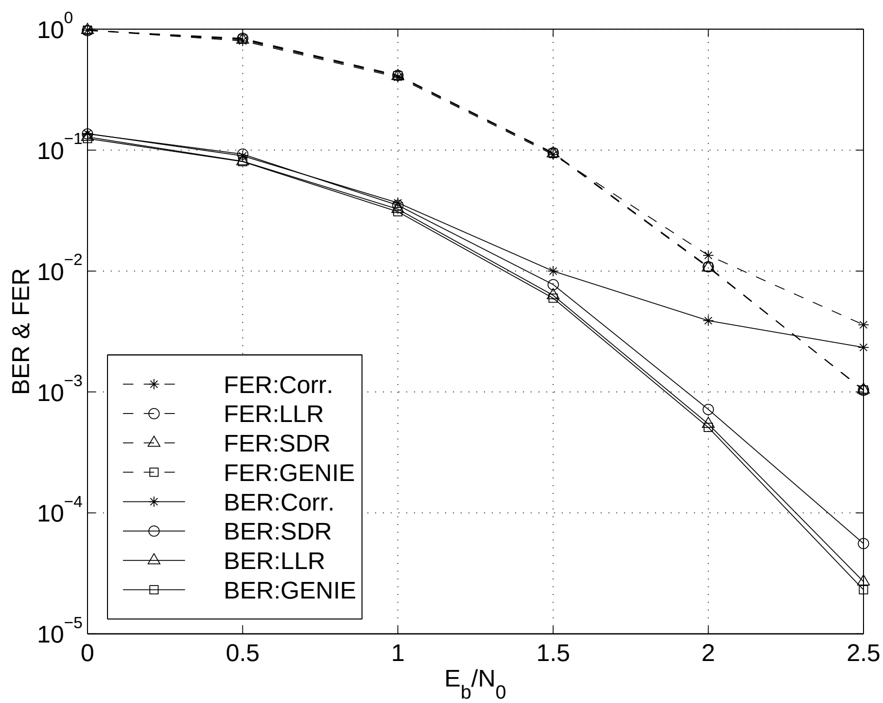



Figure 8.91: BER and FER with the same setting as Figure 8.90.

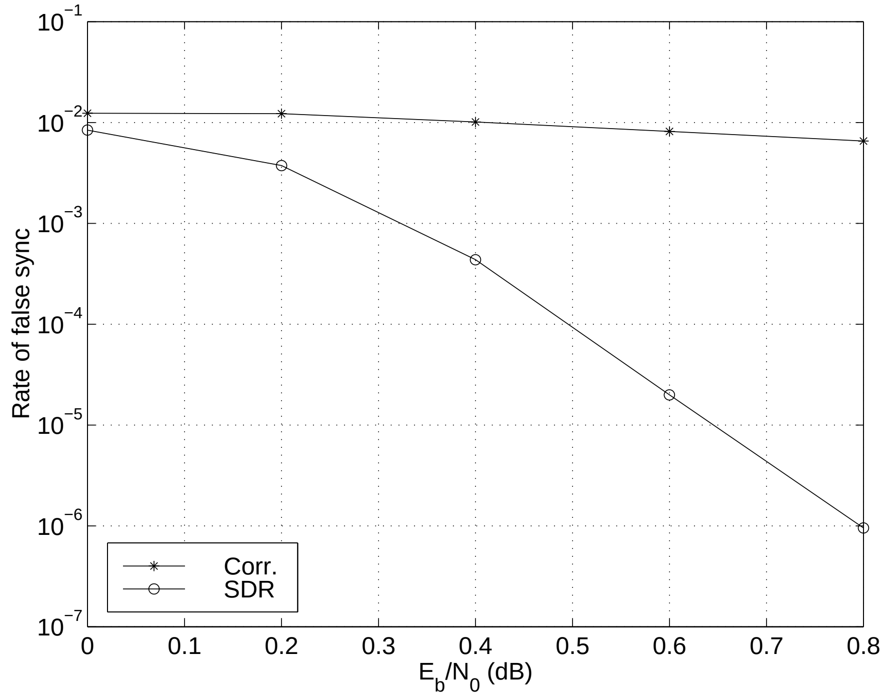



Figure 8.92: False alarm rate with $N_\omega = 20$ ($N_\theta = 40$), $N = 2028$, $N_\Delta = 20$, rate 1/3, code generator $g = (13, 15)_{octal}$, 8 iterations per decoding operation, and $N_\xi = 3, 6, 8, 10, 12$ for $E_b/N_0 = 0.0, 0.5, 1.0, 1.5$ dB, respectively.

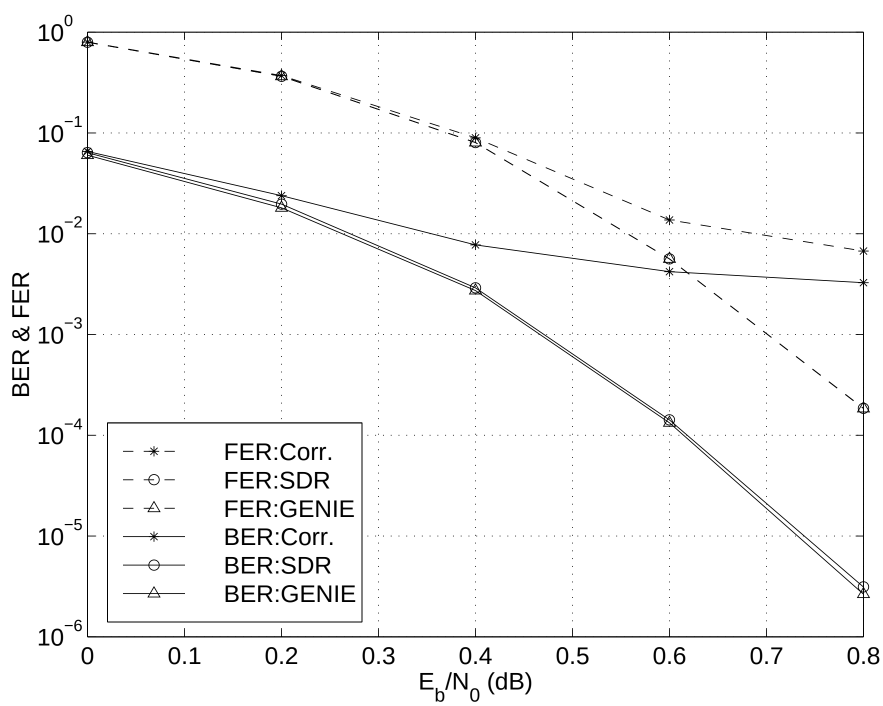



Figure 8.93: BER and FER with the same setting as Figure 8.92.

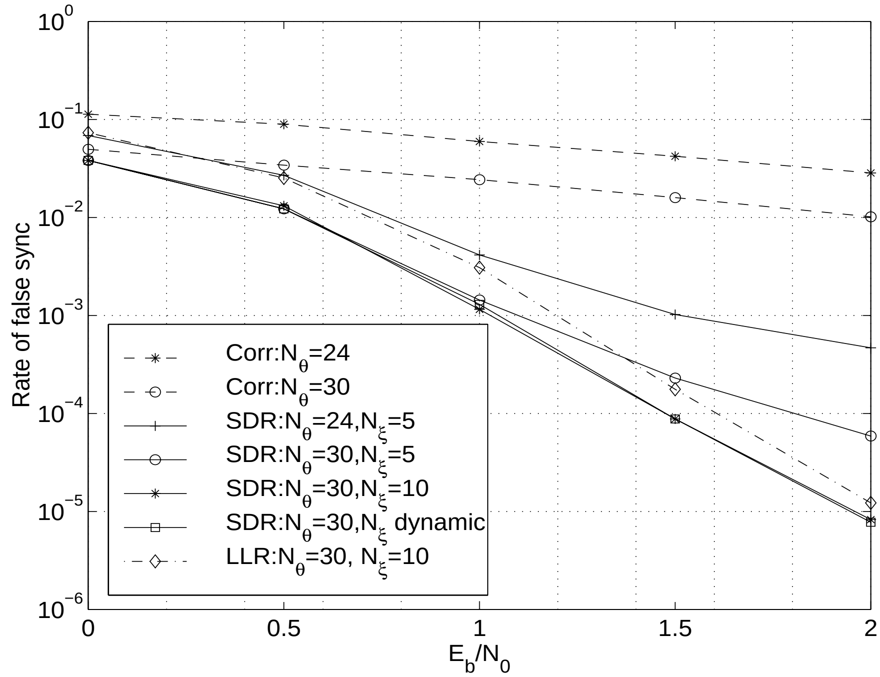



Figure 8.94: Influence of parameters on false alarm rate with $N_\omega + N = 320$, rate 1/3, code generator $g = (13, 15)_{octal}$, and 6 iterations per decoding operation. $N_\xi = 3, 6, 8, 10, 12$ for $E_b/N_0 = 0.0, 0.5, 1.0, 1.5$ dB, respectively, for the dynamic $N_\xi$ case.

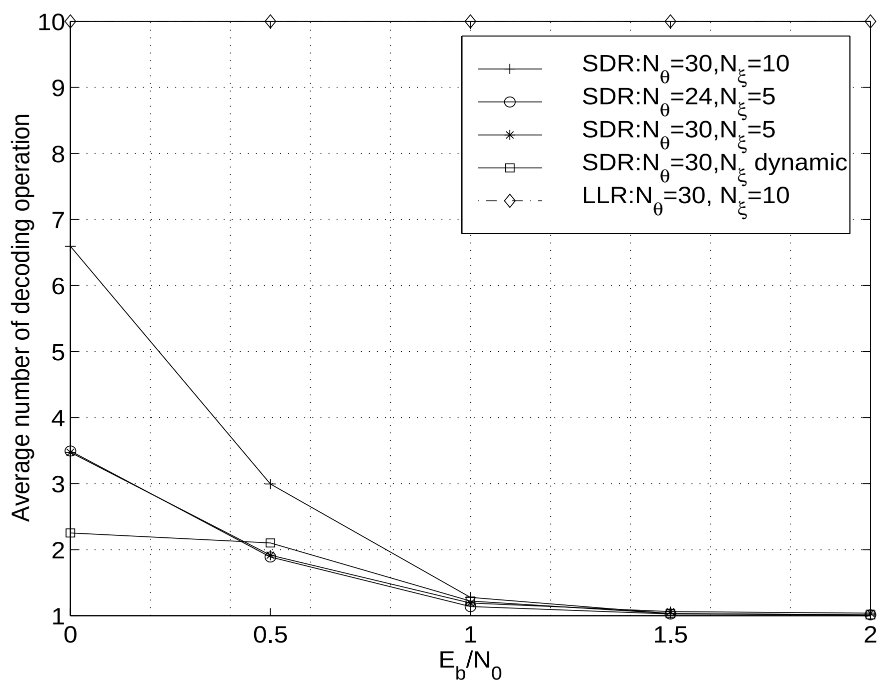



Figure 8.95: Influence of parameters on the average number of decoding operations with SDR sync and the same setting as Figure 8.94.

# Chapter 9

# Implementation of a Real-Time Turbo Decoder

Previous chapters of this dissertation have established the performance of various turbo codes in the AWGN channel. Furthermore, efficient decoding algorithms have been presented and practical issues such as the effect of fixed point arithmetic in the turbo decoder have been studied.

In order to investigate issues encountered in implementing turbo decoders in real-world systems, a prototype real-time turbo decoder was developed. Since this research was performed under the auspices of the GloMo program at Virginia Tech, among the goals of which is the application of reconfigurable baseband processing in wireless communications systems, the turbo decoder was targeted to a FPGA.

This chapter summarizes the work reported in [84]. The real-time turbo decoder design procedure is described from the high level design down through the low level design and realization. In addition, ideas for potential improvements to the design are supplied.

## 9.1 Design Methodology

Typically, in practical wireless systems, requirements would be placed on the throughput and BER/FER performance based on such factors as range, interference, power consumption, and quality of service. A phase of systems engineering studies would commence to select coding schemes and algorithms capable of meeting the minimum requirements. The research presented in previous chapters of this dissertation could rightly be construed as representing the basic systems engineering studies for the real-time turbo decoder.

Systems engineering studies for this design are characterized by the development of computer-based models using the C programming language and/or Signal Processing Workstation (SPW) [85] development system. For brevity, only the SPW modeling is described below since the results presented previously in this dissertation were obtained from C-language models. At the start, the algorithms are developed using floating point arithmetic to study the coding schemes using highly precise arithmetic. This work was presented in Chapters 3 and 4.

Since floating point arithmetic is generally not suitable for low-power real-time implementations, models using fixed point arithmetic were developed from the floating point models. The fixed point models are also used to produce input and output test vectors by which the hardware can be verified. This work was presented in Chapter 5.

Much research is currently focused on decreasing the hardware design cycle through the use of schematic capture tools such as SPW or COSSAP [86] and the use of C-to-HDL translators. Unfortunately, using these tools for the entire design cycle is not yet a mature technique and it requires a significant investment of learning time. Therefore, hardware developers generally resort to manual hardware description language (HDL) programming following the development of efficient fixed point algorithms. This is the approach that was followed in this research.

In-situ performance prediction of hardware is accomplished by using realistic channel models in the C-language simulations and verifying that the hardware produces the exact numerical results as the fixed point C-language models. This can either be accomplished at the stage of HDL coding by using the HDL testbenches or the prototyping hardware. In this research, verification was done using the prototyping hardware, which had a personal computer development environment that allowed the external memories on the prototyping board to be read using C-language subroutine calls.

## 9.2 High Level Design

### 9.2.1 Requirements and Encoder-Decoder Specification

As the purpose of the exercise was oriented towards research into implementation details no specific BER or FER criteria were established. However, for the experimental GloMo system under development at Virginia Tech BERs smaller than $< 10^{-6}$ are not required. Furthermore, an information rate of 32 kbps was stipulated.

Therefore, based upon the studies in Chapter 4, a PCCC with rate 1/3 (no puncturing) or 1/2 (puncturing) was selected. To keep the implementation complexity relatively low, a constraint length three constituent code was selected. As shown in Chapter 4, given such an encoder the generator $g(D) = [1, 1 + D^2/1 + D + D^2]$ provides the best performance, and therefore was selected. Although it was decided that the implementation should allow for flexible frame sizes, the design was verified for a frame size of $N = 1024$. It was decided that interleaver was to be implemented as a lookup table. Therefore, a spread interleaver [87] [63] was designed with a C-language program and stored in the decoder memory. For termination, the trellis of RSC1 was terminated to the all-zero state with two tail bits while that of RSC2 was left indeterminate.

For decoding, the max-Log-MAP algorithm was selected so as to avoid the difficulties associated with implementing the nonlinear term of the $\max^*(\cdot)$ operator. Although it was decided that a flexible number of decoding iterations was to be accommodated, the implementation was verified for six decoding iterations. Based upon the studies of Chapter 5, eight-bit signed numbers were used to represent the received symbols from channel, while all quantities internal to the decoder were represented as 16-bit signed numbers.

### 9.2.2 SPW Models

SPW is used in industry and academia for simulating digital communication system designs and quickly translating these designs into hardware and/or software. In SPW, systems are specified by block diagrams composed of various processing blocks with inputs and outputs through which the signals flow. A group of interconnected blocks which work together as a functional unit can be "packaged" and used as a single block at a higher level. In this way the entire system can be built up in a hierarchical fashion.

SPW models were developed for both PCCCs and SCCCs in order to investigate their implementation issues. For brevity, only the rate 1/2 PCCC model is discussed here.

Figure 9.96 shows the SPW model for the encoder, BPSK modulator, and additive channel noise, culminating in the received signal 'rec_ch'. Figure 9.97 shows the SPW model for the input LLR generator which scales 'rec_ch' by the channel reliability 'L_c' and the demultiplexer which splits the input LLRs into two data streams, one for SISO1, 'Lc_i1' , and the other for SISO2, 'Lc_i2'.

Figure 9.98 shows the SPW model for the rate 1/2 PCCC decoder. Multiple instantiations of the SISO1/SISO2 block, one per iteration, are employed. Although four iterations are used in Figure 9.98, any number of iterations may be used by linking the appropriate number of blocks. Hard decisions are made following the last iteration. The SPW model for a SISO1/SISO2 block is shown in Figure 9.99.

Figure 9.100 shows the SPW model for the postprocessor that compares 'bit_rx', the decoded bits from the decoder, with 'bit_tx', the actual source bits and records errors. The postprocessing results are graphically displayed while a simulation runs.

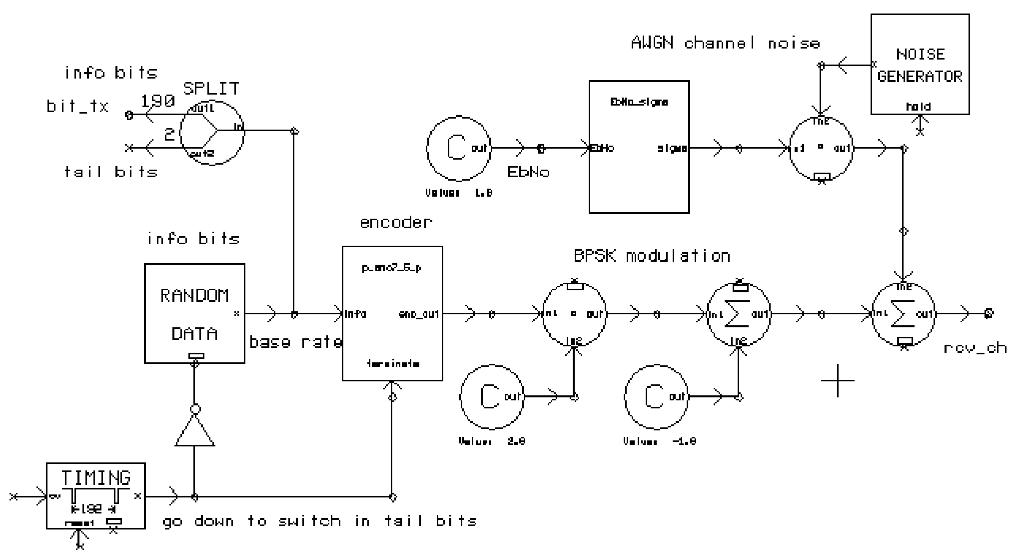



Figure 9.96: SPW model of the transmitter for a rate 1/2 PCCC system.

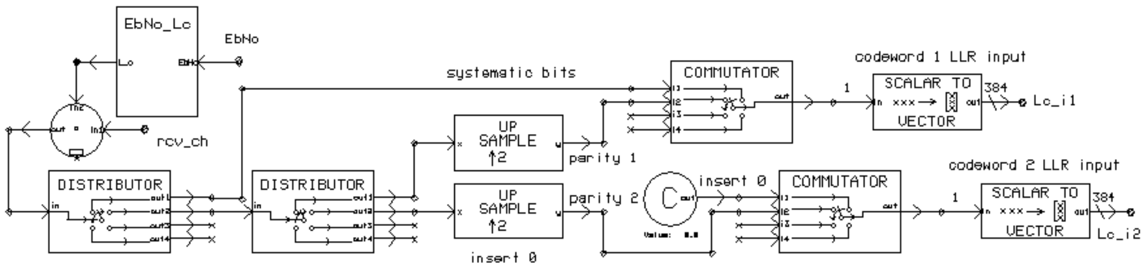



Figure 9.97: SPW model of the LLR generator and demultiplexer of a rate 1/2 PCCC system.

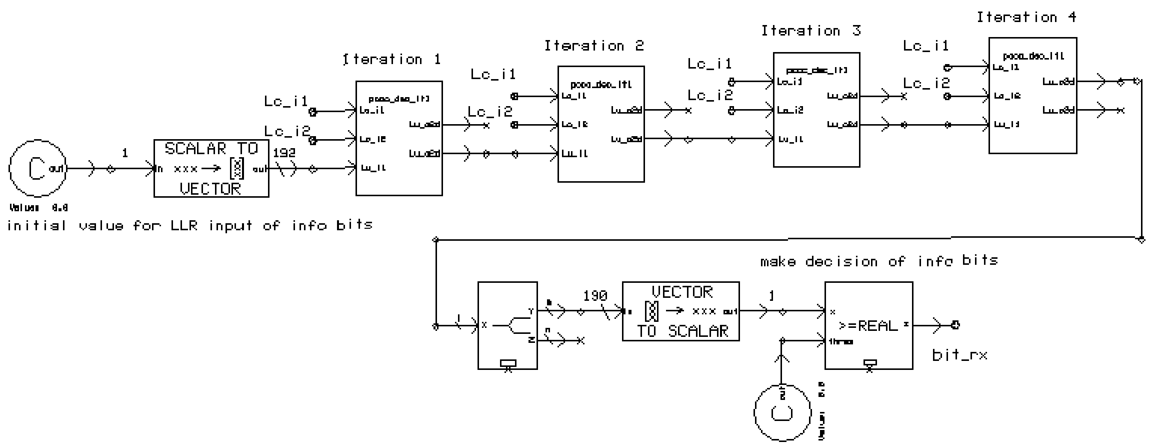



Figure 9.98: SPW model of the decoder for a PCCC system.

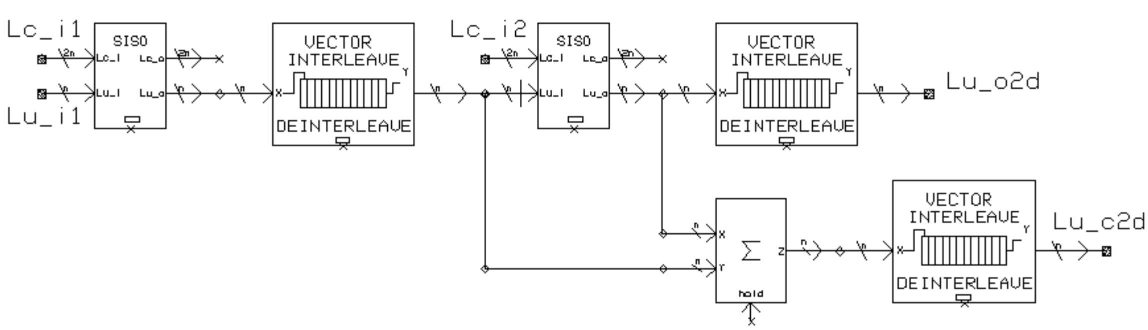



Figure 9.99: SPW model of one decoding iteration of PCCC.

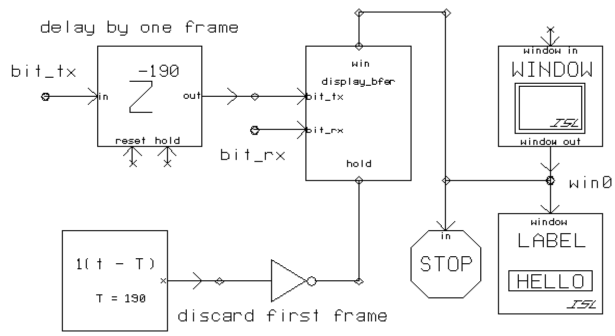



Figure 9.100: Postprocessor for the SPW PCCC system.

## 9.3 Low Level Design and Results

### 9.3.1 Prototyping Hardware and Software

The decoder was prototyped on a WILDFORCE board from Annapolis Micro Systems [88], which hosts five Xilinx 4062XL-3 FPGAs and the external memory. The WILDFORCE board was configured and controlled by a host personal computer (PC) through an application programming interface (API). Since all the simulation modules and chip interfaces of the WILDFORCE board are written in Very High Speed Integrated Circuit (VHSIC) HDL (VHDL), it was adopted as the language by which the turbo decoder logic was designed.

The Xilinx 4062XL-3 FPGA consists of a grid of 2,304 interconnected configurable logic blocks (CLBs). Each CLB consists of two four-input lookup tables (LUTs), a three-input LUT, two D-type flip-flops (FFs), and several multiplexers that are used for internal routing of signals in the CLB. A set of input/output buffers (IOBs) provide the interface between the internal logic of the FPGA and external components such as memories. The FPGA can be configured within a few milliseconds by loading an internal static random access memory (SRAM) known as the configuration memory.

### 9.3.2 Decoder Architecture and Design Flow

Using the fixed and floating point C and SPW models described earlier and an understanding of the FPGA and WILDFORCE architecture, the decoder was partitioned into functional modules to facilitate design and verification. This high level structure is shown in Figure 9.101, and the state diagram of the decoder module, which is slightly different for constituent decoder one and decoder two, is shown in Figure

9.102.

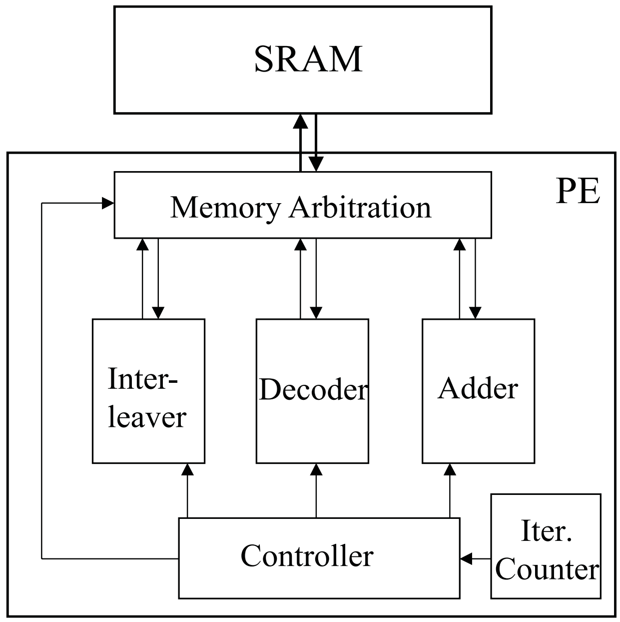



Figure 9.101: Top-level diagram of turbo decoder implementation.

The various modules of the turbo decoder were manually coded and simulated in VHDL. Workview Office from Viewlogic [89] was used as the VHDL simulator to verify the function of the design. FPGA Express from Synopsys [90] was used to synthesize the validated VHDL design to obtain a netlist. Finally, the Xilinx M1 software [91] was used to map the netlist into the FPGA and obtain the configuration code. The synthesis and mapping tools also performed various optimizations for the targeted device to ensure efficient usage of the FPGA.

The host PC was responsible for downloading configuration code to the FPGA board, generating source data, simulating the modulation and the channel, loading vectors for decoding to the external memory, postprocessing the soft output of the turbo decoder, and validating the decoder. In other words, among all the SPW blocks shown in Figure 9.96 to 9.100, only the decoder model in Figure 9.98 and 9.99 was implemented in hardware, and the rest was taken care of by the host C code. A large part of the host code was adapted from the C-language software written during the high level design phase.

Verification was accomplished by generating coded noisy test vectors which were fed both to the FPGA and to the fixed point C-language software written in the high level design phase. Debugging was carried out on the host code and VHDL code until

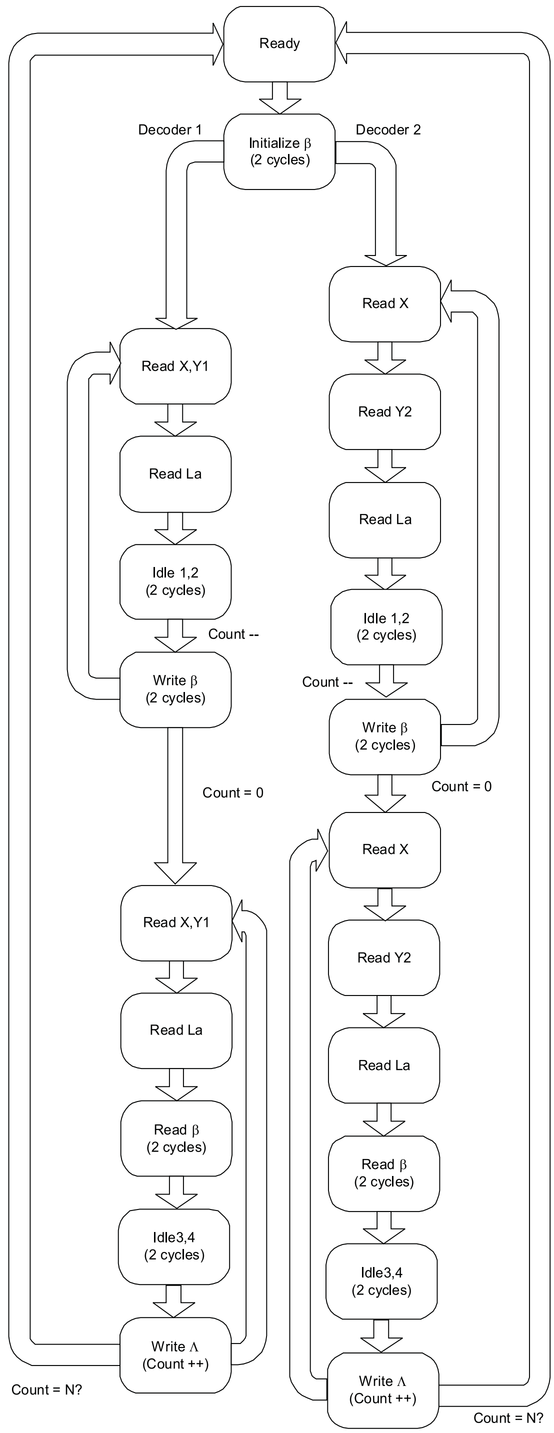



Figure 9.102: State diagram of the turbo decoder module.

the data produced by the FPGA exactly matched that produced by the C-language routines.

### 9.3.3 Results

Although there were five FPGAs on the WILDFORCE board, the turbo decoder design only required one FPGA. The design utilized about 3/4 of the FPGA's logic resources. Therefore, since external memory was shared between the various modules, a memory arbitration unit, as shown in Figure 9.101, was used to resolve memory access conflicts.

The decoder required about 50 clock cycles/iteration/information bit. Furthermore, the FPGA produced correct output at clock rates of up to 11.5 MHz. Therefore, for six decoding iterations, the measured throughput reached 38.9 kbps, which is higher than the target rate of 32 kbps.

For more details on the FPGA implementation of the turbo decoder, the reader is referred to [84].

## 9.4 Improvements

In the following four changes to the turbo decoder design presented above are discussed. The first three changes would allow some hardware logic or clock cycles to be eliminated at no expense to the decoding quality. The last change would improve the BER/FER performance of the decoder at the expense of a slight increase in hardware logic.

First, the design presented above has the input/output structure of the conventional Log-MAP decoder as shown in Figure 3.17 and discussed in Section 3.3. The "Adder" module in Figure 9.101 was used to obtain the extrinsic information by subtracting both the *a priori* information and the systematic information from the complete information. If the simplified structure shown in Figure 3.18 and discussed in Section 3.4 is adopted, then the extrinsic information would be calculated directly using Equation 3.118. This would allow removal of the "Adder" module in Figure 9.101.

Second, as discussed in Section 3.4, in an iterative decoding scheme the systematic information need not be removed from the output of the first decoder. Unfortunately, this is exactly what the design discussed above does. Referring to Figure 9.102, the extra cycle required to fetch "X" for decoder two is thus unnecessary. Furthermore, the cycle required to interleave "X" before passing it to decoder 2 is also unnecessary. Thus, adoption of the simplified scheme discussed in Section 3.4 would eliminate clock cycles, thus increasing the throughput of the decoder.

Third, currently the subtractive normalization technique of Equations (3.101) and (3.102) in Section 3.3.2 is used for the $\alpha$ and $\beta$ recursions. If the modular renormalization technique developed in Section 5.2.3 is adopted, then subtractive normalization is unnecessary. This observation would allow elimination of six $\max(\cdot)$ circuits and $2^{m+1}$ subtraction circuits.

Fourth, for simplicity, the Max-Log-MAP algorithm instead of the Log-MAP algorithm was used in the design presented above. As discussed in Section 3.1, the Max-Log-MAP algorithm approximates the $\max^*(\cdot)$ operation of the Log-MAP algorithm,

$$\max{}^*(x, y) = \max(x, y) + f_c(|x - y|),$$

where $f_c(x) = \ln(1 + e^{-x})$, $x > 0$, by the $\max(x, y)$ operation. This is usually done to alleviate the difficulty of implementing the correction term $f_c(x)$. Unfortunately, as discussed in Section 3.1 the Max-Log-MAP algorithm provides about 0.5 dB less coding gain than the Log-MAP algorithm in high BER region.

There are at least three ways to implement the correction function $f_c(x)$.

- First, a lookup table could be used. It has been shown [48] that an eight-value table of $f_c(x)$ with $x$ ranging between 0.0 and 5.0 works well. One way to construct such a table would be to evenly pick eight ordinate values of $f_c(x)$ and determine the corresponding abscissa values $x_1, x_2, ..., x_8$, as shown in Figure 9.103. Then

$$f_c(x) \approx \begin{cases} f_c(x_1), & 0 \le x < (x_1 + x_2)/2, \\ f_c(x_i), & (x_{i-1} + x_i)/2 \le x < (x_i + x_{i+1})/2,\ 2 \le i \le 7, \\ f_c(x_8), & (x_7 + x_8)/2 \le x < 5 \\ 0, & x \ge 5. \end{cases} \tag{9.249}$$

- Second, when the size of the lookup table for $f_c(x)$ is reduced to one, then the $\max^*(\cdot)$ logic becomes very simple: a constant is either added to $\max(x, y)$ or not, depending upon the relationship of $|x - y|$ to a predetermined threshold.

In [93], the following approximation was suggested:

$$f_c(x) \approx \begin{cases} 0.375, & 0 \leq x < 2, \\ 0, & \text{otherwise.} \end{cases} \tag{9.250}$$

Furthermore, it was shown in [93] that very little performance degradation is incurred by using this approximation in an AWGN channel.

– Third, a linear function may be used to approximate $f_c(x)$ [94]. If the slope is $-1/4$, the function $f_c'(x)$, where

$$f_c'(x) = \begin{cases} 0.61 - 0.25x, & 0 \leq x < 2.44, \\ 0, & x > 2.44, \end{cases} \tag{9.251}$$

is found to approximate $f_c(x)$. The slope is constrained to be $-1/4$ so that the multiplication can be realized by a right shift operation. Furthermore, if repeated applications of $\max^*(\cdot)$ are performed in a pairwise fashion, e.g.,

$$\max{}^*(x_1, x_2, x_3, x_4) = \max{}^*(\ \max{}^*(x_1, x_2),\ \max{}^*(x_3, x_4)\ ),$$

instead of in a sequential fashion, e.g.,

$$\max{}^*(x_1, x_2, x_3, x_4) = \max{}^*(\ \max{}^*(\ \max{}^*(x_1, x_2), x_3), x_4)\ ),$$

then the constant 0.61 can be eliminated from $f_c'(x)$ since the $\max^*(x, y)$ operation only depends upon the difference between $x$ and $y$. This leads to the function $f_c''(x)$, where

$$\begin{aligned} f_c''(x) &= \begin{cases} -0.25x, & 0 \leq x < 2.44, \\ -0.61, & x > 2.44, \end{cases} && (9.252) \\ &= -0.25 \times \min(x, 2.44). && (9.253) \end{aligned}$$

Equation (9.253) is relatively easier to implement.

Notice that in Figures 3.17 and 3.18 the received signals $y_k^{(i)}$ are all scaled by $L_c$ before being sent to the decoder. Such a scaling is not necessary if the approximation of $f_c(x)$ is modified. Since $f_c(x)$ is the only nonlinear operation inside the decoder, if $f_c''(x)$ is replaced by

$$f_c'''(x) = -0.25 \times \min(x, 2.44/L_c),$$

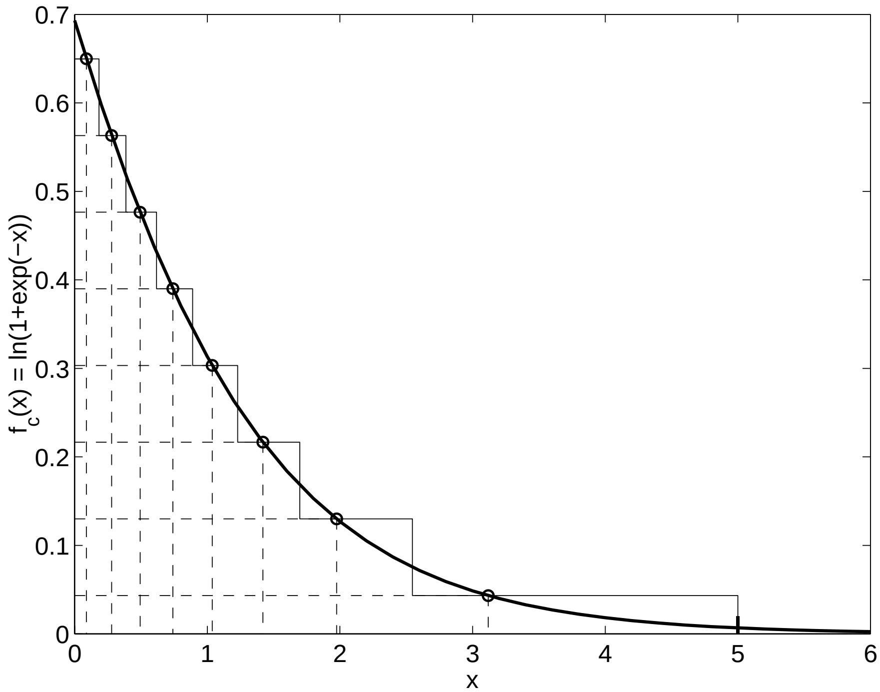



Figure 9.103: Construct a eight value table to represent correction function $f_c(x)$.

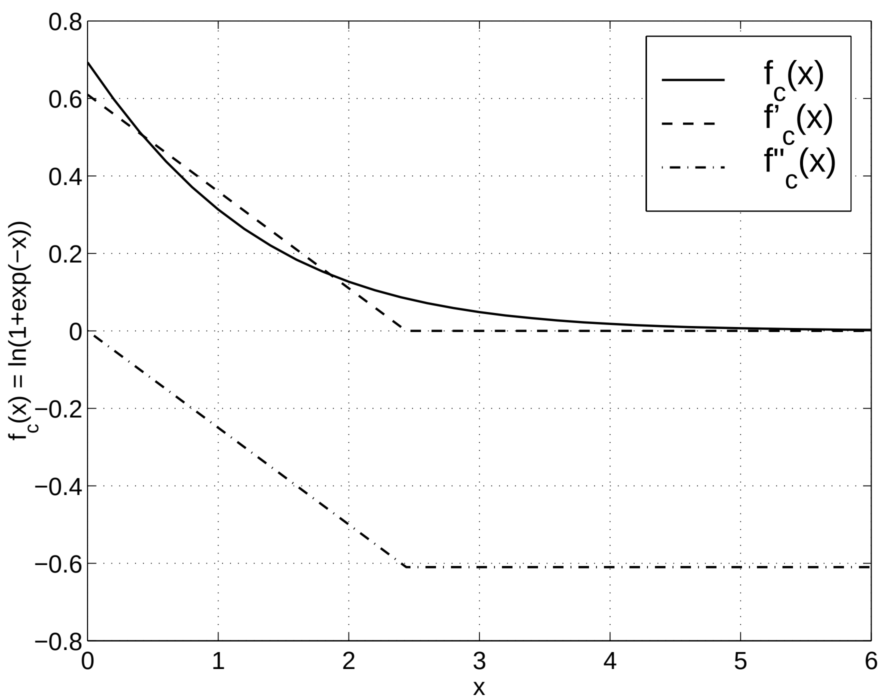



Figure 9.104: Approximate $f_c(x)$ with a linear function with slope 1/4.

then $y_k^{(i)}$, instead of $L_c y_k^{(i)}$, can be sent to the decoder. Using $f_c'''(x)$ as the $\max^*(\cdot)$ correction term and $y_k^{(i)}$ as the decoder input essentially divides every number inside the decoder by $L_c$, which does not affect the hard decision [94]. The same removal of the scaling by $L_c$ can be applied to the lookup table methods for $f_c(x)$ if the table entries are all divided by $L_c$.

## 9.5 Summary

This chapter has presented an implementation of real-time turbo decoder. The design methodology was presented first. This was followed by a discussion of the high level modeling and design decisions. The low level design was described next, including a look at the hardware and software tools used in the implementation, the decoder architecture and design flow, and the implementation results. Although the implementation was successful, several possible improvements were noted, including implementing simplified Log-MAP algorithm, removing the normalization circuits, and upgrading the implementation to a better approximation of the Log-MAP to improve the error rate performance.

# Chapter 10

# Conclusion

In this dissertation, two types of concatenated convolutional codes, PCCCs and SC-CCs, were discussed. PCCCs and SCCCs share many of the same characteristics such as code concatenation, recursive convolutional constituent codes, pseudorandom interleaver, and maximum *a posteriori* probability iterative decoding. However, they display fundamental differences in error rate performance, as was illustrated both by theoretic and simulation studies that were presented in this dissertation. Various decoding complexity issues, such as the development of efficient decoding algorithms, the use of fixed point arithmetic, and an iteration stopping criteria were addressed in this research. A novel ARQ technique, which leverages the strengths of both PCCCs and SCCCs, was proposed and studied. Finally, soft information of turbo codes were applied to solve the problem of frame synchronization at low $E_b/N_0$. Much of the knowledge obtained through this research has been applied to the real-time turbo decoder whose implementation was detailed in Chapter 9.

## 10.1 Contributions

Following a detailed study of the extant literature on turbo codes, this research has made the following original contributions.

1. A technique to compute the backward path metrics in the forward direction was developed. (Section 3.8)

2. The performance of PCCCs and SCCCs with various parameters such as the iteration number, the frame size, the code rate, and the code generator was extensively simulated. (Chapter 4)

3. The influence of quantization and fixed point arithmetic on the decoding performance was studied, and the optimal gain to scale received signals prior to quantization was derived. (Section 5.1)

4. The dynamic ranges of $\Delta\alpha_k$, $\Delta\beta_k$, and $\lambda_k(u; O)$ in the SISO decoding module were analytically derived, which led to an expression for the minimum internal data width. (Section 5.2)

5. The modular renormalization technique was demonstrated to be applicable to the SISO decoding. (Section 5.2.3)

6. A new iterative decoding stopping criterion, SDR, was proposed and found to have comparable performance to existing stopping criteria with yet lower complexity. (Chapter 6)

7. A new type-II ARQ scheme using both PCCCs and SCCCs was developed and simulated. (Chapter 7)

8. A list technique for frame synchronization with two variants, LLR sync and SDR sync, was developed and simulated. (Chapter 8)

9. SPW models for both PCCC and SCCC encoding/decoding systems were constructed. (Section 9.2.2)

10. Assistance was provided in the implementation of a real-time turbo decoder on a FPGA, including contribution of the high level design, cooperation in the host code development, and verification of the implementation.

## 10.2 Summary and Publications

This dissertation started with a theoretic analysis of PCCCs and SCCCs under the assumption of maximum likelihood decoding. Various iterative decoding algorithms, which surmount the practical infeasibility of maximum likelihood decoding, were experimented with and compared. Due to its superior performance, the Log-MAP algorithm was ultimately selected for extensive investigation in order to develop low-complexity algorithms which achieve near-optimal performance. For instance, a technique for computing the backward path metrics in the forward direction, which requires much less memory than the conventional backward recursion technique, was developed in this research. This technique will be presented at the Vehicular Technology Conference [95] in the spring of 2000.

Since low-power real-time turbo decoding is bound to use fixed point arithmetic, this topic was studied in depth. First, the initial quantization of the signal was considered. The pre-quantization scaling factor minimizing the quantization signal-to-distortion ratio was derived, and the decoding performance under various degrees of quantization was studied. A paper showing that eight bits of quantization is sufficient to guarantee near-floating point decoding performance was presented at the Vehicular Technology Conference [65] in the spring of 1999.

Next, the word size required inside a SISO decoding module for a given input word size was considered. It was shown that, since the Log-MAP algorithm depends only upon differences between quantities, the range which ensures representation of the maximum possible difference is sufficient. Bounds on these maximum possible differences were derived, which were directly translated in required SISO decoding module word sizes. It was proven that if these word sizes are adhered to and two's complement arithmetic is employed, normalization of the forward and backward recursion metrics is not required. This technique, which has been widely used in Viterbi decoding [69], is referred to as modular renormalization. These results will be presented at the Vehicular Technology conferences in the spring [96] and fall [97] of 2000. A paper detailing these results has also been submitted for publication in the *IEEE Transactions on Communications* [98].

Continuing on the theme of minimizing complexity, it was reasoned that stopping the iterative decoding procedure upon convergence would decrease the average complexity. The existing stopping criteria were studied and a new criterion, SDR, was conceived. SDR has smaller complexity and storage requirements than all other stopping criteria which were studied. Moreover, the BER/FER performance of SDR and the average number of iterations required are comparable to all other stopping criteria which were studied. These finding were reported on in a paper which has been accepted for publication in the *IEEE Communications Letters* [99].

The application of PCCCs and SCCCs to ARQ systems was also explored. A type-II code combining ARQ scheme was developed using related PCCCs and SCCCs. A paper on the subject will be presented at the IEEE International Conference on Communications [100] in 2000.

A paradox in turbo coded systems is that although turbo codes provide unparalleled error rate performance in the low $E_b/N_0$ region, synchronization is more difficult at low $E_b/N_0$, thus potentially diminishing their effectiveness. An effective solution to this conundrum was determined to lie in allowing the synchronization determination and turbo decoding to work collaboratively. Two such methods, LLR sync and SDR sync, were proposed and studied. Of these two methods, the SDR sync is preferable on the basis of complexity and false alarm probability. A paper on these studies in combination with those of other researchers on synchronization at MPRG has been submitted to the Second International Symposium on Turbo Codes and Related Topics [101]. A submission is also in preparation on this subject for *IEEE Journal on Selected Areas in Communications*.

## 10.3 Future Work

The field of turbo codes is still in its infancy. Hence there is yet much to be learned and much to be developed. The number of papers and researchers being added to the field each year is growing at a staggering rate, and the urgency is heightened by the adoption of turbo codes into the third generation wireless standards.

It is sincerely desired that this dissertation has ameliorated and innovated in the areas of efficient real-time turbo decoding, turbo code-based ARQ, and turbo code-assisted frame synchronization. It is recognized, however, that the research presented here may be readily built upon in four immediate ways.

First, this dissertation has restrained the channel to be AWGN. In reality, fading exists, especially in a wireless communication system [102]. The application of turbo coding in Rayleigh/Rician fading channels have been discussed in papers such as [103], [104], [105], and [106]. To comprehensively examine the techniques developed in this dissertation, such as the optimal gain, the SDR stopping criterion, the list synchronization methods, etc., they need to be tested in a fading environment.

Second, although error probability performance of both PCCCs and SCCCs have been studied and simulated, emphasis has been put into PCCCs when further investigations were pursued for realistic application. This is because PCCCs work better in the concerned BER region and they have been adopted in the third generation wireless communications standard [71]. However, if any application requires BER$< 10^{-6}$, SCCCs should be considered instead of PCCCs as discussed in Chapter 4. In parallel to the techniques developed for PCCCs in this dissertation, similar schemes need to be formulated for SCCCs. Minor changes are anticipated while the principles behind the techniques should readily apply.

Third, in Chapter 7, PCCC and SCCC were used to build a code combining ARQ system and the technique was examined with one retransmission. A method could be developed to combine the received signals ingeniously when multiple retransmissions are allowed. The combining technique described in [78] may be borrowed. Also, our recent study showed that if the received signals of previous transmissions were stored and combined with current transmission in PC-ARQ, its $E_s/N_0$ would be effectively increased, and PC-ARQ may perform better than PS-ARQ. It is not very clear which technique is the best for a real-time ARQ system. More comprehensive examination is necessary to determine the best solution.

Fourth, in Chapter 8, a sync word was added in front of the frame to assist frame synchronization where perfect channel knowledge was assumed. It is possible to use the sync word as the pilot for channel estimation at the same time. The technique that estimates the flat fading channel using pilot symbols has been reported in [107] where perfect frame synchronization was assumed. It would be useful to test the performance of the turbo coding system when both channel estimation and frame synchronization depends on the header.

# Appendix A

# List of Variables

| | |
|---|---|
| $A$ | amplitude of the antipodal signal |
| $A_k(s)$ | forward path metric of state $s$ at time $k$ in MAP algorithm |
| $A_{w,h}$ | number of codeword sequences with Hamming weight $h$ and input weight $w$ |
| $A_{w,h,n_i}$ | number of codeword sequences with input weight $w$, codeword sequence weight $h$, and $n_i$ concatenated error events |
| $a_k$ | input to the first RSC shift register cell at time $k$ |
| $B$ | channel bandwidth |
| $B_h$ | average Hamming weight per input bit corresponding to codeword sequences of weight $h$ |
| $B_k(s)$ | backward path metric of state $s$ at time $k$ in MAP algorithm |
| $B_{\alpha,k}$ | bound on $\Delta\alpha_k$ |
| $B_\alpha$ | bound on $\Delta\alpha_k$ for all $k$ |
| $B_{\beta,k}$ | bound on $\Delta\beta_k$ |
| $B_{\lambda,k}$ | bound on $\lvert\lambda_k(u;O)\rvert$ |
| $B_\lambda$ | bound on $\lvert\lambda_k(u;O)\rvert$ for all $k$ |
| $C$ | channel capacity |
| $C(\tau)$ | number of sign changes of the extrinsic information between iteration $(\tau-1)$ and $\tau$ |
| $\mathbf{c}$ | codeword sequence |
| $c$ | codeword |
| $c^{(j)}$ | the $j$-th bit of the codeword |
| $D$ | distortion function of the quantizer |

| | |
|---|---|
| $D_j(\tau)$ | number of sign differences between extrinsic information and *a priori* information of $j$-th SISO at iteration $\tau$ |
| $d_i$ | the $i$-th CRC bit |
| $d_f$ | free distance |
| $d_{f,eff}$ | effective free distance |
| $d_{min}$ | minimum distance |
| $E_b$ | average binary symbol energy when no channel coding |
| $E_s$ | average symbol energy |
| $f_c(x)$ | error correction term of $\max^*(\cdot)$ function |
| $\mathbf{G}$ | code generator matrix |
| $g(D)$ or octal $g$ | RSC code generator polynomial |
| $g(x)$ | CRC code generator polynomial |
| $H_b(e)$ | entropy of binary variables |
| $h$ | Hamming weight of codeword sequence |
| $h_*$ | a constant |
| $\mathbf{I_n}$ | $n \times n$ identity matrix |
| $I_x, I_y$ | arbitrary integers |
| $I(X,Y)$ | mutual information between $X$ and $Y$ |
| $J$ | decoding complexity |
| $K$ | constraint length of RSC encoder |
| $k$ | time index |
| $k_0$ | number of information bits in an information word |
| $L$ | number of quantization levels |
| $M$ | index of butterfly pair |
| $M_k(e)$ | branch metric of edge $e$ at time $k$ in MAP algorithm |
| $\bar{M}_k(e)$ | a fraction of $M_k(e)$, as defined in Equation (5.211) |
| $m$ | memory size of the encoder |
| $m_i$ | the $i$-th message bit of CRC |
| $\max^*(\cdot)$ | a function used in Log-MAP, as defined by Equation (3.86) |
| $N$ | PCCC frame size before encoding; SCCC interleaver size |
| $N_0$ | one-sided noise power spectral density |
| $N_{if,eff}$ | multiplicity of error events of the $i$-th constituent code with output weight equal to the effective free distance |
| $N_b$ | block size when forwardly computing $B_k(s)$ |

| | |
|---|---|
| $N_{blk}$ | number of blocks in a frame when forwardly computing $B_k(s)$ |
| $N_f$ | frame size after channel encoding |
| $N_u$ | updating window size of SW-SISO |
| $N_t$ | training window size of SW-SISO |
| $N_w$ | window size of SW-SISO |
| $N_{wind}$ | number of windows in SW-SISO |
| $N_\theta$ | size of sync word |
| $N_\xi$ | size of the list in list synchronization method |
| $N_\omega$ | size of the known uncoded sync bits for LLR sync method |
| $n$ | number of bits to represent a data inside the decoder |
| $n_{crc}$ | number of CRC bits |
| $n_q$ | number of quantization bits to represent a data |
| $n_0$ | number of code bits in a codeword |
| $n_{ce}$ | number of constituent encoders |
| $n_i$ | number of concatenated error events of code $i$ |
| $n_M$ | largest number of concatenated error events |
| $P$ | signal power |
| $P(\cdot)$ | probability |
| $P_b(e)$ | bit error probability |
| $P_s(e)$ | channel symbol error probability |
| $P_w(e)$ | word error probability |
| $P_k^A(c;O)$ | *a posteriori* probability of codeword $c$ |
| $P_k^A(u;O)$ | *a posteriori* probability of information word $u$ |
| $P_{\overline{sync}}$ | probability of false sync |
| $P_{b,sync}(e)$ | bit error probability when considering imperfect synchronization |
| $\mathbf{p}$ | the vector of parity bits |
| $p(\cdot)$ | probability density function |
| $Q(x)$ | $Q$ function as defined in Equation (2.26) |
| $q_c$ | threshold of the sign change ratio in SCR criterion |
| $q_d$ | threshold of the sign difference ratio in SDR criterion |
| $\mathbf{R}$ | code generator matrix for parity bits |
| $r$ | code rate |
| $S_k$ | RSC shift register state at time $k$ |
| $S_{k,i}$ | state of cell $i$ of the RSC shift register at time $k$ |

| | |
|---|---|
| $s$ | state index |
| $s_k^S(e)$ | starting shift register state of edge $e$ at time $k$ |
| $s_k^E(e)$ | ending shift register state of edge $e$ at time $k$ |
| $T$ | bit duration |
| $T(\tau)$ | cross entropy between LLRs of the constituent decoders at iteration $\tau$ |
| $\Delta T$ | signal duration |
| $\mathbf{u}$ | information sequence |
| $u$ | information word |
| $u^{(j)}$ | $j$-th bit of the information word |
| $\hat{u}$ | estimation of $u$ |
| $V_{low}, V_{hi}$ | lower and higher voltage limit of the quantizer |
| $v$ | symmetric voltage limit of the quantization range |
| $w$ | Hamming weight of encoder input sequence |
| $X$ | channel input signal |
| $x$ | value of channel input signal |
| $x^{(j)}$ | transmitted signal corresponding to $c^{(j)}$ |
| $Y$ | channel output signal |
| $Y_i^j$ | received signals from time $i$ to time $j$, inclusive |
| $\mathbf{y}$ | vector of received signals |
| $y$ | value of channel output signal |
| $y^{(j)}$ | received signal corresponding to $c^{(j)}$ |
| $Z$ | additive channel noise |
| $z$ | value of additive channel noise |
| $\alpha(k)$ | the original position of the number which is at the $k$-th position after interleaving |
| $\alpha_k(s)$ | forward path metric of state $s$ at time $k$ in logarithmic domain |
| $\tilde{\alpha}_k(s)$ | subtractively normalized $\alpha_k(s)$ |
| $\bar{\alpha}_k(s)$ | modular renormalized $\alpha_k(s)$ |
| $\Delta\alpha_k$ | maximum absolute difference between forward path metrics $\alpha_k(s)$ at time $k$ |
| $\beta_k(s)$ | backward path metric of state $s$ at time $k$ in logarithmic domain |
| $\tilde{\beta}_k(s)$ | subtractively normalized $\beta_k(s)$ |
| $\bar{\beta}_k(s)$ | modular renormalized $\beta_k(s)$ |

| | |
|---|---|
| $\Delta\beta_k$ | maximum absolute difference between backward path metrics $\beta_k(s)$ at time $k$ |
| $\gamma_k(e)$ | branch metric of state $s$ at time $k$ in logarithmic domain |
| $\delta$ | bin width of the quantizer |
| $\delta(x)$ | impulse (Dirac delta) function |
| $\Lambda_a(u_k)$ | *a priori* information of $k$-th information bit |
| $\Lambda_e(u_k)$ | extrinsic information of $k$-th information bit |
| $\lambda$ | LLR soft information |
| $\lambda_k(c; I)$ | input LLR information of codeword $c$ at time $k$ |
| $\lambda_k(u; I)$ | input LLR information of information word $u$ at time $k$ |
| $\lambda_k(c; O)$ | output LLR information of codeword $c$ at time $k$ |
| $\lambda_k(u; O)$ | output LLR information of information word $u$ at time $k$ |
| $\lambda_k^A(c; O)$ | complete LLR information of codeword $c$ at time $k$ |
| $\lambda_k^A(u; O)$ | complete LLR information of information word $u$ at time $k$ |
| $\lambda_k(c^{(j)}; O)$ | output LLR information of $j$-th code bit at time $k$ |
| $\lambda_k(u^{(j)}; O)$ | output LLR information of $j$-th information bit at time $k$ |
| $\lambda_k^A(c^{(j)}; O)$ | complete LLR information of $j$-th code bit at time $k$ |
| $\lambda_k^A(u^{(j)}; O)$ | complete LLR information of $j$-th information bit at time $k$ |
| $\nu$ | scaled received signal to be quantized |
| $\nu_k$ | $k$-th quantization boundary |
| $\bar{\nu}_k$ | $k$-th quantization level |
| $\mathbf{\Pi}$ | interleaver matrix |
| $\rho$ | scaling factor of the received signals before quantization |
| $\sigma_k(e)$ | joint probability of edge $e$ at time $k$ in MAP algorithm |
| $\sigma$ | standard deviation of Gaussian noise |
| $\tau$ | iteration number |
| $\varphi(h)$ | largest exponent of $N$ with weight $h$ in expression of $P_b(e)$ |
| $\eta$ | throughput efficiency |
| $\chi$ | computational complexity of the decoder |
| $\theta$ | sync word |
| $\xi$ | the list of the most likely sync positions |
| $\omega$ | known uncoded sync bits for LLR sync method |
| $\Gamma_k(e)$ | state metric of edge $e$ at time $k$, as defined in Equation (5.192) |

# Appendix B

# Channel Capacity of a Binary AWGN Channel

Assume the channel input X has value $x_i \in (-\sqrt{E_s}, +\sqrt{E_s})$, the independent and identically distributed noise $Z$ has value $z_i$, and the channel output $Y$ has value $y_i = x_i + z_i$, where $E_s$ is the average symbol energy. $X$ is independent from $Z$, and $Z \sim \mathcal{N}(0, \sigma^2)$. If AWGN channel is assumed, we can replace the noise variance $\sigma^2$ by $N_0/2$, where $N_0/2$ is the two-sided power spectral density [81]. The mutual information between the input and the output is

$$\begin{aligned} I(X,Y) &= h(Y) - h(Y|X) && \text{(B.254)} \\ &= h(Y) - h(X+Z|X) && \text{(B.255)} \\ &= h(Y) - h(Z) && \text{(B.256)} \end{aligned}$$

where $h(y)$ stands for the differential entropy of $Y$ and

$$h(Y) = -\int p(y) \log_2 p(y) dy.$$

The channel capacity is

$$\begin{aligned} C &= \max_{p(x)} I(X,Y) && \text{(B.257)} \\ &= I(X,Y) && \text{(B.258)} \end{aligned}$$

since the pdf of $X$ is fixed to be

$$p(x) = 0.5(\delta(x + \sqrt{E_s}) + \delta(x - \sqrt{E_s})). \tag{B.259}$$

The pdf of AWGN noise $Z$ is

$$p(z) = \frac{1}{\sqrt{\pi N_0}} \exp\left(\frac{-z^2}{N_0}\right), \tag{B.260}$$

therefore the pdf of $Y$ is:

$$p(y) = \frac{1}{2\sqrt{\pi N_0}} \left(\exp\left(\frac{-(y-\sqrt{E_s})^2}{N_0}\right) + \exp\left(\frac{-(y+\sqrt{E_s})^2}{N_0}\right)\right). \tag{B.261}$$

As a result, the channel capacity is

$$\begin{aligned}
C &= h(Y) - h(Z) \\
&= -\int p(y)\log_2 p(y)dy + \int p(z)\log_2 p(z)dz \\
&= -\int \frac{1}{2\sqrt{\pi N_0}} \left(\exp\left(\frac{-(y-\sqrt{E_s})^2}{N_0}\right) + \exp\left(\frac{-(y+\sqrt{E_s})^2}{N_0}\right)\right) \\
&\quad \cdot \log_2\left[\frac{1}{2\sqrt{\pi N_0}} \left(\exp\left(\frac{-(y-\sqrt{E_s})^2}{N_0}\right) + \exp\left(\frac{-(y+\sqrt{E_s})^2}{N_0}\right)\right)\right] dy \\
&\quad + \int \frac{1}{\sqrt{\pi N_0}} \exp\left(\frac{-z^2}{N_0}\right) \cdot \log_2\left[\frac{1}{\sqrt{\pi N_0}} \exp\left(\frac{-z^2}{N_0}\right)\right] dz \\
&= -\int \frac{\exp\left(\frac{-(y-\sqrt{E_s})^2}{N_0}\right)}{2\sqrt{\pi N_0}} \\
&\qquad \cdot \log_2\left[\frac{1}{2\sqrt{\pi N_0}} \left(\exp\left(\frac{-(y-\sqrt{E_s})^2}{N_0}\right) + \exp\left(\frac{-(y+\sqrt{E_s})^2}{N_0}\right)\right)\right] dy \\
&\quad - \int \frac{\exp\left(\frac{-(y+\sqrt{E_s})^2}{N_0}\right)}{2\sqrt{\pi N_0}} \\
&\qquad \cdot \log_2\left[\frac{1}{2\sqrt{\pi N_0}} \left(\exp\left(\frac{-(y-\sqrt{E_s})^2}{N_0}\right) + \exp\left(\frac{-(y+\sqrt{E_s})^2}{N_0}\right)\right)\right] dy \\
&\quad + \int \frac{1}{\sqrt{\pi N_0}} \exp\left(\frac{-z^2}{N_0}\right) \cdot \log_2\left[\frac{1}{\sqrt{\pi N_0}} \exp\left(\frac{-z^2}{N_0}\right)\right] dz \\
&= -\int \frac{1}{2\sqrt{\pi N_0}} \exp\left(\frac{-t^2}{N_0}\right) \cdot \log_2\left[\frac{1}{2\sqrt{\pi N_0}} \left(\exp\left(\frac{-t^2}{N_0}\right) + \exp\left(\frac{-(t+2\sqrt{E_s})^2}{N_0}\right)\right)\right] dt \\
&\quad - \int \frac{1}{2\sqrt{\pi N_0}} \exp\left(\frac{-t^2}{N_0}\right) \cdot \log_2\left[\frac{1}{2\sqrt{\pi N_0}} \left(\exp\left(\frac{-(t-2\sqrt{E_s})^2}{N_0}\right) + \exp\left(\frac{-t^2}{N_0}\right)\right)\right] dt \\
&\quad + \int \frac{1}{\sqrt{\pi N_0}} \exp\left(\frac{-z^2}{N_0}\right) \cdot \log_2\left[\frac{1}{\sqrt{\pi N_0}} \exp\left(\frac{-z^2}{N_0}\right)\right] dz \\
&= -\int \frac{1}{2\sqrt{\pi N_0}} \exp\left(\frac{-t^2}{N_0}\right) \cdot \log_2\left[\frac{1}{2}\left(1 + \exp\left(\frac{-4\sqrt{E_s}t - 4E_s}{N_0}\right)\right)\right] dt
\end{aligned}$$

$$\begin{aligned}
& -\int \frac{1}{2\sqrt{\pi N_0}} \exp\left(\frac{-t^2}{N_0}\right) \cdot \log_2 \left[\frac{1}{2}\left(1 + \exp\left(\frac{4\sqrt{E_s}t - 4E_s}{N_0}\right)\right)\right] dt \\
= & -\int \frac{1}{2\sqrt{\pi N_0}} \exp\left(\frac{-t^2}{N_0}\right) \cdot \log_2 \cosh\left(\frac{2\sqrt{E_s}t + 2E_s}{N_0}\right) dt \\
& -\int \frac{1}{2\sqrt{\pi N_0}} \exp\left(\frac{-t^2}{N_0}\right) \cdot \log_2 \cosh\left(\frac{2\sqrt{E_s}t - 2E_s}{N_0}\right) dt \\
& +2\int \frac{1}{2\sqrt{\pi N_0}} \exp\left(\frac{-t^2}{N_0}\right) (\log_2 e) \left(\frac{2E_s}{N_0}\right) dt \\
= & \frac{2E_s}{(\ln 2)N_0} - \frac{1}{\sqrt{2\pi}} \int \exp\left(\frac{-t^2}{2}\right) \cdot \log_2 \cosh\left(t\sqrt{\frac{2E_s}{N_0}} + \frac{2E_s}{N_0}\right) dt \\
= & \frac{2rE_b}{(\ln 2)N_0} - \frac{1}{\sqrt{2\pi}} \int \exp\left(\frac{-t^2}{2}\right) \cdot \log_2 \cosh\left(t\sqrt{\frac{2rE_b}{N_0}} + \frac{2rE_b}{N_0}\right) dt
\end{aligned} \tag{B.262}$$

where $r$ is the code rate and $E_s$ is the energy of a symbol in a codeword. The following relationships are used:

$$E_s = rE_b \tag{B.263}$$

$$\cosh x = \frac{1}{2}\left(e^x + e^{-x}\right) \tag{B.264}$$

$$\int \exp\left(\frac{-t^2}{N_0}\right) \cdot t dt = 0 \tag{B.265}$$

$$\int \exp\left(\frac{-t^2}{N_0}\right) dt = \sqrt{\pi N_0} \tag{B.266}$$

$$\frac{1}{2}\left[1 + \exp\left(\frac{-4\sqrt{E_s}t - 4E_s}{N_0}\right)\right] = \exp\left(\frac{-2\sqrt{E_s}t - 2E_s}{N_0}\right) \cdot \cosh\left(\frac{2\sqrt{E_s}t + 2E_s}{N_0}\right) \tag{B.267}$$

$$\frac{1}{2}\left[1 + \exp\left(\frac{4\sqrt{E_s}t - 4E_s}{N_0}\right)\right] = \exp\left(\frac{2\sqrt{E_s}t - 2E_s}{N_0}\right) \cdot \cosh\left(\frac{2\sqrt{E_s}t - 2E_s}{N_0}\right) \tag{B.268}$$

$$\int \exp\left(\frac{-t^2}{N_0}\right) \cdot \log_2 \cosh\left(\frac{2\sqrt{E_s}t + 2E_s}{N_0}\right) dt = \int \exp\left(\frac{-t^2}{N_0}\right) \cdot \log_2 \cosh\left(\frac{2\sqrt{E_s}t - 2E_s}{N_0}\right) dt \tag{B.269}$$

# Appendix C

# CRC Encoding and Decoding

Cyclic redundancy check (CRC) error detecting codes are actually shortened cyclic codes [77]. CRC codes are generally not cyclic, but they are derived from cyclic codes and hence the name. CRC codes are very popular error detection codes since they take advantage of the considerable burst-error detection capability provided by cyclic codes with extremely simple encoder and decoder implementations.

The shift register syndrome computation circuit for the original cyclic code can be used to encode/decode CRC codes. For example, with the generator polynomial $g(x) = x^{16} + x^{15} + x^2 + 1$, the shift register circuit of a systematic encoder is shown in Figure C.105.

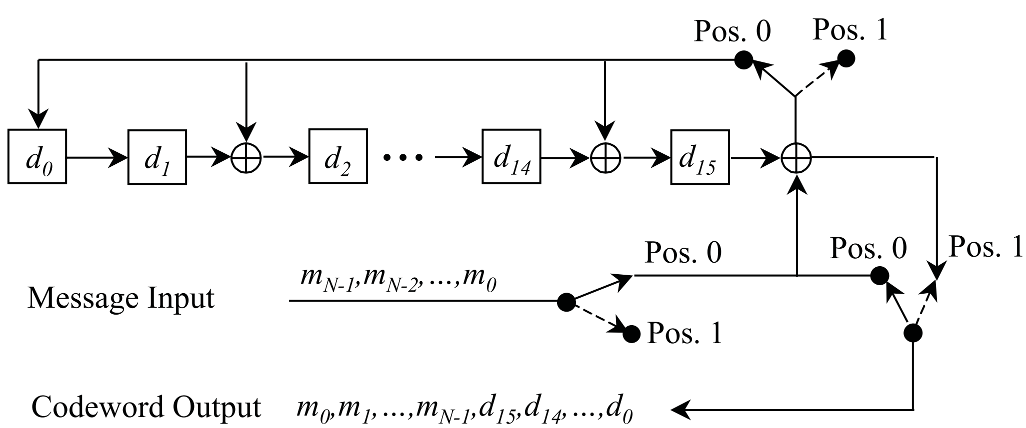



Figure C.105: Shift Register circuit for CRC encoding and decoding with $g(x) = x^{16} + x^{15} + x^2 + 1$.

The procedure of encoding a message sequence $(m_0, m_1, ..., m_{N-1})$ is as follows:

1. Place all switches at Position 0.

2. Feed $N$ message symbols into the encoder in order of increasing index.

3. After all $N$ bits have been fed into the shift register, move the switch to Position 1.

4. Shift the CRC bits $(d_{15}, d_{14}, ..., d_0)$ out.

The decoding procedure is very similar to the encoding. Using the same structure in Figure C.105, the received sequence $(m'_0, m'_1, ..., m'_{N-1}, d'_{15}, d'_{14}, ..., d'_0)$ is fed into the circuit. Let the CRC bits be $(d''_{15}, d''_{14}, ..., d''_0)$ after all $(N + 16)$ bits are finished, error detection decision is made based on the following:

$$(d''_{15}, d''_{14}, ..., d''_0) \text{ is } \begin{cases} \text{all-zero,} & \Rightarrow \text{assume no error,} \\ \text{not all-zero,} & \Rightarrow \text{detect an error.} \end{cases}$$

Note that the initial state of the shift register should be the same for both the encoder and the decoder.

# Bibliography

[1] R. E. Ziemer and R. L. Peterson, *Introduction to Digital Communication.* New York: Macmillan, Inc., 1992.

[2] J. Proakis, *Digital Communications.* New York: McGraw-Hill, Inc., third ed., 1995.

[3] G. Ungerboeck, "Channel coding with multilevel/phase signals," *IEEE Trans. Inform. Theory*, vol. IT-28, pp. 55–67, Jan. 1982.

[4] G. Ungerboeck, "Trellis-coded modulation with redundant signal sets, Part I: introduction," *IEEE Communications Magazine*, vol. 25, pp. 5–11, Feb. 1987.

[5] G. Ungerboeck, "Trellis-coded modulation with redundant signal sets, Part II: state of the art," *IEEE Communications Magazine*, vol. 25, pp. 12–21, Feb. 1987.

[6] S. Lin and D. J. Costello, *Error Control Coding: Fundamentals and Applications.* Englewood Cliffs, NJ: Prentice Hall, Inc., 1983.

[7] R. W. Hamming, "Error detecting and correcting codes," *Bell Sys. Tech. J.*, vol. 29, pp. 147–160, 1950.

[8] I. S. Reed and G. Solomon, "Polynomial codes over cerain finite fields," *SIAM Journal on Applied Mathematics*, vol. 8, pp. 300–304, 1960.

[9] A. Hocquenghem, "Codes correcteurs d'erreurs," *Chiffres*, vol. 2, pp. 147–156, 1959.

[10] R. C. Bose and D. K. Ray-Chaudhuri, "On a class of error correcting binary group codes," *Information and Control*, vol. 3, pp. 68–79, Mar. 1960.

[11] G. D. Forney, *Concatenated Codes.* Cambridge, MA: MIT Press, 1966.

[12] I. S. Reed, "A class of multiple-error-correcting codes and the decoding scheme," *IRE Trans. Inform. Theory*, vol. IT-4, pp. 38–49, Sept. 1954.

[13] R. G. Gallager, "Low-dentity parity-check codes," *IRE Trans. Inform. Theory*, vol. IT-8, pp. 21–28, Jan. 1962.

[14] W. W. Peterson, "Encoding and error-correction procedures for the Bose-Chaudhuri codes," *IRE Trans. Inform. Theory*, vol. IT-6, pp. 459–470, Sept. 1960.

[15] R. T. Chien, "Cyclic decoding procedures for Bose-Chaudhuri-Hocquenghem codes," *IEEE Trans. Inform. Theory*, vol. IT-10, pp. 357–363, Oct. 1964.

[16] G. D. Forney, "On decoding BCH codes," *IEEE Trans. Inform. Theory*, vol. IT-11, pp. 549–557, Oct. 1965.

[17] E. R. Berlekamp, *Algebraic Coding Theory.* New York: McGraw-Hill, 1968.

[18] P. Elias, "Coding for noisy channels," *IRE Convention Record*, vol. 3, pp. 37–46, 1955.

[19] J. M. Wozencraft and B. Reiffen, *Sequential Decoding.* Cambridge, MA: MIT Press, 1961.

[20] J. M. Wozencraft and I. M. Jacobs, *Principles of Communication Engineering.* New York: John Wiley, 1965.

[21] J. L. Massey, *Threshold Decoding.* Cambridge, MA: MIT Press, 1963.

[22] A. J. Viterbi, "Error bounds for convolutional codes and an asymptotically optimum decoding algorithm," *IEEE Trans. Inform. Theory*, vol. IT-13, pp. 260–269, Apr. 1967.

[23] L. R. Bahl, J. Cocke, F. Jelinek, and J. Raviv, "Optimal decoding of linear codes for minimizing symbol error rate," *IEEE Trans. Inform. Theory*, vol. IT-20, pp. 284–287, Mar. 1974.

[24] Consultative Committee for Space Data Systems, "Recommendations for space data standard: Telemetry channel coding." Blue Book Issue 2, CCSDS 101.0-B2, Jan. 1987.

[25] J. Hagenauer, E. Offer, and L. Papke, "Matching Viterbi decoders and Reed-Solomon decoders in concatenated systems," in *Reed-Solomon Codes and Their Applications* (S. B. Wicker and V. K. Bhargava, eds.), pp. 242–271, Piscataway, NJ: IEEE press, 1994.

[26] J. Hagenauer and P. Hoeher, "A Viterbi algorithm with soft-decision outputs and its applications," in *Proc., IEEE GLOBECOM*, pp. 1680–1686, 1989.

[27] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near Shannon limit error-correcting coding and decoding: turbo-codes (1)," in *Proc., IEEE Int. Conf. on Commun.*, (Geneva, Switzerland), pp. 1064–1070, May 1993.

[28] D. Divsalar and F. Pollara, "Serial and hybrid concatenation codes with applications," in *Proc., Int. Symp. on Turbo Codes and Related Topics*, (Brest, France), pp. 80–87, Sept. 1997.

[29] C. E. Shannon, "A mathematical theory of communication," *Bell Sys. Tech. J.*, vol. 27, pp. 379–423 and 623–656, 1948.

[30] R. G. Gallager, "Simple derivation of the coding theorem and aome applications," *IEEE Trans. Inform. Theory*, vol. IT-11, pp. 3–18, Jan. 1965.

[31] T. Cover and J. Thomas, *Elements of Information Theory*. New York: Wiley Interscience, 1991.

[32] L. W. Couch, *Digital and Analog Communication Systems*. Macmillan Publishing Company, 4th ed., 1993.

[33] C. Berrou and A. Glavieux, "Near optimum error correcting coding and decoding: Turbo-codes," *IEEE Trans. Commun.*, vol. COM-44, pp. 1261–1271, Oct. 1996.

[34] G. Battail, "A conceptual framework for understanding turbo codes," in *Proc., Int. Symp. on Turbo Codes and Related Topics*, (Brest, France), pp. 55–62, Sept. 1997.

[35] P. Hoeher, "On channel coding and multi-user detection for DS-CDMA," in *Proc., IEEE Int. Conf. on Universal Personal Commun.*, (Ottawa, Canada), pp. 641–646, Oct. 1993.

[36] J. Hagenauer, "Source-controlled channel decoding," *IEEE Trans. Commun.*, vol. COM-43, pp. 2449–2457, Sep. 1995.

[37] P. Robertson, "An overview of bandwidth efficient turbo coding schemes," in *Proc., Int. Symp. on Turbo Codes and Related Topics*, (Brest, France), pp. 103–110, Sept. 1997.

[38] A. Glaviex, C. Laot, and J. Labat, "Turbo equalization over a frequency selective channel," in *Proc., Int. Symp. on Turbo Codes and Related Topics*, (Brest, France), pp. 96–102, Sept. 1997.

[39] S. Benedetto, G. Montorsi, D. Divsalar, and F. Pollara, "Serial concatenation of interleaved codes: performance analysis, design, and iterative decoding," *JPL TDA Progress Report*, vol. 42-126, Aug. 1996.

[40] D. J. Costello, J. Hagenauer, H. Imai, and S. B. Wicker, "Applications of error-control coding," *IEEE Trans. Inform. Theory*, vol. 44, pp. 2531–2560, Oct. 1998.

[41] J. Hagenauer, "The turbo principle: Tutorial introduction and state of the art," in *Proc., Int. Symp. on Turbo Codes and Related Topics*, (Brest, France), pp. 1–11, Sept. 1997.

[42] M. Moher and T. A. Gulliver, "Cross-entropy and iterative decoding," *IEEE Trans. Inform. Theory*, vol. 44, pp. 3097–3104, Nov. 1998.

[43] S. Benedetto and G. Montorsi, "Unveiling turbo codes: Some results on parallel concatenated coding schemes," *IEEE Trans. Inform. Theory*, vol. IT-42, pp. 409–428, Mar. 1996.

[44] S. Benedetto and G. Montorsi, "Design guidelines of parallel concatenated convolutional codes," in *Proc., IEEE GLOBECOM*, (Singapore), pp. 2273–2277, Nov. 1995.

[45] S. Benedetto and G. Montorsi, "Design of parallel concatenated convolutional codes," *IEEE Trans. Commun.*, vol. COM-44, pp. 591–600, May 1996.

[46] D. Divsalar and R. J. McEliece, "Effective free distance of turbo codes," *Electronics Letters*, vol. 32, pp. 445–446, Feb. 29th 1996.

[47] M. C. Valenti, *Itarative Detection and Decoding for Wireless Communications.* PhD thesis, Virginia Polytechnic Institute and State University, July 1999.

[48] P. Robertson, P. Hoeher, and E. Villebrun, "Optimal and sub-optimal maximum a posteriori algorithms suitable for turbo decoding," *European Trans. on Telecommun.*, vol. 8, pp. 119–125, Mar./Apr. 1997.

[49] J. Hagenauer, P. Robertson, and L. Papke, "Iterative (turbo) decoding of systematic convolutional codes with the MAP and SOVA algorithms," in *Proc., ITG Conf.*, (Munchen, Germany), pp. 21–29, Sept. 1994.

[50] C. Nill and C.-E. W. Sundberg, "List and soft symbol output Viterbi algorithms: extensions and comparisons," in *IEEE Trans. Commun.*, vol. 43, pp. 277–287, Feb./Mar./Apr. 1995.

[51] K. R. Narayanan and G. L. Stüber, "List decoding of turbo codes," *IEEE Trans. Commun.*, vol. 46, pp. 754–761, June 1998.

[52] N. Wiberg, H.-A. Loeliger, and R. Kotter, "Codes and iterative decoding on general graphs," *European Trans. on Telecommun.*, vol. 6, pp. 513–525, Sept./Oct. 1995.

[53] N. Wiberg, *Codes and Decoding on General Graphs.* PhD thesis, U. Linkoping, Sweden, Apr. 1996. Dept. Elec. Eng.

[54] F. R. Kschischang and B. J. Frey, "Iterative decoding of compound codes by probability propagation in graphical models," *IEEE J. Select. Areas Commun.*, vol. SAC-16, pp. 219–230, Feb. 1998.

[55] J. Hagenauer, E. Offer, and L. Papke, "Iterative decoding of binary block and convolutional codes," *IEEE Trans. Inform. Theory*, vol. IT-42, pp. 429–445, Mar. 1996.

[56] P. Robertson, E. Villebrun, and P. Hoeher, "A comparison of optimal and sub-optimal MAP decoding algorithms operating in the log domain," in *Proc., IEEE Int. Conf. on Commun.*, pp. 1009–1013, 1995.

[57] S. Benedetto, D. Divsalar, G. Montorsi, and F. Pollara, "Soft-output decoding algorithms in iterative decoding of turbo codes," *JPL TDA Progress Report*, vol. 42-124, Feb. 15, 1996.

[58] M. P. C. Fossorier, F. Burkert, S. Lin, and J. Hagenauer, "On the equivalence between SOVA and max-log-MAP decodings," *IEEE Commun. Letters*, vol. 2, pp. 137–139, May 1998.

[59] J. Erfanian, S. Pasupathy, and G. Gulak, "Reduced complexity symbol detectors with parallel structures for ISI channels," *IEEE Trans. Commun.*, vol. COM-42, pp. 1661–1671, Feb./Mar./Apr. 1994.

[60] A. J. Viterbi, "An intuitive justification and a simplified implemetation of the MAP decoder for convolutional codes," *IEEE J. Select. Areas Commun.*, vol. 16, pp. 260–264, Feb. 1998.

[61] Y. Wu, "Design and Implementation of Parallel and Serial Concatenated Convolutional Codes," Ph. D. Preliminary Examination Report, Virginia Tech, May 1999.

[62] C. Schlegel and L. Perez, "On error bounds and turbo-codes," *IEEE Commun. Letters*, vol. 3, pp. 205–207, July 1999.

[63] S. Dolinar and D. Divsalar, "Weight distributions for turbo codes using random and nonrandom permutations," *JPL TDA Progress Report*, vol. 422-122, pp. 56–65, Aug. 15th, 1995.

[64] S. Benedetto, R. Garello, and G. Montorsi, "Parallel concatenated coding schemes for wireless application," in *Proc., IEEE Int. Conf. on Universal Personal Commun.*, vol. 2, pp. 807–811, Oct. 1998.

[65] Y. Wu and B. D. Woerner, "The influence of quantization and fixed point arithmetic upon the BER performance of turbo codes," in *Proc., IEEE Veh. Tech. Conf.*, (Houston, TX), May 1999.

[66] M. J. Demler, *High-Speed Analog-to-Digital Conversion*. San Diego: Academic Press, 1991.

[67] T. K. Blankenship, "Design and implementation of a pilot signal scanning receiver for CDMA personal communication services systems," Master's thesis, Virginia Tech, Apr. 1998.

[68] G. Masera, G. Piccinini, M. R. Roch, and M. Zamboni, "VLSI architectures for turbo codes," *IEEE Trans. Very Large Scale Integration (VLSI) Systems*, vol. 7, pp. 369–379, Sept. 1999.

[69] A. P. Hekstra, "An alternative to metric rescaling in Viterbi decoders," *IEEE Trans. Commun.*, vol. 37, pp. 1220–1222, Nov. 1989.

[70] S. Benedetto, D. Divsalar, G. Montorsi, and F. Pollara, "A soft-input soft-output maximum a posteriori (MAP) module to decode parallel and serial concatenated codes," *JPL TDA Progress Report*, vol. 42-127, Nov. 15, 1996.

[71] 3rd Generation Partnership Project, "Technical Specification TS 25.212 v3.0.0," Nov. 1999.

[72] A. Shibutani, H. Suda, and F. Adachi, "Reducing average number of tubo decoding iterations," *Electronics Letters*, vol. 35, pp. 701–702, Apr. 1999.

[73] O. Y. Takeshita, O. M. Collins, P. C. Massey, and D. J. Costello, "Concatenated codes that achieve $10^{-6}$ frame error rates at SNR's 0.6dB from the sphere packing bound," in *Proc., 36th Annual Allerton Conf. on Communication, Control, and Computing*, Sept. 1998.

[74] R. Y. Shao, S. Lin, and M. P. C. Fossorier, "Two simple stopping criteria for turbo decoding," *IEEE Trans. Commun.*, vol. 47, pp. 1117–1120, Aug. 1999.

[75] D. Agrawal and A. Vardy, "On the phase trajectories of the turbo-decoding algorithm," in *IMA Summer Program*, Aug. 1999.

[76] L. C. Perez, J. Seghers, and D. J. Costello, "A distance spectrum interpretation of turbo codes," *IEEE Trans. Inform. Theory*, vol. IT-42, pp. 1698–1708, Nov. 1996.

[77] S. Wicker, *Error Control Systems for Digital Communicationsand Storage*. Englewood Cliffs, NJ: Prentice Hall, Inc., 1995.

[78] K. R. Narayanan and G. L. Stüber, "A novel ARQ technique using the turbo coding principle," *IEEE Commun. Letters*, vol. 1, pp. 49–51, Mar. 1997.

[79] G. Caire and E. Biglieri, "Parallel concatenated codes with unequal error protection," *IEEE Trans. Commun.*, vol. 46, pp. 565–567, May 1998.

[80] L. Lu and S. G. Wilson, "Synchronization of turbo coded modulation schemes at low SNR," in *Proc., IEEE Int. Conf. on Commun.*, 1998.

[81] B. Sklar, *Digital Communications.* P T R Prentice Hall, 1988.

[82] S. S. Pietrobon, "Efficient implementation of continuous MAP decoders and a synchronisation technique for turbo decoders," in *Int. Symp. on Inform. Theory and its Applications*, (Victoria, BC, Canada), pp. 586–589, Sept. 1996.

[83] S. S. Pietrobon, "Implementation and performance of a turbo/MAP decoder," *Int. J. Satell. Commun.*, vol. 16, pp. 23–46, 1998.

[84] J. R. Hess, "Implementation of a turbo decoder on a configurable computing platform," Master's thesis, Virginia Polytechnic Institute and State University, Sept. 1999.

[85] Cadence Design Systems, Inc., *Wireless Digital Communications System Design with SPW.* Lab Book Version 4.0, Jan. 1998.

[86] Synopsys, "COSSAP Design Environment." Datasheet, Apr. 1999.

[87] D. Divsalar and F. Pollara, "Turbo codes for PCS applications," in *Proc., IEEE Int. Conf. on Commun.*, pp. 54–59, May 1995.

[88] Annapolis Micro Systems website, *http://www.annapmicro.com/*.

[89] Viewlogic website, *http://www.viewlogic.com/*.

[90] Synopsys, "FPGA Express." Datasheet, July 1998.

[91] Xilinx, "Development Systems Products Overview." Product verview, Dec. 1997.

[92] S. F. Swanchara, "An FPGA-based multiuser receiver employing parallel interference cancellation," Master's thesis, Virginia Polytechnic Institute and State University, July 1998.

[93] W. J. Gross and P. G. Gulak, "Simplified MAP algorithm suitable for implementation of turbo decoders," *Electronics Letters*, vol. 34, Aug. 1998.

[94] T. K. Blankenship, private communication, July 1999.

[95] Y. Wu, W. J. Ebel, and B. D. Woerner, "Forward computation of backward path metrics for MAP decoder," in *Proc., IEEE Veh. Tech. Conf.*, May 2000.

[96] Y. Wu and B. D. Woerner, "Internal data width in SISO decoding module with modular renormalization," in *Proc., IEEE Veh. Tech. Conf.*, May 2000.

[97] Y. Wu and B. D. Woerner, "Analysis of internal data width requirements for SISO decoding modules," in *IEEE Veh. Tech. Conf.*, Sept. 2000. submitted.

[98] Y. Wu, B. D. Woerner, and T. K. Blankenship, "Analysis of internal data width requirements and modular renormalization for SISO decoding modules," *IEEE Trans. Commun.*, Nov. 1999. Submitted.

[99] Y. Wu, B. D. Woerner, and W. J. Ebel, "A simple stopping criterion for turbo decoding," *IEEE Commun. Letters*, Nov. 1999. Accepted.

[100] Y. Wu and M. C. Valenti, "An ARQ technique using related parallel and serial concatenated convolutional codes," in *Proc., IEEE Int. Conf. on Commun.*, June 2000.

[101] M. M. Howlader, Y. Wu, and B. Woerner, "Decoder-assisted frame synchronization for turbo coded systems," in *Int. Symp. on Turbo Codes and Related Topics*, Sept. 2000.

[102] T. S. Rappaport, *Wireless Communications: Principles and Practice.* Upper Saddle River, NJ: Prentice Hall PTR, 1996.

[103] E. K. Hall and S. G. Wilson, "Design and analysis of turbo codes on Rayleigh fading channels," *IEEE J. Select. Areas Commun.*, vol. SAC-16, pp. 160–174, Feb. 1998.

[104] I. D. Marsland and P. T. Mathiopoulos, "Multiple differential detection of parallel concatenated convolutional (turbo) codes in correlated fast Rayleigh fading," *IEEE J. Select. Areas Commun.*, vol. SAC-16, pp. 265–275, Feb. 1998.

[105] F. Gagnon and D. Haccoun, "Bounds on the error performance of coding for nonindependent Rician-fading channels," *IEEE Trans. Commun.*, vol. COM-40, pp. 351–360, Feb. 1992.

[106] G. Kaplan and S. Shamai, "Achievable performance over the correlated Rician channel," *IEEE Trans. Commun.*, vol. COM-42, pp. 2967–2978, Nov. 1994.

[107] M. C. Valenti and B. D. Woerner, "Refined channel estimation for coherent detection of turbo codes over flat-fading channels," *Electronics Letters*, vol. 34, pp. 1648–1650, Aug. 20, 1998.

[108] R. A. Cameron, *Fixed-Point Implementation of a Multistage Receiver.* PhD thesis, Virginia Polytechnic Institute and State University, Jan. 1997.

[109] B. K. Yi, "On the synchronization issues of the turbo coded telemetry system," in *Proc., Int. Symp. on Turbo Codes and Related Topics*, (Brest, France), pp. 275–279, Sept. 1997.

[110] G. L. Stüber, *Principles of Mobile Communication.* Kluwer Academic Publishers, 1996.

# Vita

**Yufei Wu** was born in Zhuzhou, Hunan Province of southern China on December 12, 1972. In July 1990, she was admitted to the Teaching-Reform class of Northwestern Polytechnical University (NPU) in Xi'an, China, where she was exempt from the National Admission Exam because of her previous performance record. This allowed her to finish the B.S. requirement in three years and to become a graduate student of the Department of Automatic Control in September 1993, once again being excused from the admissions test. She received her Masters of Science in Electrical Engineering in the field of Research and Design of Aeroplane Control Systems in March of 1996.

After receiving her M.S., she joined the Aerospace and Ocean Engineering Department at Virginia Polytechnic Institute and State University in August 1996 as a graduate research assistant in the pursuit of a doctoral degree. Realizing that Electrical Engineering was her field of choice, she transferred to the Mobile and Portable Radio Research Group (MPRG) of the Bradley Department of Electrical and Computer Engineering at Virginia Tech in May of 1997, where she works with Dr. Brian D. Woerner. She will begin the employment with Motorola Labs in Schaumburg, IL after graduation.

Her primary research interests are in wireless communications, spread spectrum, and channel coding.