DESIGN AND IMPLEMENTATION OF PARALLEL AND SERIAL CONCATENATED CONVOLUTIONAL CODES

by

Yufei Wu

A Preliminary Review of Initial Research and Proposal
for Current and Future Work toward Doctor of Philosophy Degree.

Approved:

Brian D. Woerner
(Chairman)

Peter M. Athanas

F. Gail Gray

Jeffrey H. Reed

Lee Johnson
Math Department

May 1999

Blacksburg, Virginia

Design and Implementation of Parallel and Serial Concatenated Convolutional Codes

by
Yufei Wu
Committee Chairman: Brian D. Woerner
Electrical Engineering

Abstract

This report concentrates on the application of parallel and serial concatenated convolutional codes (PCCC and SCCC) to wireless communication systems. PCCCs, coined "turbo codes" by their discoverers, have been shown to perform close to the Shannon bound at low to medium signal-to-noise ratio (SNR). SCCCs, who perform better than PCCCs at high SNR, were developed borrowing the same principle.

The first part of the report presents the theoretic background of channel coding. Then the performance bounds for PCCC and SCCC are evaluated, providing insight into the reasons for their excellent performance. Next, a general and efficient maximum *a posteriori* soft-input soft-output (SISO) decoding algorithm is presented. A sliding window version is later developed to exchange higher complexity for lower latency and smaller storage. A PCCC decoder and a SCCC decoder are presented for this implementation. Afterwards, a practical analysis for implementation is performed. A method is proposed to compute the backward metric forwardly in the MAP (maximum *a posteriori*) algorithm, so that the storage will be significantly reduced. Practical considerations, such as the interleaver design, trellis termination and termination of the iterative decoding, are discussed.

In the second part, the PCCC and SCCC encoding-decoding systems have been constructed for SPW as a bridge to hardware implementation. Using Monte Carlo simulation, a comprehensive study has been carried out to show the good performance of the concatenated codes, the influence of various parameters, and the difference between the parallel and serial structures. Finally, the effect of quantization on the performance of the decoder has been analyzed and simulated.

Using this work as a foundation, we conclude this report by proposing a program of study that will lead to fulfillment of the research requirement for the Ph.D. degree.

Contents

List of Tables

List of Figures

Abbreviation

APP	*a posteriori* probability
ARQ	automatic repeat request
AWGN	additive white Gaussian noise
BCH	Bose-Chaudhuri-Hocquenghem
BCJR	Bahl-Cocke-Jelinek-Raviv
BER	bit error rate
BSC	binary symmetric channel
CH-UC	unconstrained AWGN channel
CH-BI	binary-input constrained AWGN channel
CH-BIBO	binary-input binary-output channel
BPSK	binary phase shift keying
CWEF	conditional weight enumerating function
DSP	digital signal processing
FER	frame error rate
FEC	forward error correction
FPGA	field programmable gate array
HCCC	hybrid concatenated convolutional code
IOWEF	input-output weight enumerating function
LLR	log-likelihood ratio
MAP	maximum *a posteriori*
ML	most likelihood, most likely
PCCC	parallel concatenated convolutional code
pdf	probability density function
RS	Reed-Solomon
RSC	recursive systematic convolutional
SCCC	serial concatenated convolutional code
SIR	signal-to-interference ratio
SDR	signal-to-distortion ratio
SISO	soft-input soft-output
SNR	signal-to-noise ratio
SPW	Signal Processing WorkSystem
SOVA	soft-output Viterbi algorithm
SW-SISO	sliding window soft-input soft-output
TTCM	turbo trellis coded modulation
VA	Viterbi algorithm

Chapter 1

Introduction

Error-control coding, an outgrowth of Shannon's Information Theory, has developed from a mathematical curiosity into a fundamental element of almost any system that transmits or stores digital information over the past fifty years. 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. In this preliminary report, the focus is a channel coding technique for wireless communication systems.

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 reduce 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. The analog signal is sampled and quantized before it is transmitted 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 equal. Thus, the source encoder is used to remove the redundancy before transmission so that the least number of bits are necessary to represent the information. The information source is transformed by the source encoder into a sequence of bits called the information

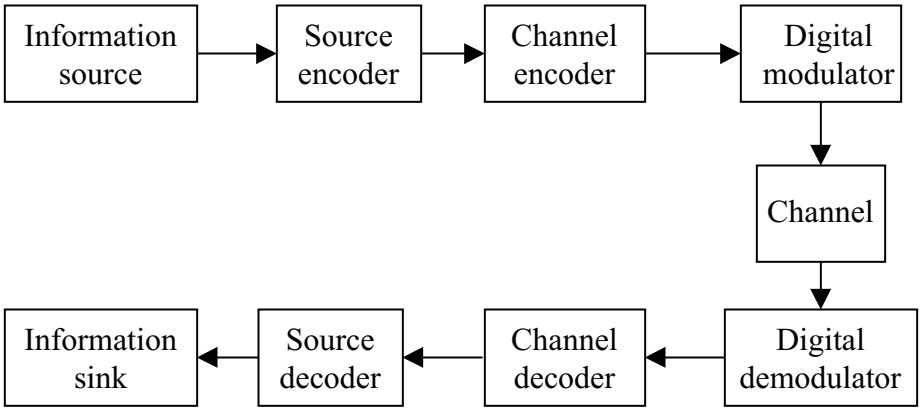


Figure 1.1: Block diagram of a communication system.

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 the binary bits are not suitable for transmission over a physical channel, the digital modulator is used to transform them into a continuous-time waveform proper 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 contaminated in a random manner by, e.g., the thermal noise generated by the electronic devices, 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 added by the physical media.

Finally, the source decoder accepts the decoded bits and attempts to reconstruct the original information source with the 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 separated in channel limited wireless systems.

1.2 Development of Channel Coding Technique

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 ARQ (automatic repeat request) with error detection and retransmission, the error control strategy for a one-way system must be FEC (forward error correction), 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 independently collect blocks of k_0 information symbols and map them to codewords of n_0 symbols. Hamming discovered the Hamming code, which was the first class of error correction codes to be described in a combinatorial, constructive manner [7]. After that, a major breakthrough was made with the discovery of RS (Reed-Solomon) codes and BCH (Bose-Chaudhuri-Hocquenghem) 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 broke the asymptotic difficulty 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 various types were asymptotically 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 become 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 MAP (maximum *a posteriori*) 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 invention 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 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 capacity around BER= 10^{-5} is usually achievable with short constraint length, long block length, and 10 to 20 decoding iterations.

Prompted by turbo codes, which are also called parallel concatenated convolutional codes (PCCCs), serial concatenated convolutional codes (SCCCs) and hybrid concatenated convolutional codes (HCCCs) are constructed with the same components to provide similar coding gains [28]. SCCC and HCCC can perform better than PCCC 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 code in the concatenation.

1.3 Outline of Report

This report outlines a program of study intended to bring the initial promise of turbo codes closer to practical fruition. The proposed thesis will address implementation issues for PCCC and SCCC in the theoretical and algorithmic manner.

In Chapter 2, the fundamentals of channel coding technology are explained. 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, including the minimum bit error probability for a given code rate, the lowest possible signal-to-noise ratio to achieve a certain performance, and the largest coding gain possible for a scheme. 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, theoretic analysis is presented to find the reason why they excel. Interleaver gain[1], a key feature of PCCC and SCCC in comparison to convolutional codes, is emphasized in the explanation. Following the analysis, design guidelines are summarized.

In Chapter 3, the conventional Log-MAP algorithm for PCCC is presented. As an extension, an enhanced implementation of the general soft-input soft-output (SISO) decoding technique is developed. For binary transmission in AWGN noise, the computation is simplified significantly. Then, the procedures to decode PCCC and SCCC are shown with the general decoding module. For continuous transmission or blockwise transmission with very large frame sizes, a sliding window can be incorporated into this algorithm to reduce memory requirements and delay. Finally, a novel method is proposed to compute the backward metrics forwardly so that the storage requirement is cut down by at least 50%. A stability analysis is explained to validate the proposal.

In Chapter 4, encoding-decoding systems are presented which have been built with SPW, a block diagram simulator that is based upon hardware modeling. Three systems have been constructed: a rate 1/2 and a rate 1/3 PCCC system, and a rate 1/4 SCCC system. The details of the blocks in the systems are demonstrated. The SPW models serve as an intermediate tool for constructing an actual hardware implementation.

In Chapter 5, groups of simulation curves are graphed to show the good performance of PCCC and SCCC. The influence of the parameters, such as the number of iterations, frame size, code rate, and code generator, becomes obvious with the examples. Also, the difference between PCCC and SCCC is clarified.

[1]The interleaver gain is the additional coding gain obtainable by increasing the frame size.

In Chapter 6, the effect of quantization noise is discussed. Quantization noise is an important issue in practical application. A method is presented to minimize the quantization error by finding the optimal gain to scale the received signal properly. Simulation shows that 8-bit representation is enough to obtain performance curves without noticeable degradation.

Finally in Chapter 7, the contributions of this report are summarized, and plans are outlined for future research, including techniques for improved computational efficiency within the Log-MAP algorithm, improved frame synchronization, and enhanced techniques for terminating decoding iteration.

Chapter 2

Theoretic Foundation of Concatenated Codes

2.1 Introduction

This chapter aims to explain the characteristics of PCCC and SCCC, and the reasons why they perform better than the other coding techniques such as block codes and convolutional codes. First the performance bounds of coding techniques in general are established. Then the structures of PCCC and SCCC systems are introduced and the analysis of their performance is presented.

2.2 Channel Model

A communication system model is shown in Figure 2.2 to facilitate the analysis of channel coding. The modulator and demodulator are included as a part of the composite channel [2]. If the channel noise is AWGN (additive white Gaussian noise) and the modulator accepts binary input, and the demodulator makes hard decisions, then the composite channel is a BSC (binary symmetric channel). 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 important to turbo codes, whose decoder requires soft output from the demodulator.

In the system the information symbol u of dimension k_0 is encoded into a codeword c of dimension n_0. The codeword c is mapped to determine the channel input X. The channel output signal $Y = X + Z$, where Z is the additive channel noise. The terms X, Y

and Z are random variables, and their respective values are denoted by x_i, y_i and z_i, where $y_i = x_i + z_i$.

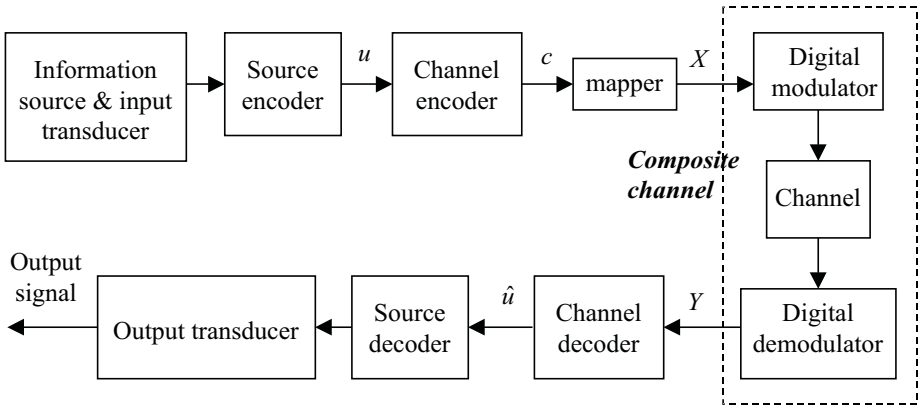


Figure 2.2: A general communication system.

2.3 Channel Capacity

In his 1948 paper, "A Mathematical Theory of Communication" [29], Claude Shannon introduced the important concept of channel capacity C, which is defined as the maximum mutual information over all possible channel input distributions:

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

and $I(X,Y)$ is the mutual information between the input and output of a channel:

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

where $p(x)$ and $p(y)$ are the probability density functions (pdf) of X and Y, respectively, and $p(x,y)$ is the joint pdf. For an AWGN channel, the capacity is

$$C = \frac{1}{2} \log_2 \left(1 + \frac{2P}{N_0}\right) \quad \text{bits per transmission,} \tag{2.3}$$

where $Z \sim N(0, N_0/2)$, and P is the power constraint: $\frac{1}{n_0} \sum_{i=1}^{n_0} x_i^2 \leq P$.

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

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

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 ΔT. 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 W, 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 W. When $W \to \infty$, the channel capacity approaches its asymptotic value [2]:

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

2.4 Channel Coding Tradeoffs

The **noisy channel coding theorem** [29] states that: there exist channel codes (and decoders) that make it possible to achieve reliable communication with as small an error probability as desired, if code rate $r < C$, where C is the channel capacity; conversely, if $r > C$, it is impossible to make the probability of error tend toward 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 randomly chosen code 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 which is close to random, yet allows efficient decoding methods.

It has been shown by Gallager [30] that for any discrete-input memoryless channel, there exists an n_0-symbol code of rate r for which the word error probability with maximum likelihood decoding is bounded by

$$P_w(e) < \exp[-n_0 \cdot \mathrm{E}(r)], \quad 0 \le r \le C, \tag{2.6}$$

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

1. Decrease the code rate $r = k_0/n_0$, as in Figure 2.3. However, for a given source rate, decreasing r means increasing the transmission rate, and hence increasing 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.

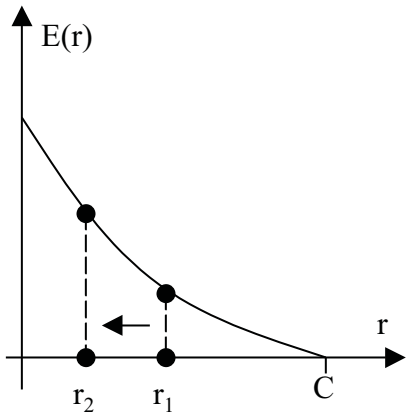


Figure 2.3: $E(r)$ increases when r decreases.

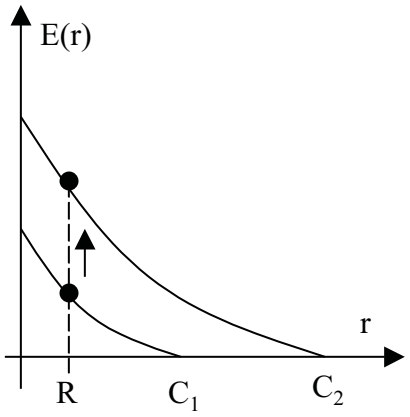


Figure 2.4: $E(r)$ increases when C increases.

3. Increase the code symbol length n_0, 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 D is on the order of the number of codewords, or,

$$\begin{aligned} D &\propto 2^{k_0} = 2^{n_0 r} \\ &\propto \exp(n_0 r) \end{aligned}$$

Consequently,

$$\begin{aligned} P_w(e) &< \exp[-n_0 \mathrm{E}(r)] \\ &\propto \exp\left(-\ln D \frac{\mathrm{E}(r)}{r}\right) \\ &\propto D^{-\mathrm{E}(r)/r} \end{aligned} \tag{2.7}$$

This implies that the word error probability decreases only algebraically with respect to the decoding complexity. In comparison, for convolutional codes (with Viterbi algorithm) and the concatenated codes (with MAP or SOVA), the decoding complexity per information bit is invariant to frame size. Or, for the entire frame,

$$D \propto k_0 = n_0 r$$

Thus,

$$\begin{aligned} P_w(e) &< \exp[-n_0 \mathrm{E}(r)] \\ &\propto \exp\left(-D\frac{\mathrm{E}(r)}{r}\right) \end{aligned} \tag{2.8}$$

This implies that the word error probability decreases exponentially with an increase in the decoding complexity. Nevertheless, when it comes to the bit error probability $P_b(e)$, there is a distinction between convolutional codes and concatenated codes because of the different mapping of the information weight to the codeword weight. In consequence, $P_b(e)$ for concatenated codes decreases with an increase of frame size, while that of convolutional codes does not. Since both $P_w(e)$ and $P_b(e)$ decrease when the frame size increases and the decoding complexity of a bit is decoupled from the frame size, concatenated codes become very attractive to engineers.

2.5 Channel Coding Bounds

If the information bits are received 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 $P_b(e)$. Given the channel capacity C, the code rate r is bounded by:

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

From this inequality, several theoretic coding bounds can be derived. In this report three types of channel 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.

2.5.1 $P_b(e)$ Bounds

CH-UC

Let E_b be the average power of a transmitted signal when channel coding is not used. Thus, when channel coding of rate r is applied, the average power P of the transmitted signal becomes $P = rE_b$. Substituting into Equation 2.3, the capacity of CH-UC is obtained:

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

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. $P_b(e)$ limits are numerically calculated and plotted in Figure 2.5 for various code rates r. When $r \to 0$, the transmission bandwidth approaches infinity. For this case, the lowest possible SNR is reached for transmission with arbitrarily low bit error probability. When $r \to 0$, Equation 2.12 yields

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

When $P_b(e) \to 0$, $1 - H_b(e) \to 1$. This implies that to achieve reliable transmission, the minimum power for all coding schemes is: $E_b/N_0 = \ln 2 = -1.6$ 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 can be shown to be

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

The derivation is shown in Appendix A.

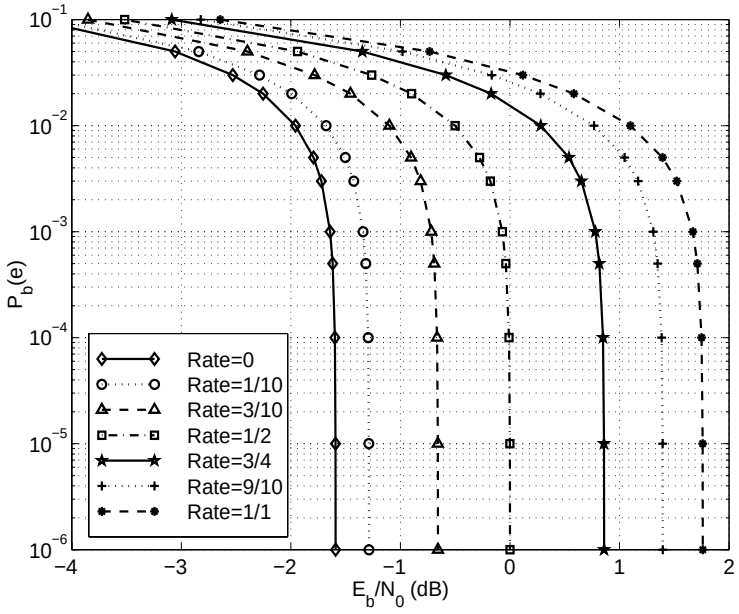


Figure 2.5: Bit error probability bounds for CH-UC with various code rates.

Substituting Equation 2.14 into Equation 2.10 and taking the equality, the bit error probability bound vs. E_b/N_0 for a given r is obtained. In Figure 2.6, the bounds on $P_b(e)$ for various code rates are plotted. As $r \to 0$,

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

Let

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

Then

$$\frac{dt'}{dr} = t\sqrt{\frac{E_b}{2N_0}} r^{-1/2} + \frac{2E_b}{N_0} \tag{2.16}$$

$$\lim_{r\to 0} t' = 0 \tag{2.17}$$

$$\lim_{r\to 0} \frac{\sinh t'}{\sqrt{r}} = \lim_{r\to 0} \frac{(\cosh t')\left(t\sqrt{\frac{E_b}{2N_0}} r^{-1/2} + \frac{2E_b}{N_0}\right)}{0.5 r^{-1/2}} = t\sqrt{\frac{2E_b}{N_0}} \tag{2.18}$$

Thus,

$$\lim_{r\to 0}[1 - H_b(e)] = \frac{2E_b}{(\ln 2)N_0} - \frac{1}{\sqrt{2\pi}} \lim_{r\to 0} \int \exp\left(\frac{-t^2}{2}\right) \cdot \log_2 e \cdot \frac{\sinh t'}{\cosh t'}$$

$$
\begin{aligned}
& \cdot \left(t\sqrt{\frac{E_b}{2N_0}} r^{-1/2} + \frac{2E_b}{N_0} \right) dt \\
= \quad & \frac{2E_b}{(\ln 2)N_0} - \frac{1}{\sqrt{2\pi}} \int \exp\left(\frac{-t^2}{2}\right) \cdot \log_2 e \cdot t^2 \frac{E_b}{N_0} dt \\
= \quad & \frac{2E_b}{(\ln 2)N_0} - \frac{1}{\sqrt{2\pi}} \frac{E_b}{(\ln 2)N_0} \cdot \sqrt{2\pi} \\
= \quad & \frac{E_b}{(\ln 2)N_0}
\end{aligned} \tag{2.19}
$$

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

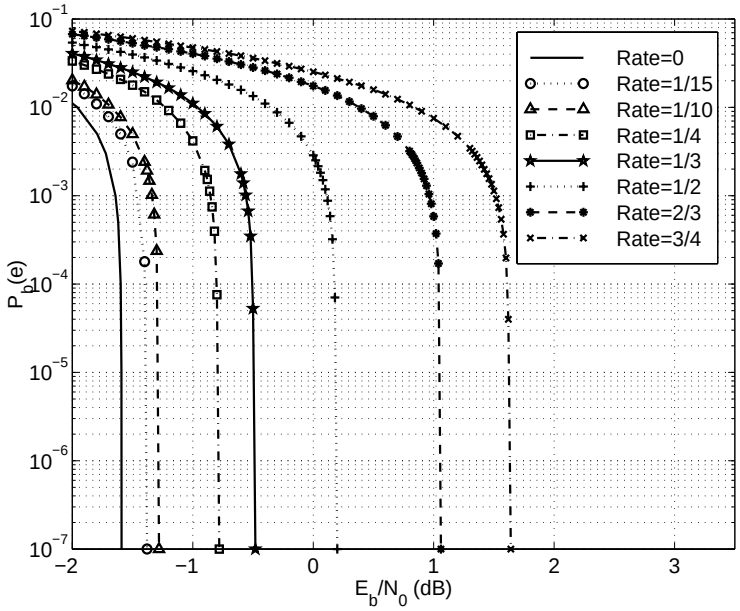


Figure 2.6: Bit error probability bounds for CH-BI with various code rates.

2.5.2 Minimum E_b/N_0

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

$$r \leq C \tag{2.20}$$

1. For CH-UC, Equation 2.20 gives

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

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 implies

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

3. For CH-BIBO, Equation 2.20 gives:

$$r \leq C = 1 - H(p) = 1 + p\log_2 p + (1-p)\log_2(1-p) \tag{2.24}$$

with the binary channel symbol error probability

$$p = Q\left(\sqrt{\frac{2rE_b}{N_0}}\right) \tag{2.25}$$

when BPSK is used for transmission.

Taking equalities in Equations 2.21, 2.23, and 2.24, the relationship between the minimum E_b/N_0 and r can be found for errorless transmission. The curves are drawn in Figure 2.7 for the above three channels, where numerical calculation is used for Equations 2.23 and 2.24.

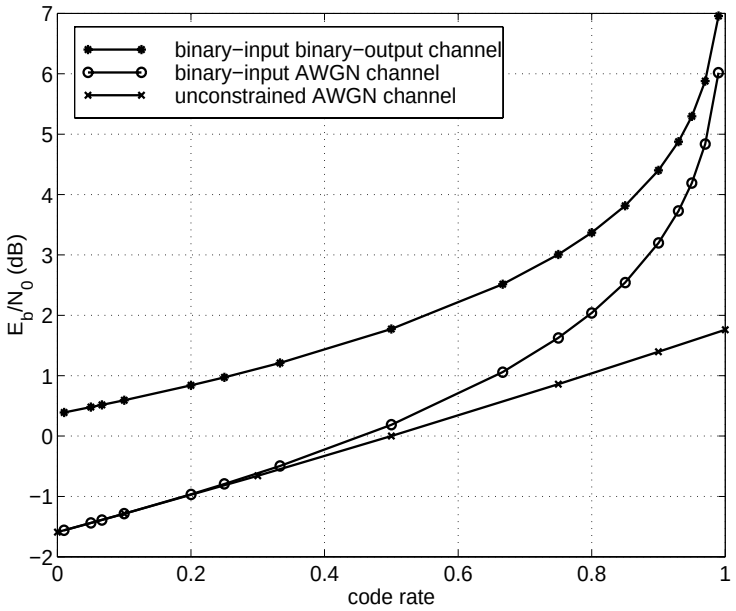


Figure 2.7: Minimum E_b/N_0 vs. rate r for CH-UC, CH-BI and CH-BIBO.

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 realized in going from hard decision decoding to soft decision decoding.

2.5.3 Maximum Achievable Coding Gain

As $r \to 0$, the transmission bandwidth approaches infinity. This provides the upper bound on the coding gain among all possible coding schemes. Again, three types of channel are discussed below.

1. For CH-UC, the information capacity limit as $r \to 0$ is derived in Equation 2.13. Rewriting it, we obtain

$$\begin{aligned} \frac{E_b}{N_0} &= \ln 2(1 - H_b(e)) \\ &= \ln 2(1 + P_b(e)\log_2 P_b(e) + (1 - P_b(e))\log_2(1 - P_b(e))). \end{aligned} \tag{2.26}$$

2. Equation 2.19 indicates that CH-BI has the same performance as CH-UC when the transmission bandwidth is infinite.

3. For CH-BIBO,

$$\begin{aligned} \lim_{r\to 0}[1 - H_b(e)] &= \lim_{r\to 0}\frac{C}{r} \\ &= \lim_{r\to 0}\frac{1 + p\log_2 p + (1-p)\log_2(1-p)}{r} \\ &= \lim_{r\to 0}\Big[p'\log_2 p + p\frac{1}{(\ln 2)p}p' \\ &\qquad -p'\log_2(1-p) + (1-p)\frac{1}{(\ln 2)(1-p)}(-p')\Big] \\ &= \lim_{r\to 0} p'(\log_2 p - \log_2(1-p)) \\ &= \lim_{r\to 0}\left(-\frac{1}{2}\sqrt{\frac{E_b}{\pi N_0}}\exp\left(-\frac{rE_b}{N_0}\right)r^{-1/2}\right)(\log_2 p - \log_2(1-p)) \\ &= \lim_{r\to 0} -\frac{1}{2}\sqrt{\frac{E_b}{\pi N_0}}\frac{\frac{p'}{\ln 2}(\frac{1}{p} + \frac{1}{1-p})}{0.5r^{-1/2}} \\ &= \lim_{r\to 0} -\frac{4}{\ln 2}\sqrt{\frac{E_b}{\pi N_0}}\frac{\left(-\frac{1}{2}\sqrt{\frac{E_b}{\pi N_0}}\exp\left(-\frac{rE_b}{N_0}\right)r^{-1/2}\right)}{r^{-1/2}} \\ &= \frac{2}{(\ln 2)\pi}\frac{E_b}{N_0} \end{aligned} \tag{2.27}$$

where BPSK modulation is assumed. Equation 2.25 provides an expression for p, and the following relationships are used in the derivation of Equation 2.27:

$$Q(x) = \frac{1}{\sqrt{2\pi}} \int_x^\infty \exp(-t^2/2)dt \tag{2.28}$$

$$\lim_{r\to 0} p = \lim_{r\to 0} Q\left(\sqrt{\frac{2rE_b}{N_0}}\right) = \frac{1}{2} \tag{2.29}$$

$$\begin{aligned} \frac{dp}{dr} &= \frac{dQ\left(\sqrt{\frac{2rE_b}{N_0}}\right)}{dr} \\ &= -\frac{1}{2}\sqrt{\frac{E_b}{\pi N_0}} \exp\left(-\frac{rE_b}{N_0}\right) r^{-1/2} \end{aligned} \tag{2.30}$$

From Equation 2.27, it is clear that to achieve errorless transmission, or $P_b(e) \to 0$, $H_b(e) \to 0$, the lowest possible E_b/N_0 is: $E_b/N_0 = (\ln 2)\pi/2 = 0.37$ dB.

Summing up the above results, two curves are plotted in Figure 2.8 for Equations 2.26 and 2.27. 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 terribly corrupted. Also, as $r \to 0$, the CH-UC and CH-BI channels can provide about 2 dB more coding gain than CH-BIBO.

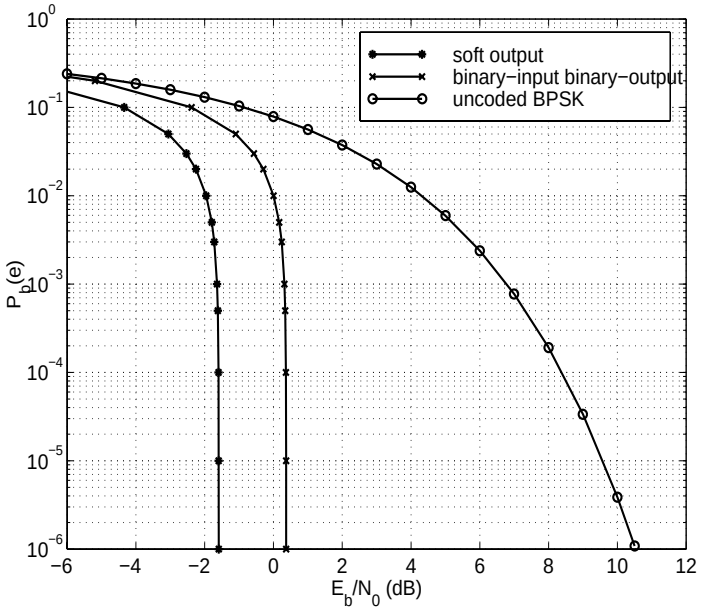


Figure 2.8: Lower bound of $P_b(e)$ for CH-UC/CH-BI and CH-BIBO.

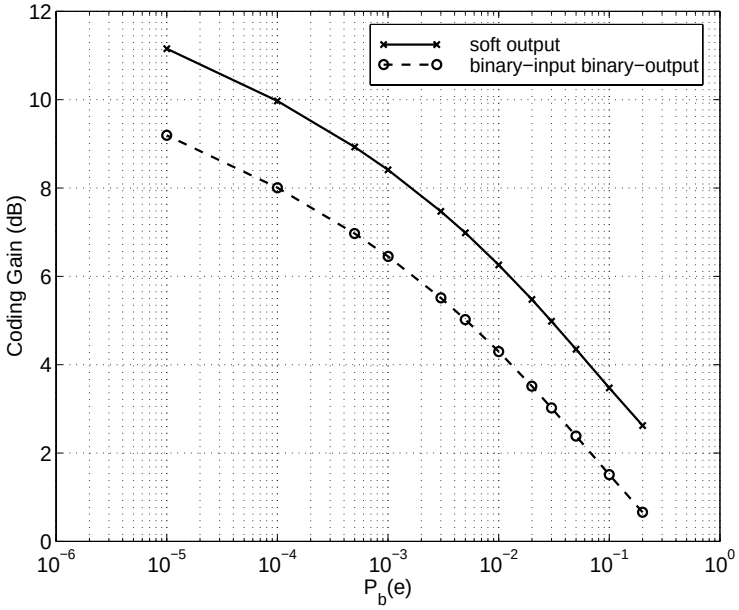


Figure 2.9: Upper bound of coding gain for CH-UC/CH-BI and CH-BIBO.

2.6 Theoretic Performance Bounds for PCCC and SCCC

Since the channel coding theorem was discovered by Claude Shannon [29], researchers have been searching 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 decent coding gain while retaining enough structure so that a decoder with reasonable complexity is possible. A major breakthrough was made by French researchers Berrou *et al.* [27], who discovered the so-called "turbo codes." Turbo codes, also known as parallel concatenated convolutional codes (PCCCs), were demonstrated to have performance within 0.5 dB of the Shannon limit around BER=10^{-5} [33] [34].

Turbo codes have attracted a large group of researchers throughout the world to explore their advantageous features and to extend their principles to other structures [35] [36] [37] [38]. Serial concatenated convolutional codes (SCCCs) were an important discovery made by Benedetto *et al.* [39] along the way, which were found to outperform PCCC at high SNR in terms of both bit error probability and frame error probability.

2.6.1 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.

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 perforation pattern, can be used to increase the code rate of the overall code [27]. In general, the encoder is formed by two or more parallel constituent encoders joined by one or more interleavers.

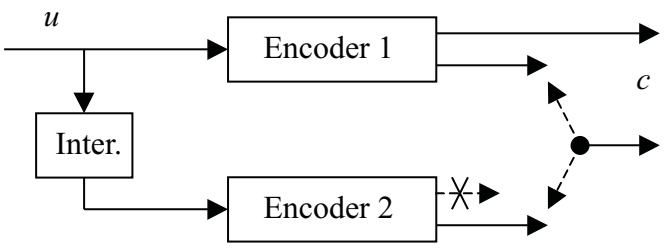


Figure 2.10: Turbo (PCCC) encoder.

The interleaver has the effect of matching low-weight[1] parity sequences with high-weight

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

parity sequences in almost all cases, thus generating a code with very few low-weight output sequences. Usually the free distance of turbo codes is not large. The excellent performance at moderate BER is due rather to a drastic reduction in the number of nearest neighboring output 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. Thus, the information is shared between two constituent decoders. Since MAP gives the maximum *a posteriori* estimation of each information bit, the iterative decoding procedure will converge to a certain number as the number of iteration increases.

At almost any bandwidth efficiency (the ratio of data rate to bandwidth), performance about 0.5 dB away from the capacity limit at BER= 10^{-5} is achievable with short constraint length[2] 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].

2.6.2 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 codes, serial concatenated convolutional codes (SCCCs), were proposed [39]. The good performance of SCCCs has incited a lot of 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 Reed-Solomon (RS) outer code concatenated with a convolutional inner code separated by a 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 RS and the convolutional code, a bit interleaver is used in SCCCs to introduce randomness.

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

[2]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.

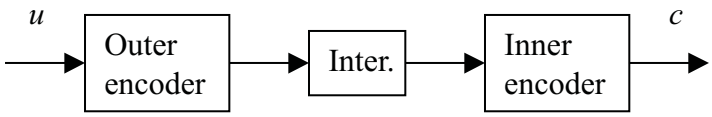


Figure 2.11: SCCC encoder.

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 two or more serially cascaded constituent encoders separated by one or more interleavers.

The code rate of SCCC is the product of the outer code rate and the inner code rate. Just as for PCCC, SCCC also creates an overall code trellis with a huge number of states because of the bit interleaver; however, it can be decoded with a relatively simple iterative MAP decoding procedure.

In comparison to PCCC whose $P_b(e)$ decreases at the rate of N^{-1} as N increases, N being the interleaver size, $P_b(e)$ of SCCC decreases at a faster rate, e.g., $N^{-2}, N^{-3}, ...$, as N increases. The error floor associated with PCCC, where the bit error probability flattens, is eliminated by SCCC as a result. This feature is illustrated by the simulation curves in Figure 2.12. A disadvantage of SCCC is that it is computationally more complex than a PCCC with constituent codes of the same memory size. Also, SCCC tends to have a higher bit error probability than PCCC at low SNR.

2.6.3 Performance Upper Bound

For an (n_0, k_0) code, the input-output weight enumerating function (IOWEF) is defined to be [42]

$$A(W, H) = \sum_{w=0}^{k_0} \sum_{h=0}^{n_0} A_{w,h} W^w H^h \tag{2.32}$$

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:

$$P_w(e) \leq \sum_{h=d_{\min}}^{n_0} \left(\sum_{w=1}^{k_0} A_{w,h} \right) Q\left(\sqrt{\frac{2hrE_b}{N_0}} \right) \tag{2.33}$$

where d_{min} is lowest hamming weight of the nonzero codeword sequences. For linear codes such as block codes, convolutional codes, PCCC and SCCC, d_{min} is also the smallest hamming distance between any two distinct codeword sequences. The bit error probability is

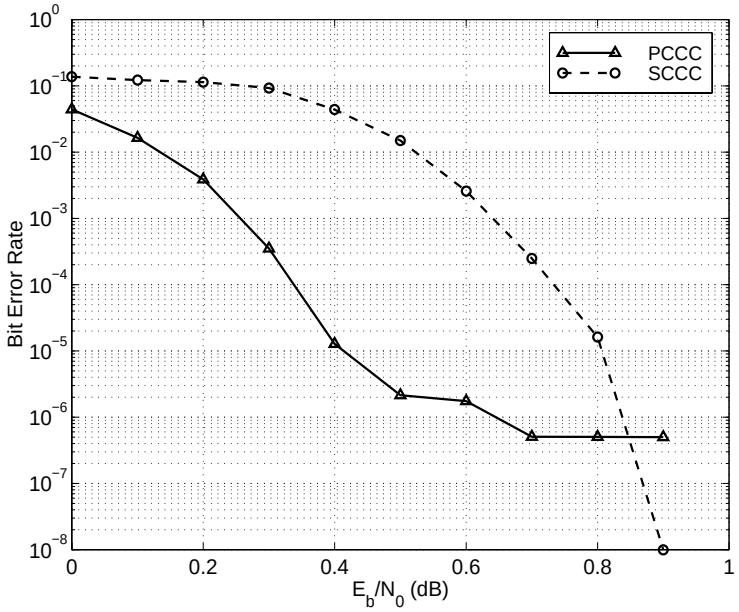


Figure 2.12: Performance comparison of PCCC ($g = (15, 17)_{octal}$) and SCCC ($g = (7, 5)_{octal}$): rate 1/3, frame size 5120, 10 iterations.

upper bounded by

$$\begin{aligned} P_b(e) &= \sum_w \frac{w}{k_0} P_w(e, w) \\ &\leq \sum_{h=d_{\min}}^{n_0} \left(\sum_{w=1}^{k_0} \frac{w}{k_0} A_{w,h} \right) Q\left(\sqrt{\frac{2hrE_b}{N_0}} \right) \\ &\leq \sum_{h=d_{\min}}^{n_0} B_h Q\left(\sqrt{\frac{2hrE_b}{N_0}} \right) \end{aligned} \tag{2.34}$$

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

$$B_h = \sum_{w=1}^{k_0} \frac{w}{k_0} A_{w,h}, \tag{2.35}$$

and $k_0 B_h$ 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 approach 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 u produces a codeword sequence c, its delayed version du produces dc, which has the same weight as 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 k_0 increases, B_h does not decrease accordingly. This makes the interleaver gain impossible. On the other hand, B_h increases rapidly with h. The large multiplicities make the contribution of the higher distance spectral lines[3] to $P_b(e)$ to be greater than the free distance asymptote at low SNR. This is why convolutional codes do not have a steep BER slope at low SNR.

However, because of the pseudorandom interleaver between two constituent codes, turbo codes are time varying. Hence, u and du 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 characterized by spectral thinning. For very large interleavers, spectral thinning results in a 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 get a sparse distance spectrum. As the interleaver size approaches infinity, the thinning of the spectrum enables the free distance asymptote to dominate the performance at low SNR. This is why turbo codes can achieve performance close to the Shannon limit at low to medium SNR.

At medium to high SNR, the performance asymptotically approaches

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

where N is the interleaver size and d_f is the free distance of turbo codes. Since d_f 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_f, its performance would eventually surpass that of a turbo code at high SNR because of its steeper free distance asymptote.

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

In the following sections, union bound analysis following the work of Benedetto, Montorsi, and Divsalar [39] [42] is used to explore the performance characteristics of PCCC and

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

SCCC.

2.6.4 Uniform Interleaver

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

$$A(w,H) = \sum_{h=0}^{n_0} A_{w,h} H^h. \tag{2.37}$$

The CWEF describes the weight distribution of codeword sequences generated by the information sequences of a certain 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. The knowledge of the input by itself is not sufficient to obtain the weight of the codeword sequence from the second encoder.

To facilitate the theoretical analysis of turbo codes, the concept of a uniform interleaver [43] 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 $p = 1/\binom{N}{w}$. 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 outputs corresponding to the input weight w will be generated after passing the second encoder C_2. As a 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 of the parallel concatenated code as

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

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 [44]. Thus, for any SNR, the performance obtained by the uniform interleaver is achievable by at least one deterministic interleaver.

2.6.5 Performance Upper Bound of PCCC

Assuming two rate 1/2 constituent codes and a uniform interleaver of length N between them, the PCCC has CWEF with the following coefficients for any given w, according to Equation 2.38:

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

Note that the codeword sequence weight of C_2 should not include the weight of the systematic bits since they are not transmitted. The bit error probability is thus

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

based on Equation 2.34.

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, output weight h, and n_i concatenated error events. The index of the constituent code is $i = 1, 2$. When the interleaver size N is much larger than the constraint length of the constituent code, the coefficient $A_{w,h}$ can be approximated by

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

where n_M is the largest number of error events concatenated in a codeword sequence of weight h generated by an input 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, expression 2.39 can be written as

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

When $N \gg n$, the approximation

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

is valid. Applying it to Equation 2.42, we obtain

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

Substituting Equation 2.44 into Equation 2.40, we have

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

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

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

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, the bit error probability can be approximated by

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

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

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

1. $\alpha(d_{fp})$
   When h is equal to the free distance d_{fp}, the input must bring a single error event for both constituent codes. Or, $n_1(d_{fp}) = 1, n_2(d_{fp}) = 1$. So

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

   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 of minimum weight one produces the error event of smallest output weight. Thus $w_{fp} = 1$, $\alpha(d_{fp}) = 0$, and therefore there is no interleaver gain at high E_b/N_0. For recursive convolutional codes, an input of weight one will produce an output of infinite weight. Thus, the smallest output weight event has to be produced by an input of weight greater than one, or $w_{fp} \geq 2$ [43]. As a result, $\alpha(d_{fp}) \leq -1$. For the recursive convolutional encoders of rate $1/n$, $w_{fp} = 2$ [43], which implies $\alpha(d_{fp}) = -1$. This indicates an interleaver gain of N^{-1} at high E_b/N_0. Hence, the constituent encoders must be recursive to have interleaver gain at high SNR.

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

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

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

   - For nonrecursive convolutional codes and block codes, $w_{im} = 1$, and consequently $n_{iM} = w$. 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

$$\alpha_M = w - 1 \geq 0,$$

     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 $w_{im} = 2$. In this case, $n_{iM} \leq \lfloor w/2 \rfloor, i = 1, 2$. The maximum exponent is

$$\alpha_M \leq 2\left\lfloor \frac{w}{2} \right\rfloor - w - 1 \tag{2.49}$$

$$= \begin{cases} -1, & w \text{ even.} \\ -2, & w \text{ odd.} \end{cases} \tag{2.50}$$

     This implies the existence of interleaver gain when $N \to \infty$. Particularly, $\alpha_M = -1$ for rate $1/n$ 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

$$P_b(e) \approx N^{-1} N_{1f,eff} N_{2f,eff} Q\left(\sqrt{\frac{2 d_{fp,eff} r E_b}{N_0}}\right) \tag{2.51}$$

     where $d_{fp,eff}$ is the **effective free distance** of the PCCC [45], which is defined to be the minimum weight of the PCCC 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. Obviously the effective free distance of the constituent codes should be maximized to minimize $P_b(e)$.

2.6.6 Conclusion 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 input sequence with $w = 3$), then d_4 (minimum weight of codeword sequences produced by input sequence with $w = 4$), and so on, should be chosen.

A comment is that, to achieve a good performance, it is important that the constituent codes are recursive, 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 $1/n$ recursive convolutional encoder with memory m, the effective free distance is upper bounded by [43] [44]

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

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

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

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

2.6.7 Performance Upper Bound of SCCC

Since SCCC has the same components as PCCC, it can be analyzed in a similar way. In particular, the uniform interleaver concept can still be used.

Let the superscripts o and i refer to the outer and inner constituent codes, respectively. N is 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 is 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 to be transmitted as the output of the whole SCCC encoder. The overall code rate r equals $r^o r^i$.

Similar to PCCC, let $A_{w,h}^{C_s}$ 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, and produced by information sequences of weight w. According to Equation 2.34, the upper bound on the bit error probability is

$$P_b(e) \leq \sum_{h=d_{fs}}^{N/r^i} \sum_{w=1}^{Nr^o} \frac{w}{Nr^o} A_{w,h}^{C_s} Q\left(\sqrt{\frac{2hrE_b}{N_0}}\right), \tag{2.52}$$

where d_{fs} is the minimum weight of all nonzero output 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_f^i. For SCCC with a pseudorandom interleaver, it is valid to assume $\lfloor d_{fs}/d_f^i \rfloor = 1$.

Suppose a uniform interleaver is used between the inner and outer encoders. When the coefficients $A_{w,l}^{C_o}$ and $A_{l,h}^{C_i}$ of the outer and inner codes are known, the coefficients of $A_{w,h}^{C_s}$ can be obtained as [28] [39]

$$A_{w,h}^{C_s} = \sum_{l=d_f^o}^{N} \frac{A_{w,l}^{C_o} A_{l,h}^{C_i}}{\binom{N}{l}} \tag{2.53}$$

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

$$A_{w,h}^{C_s} \approx \sum_{l=d_f^o}^{N} \sum_{n^o=1}^{n_M^o} \sum_{n^i=1}^{n_M^i} \frac{\binom{Nr^o}{n^o}\binom{N}{n^i}}{\binom{N}{l}} A_{w,l,n^o}^{o} A_{l,h,n^i}^{i} \tag{2.54}$$

where M indicates the maximum number of error events concatenated in a codeword sequence. Using the approximation in Equation 2.43, Equation 2.54 can be further approximated by

$$A_{w,h}^{C_s} \approx \sum_{l=d_f^o}^{N} \sum_{n^o=1}^{n_M^o} \sum_{n^i=1}^{n_M^i} N^{n^o+n^i-l} \frac{(r^o)^{n^o} l!}{n^o! n^i!} A_{w,l,n^o}^{o} A_{l,h,n^i}^{i} \tag{2.55}$$

Substituting Equation 2.55 into Equation 2.52, the upper bound on $P_b(e)$ is approximately

$$P_b(e) \leq \sum_{h=d_{fs}}^{N/r^i} \left(\sum_{w=1}^{Nr^o} \sum_{l=d_f^o}^{N} \sum_{n^o=1}^{n_M^o} \sum_{n^i=1}^{n_M^i} N^{n^o+n^i-l-1} \frac{w(r^o)^{n^o-1} l!}{n^o! n^i!} A^o_{w,l,n^o} A^i_{l,h,n^i} \right) Q\left(\sqrt{\frac{2hrE_b}{N_0}}\right) \tag{2.56}$$

Similar to PCCC, a very important parameter of SCCC is the exponent of N, which determines the interleaver gain. Denote the exponent as $\alpha(w,h) = n^o + n^i - l - 1$. As $N \to \infty$, the dominant term in Equation 2.56 for a given h is the one with the largest exponent of N,

$$\alpha(h) = \max_{n^o, n^i, w} \{n^o(h) + n^i(h) - l - 1\}. \tag{2.57}$$

Investigation of $\alpha(h)$ yields a better understanding of the performance of SCCC. The following two important cases are discussed:

- $\alpha(d_{fs})$, which corresponds to $h = d_{fs}$, the free distance of SCCC. The term with exponent $\alpha(d_{fs})$ in Equation 2.56 dominates the bit error performance at high E_b/N_0;
- the largest value $\alpha_M = \max_h \alpha(h)$, which determines the dominant term as $N \to \infty$.

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

   By definition, we know that

$$n^i \leq \left\lfloor \frac{d_{fs}}{d_f^i} \right\rfloor \tag{2.58}$$

$$n^o \leq \left\lfloor \frac{l}{d_f^o} \right\rfloor \tag{2.59}$$

   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

$$\begin{aligned} \alpha(d_{fs}) &\leq \max_l \left\{ \left\lfloor \frac{d_{fs}}{d_f^i} \right\rfloor + \left\lfloor \frac{l}{d_f^o} \right\rfloor - l - 1 \right\} \\ &\leq \max_l \left\{ \left\lfloor \frac{l}{d_f^o} \right\rfloor - l \right\} \\ &= \left(\left\lfloor \frac{l}{d_f^o} \right\rfloor - l \right) \Big|_{l=d_f^o} \\ &= 1 - d_f^o \end{aligned} \tag{2.60}$$

   Thus, when E_b/N_0 is large, it can be deduced from Equation 2.56 that

$$P_b(e) \leq C_h N^{1-d_f^o} Q\left(\sqrt{\frac{2d_{fs}rE_b}{N_0}}\right), \tag{2.61}$$

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. $\alpha_M = \max_h \alpha(h)$

   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 combinations with equal probability. In this case, $n_M^i = l$ and

   $$\alpha_M = n_M^o - 1 \geq 0 \tag{2.62}$$

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

   - For a recursive convolutional code, 2 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 $n_M^i = \lfloor l/2 \rfloor$ error events. Notice that $n_M^o \leq \lfloor l/d_f^o \rfloor$ by definition. So

   $$\begin{aligned} \alpha_M &\leq \max_l \left\{ \left\lfloor \frac{l}{d_f^o} \right\rfloor + \left\lfloor \frac{l}{2} \right\rfloor - l - 1 \right\} \\ &\leq \max_l \left\{ \left\lfloor \frac{l}{d_f^o} \right\rfloor - \left\lfloor \frac{l+1}{2} \right\rfloor - 1 \right\} \\ &= \left(\left\lfloor \frac{l}{d_f^o} \right\rfloor - \left\lfloor \frac{l+1}{2} \right\rfloor - 1 \right) \Bigg|_{l=d_f^o} \\ &= - \left\lfloor \frac{d_f^o + 1}{2} \right\rfloor \\ &< 0, \qquad \text{since} \quad d_f^o > 0. \end{aligned} \tag{2.63}$$

   This shows that the exponent α_M is always a negative integer. Consequently, $\alpha(h) < 0, \forall h$, and interleaver gain exists. Substituting Equation 2.63 into Equation 2.56, we get the approximate upper bound of the bit error probability:

   $$P_b(e) \leq C_h N^{-\lfloor (d_f^o+1)/2 \rfloor} Q\left(\sqrt{\frac{2h(\alpha_M) r E_b}{N_0}} \right). \tag{2.64}$$

An important point is that for PCCC with rate $1/n$ RSCs, the interleaver gain can only be N^{-1}, while it can be $N^{-2}, N^{-3}, ...$ for SCCC depending on the value of d_f^o [39]. As a result, the slope of $P_b(e)$ vs. 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 2.64, $h(\alpha_M)$ is the minimum output weight associated with the largest exponent α_M. 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,

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

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,

$$h(\alpha_M) = \frac{(d_f^o - 3)d_{f,eff}^i}{2} + h_m^{(3)}$$

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

2.6.8 Conclusion on SCCC Design

As a summary of the above discussion, several guidelines for the design of SCCC are listed here.

- 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 recursiveness ensures an interleaver gain.
- The free distance d_f^o of the outer code should be as large as possible. Examining Equation 2.64, we see that this is required to minimize the exponent of N for a higher interleaver gain.
- The effective free distance of the inner code $d_{f,eff}^i$ needs to be maximized. Increasing $d_{f,eff}^i$ will increase the output weight $h(\alpha_M)$, which diminishes the $Q(\cdot)$ function in Equation 2.64.

2.7 Summary

In this chapter, the 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. It was shown that they both provide BER performance close to Shannon limit at around 10^{-4} <BER< 10^{-7}. The most significant advantage of them in comparison to convolutional codes and block codes is that their probability of error 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 SNR region) or $N^{-\lfloor (d_f^o+1)/2 \rfloor}$ (at low-to-medium SNR region and N is large) with respect to the increase of frame size N. Based on the analysis, design guidlines for both PCCC and SCCC were summarized.

The intent of this chapter is 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 two important advances described in Chapter 2 is the subject of the remainder of this report.

Chapter 3

Iterative Decoding of Concatenated Codes

3.1 Introduction

In addition to the concatenation of several convolutional codes and the pseudorandom interleaver, a very important feature of PCCC (turbo code) and SCCC is that their decoding procedure is done iteratively and that 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. Since the complexity of both PCCC and SCCC lies in the iterative decoder, it is very important to understand the decoding algorithms and find the simplest way to realize the decoder without compromising performance.

While the iteration proceeds, the constituent decoders exchange soft information between each other. As proved by Forney [11], the optimal soft output should be the *a posteriori* probability (APP) distribution, the sequence of probability distribution conditioned on the received signal. Two major types of decoding algorithms have been proposed to decode the turbo codes. One is the MAP family. This family of decoding algorithms includes MAP which is the optimal method that produces the APP information, its additive form, Log-MAP, and its suboptimal additive form, Max-Log-MAP. The other class of decoding algorithm is based on the modification of the VA, the so called soft-output Viterbi algorithm (SOVA), which is suboptimal since the required APP can not be provided.

The symbol-by-symbol maximum *a posteriori* (MAP) [23] algorithm is a trellis-based soft-output decoding algorithm. Unlike the VA, which is a maximum likelihood trellis decoding method and minimizes the word error probability, MAP 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 too difficult for practical application, mainly because of the numerical 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 without its problems in practical implementation. Like Max-Log-MAP and SOVA, Log-MAP works exclusively in the logarithmic domain. Multiplication is converted to addition, while addition is converted to a $\max^*(\cdot)$ operation:

$$\max^*(x,y) = \ln(e^x + e^y) = \max(x,y) + \ln(1 + e^{-|x-y|}) = \max(x,y) + f_c(|x-y|) \quad (3.65)$$

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

$$\max^*(x,y,z) = \max^*(x,\ \max^*(y,z)) \quad (3.66)$$

$$\max^*(x+y, x+z) = x + \max^*(y,z) \quad (3.67)$$

$$\max^*(-\infty, x) = x. \quad (3.68)$$

Max-Log-MAP and SOVA are the suboptimal algorithms designed for easy implementation [26] [46] [36] [47]. However, their performance suffers, especially at the low signal-to-noise region. Max-Log-MAP uses exactly the same approach as MAP and Log-MAP with forward and backward recursions. It reduces the complexity by approximating the $\max^*(\cdot)$ function by 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 [48] [49]. In comparison, SOVA builds upon the VA with some additional real value additions and storages. In essence, SOVA uses the same metric and gives identical hard decisions as Max-Log-MAP [50]. The complexity of SOVA is approximately half of that of Log-MAP.

At each step k in a trellis, MAP (and Log-MAP) splits all paths into two sets, one with the information bit $u_k = 1$, and the other with $u_k = 0$ [48] [51]. MAP computes the LLR of these two sets. 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 Max-Log-MAP algorithm is the LLR of these two paths. On the other hand, SOVA considers only the survivor path and the competing path which joins it. SOVA can always find the most likely (ML) path, but the best competing path may have been eliminated before merging with the ML path. Consequently, soft output of SOVA can deviate from that of Max-Log-MAP. The iterative decoder using SOVA degrades from that using Max-Log-MAP.

Since SOVA performs worse than Log-MAP for similar complexity and since Log-MAP

is more suited to parallel processing [52], only MAP and Log-MAP are presented in detail in this report. Max-Log-MAP can be readily obtained by replacing all $\max^*(\cdot)$ with $\max(\cdot)$.

3.2 Conventional Log-MAP Algorithm

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 decoding algorithm for turbo codes. A more general form of it is called SISO. To make the notation clear, an edge, sometimes called branch, of the encoder trellis is introduced in Figure 3.13. Here $s_k^S(e)$ and $s_k^E(e)$ are the starting and ending states of the 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.

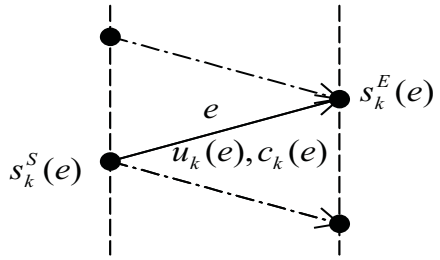


Figure 3.13: An edge of the trellis.

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

$$\begin{aligned} M_k(e) &= P(s_k^E(e), Y_k | s_k^S(e)) \\ &= \sum_{X_k} P(s_k^E(e) | s_k^S(e)) \cdot P(X_k | e) \cdot P(Y_k | X_k) \end{aligned} \tag{3.69}$$

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

$$\begin{aligned} A_k(s) &= P(s_k^E(e) = s, Y_1^k) \\ &= \sum_{e: s_k^E(e) = s} A_{k-1}(s_k^S(e)) \cdot M_k(e), \qquad k = 1, ...N-1. \end{aligned} \tag{3.70}$$

$$\begin{aligned} B_k(s) &= P(Y_{k+1}^N | s_{k+1}^S(e) = s) \\ &= \sum_{e: s_{k+1}^S(e) = s} B_{k+1}(s_{k+1}^E(e)) \cdot M_{k+1}(e), \quad k = N-1, ..., 1. \end{aligned} \tag{3.71}$$

Suppose the encoder starts with a known state S_0; then the $A_k(s)$ computation will be initialized as

$$A_0(s) = \begin{cases} 1, & s = S_0. \\ 0, & \text{otherwise.} \end{cases} \tag{3.72}$$

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

$$B_N(s) = \begin{cases} 1, & s = S_N. \\ 0, & \text{otherwise.} \end{cases} \tag{3.73}$$

However, if the final state of the trellis is unknown,

$$B_N(s) = \frac{1}{2^m}, \qquad \forall s. \tag{3.74}$$

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

$$\begin{aligned} \sigma_k(e) &= P(e, Y_1^N) \\ &= A_{k-1}(s_k^S(e)) \cdot M_k(e) \cdot B_k(s_k^E(e)), \end{aligned} \tag{3.75}$$

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

$$\begin{aligned} P_k^A(c;O) &= P(c_k = c|Y_1^N) \\ &= \frac{1}{P(Y_1^N)} \sum_{e:c(e)=c} \sigma_k(e) \\ &= \frac{1}{P(Y_1^N)} \sum_{e:c(e)=c} A_{k-1}(s_k^S(e)) \cdot M_k(e) \cdot B_k(s_k^E(e)) \end{aligned} \tag{3.76}$$

$$\begin{aligned} P_k^A(u;O) &= P(u_k = u|Y_1^N) \\ &= \frac{1}{P(Y_1^N)} \sum_{e:u(e)=u} \sigma_k(e) \\ &= \frac{1}{P(Y_1^N)} \sum_{e:u(e)=u} A_{k-1}(s_k^S(e)) \cdot M_k(e) \cdot B_k(s_k^E(e)) \end{aligned} \tag{3.77}$$

The term $1/P(Y_1^N)$ is a constant for a given frame, so it is ignored from the computation in the following descriptions.

Convert the variables into logarithmic domain so that the computation will be mainly addition instead of multiplication. Also, both the input and the output of the decoder module will become LLRs. In the following equations, $P_k^A(\cdot)$ represents the complete probability, $P_k(\cdot)$ is the extrinsic probability, $\lambda_k^A(\cdot)$ is the complete LLR information, $\lambda_k(\cdot)$ is the extrinsic LLR information, I denotes an input variable, O denotes an output variable, u is the

information symbol $(u^1, ..., u^j, ..., u^{k_0})$ 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:

$$\alpha_k(s) = \ln A_k(s) \tag{3.78}$$

$$\beta_k(s) = \ln B_k(s) \tag{3.79}$$

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

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

$$\lambda_k^A(c;O) = \ln \frac{P_k^A(c;O)}{P_k^A(c=\underline{0};O)} \tag{3.82}$$

$$\lambda_k^A(u;O) = \ln \frac{P_k^A(u;O)}{P_k^A(u=\underline{0};O)} \tag{3.83}$$

$$\lambda_k(c;O) = \ln \frac{P_k(c;O)}{P_k(c=\underline{0};O)} \tag{3.84}$$

$$\lambda_k(u;O) = \ln \frac{P_k(u;O)}{P_k(u=\underline{0};O)} \tag{3.85}$$

$$\lambda_k^A(c^j;O) = \ln \frac{P_k^A(c^j=1;O)}{P_k^A(c^j=0;O)}, \quad j=1,...,n_0. \tag{3.86}$$

$$\lambda_k^A(u^j;O) = \ln \frac{P_k^A(u^j=1;O)}{P_k^A(u^j=0;O)}, \quad j=1,...,k_0. \tag{3.87}$$

$$\lambda_k(c^j;O) = \ln \frac{P_k(c^j=1;O)}{P_k(c^j=0;O)}, \quad j=1,...,n_0. \tag{3.88}$$

$$\lambda_k(u^j;O) = \ln \frac{P_k(u^j=1;O)}{P_k(u^j=0;O)}, \quad j=1,...,k_0. \tag{3.89}$$

For the 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, y_k^1 corresponding to the k-th systematic bits. Antipodal transmission with amplitude A is assumed; thus, $x_k^1, x_k^2 \in (A, -A)$. The LLR is the logarithmic ratio of the probability that a bit equals a 1 to the probability that a bit equals a 0. The path metric of the conventional Log-MAP decoder is deduced below.

In Equation 3.69, $P(X_k|e)$ 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 $\lambda_k(u;I)$ of the information bit:

$$P(s_k^E(e)|s_k^S(e)) = \begin{cases} P(u_k=1) = \frac{e^{\lambda_k(u;I)}}{1+e^{\lambda_k(u;I)}}, & u_k = 1. \\ P(u_k=0) = \frac{1}{1+e^{\lambda_k(u;I)}}, & u_k = 0. \end{cases} \tag{3.90}$$

Thus,

$$\ln P(s_k^E(e)|s_k^S(e)) = \begin{cases} \lambda_k(u;I) - \ln\left(1+e^{\lambda_k(u;I)}\right), & u_k = 1. \\ -\ln\left(1+e^{\lambda_k(u;I)}\right), & u_k = 0. \end{cases} \tag{3.91}$$

For an AWGN channel with noise variance σ^2,

$$\begin{aligned} P(Y_k|X_k) &= P(y_k^1|u_k)\cdot P(y_k^2|u_k, s_k^S(e), s_k^E(e)) \\ &= \frac{1}{\sqrt{2\pi\sigma^2}}\exp\left(\frac{-(y_k^1-x_k^1)^2}{2\sigma^2}\right)\cdot\frac{1}{\sqrt{2\pi\sigma^2}}\exp\left(\frac{-(y_k^2-x_k^2)^2}{2\sigma^2}\right). \end{aligned} \tag{3.92}$$

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

$$\begin{aligned} \ln P(Y_k|X_k) &= \frac{y_k^1 x_k^1}{\sigma^2} + \frac{y_k^2 x_k^2}{\sigma^2} \\ &= \frac{A}{\sigma^2}\left[y_k^1(2c_k^1-1) + y_k^2(2c_k^2-1)\right]. \end{aligned} \tag{3.93}$$

Since

$$\lambda_k(c^j;I) = (2A/\sigma^2)y_k^j$$

which is explained in Equation 3.149, Equation 3.93 can be also written as

$$\ln P(Y_k|X_k) = 0.5\lambda_k(c^1;I)(2c_k^1-1) + 0.5\lambda_k(c^2;I)(2c_k^2-1). \tag{3.94}$$

Using Equation 3.91 and 3.94, the path metric in the logarithmic domain is obtained as follows:

$$\begin{aligned} & \gamma_k(e) = \ln M_k(e) \\ = \; & \ln P(s_k^E(e)|s_k^S(e)) + \ln P(Y_k|X_k) \\ = \; & \begin{cases} \lambda_k(u;I) - \ln(1+e^{\lambda_k(u;I)}) + 0.5\lambda_k(c^1;I) + 0.5\lambda_k(c^2;I)(2c_k^2-1), & u_k = 1. \\ -\ln(1+e^{\lambda_k(u;I)}) - 0.5\lambda_k(c^1;I) + 0.5\lambda_k(c^2;I)(2c_k^2-1), & u_k = 0. \end{cases} \end{aligned} \tag{3.95}$$

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 zero. Although the systematic bit for RSC2 is not transmitted, it is known to be the systematic bit of RSC1 in the interleaved order. Since the LLR input $\lambda_k(c^1;I)$ is obtained by scaling the received signal of the systematic bit (see Equation 3.149), we can interleave $\lambda_{1k}(c^1;I)$ of RSC1 to make $\lambda_{2k}(c^1;I)$ 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 for

the forward path metrics $\alpha_k(s)$ and backward path metrics $\beta_k(s)$. The terms $\alpha_k(s)$ and $\beta_k(s)$ are computed as follows:

$$\begin{aligned}
\alpha_k(s) &= \ln \sum_{e:s_k^E(e)=s} A_{k-1}(s_k^S(e)) \cdot M_k(e) \\
&= \max_{e:s_k^E(e)=s}{}^{*} \left[\alpha_{k-1}(s_k^S(e)) + \gamma_k(e)\right], \quad k = 1, ..., N-1. \qquad (3.96) \\
\beta_k(s) &= \ln \sum_{e:s_{k+1}^S(e)=s} B_{k+1}(s_{k+1}^E(e)) \cdot M_{k+1}(e) \\
&= \max_{e:s_{k+1}^S(e)=s}{}^{*} \left[\beta_{k+1}(s_{k+1}^E(e)) + \gamma_{k+1}(e)\right], \quad k = N-1, ..., 1. \qquad (3.97)
\end{aligned}$$

where the operation max* is defined in Equation 3.65. The summation in [ ] represents all the terms on which the max* function operates. 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

$$\alpha_k(s) \Leftarrow \alpha_k(s) - \max_s \alpha_k(s) \qquad (3.98)$$

$$\beta_k(s) \Leftarrow \beta_k(s) - \max_s \beta_k(s) \qquad (3.99)$$

This practice will make sure that $\alpha_k(s)$ and $\beta_k(s)$ do not overflow in a real implementation. It can be proved that adding a constant to $\alpha_k(s)$ or $\beta_k(s)$ has no influence on the soft output since the constant will be canceled eventually.

Suppose the encoder starts with a known state S_0; the $\alpha_k(s)$ computation will then be initialized as

$$\alpha_0(s) = \begin{cases} 0, & s = S_0. \\ -\infty, & \text{otherwise.} \end{cases} \qquad (3.100)$$

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

$$\beta_N(s) = \begin{cases} 0, & s = S_N. \\ -\infty, & \text{otherwise.} \end{cases} \qquad (3.101)$$

If the final state of the trellis is unknown,

$$\beta_N(s) = 0 \quad \text{(or any other constant)}, \qquad \forall s. \qquad (3.102)$$

For numerical computation, a very large value is assigned to take place of ∞.

The complete LLR information of the information bit is

$$\begin{aligned}\lambda_k^A(u;O) &= \ln \sum_{e:u(e)=1} A_{k-1}(s_k^S(e)) \cdot M_k(e) \cdot B_k(s_k^E(e)) \\ &\quad - \ln \sum_{e:u(e)=0} A_{k-1}(s_k^S(e)) \cdot M_k(e) \cdot B_k(s_k^E(e)) \\ &= \max_{e:u(e)=1}^{*} \left[\alpha_{k-1}(s_k^S(e)) + \gamma_k(e) + \beta_k(s_k^E(e))\right] \\ &\quad - \max_{e:u(e)=0}^{*} \left[\alpha_{k-1}(s_k^S(e)) + \gamma_k(e) + \beta_k(s_k^E(e))\right]. \end{aligned} \tag{3.103}$$

Substituting Equation 3.95 for $\gamma_k(e)$ and pulling out the common factor, we have

$$\begin{aligned}\lambda_k^A(u;O) &= \max_{e:u(e)=1}^{*} \Big[\lambda_k(u;I) - \ln(1+e^{\lambda_k(u;I)}) + 0.5\lambda_k(c^1;I) + 0.5\lambda_k(c^2;I)(2c_k^2-1) \\ &\qquad +\alpha_{k-1}(s_k^S(e)) + \beta_k(s_k^E(e))\Big] \\ &\quad - \max_{e:u(e)=0}^{*} \Big[-\ln(1+e^{\lambda_k(u;I)}) - 0.5\lambda_k(c^1;I) + 0.5\lambda_k(c^2;I)(2c_k^2-1) \\ &\qquad +\alpha_{k-1}(s_k^S(e)) + \beta_k(s_k^E(e))\Big] \\ &= (\lambda_k(u;I) - \ln(1+e^{\lambda_k(u;I)}) + 0.5\lambda_k(c^1;I)) \\ &\quad + \max_{e:u(e)=1}^{*} \left[\alpha_{k-1}(s_k^S(e)) + 0.5\lambda_k(c^2;I)(2c_k^2-1) + \beta_k(s_k^E(e))\right] \\ &\quad -(-\ln(1+e^{\lambda_k(u;I)}) - 0.5\lambda_k(c^1;I)) \\ &\quad - \max_{e:u(e)=0}^{*} \left[\alpha_{k-1}(s_k^S(e)) + 0.5\lambda_k(c^2;I)(2c_k^2-1) + \beta_k(s_k^E(e))\right] \\ &= \lambda_k(u;I) + \lambda_k(c^1;I) \\ &\quad + \max_{e:u(e)=1}^{*} \left[\alpha_{k-1}(s_k^S(e)) + 0.5\lambda_k(c^2;I)(2c_k^2-1) + \beta_k(s_k^E(e))\right] \\ &\quad - \max_{e:u(e)=0}^{*} \left[\alpha_{k-1}(s_k^S(e)) + 0.5\lambda_k(c^2;I)(2c_k^2-1) + \beta_k(s_k^E(e))\right]. \end{aligned} \tag{3.104}$$

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

Since both constituent decoders share the same systematic information $\lambda_k(c^1;I)$, it has to be removed from the complete information as well as $\lambda_k(u;I)$ to provide the proper extrinsic information [47]. The extrinsic information will be sent to the next constituent decoder and used as its *a priori* information. The extrinsic information is

$$\begin{aligned}\lambda_k(u;O) &= \lambda_k^A(u;O) - \lambda_k(u;I) - \lambda_k(c^1;I) \\ &= \max_{e:u(e)=1}^{*} \left[\alpha_{k-1}(s_k^S(e)) + 0.5\lambda_k(c^2;I)(2c_k^2-1) + \beta_k(s_k^E(e))\right] \\ &\quad - \max_{e:u(e)=0}^{*} \left[\alpha_{k-1}(s_k^S(e)) + 0.5\lambda_k(c^2;I)(2c_k^2-1) + \beta_k(s_k^E(e))\right]. \end{aligned} \tag{3.105}$$

Equation 3.95, 3.96, 3.97, 3.103, and 3.105 make up the conventional Log-MAP algorithm for a constituent decoder, where $\lambda_k(c;I)$ is computed using Equation 3.149.

Log-MAP algorithm can be embedded in the turbo decoder and process the received signal iteratively. Let the first constituent decoder be DEC1, and the second constituent decoder be DEC2. In the first iteration, the *a priori* information $\lambda_{1k}(u;I)$ of DEC1 is initialized to be all-zero, assuming that 1 and 0 are equally probable for an information bit.

After DEC1 produces its output, its extrinsic information $\lambda_{1k}(u;O)$ will be interleaved and then passed as the *a priori* information for DEC2: $\lambda_{2k}(u;I) = \lambda_{1\alpha(k)}(u;O)$. After DEC2 generates its output, one iteration is completed. The extrinsic information $\lambda_{2k}(u;O)$ from DEC2 can be deinterleaved and fed back as the *a priori* information of DEC1 in the next iteration: $\lambda_{1\alpha(k)}(u;I) = \lambda_{2k}(u;O)$. All the other iterations can be done by repeating the above procedure.

3.3 Simplified Log-MAP Algorithm

The conventional Log-MAP is more suitable for practical implementation than MAP. However, it is observed that three modifications can be made to further simplify the algorithm.

1. Notice that if there is any constant common to $u_k = 0$ and $u_k = 1$ in $\gamma_k(e)$, it will eventually be canceled during the computation of the soft output; thus, it can be ignored from the computation. For Equation 3.95, we found that $\gamma_k(e)$ can be rewritten as:

$$\begin{aligned}\gamma_k(e) &= \begin{cases} [\lambda_k(u;I) + c_k^1\lambda_k(c^1;I) + c_k^2\lambda_k(c^2;I)] \\ \quad + \left[-\ln(1+e^{\lambda_k(u;I)}) - 0.5\lambda_k(c^1;I) - 0.5\lambda_k(c^2;I)\right], \\ \qquad\qquad\qquad\qquad\qquad\qquad c_k^1 = u_k = 1. \\ \qquad [c_k^1\lambda_k(c^1;I) + c_k^2\lambda_k(c^2;I)] \\ \quad + \left[-\ln(1+e^{\lambda_k(u;I)}) - 0.5\lambda_k(c^1;I) - 0.5\lambda_k(c^2;I)\right], \\ \qquad\qquad\qquad\qquad\qquad\qquad c_k^1 = u_k = 0. \end{cases} \\ &= u_k\lambda_k(u;I) + c_k^1\lambda_k(c^1;I) + c_k^2\lambda_k(c^2;I) + C, \qquad c_k^1 = u_k \in (0,1). \end{aligned} \tag{3.106}$$

The constant is

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

and can be ignored from 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. Instead of interleaving the systematic information $\lambda_k(c^1;I)$ to share it between DEC1 and DEC2, $\lambda_k(c^1;I)$ can be kept only for DEC1. This results when we treat RSC1 as a rate 1/2 code, but RSC2 as a rate 1/1 code whose codeword has only a single parity bit. 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

$$\begin{cases} \gamma_{1k}(e) = u_k\lambda_{1k}(u;I) + c_{1k}^1\lambda_{1k}(c^1;I) & +c_{1k}^2\lambda_{1k}(c^2;I), & \text{for RSC1.} \\ \gamma_{2k}(e) = u_k\lambda_{2k}(u;I) & +c_{2k}^2\lambda_{2k}(c^2;I), & \text{for RSC2.} \end{cases} \tag{3.107}$$

The computation for $\gamma_{2k}(e)$ is simpler because the systematic bit c_{2k}^1 is not considered.

3. Since $\lambda_k(c^1;I)$ is included in the branch metrics for DEC1, but not for DEC2, the expression of the extrinsic information needs to be modified. The complete information for DEC1 is the same as that in Equation 3.104:

$$\begin{aligned} \lambda_{1k}'^{A}(u;O) &= \lambda_{1k}'(u;I) + \lambda_{1k}(c^1;I) \\ &\quad + \max_{e:u(e)=1}^{*}\left[\alpha_{1,k-1}(s_k^S(e)) + 0.5\lambda_{1k}(c^2;I)(2c_{1k}^2-1) + \beta_{1k}(s_k^E(e))\right] \\ &\quad - \max_{e:u(e)=0}^{*}\left[\alpha_{1,k-1}(s_k^S(e)) + 0.5\lambda_{1k}(c^2;I)(2c_{1k}^2-1) + \beta_{1k}(s_k^E(e))\right]. \end{aligned} \tag{3.108}$$

Since $\lambda_{2k}(c^1;I)$ is not included in $\gamma_{2k}(e)$ for DEC2, $\lambda_{1k}(c^1;I)$ should be part of the extrinsic information from DEC1. Therefore, the systematic information comes to DEC2 as a part of the *a priori* information. So,

$$\begin{aligned} \lambda_{1k}'(u;O) &= \lambda_{1k}'^{A}(u;O) - \lambda_{1k}'(u;I) \\ &= \lambda_{1k}(c^1;I) \\ &\quad + \max_{e:u(e)=1}^{*}\left[\alpha_{1,k-1}(s_k^S(e)) + 0.5\lambda_{1k}(c^2;I)(2c_{1k}^2-1) + \beta_{1k}(s_k^E(e))\right] \\ &\quad - \max_{e:u(e)=0}^{*}\left[\alpha_{1,k-1}(s_k^S(e)) + 0.5\lambda_{1k}(c^2;I)(2c_{1k}^2-1) + \beta_{1k}(s_k^E(e))\right] \\ &= \lambda_{1k}(u;O) + \lambda_{1k}(c^1;I) \end{aligned} \tag{3.109}$$

is the extrinsic information for DEC1, which is to be interleaved as the *a priori* information $\lambda_{2k}'(u;I)$ for DEC2. The new *a priori* information is related to the old one by

$$\lambda_{2k}'(u;I) = \lambda_{1\alpha(k)}'(u;O)$$

$$\begin{aligned} &= \lambda_{1\alpha(k)}(u;O) + \lambda_{1\alpha(k)}(c^1;I) \\ &= \lambda_{2k}(u;I) + \lambda_{1\alpha(k)}(c^1;I), \end{aligned} \tag{3.110}$$

which is the summation of the old *a priori* information and the old systematic information. The complete information for DEC2 is similar to that of Equation 3.104 minus the $\lambda_k(c^1;I)$ term:

$$\begin{aligned} \lambda'^A_{2k}(u;O) &= \lambda'_{2k}(u;I) \\ &\quad + \max^*_{e:u(e)=1}\left[\alpha_{2,k-1}(s^S_k(e)) + 0.5\lambda_{2k}(c^2;I)(2c^2_{2k}-1) + \beta_{2k}(s^E_k(e))\right] \\ &\quad - \max^*_{e:u(e)=0}\left[\alpha_{2,k-1}(s^S_k(e)) + 0.5\lambda_{2k}(c^2;I)(2c^2_{2k}-1) + \beta_{2k}(s^E_k(e))\right] \\ &= \left[\lambda_{2k}(u;I) + \lambda_{1\alpha(k)}(c^1;I)\right] + \lambda_{2k}(u;O) \\ &= \lambda^A_{2k}(u;O). \end{aligned} \tag{3.111}$$

The extrinsic information of DEC2 is

$$\begin{aligned} \lambda'_{2k}(u;O) &= \lambda'^A_{2k}(u;O) - \lambda'_{2k}(u;I) \\ &= \lambda_{2k}(u;O). \end{aligned} \tag{3.112}$$

We see that the extrinsic information of DEC2 is not changed when we treat RSC2 as a rate 1/1 code. Consequently, the *a priori* information for DEC1 remains the same, $\lambda'_{1k}(u;I) = \lambda_{1k}(u;I)$. As a consequence, the complete informations $\lambda^A_{1k}(u;O)$ and $\lambda^A_{2k}(u;O)$ remain unchanged. Also, we see that the computation for the extrinsic information is unified to be

$$\lambda'_k(u;O) = \lambda^A_k(u;O) - \lambda_k(u;I) \tag{3.113}$$

for both DEC1 and DEC2. In comparison to that for conventional Log-MAP in Equation 3.105, half of the subtraction is eliminated.

In summary, the simplified Log-MAP is composed of Equations 3.107, 3.96, 3.97, 3.103 and 3.113. The iterative decoding procedure is the same as that of conventional Log-MAP.

In the simplified form, the function of DEC1 and DEC2 is realized exactly as Log-MAP. Only the form is modified to reduce complexity. The DEC2 of simplified Log-MAP has two inputs, $\lambda_{2k}(u;I)$ and $\lambda_{2k}(c^2;I)$, instead of three, $\lambda_{2k}(u;I)$, $\lambda_{2k}(c^1;I)$ and $\lambda_{2k}(c^2;I)$, as in the conventional Log-MAP. The computation of the branch metric is reduced by about 90%, while the computation of the extrinsic information is reduced by 50%.

The complete information in simplified Log-MAP is the summation of two terms, the new *a priori* information and the new extrinsic information. However, the old systematic information is still there, combined with some other terms. For DEC1, its *a priori*

information is unchanged, but the new extrinsic information is the summation of the old systematic information and the old extrinsic information; for DEC2, the extrinsic information is unchanged, but the new *a priori* information is the summation of the old systematic information and the old *a priori* information. The complete information of both DEC1 and DEC2 stays the same.

3.4 General SISO module for PCCC and SCCC

Extending the derivation of simplified Log-MAP, we can obtain an general and efficient decoding algorithm, the SISO (soft-input soft-output) module, which can be used to decode both PCCC and SCCC [49]. In the following section, the algorithm is deduced thoroughly, starting with the 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 derivation, while additive SISO is the transformation of multiplicative SISO. Additive SISO is usually employed in implementations and is the one referred to when the term "SISO" is used.

SISO module has two input vectors, $\left[\lambda(u^1;I),...,\lambda(u^{k_0};I)\right]$ (*a priori* information of information bits) and $[\lambda(c^1;I),...,\lambda(c^{n_0};I)]$ (*a priori* information of codeword bits). After the processing, two output vectors are produced, $\left[\lambda(u^1;O),...,\lambda(u^{k_0};O)\right]$ (extrinsic information of information bits) and $[\lambda(c^1;O),\ ...,\ \lambda(c^{n_0};O)]$ (extrinsic information of codeword bits). $\lambda(c^i;I)$ is usually determined by the soft value of the received signal (see Equation 3.149). In particular, for a systematic constituent code, $\lambda(c^1;I)$ is usually called the systematic information. Generally, it is treated as *a priori* information of the first codeword bit, just as any other $\lambda(c^i;I)$.

3.4.1 Output Probabilities for Information Symbol and Codeword Symbol

Recall the branch metric in Equation 3.69,

$$M_k(e) = \sum_{X_k} P(s_k^E(e)|s_k^S(e)) \cdot P(X_k|e) \cdot P(Y_k|X_k). \tag{3.114}$$

The first term can be expressed as

$$\begin{aligned} P(s_k^E(e)|s_k^S(e)) &= P(u_k|Y_1^N) \\ &= P_k(u(e);I). \end{aligned} \tag{3.115}$$

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 equally probable, the third term is

$$\begin{aligned} P(Y_k|X_k) &= \frac{P(X_k|Y_k)P(Y_k)}{P(X_k)} \\ &= h_M P(X_k|Y_k) \\ &= h_M P(c_k|Y_k) \\ &= h_M P_k(c(e);I). \end{aligned} \tag{3.116}$$

In a trellis without multiple connections between two states, if $s_k^S(e)$ and $s_k^E(e)$ are connected,

$$M_k(e) = h_M P_k(u(e);I) \cdot P_k(c(e);I). \tag{3.117}$$

If there is no connection between $s_k^S(e)$ and $s_k^E(e)$, $M_k(e) = 0$. The *a posteriori* probabilities (APP) for the codeword symbol and the information symbol are

$$\begin{aligned} P_k^A(c;O) &= P(c_k = c|Y_1^N) \\ &= \frac{1}{P(Y_1^N)} \sum_{e:c(e)=c} \sigma_k(e) \\ &= \frac{1}{P(Y_1^N)} \sum_{e:c(e)=c} A_{k-1}(s_k^S(e)) \cdot M_k(e) \cdot B_k(s_k^E(e)) \\ &= h_c \sum_{e:c(e)=c} A_{k-1}(s_k^S(e)) \cdot P_k(u(e);I) \cdot P_k(c(e);I) \cdot B_k(s_k^E(e)) \\ &= h_c P_k(c(e);I) \sum_{e:c(e)=c} A_{k-1}(s_k^S(e)) \cdot P_k(u(e);I) \cdot B_k(s_k^E(e)) \end{aligned} \tag{3.118}$$

$$\begin{aligned} P_k^A(u;O) &= P(u_k = u|Y_1^N) \\ &= \frac{1}{P(Y_1^N)} \sum_{e:u(e)=u} \sigma_k(e) \\ &= \frac{1}{P(Y_1^N)} \sum_{e:u(e)=u} A_{k-1}(s_k^S(e)) \cdot M_k(e) \cdot B_k(s_k^E(e)) \\ &= h_c \sum_{e:u(e)=u} A_{k-1}(s_k^S(e)) \cdot P_k(u(e);I) \cdot P_k(c(e);I) \cdot B_k(s_k^E(e)) \\ &= h_c P_k(u(e);I) \sum_{e:u(e)=u} A_{k-1}(s_k^S(e)) \cdot P_k(c(e);I) \cdot B_k(s_k^E(e)). \end{aligned} \tag{3.119}$$

In Equations 3.116 to 3.119 above, $h_M = P(Y_k)/P(X_k)$ and $h_c = 1/P(Y_1^N)$ are constants for a given frame, so they will be ignored in the following descriptions. Computing the ratio of the output APP and the input conditional probability, we get the extrinsic information for the codeword symbol and the information symbol as follows:

$$P_k(c;O) = \frac{P_k^A(c;O)}{P_k(c;I)} = \sum_{e:c(e)=c} A_{k-1}(s_k^S(e)) \cdot P_k(u(e);I) \cdot B_k(s_k^E(e)) \tag{3.120}$$

$$P_k(u;O) = \frac{P_k^A(u;O)}{P_k(u;I)} = \sum_{e:u(e)=u} A_{k-1}(s_k^S(e)) \cdot P_k(c(e);I) \cdot B_k(s_k^E(e)). \quad (3.121)$$

The extrinsic information 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 extrinsic information can be used as the *a priori* information for the next decoding module.

3.4.2 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,

$$P_k(c;I) = \prod_{j=1}^{n_0} P_k(c^j;I) \quad (3.122)$$

$$P_k(u;I) = \prod_{j=1}^{k_0} P_k(u^j;I) \quad (3.123)$$

where $c^j \in (0,1)$ represents the value of the j-th bit of the output symbol c, and u^j represents the value of the j-th bit of the input symbol u. Equation 3.122 and 3.123 are valid when a bit interleaver is used ahead of the channel instead of a symbol interleaver. If the channel is AWGN, this criteria is always satisfied. Using Equation 3.122 along with Equation 3.120, the complete information for bit c^j is

$$\begin{aligned} P_k^A(c^j;O) &= \sum_{c:c^j(e)=c^j} P_k^A(c(e);O) \\ &= \sum_{c:c^j(e)=c^j} P_k(c(e);I) \cdot P_k(c(e);O) \\ &= P_k(c^j;I) \sum_{c:c^j(e)=c^j} P_k(c(e);O) \prod_{i=1,i\neq j}^{n_0} P_k(c^i(e);I), \quad j=1,...,n_0. \end{aligned} \quad (3.124)$$

The extrinsic information for bit c^j is

$$\begin{aligned} P_k(c^j;O) &= \frac{P_k^A(c^j;O)}{P_k(c^j;I)} \\ &= \sum_{c:c^j(e)=c^j} P_k(c(e);O) \prod_{i=1;i\neq j}^{n_0} P_k(c^i(e);I), \quad j=1,...,n_0. \end{aligned} \quad (3.125)$$

Similarly, the complete information and the extrinsic information can be deduced for an information bit as follows:

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

(3.126)

$$P_k(u^j;O) = \frac{P_k^A(u^j;O)}{P_k(u^j;I)}$$

$$= \sum_{u:u^j(e)=u^j} P_k(u(e);O) \prod_{i=1,i\neq j}^{k_0} P_k(u^i(e);I), \quad j=1,...,k_0. \tag{3.127}$$

3.4.3 Binary Transmission

Suppose the codeword bits are binary and the relationship between the codeword bit and the transmitted symbol is:

$$x_k^j = \begin{cases} A, & \text{when} \quad c_k^j = 1. \\ -A, & \text{when} \quad c_k^j = 0. \end{cases} \qquad k=1,...,N;\ j=1,...,n_0. \tag{3.128}$$

Define the ratios

$$R_{kj}^c \triangleq \frac{P_k(c^j=1;I)}{P_k(c^j=0;I)} = \frac{P(c_k^j=1|Y_1^N)}{P(c_k^j=0|Y_1^N)} \tag{3.129}$$

$$R_{kj}^u \triangleq \frac{P_k(u^j=1;I)}{P_k(u^j=0;I)} = \frac{P(u_k^j=1|Y_1^N)}{P(u_k^j=0|Y_1^N)}. \tag{3.130}$$

Considering Equation 3.129 and that $P(c_k^j=1|Y_1^N)+P(c_k^j=0|Y_1^N)=1$, we get

$$P_k(c^j=1;I) = \frac{R_{kj}^c}{1+R_{kj}^c} \tag{3.131}$$

$$P_k(c^j=0;I) = \frac{1}{1+R_{kj}^c} \tag{3.132}$$

or

$$P_k(c^j;I) = \frac{(R_{kj}^c)^{c^j}}{1+R_{kj}^c}. \tag{3.133}$$

Combining Equation 3.133 with 3.122, we get

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

$$= h_{pc} \prod_{j=1}^{n_0} (R_{kj}^c)^{c^j}. \tag{3.134}$$

Similarly, for the information bit we have

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

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

$$= h_{pu} \prod_{j=1}^{k_0} (R_{kj}^u)^{u^j}. \tag{3.135}$$

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

3.4.4 Multiplicative SISO

Using the previous results, we get the following expressions for multiplicative SISO:

1. Considering Equations 3.117, 3.134 and 3.135, the branch metric is

$$M_k(e) = \prod_{j=1}^{n_0}(R_{kj}^c)^{c^j} \cdot \prod_{j=1}^{k_0}(R_{kj}^u)^{u^j}, \tag{3.136}$$

   where the constants are omitted.

2. From Equation 3.70, 3.71 and 3.136, forward and backward recursions are obtained as

$$\begin{aligned} A_k(s) &= \sum_{e:s_k^E(e)=s} A_{k-1}(s_k^S(e)) \cdot M_k(e), \quad k = 1, ..., N-1. \\ &= \sum_{e:s_k^E(e)=s} A_{k-1}(s_k^S(e)) \cdot \prod_{j=1}^{n_0}(R_{kj}^c)^{c^j} \cdot \prod_{j=1}^{k_0}(R_{kj}^u)^{u^j} \qquad (3.137) \\ B_k(s) &= \sum_{e:s_{k+1}^S(e)=s} B_{k+1}(s_{k+1}^E(e)) \cdot M_{k+1}(e), \quad k = N-1, ..., 1. \\ &= \sum_{e:s_{k+1}^S(e)=s} B_{k+1}(s_{k+1}^E(e)) \cdot \prod_{j=1}^{n_0}(R_{k+1,j}^c)^{c^j} \cdot \prod_{j=1}^{k_0}(R_{k+1,j}^u)^{u^j} \qquad (3.138) \end{aligned}$$

   The $A_k(s)$ computation is initialized by Equation 3.72, and the $B_k(s)$ computation is initialized by Equation 3.73 or 3.74 depending on the knowledge of the final state.

3. When the constants are omitted from Equation 3.118, 3.119 and 3.136, the complete information for the codeword symbol and that for the information symbol are obtained as

$$\begin{aligned} P_k^A(c;O) &= \sum_{e:c(e)=c} A_{k-1}(s_k^S(e)) \cdot M_k(e) \cdot B_k(s_k^E(e)) \qquad (3.139) \\ &= \left(\prod_{j=1}^{n_0}(R_{kj}^c)^{c^j}\right) \sum_{e:c(e)=c} A_{k-1}(s_k^S(e)) \cdot \prod_{j=1}^{k_0}(R_{kj}^u)^{u^j} \cdot B_k(s_k^E(e)) \\ P_k^A(u;O) &= \sum_{e:u(e)=u} A_{k-1}(s_k^S(e)) \cdot M_k(e) \cdot B_k(s_k^E(e)) \qquad (3.140) \\ &= \left(\prod_{j=1}^{k_0}(R_{kj}^u)^{u^j}\right) \sum_{e:u(e)=u} A_{k-1}(s_k^S(e)) \cdot \prod_{j=1}^{n_0}(R_{kj}^c)^{c^j} \cdot B_k(s_k^E(e)) \end{aligned}$$

4. From Equation 3.120, 3.121, 3.139 and 3.140, the extrinsic information for the codeword symbol and that for the information symbol are:

$$\begin{aligned} P_k(c;O) &= \frac{P_k^A(c;O)}{\prod_{j=1}^{n_0}(R_{kj}^c)^{c^j}} \\ &= \sum_{e:c(e)=c} A_{k-1}(s_k^S(e)) \cdot \prod_{j=1}^{k_0}(R_{kj}^u)^{u^j} \cdot B_k(s_k^E(e)) \qquad (3.141) \\ P_k(u;O) &= \frac{P_k^A(u;O)}{\prod_{j=1}^{k_0}(R_{kj}^u)^{u^j}} \\ &= \sum_{e:u(e)=u} A_{k-1}(s_k^S(e)) \cdot \prod_{j=1}^{n_0}(R_{kj}^c)^{c^j} \cdot B_k(s_k^E(e)) \qquad (3.142) \end{aligned}$$

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

$$\begin{aligned} P_k^A(c^j;O) &= \sum_{e:c^j(e)=c^j} A_{k-1}(s_k^S(e)) \cdot M_k(e) \cdot B_k(s_k^E(e)), \\ & \qquad j = 1, ..., n_0. \qquad (3.143) \\ P_k^A(u^j;O) &= \sum_{e:u^j(e)=u^j} A_{k-1}(s_k^S(e)) \cdot M_k(e) \cdot B_k(s_k^E(e)), \\ & \qquad j = 1, ..., k_0. \qquad (3.144) \end{aligned}$$

6. From Equation 3.125 and 3.127, the extrinsic information for the codeword bits and that for the information bits are:

$$\begin{aligned} P_k(c^j;O) &= \frac{P_k^A(c^j;O)}{P_k(c^j;I)}, \qquad j = 1, ..., n_0. \qquad (3.145) \\ &= \sum_{e:c^j(e)=c^j} A_{k-1}(s_k^S(e)) \cdot \prod_{i=1, i\neq j}^{n_0}(R_{ki}^c)^{c^i} \cdot \prod_{i=1}^{i=k_0}(R_{ki}^u)^{u^i} \cdot B_k(s_k^E(e)) \\ P_k(u^j;O) &= \frac{P_k^A(u^j;O)}{P_k(u^j;I)}, \qquad j = 1, ..., k_0. \qquad (3.146) \\ &= \sum_{e:u^j(e)=u^j} A_{k-1}(s_k^S(e)) \cdot \prod_{i=1}^{n_0}(R_{ki}^c)^{c^i} \cdot \prod_{i=1, i\neq j}^{k_0}(R_{ki}^u)^{u^i} \cdot B_k(s_k^E(e)) \end{aligned}$$

3.4.5 Additive SISO

Using Equations 3.78 to 3.89, all computations will be converted to the logarithmic domain [49]. First consider the computation of the LLR input $\lambda_k(c;I)$ and $\lambda_k(u;I)$. Substituting Equation 3.134 for $P_k(c;I)$ and considering the definition of R_{kj}^c in Equation 3.129, we have

$$\lambda_k(c;I) = \ln \frac{h_{pc}\prod_{j=1}^{n_0}(R_{kj}^c)^{c^j}}{h_{pc}\prod_{j=1}^{n_0}(R_{kj}^c)^{0}}$$

$$
\begin{aligned}
&= \sum_{j=1}^{n_0} c^j \ln R_{kj}^c \\
&= \sum_{j=1}^{n_0} c^j \lambda_k(c^j; I). \qquad (3.147)
\end{aligned}
$$

Similarly, with Equation 3.135 for $P_k(u; I)$ and 3.130 for R_{kj}^u,

$$\lambda_k(u; I) = \sum_{j=1}^{k_0} u^j \lambda_k(u^j; I). \qquad (3.148)$$

This result shows that $\lambda_k(c; I)$ can be expressed as the weighted summation of the LLR input for each codeword bit $\lambda_k(c^j; I)$ and that $\lambda_k(u; I)$ can be expressed as the weighted summation of the LLR input $\lambda_k(u^j; I)$.

If 0 and 1 are equally probable for a codeword bit, $P(x_k^j = A) = P(x_k^j = -A)$. When the channel is AWGN, $\lambda_k(c^j; I)$ can be related to the received signal as follows:

$$
\begin{aligned}
\lambda_k(c^j; I) &= \ln \frac{P_k(c^j = 1; I)}{P_k(c^j = 0; I)} \\
&= \ln R_{kj}^c \\
&= \ln \frac{P(x_k^j = A | Y_1^N)}{P(x_k^j = -A | Y_1^N)} \\
&= \ln \frac{P(x_k^j = A | y_k^j)}{P(x_k^j = -A | y_k^j)} \\
&= \ln \frac{P(y_k^j | x_k^j = A) P(x_k^j = A) / P(y_k^j)}{P(y_k^j | x_k^j = -A) P(x_k^j = -A) / P(y_k^j)} \\
&= \ln \frac{P(y_k^j | x_k^j = A)}{P(y_k^j | x_k^j = -A)} \\
&= \ln \frac{\frac{1}{\sqrt{2\pi}\sigma} \exp\left(\frac{-1}{2\sigma^2}(y_k^j - A)^2\right)}{\frac{1}{\sqrt{2\pi}\sigma} \exp\left(\frac{-1}{2\sigma^2}(y_k^j + A)^2\right)} \\
&= \frac{2A}{\sigma^2} y_k^j \qquad (3.149)
\end{aligned}
$$

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.

As to the value of $\lambda_k(u^j; I)$, it is initialized to be 0 for the first SISO decoding stage. Afterwards, it assumes the scrambled $\lambda_k(u^j; O)$ from the previous SISO decoder.

In summary, Equations 3.136 to 3.146 are converted to the following equations in logarithmic domain. Recall that Equation 3.147, 3.148 and 3.149 are used to obtain $\lambda_k(c; I)$ and $\lambda_k(u; I)$.

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

$$\alpha_k(s) = \max_{e:s_k^E(e)=s}{}^{*} \left[\alpha_{k-1}(s_k^S(e)) + \lambda_k(u(e);I) + \lambda_k(c(e);I)\right], \quad k = 1,...,N-1. \tag{3.150}$$

$$\beta_k(s) = \max_{e:s_{k+1}^S(e)=s}{}^{*} \left[\beta_{k+1}(s_{k+1}^E(e)) + \lambda_{k+1}(u(e);I) + \lambda_{k+1}(c(e);I)\right], \quad k = N-1,...,1. \tag{3.151}$$

The $\alpha_k(s)$ computation is initialized by Equation 3.100, and the $\beta_k(s)$ computation is initialized by Equation 3.101 or 3.102 depending on the knowledge of the final state. As explained for conventional Log-MAP, Equation 3.98 and 3.99 can be used to normalize $\alpha_k(s)$ and $\beta_k(s)$ for computation stability.

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

$$\lambda_k^A(c;O) = \max_{e:c(e)=c}{}^{*} \left[\alpha_{k-1}(s_k^S(e)) + \lambda_k(u(e);I) + \lambda_k(c(e);I) + \beta_k(s_k^E(e))\right] - \max_{e:c(e)=\underline{0}}{}^{*} \left[\alpha_{k-1}(s_k^S(e)) + \lambda_k(u(e);I) + \lambda_k(c(e);I) + \beta_k(s_k^E(e))\right] \tag{3.152}$$

$$\lambda_k^A(u;O) = \max_{e:u(e)=u}{}^{*} \left[\alpha_{k-1}(s_k^S(e)) + \lambda_k(u(e);I) + \lambda_k(c(e);I) + \beta_k(s_k^E(e))\right] - \max_{e:u(e)=\underline{0}}{}^{*} \left[\alpha_{k-1}(s_k^S(e)) + \lambda_k(u(e);I) + \lambda_k(c(e);I) + \beta_k(s_k^E(e))\right] \tag{3.153}$$

3. The extrinsic information for the codeword symbol and that for the information symbol are:

$$\lambda_k(c;O) = \lambda_k^A(c;O) - \lambda_k(c;I) \tag{3.154}$$

$$= \max_{e:c(e)=c}{}^{*} \left[\alpha_{k-1}(s_k^S(e)) + \lambda_k(u(e);I) + \beta_k(s_k^E(e))\right] - \max_{e:c(e)=\underline{0}}{}^{*} \left[\alpha_{k-1}(s_k^S(e)) + \lambda_k(u(e);I) + \beta_k(s_k^E(e))\right] \tag{3.155}$$

$$\lambda_k(u;O) = \lambda_k^A(u;O) - \lambda_k(u;I) \tag{3.156}$$

$$= \max_{e:u(e)=u}{}^{*} \left[\alpha_{k-1}(s_k^S(e)) + \lambda_k(c(e);I) + \beta_k(s_k^E(e))\right] - \max_{e:u(e)=\underline{0}}{}^{*} \left[\alpha_{k-1}(s_k^S(e)) + \lambda_k(c(e);I) + \beta_k(s_k^E(e))\right] \tag{3.157}$$

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

$$\begin{aligned}\lambda_k^A(c^j;O) &= \max_{e:c^j(e)=1}^{*}\left[\alpha_{k-1}(s_k^S(e))+\lambda_k(u(e);I)+\lambda_k(c(e);I)+\beta_k(s_k^E(e))\right] \\ &\quad -\max_{e:c^j(e)=0}^{*}\left[\alpha_{k-1}(s_k^S(e))+\lambda_k(u(e);I)+\lambda_k(c(e);I)+\beta_k(s_k^E(e))\right] \\ &\qquad\qquad j=1,...,n_0. \qquad (3.158)\\ \lambda_k^A(u^j;O) &= \max_{e:u^j(e)=1}^{*}\left[\alpha_{k-1}(s_k^S(e))+\lambda_k(u(e);I)+\lambda_k(c(e);I)+\beta_k(s_k^E(e))\right] \\ &\quad -\max_{e:u^j(e)=0}^{*}\left[\alpha_{k-1}(s_k^S(e))+\lambda_k(u(e);I)+\lambda_k(c(e);I)+\beta_k(s_k^E(e))\right] \\ &\qquad\qquad j=1,...,k_0. \qquad (3.159)\end{aligned}$$

5. The extrinsic information for the codeword bits and that for the information bits are:

$$\lambda_k(c^j;O)=\lambda_k^A(c^j;O)-\lambda_k(c^j;I), \qquad j=1,...,n_0. \qquad (3.160)$$

$$\begin{aligned}&= \max_{e:c^j(e)=1}^{*}\left[\alpha_{k-1}(s_k^S(e))+\lambda_k(u(e);I)+\sum_{i=1;i\neq j}^{n_0}c^i(e)\lambda_k(c^i;I)+\beta_k(s_k^E(e))\right] \\ &\quad -\max_{e:c^j(e)=0}^{*}\left[\alpha_{k-1}(s_k^S(e))+\lambda_k(u(e);I)+\sum_{i=1;i\neq j}^{n_0}c^i(e)\lambda_k(c^i;I)+\beta_k(s_k^E(e))\right] \qquad (3.161)\end{aligned}$$

$$\lambda_k(u^j;O)=\lambda_k^A(u^j;O)-\lambda_k(u^j;I), \qquad j=1,...,k_0. \qquad (3.162)$$

$$\begin{aligned}&= \max_{e:u^j(e)=1}^{*}\left[\alpha_{k-1}(s_k^S(e))+\sum_{i=1;i\neq j}^{k_0}u^i\lambda_k(u^i;I)+\lambda_k(c(e);I)+\beta_k(s_k^E(e))\right] \\ &\quad -\max_{e:u^j(e)=0}^{*}\left[\alpha_{k-1}(s_k^S(e))+\sum_{i=1;i\neq j}^{k_0}u^i\lambda_k(u^i;I)+\lambda_k(c(e);I)+\beta_k(s_k^E(e))\right] \qquad (3.163)\end{aligned}$$

3.5 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 PCCC and SCCC. In the following sections, decoding procedures are presented for both structures where the information bits are transmitted in frames.

3.5.1 Decoding Procedure for PCCC

The procedure to decode a PCCC using SISO is described with an example of two RSC codes of rate 1/2. Thus, $k_0 = 1$, $n_0 = 2$. The soft input and output for the input symbol is

identical to those for the input bit since a input symbol contains only one bit. For PCCC comprising more constituent codes or codes of different rates, the same procedure can be readily extended. In the following, subscripts 1 and 2 indicate the first and the second constituent decoder, and τ_{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 3.149 to get $\lambda_{1k}(c^j; I)$ for DEC1 and $\lambda_{2k}(c^j; I)$ for DEC2.

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 3.150 and 3.151. Then use Equations 3.159 and 3.162 to calculate the extrinsic information $\lambda_{1k}(u; O)$ for the information bits.

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

5. DEC2. First use Equations 3.150, 3.151 to obtain path metrics $\alpha_{2k}(s)$ and $\beta_{2k}(s)$. Then,

   (a) if $\tau < \tau_{max}$, use Equation 3.159 in conjuction with 3.162 to compute the extrinsic information $\lambda_{2k}(u; O)$ 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 τ, go back to Step (3) and start next iteration.

   (b) if $\tau = \tau_{max}$, use Equation 3.159 to get the complete information $\lambda^A_{2k}(u; O)$ for the interleaved bits. Deinterleave and make decisions on the transmitted bits as follows:

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

   Output the hard decisions for the current frame, and go to Step (1) to decode next frame.

This procedure is illustrated in Figure 3.14. Note that the extrinsic informations in Steps (3) and (5a) can also be obtained using Equation 3.163 directly, instead of subtracting $\lambda_k(u; I)$ from $\lambda^A_k(u; I)$. On the other hand, the complete information at Step (5b) can be obtained using $\lambda^A_{2k}(u; O) = \lambda_{2k}(u; O) + \lambda_{2k}(u; I)$, 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.

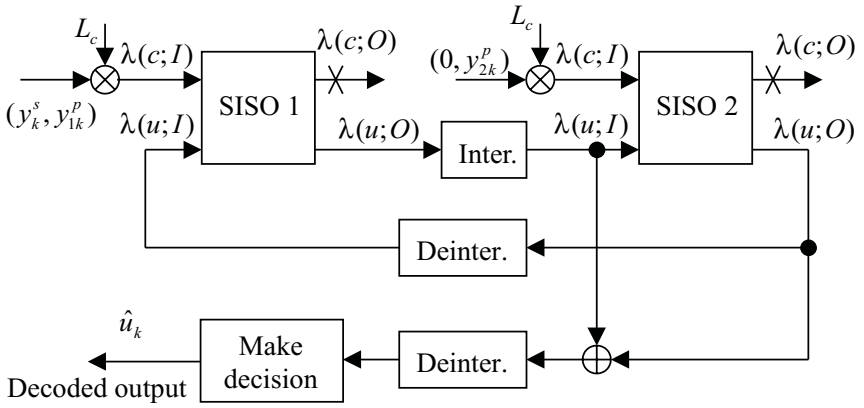


Figure 3.14: PCCC decoder with general SISO module.

3.5.2 Decoding Procedure for SCCC

To decode a SCCC with SISO, the following procedure will be employed. Here the subscripts i and o indicate the inner and outer code respectively.

1. Scale the received signal using Equation 3.149 to get the LLR input $\lambda_{ik}(c^j;I)$ for the codeword bits of the inner code.

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

3. Inner decoder. Calculate path metrics $\alpha_{ik}(s)$ and $\beta_{ik}(s)$ with Equations 3.150 and 3.151. Then use Equations 3.159 and 3.162 to compute the extrinsic information $\lambda_{ik}(u^j;O)$ for the information bits of the inner code.

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

5. Outer decoder. First calculate path metrics $\alpha_{ok}(s)$ and $\beta_{ok}(s)$ using Equations 3.150 and 3.151. Set $\lambda_{ok}(u;I) \equiv 0$, and it can be dropped from the computation. Then,

   (a) if $\tau < \tau_{max}$, use Equation 3.158 and 3.160 to calculate the extrinsic information $\lambda_{ok}(c^j;O)$ for the codeword bits of the outer code. Interleave the stream of $\lambda_{ok}(c^j;O)$ to be the LLR input $\lambda_{ik}(u^j;I)$ of the inner code. Increment τ, go back to Step (3) and start next iteration.

   (b) if $\tau = \tau_{max}$, use Equation 3.159 (with $\lambda_{ok}(u;I) = 0$) to obtain the complete information $\lambda_{ok}^A(u^j;O)$ of the outer code. Make decisions on the transmitted bits

as follows:

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

Output the hard decisions for the current frame, and go to Step (1) to decode next frame.

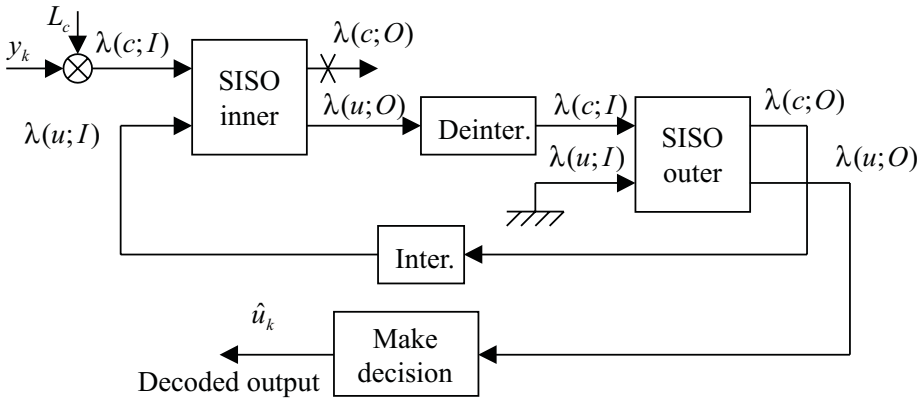


Figure 3.15: SCCC decoder with general SISO module.

This procedure is illustrated in Figure 3.15. The extrinsic information in Steps (3) and (5a) for the codeword bits and the information bits can be computed directly using Equation 3.161 and 3.163, respectively, instead of subtracting the *a priori* information from the complete information.

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

3.6 SW-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, brings about 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 can not be employed. Secondly, the delay can be intolerably large as the frame size increases. This is contradictory to the requirement that the frame size needs to be fairly large to get a decent interleaver gain.

To tackle these problems, a sliding window can be introduced into SISO [49] [53]. SW-SISO (sliding window soft-input soft-output) is the SISO algorithm with an embedded sliding window. SW-SISO segments the received sequence into small windows of size N_u. To produce the soft output for N_u trellis steps, a span of $N_w = N_u + N_t$ trellis steps will be processed together. Here N_u is the size of an update window, and N_t is the size of a training window extended beyond N_u steps. 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, 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 3.16.

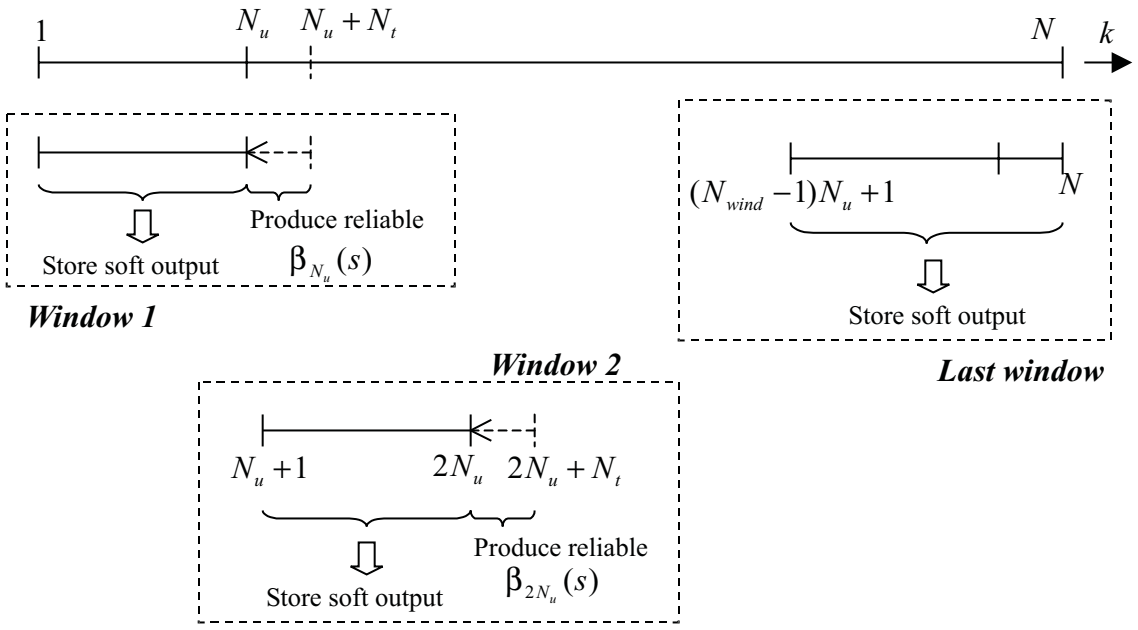


Figure 3.16: Sliding window SISO for a frame.

Here SW-SISO for a frame of size N is described. It is easy to be modified for continuous transmission. 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 will operate following these steps:

1. Window $i = 1$. Initialize $\alpha_0(s)$ as in Equation 3.100.

2. - If $i < N_{wind}$, the current window size is $N_u + N_t$. Initialize $\beta_{i(N_u+N_t)}(s) = 0, \forall s$. Go backward from $k = i(N_u+N_t)-1$ to $k = (i-1)(N_u+N_t)+1$, using Equation 3.151. 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 3.101 or 3.102 depending on the knowledge of the final state of the

trellis. Go backward from $k = N - 1$ to $k = (N_{wind} - 1)N_u + 1$, using Equation 3.151. Store $\beta_{N-1}(s)$ through $\beta_{(N_{wind}-1)N_u+1}(s)$.

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

   Compute the forward path metrics using Equation 3.150. At the same time, find the complete information or the extrinsic information, depending on the requirement. Increase i by 1: $i \Leftarrow i + 1$.

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 so fast 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 $N n_0$ 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 at the end of i-th update window. N_u should be chosen carefully to get the best compromise between complexity and latency.

3.7 Forward Computation of Backward Path Metrics for MAP Decoder

As shown in Section 3.2, 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 the soft output with increasing time index, the backward recursion is performed first to obtain all $B_k(s)$, where $k = 1, ..., N$ is the time index, $s = 0, ..., 2^m - 1$ is the state index, N is the frame size, 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 report, the backward-forward version is adopted for analysis. The extension of this procedure to the forward-backward version is obvious.

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. Only the most recent vector

$A_{k-1}(s)$ needs to be kept to obtain $A_k(s)$. However, all values of $B_k(s)$ need to be stored for the computation of 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. Sliding window method in Section 3.6 has been presented to save memory. However, it does not make use of the information in the whole frame. In this section, we develop a method to reduce storage requirements even when the entire frame is processed at a time [54]. It is proposed to compute $B_k(s)$ forwardly, and consequently the memory requirement for $B_k(s)$ can be relaxed. In addition, this method can be combined with the use of a sliding window to achieve even greater savings.

3.7.1 Existence of Butterfly Pairs

For a binary code alphabet, the branch transitions appear as butterfly pairs when the first and the last stages of the shift register are both connected in the feedback and feed-forward polynomials. This is shown below.

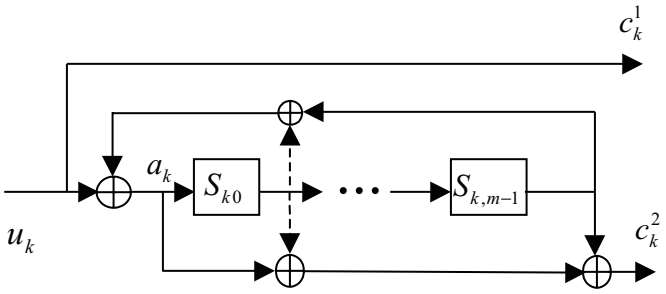


Figure 3.17: A general RSC encoder.

A RSC encoder model is shown in Figure 3.17. Denote the information bit to be encoded as u_k, the state of the encoder as $(S_{k0}, ..., S_{k,m-1})$, the substate as $(S_{k0}, ..., S_{k,m-2})$, 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. The following observations are made for a given substate $(S_{k0}, ..., S_{k,m-2})$.

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 result may be expressed as

$$\begin{cases} u_k \oplus S_{k,m-1} = 0 \\ u_k \oplus S_{k,m-1} = 1 \end{cases} \Longrightarrow \begin{cases} a_k = A \\ a_k = \bar{A} \end{cases} \quad A \in (0,1)$$

$$\Longrightarrow \text{next state at time } k+1 : \begin{cases} (A, S_{k,0}, ..., S_{k,m-2}) \\ (\bar{A}, S_{k,0}, ..., S_{k,m-2}) \end{cases}$$

If we define the index of the state as

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

and let

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

then the state pair $2M$ (when $S_{k,m-1} = 0$) and $(2M + 1)$ (when $S_{k,m-1} = 1$) always go to the next state pair M (when $a_k = 0$) and $(2^{m-1} + M)$ (when $a_k = 1$).

2. *The codewords in a butterfly pair.* The opposite value of $a_k \oplus S_{k,m-1}$ is related to the opposite value of c_k^2, or,

$$\begin{cases} a_k + S_{k,m-1} = 0 \\ a_k + S_{k,m-1} = 1 \end{cases} \Longrightarrow \begin{cases} c_k^2 = C \\ c_k^2 = \bar{C} \end{cases} \qquad C \in (0,1)$$

Since $S_{k,m-1}$ is connected to a_k by the backward polynomial and both of them 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 $(S_{k0}, ..., S_{k,m-2})$ 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 values of a codeword for a butterfly pair: (00),(11) for the (u_k, u_k) case, or (01),(10) for the $(u_k, \bar{u}_k)$ case.

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 3.18.

3.7.2 Forward Recursion of Backward Path Metrics $B_k(s)$

Using the butterfly structure shown above, the MAP algorithm can be split into a sequence of 2×2 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

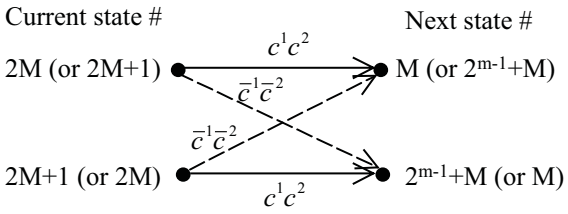


Figure 3.18: Butterfly pair model.

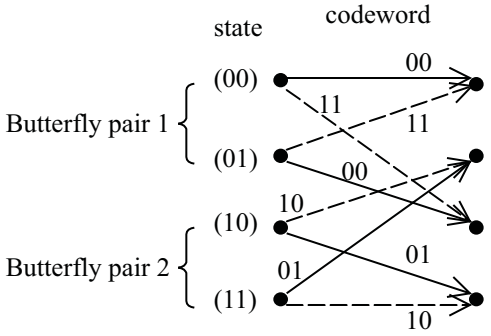


Figure 3.19: A trellis section with $g(D) = [1, 1 + D^2/1 + D + D^2]$.

this code generator at time k is shown in Figure 3.19. Clearly, there are two butterfly pairs having the form given in Figure 3.18.

There are four different branch metric values corresponding to four different codewords. Omit the constant h_M in Equation 3.117 and represent the trimmed branch metric at time k by $G_k(c(e))$, where $c(e)$ is the codeword on edge e. There are two branch metrics for butterfly pair 1:

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

There are also two branch metrics for butterfly pair 2:

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

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

- For butterfly pair 1:

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

These equations can be expressed in matrix form,

$$\begin{aligned}\begin{bmatrix} B_k(0) \\ B_k(1) \end{bmatrix} &= \begin{bmatrix} G_k(00) & G_k(11) \\ G_k(11) & G_k(00) \end{bmatrix} \begin{bmatrix} B_{k+1}(0) \\ B_{k+1}(2) \end{bmatrix} \\ &= G_{1k}^B \begin{bmatrix} B_{k+1}(0) \\ B_{k+1}(2) \end{bmatrix} \end{aligned} \tag{3.164}$$

- And for butterfly pair 2:

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

The equations above are represented in matrix form,

$$\begin{aligned}\begin{bmatrix} B_k(2) \\ B_k(3) \end{bmatrix} &= \begin{bmatrix} G_k(10) & G_k(01) \\ G_k(01) & G_k(10) \end{bmatrix} \begin{bmatrix} B_{k+1}(1) \\ B_{k+1}(3) \end{bmatrix} \\ &= G_{2k}^B \begin{bmatrix} B_{k+1}(1) \\ B_{k+1}(3) \end{bmatrix} \end{aligned} \tag{3.165}$$

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

- For butterfly pair 1:

$$\begin{aligned}\begin{bmatrix} B_{k+1}(0) \\ B_{k+1}(2) \end{bmatrix} &= (G_{1k}^B)^{-1} \begin{bmatrix} B_k(0) \\ B_k(1) \end{bmatrix} \\ &= \frac{1}{G_k^2(00) - G_k^2(11)} \begin{bmatrix} G_k(00) & -G_k(11) \\ -G_k(11) & G_k(00) \end{bmatrix} \begin{bmatrix} B_k(0) \\ B_k(1) \end{bmatrix} \\ &= h_G \times (G_k^2(10) - G_k^2(01)) \times \\ &\quad \begin{bmatrix} G_k(00) & -G_k(11) \\ -G_k(11) & G_k(00) \end{bmatrix} \begin{bmatrix} B_k(0) \\ B_k(1) \end{bmatrix} \end{aligned} \tag{3.166}$$

- For butterfly pair 2:

$$\begin{aligned}\begin{bmatrix} B_{k+1}(1) \\ B_{k+1}(3) \end{bmatrix} &= (G_{2k}^B)^{-1} \begin{bmatrix} B_k(2) \\ B_k(3) \end{bmatrix} \\ &= \frac{1}{G_k^2(10) - G_k^2(01)} \begin{bmatrix} G_k(10) & -G_k(01) \\ -G_k(01) & G_k(10) \end{bmatrix} \begin{bmatrix} B_k(2) \\ B_k(3) \end{bmatrix} \\ &= h_G \times (G_k^2(00) - G_k^2(11)) \times \\ &\quad \begin{bmatrix} G_k(10) & -G_k(01) \\ -G_k(01) & G_k(10) \end{bmatrix} \begin{bmatrix} B_k(2) \\ B_k(3) \end{bmatrix} \end{aligned} \tag{3.167}$$

In the above equations, $h_G = \frac{1}{(G_k^2(10)-G_k^2(01))(G_k^2(00)-G_k^2(11))}$. If the LLR is computed as the output, h_G will be canceled; thus, it can be omitted.

Equations 3.166 and 3.167 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 3.166 and 3.167 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 shows 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 3.166 and 3.167 to compute the $B_k(s)$ within each block. The choice of N_b will be discussed in Section 3.7.3. Finally, we have the following algorithm:

1. Perform the backward recursion using Equations 3.164 and 3.165, 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 $k = 1, ..., N-1$. Use the stored values of $B_k(s)$ for $k = 1$ and $iN_b, 0 < i < N_{blk}$. Otherwise, calculate $B_k(s)$ from the previous vector $B_{k-1}(s)$ using Equations 3.166 and 3.167. 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 metrics $A_k(s)$.

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

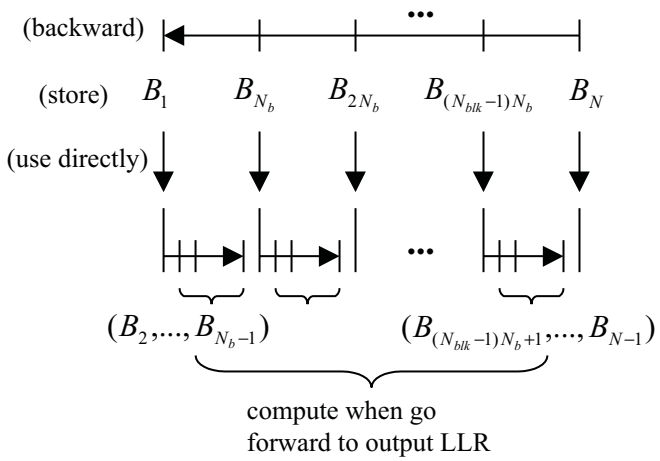


Figure 3.20: Procedure to compute $B_k(s)$.

The procedure to compute $B_k(s)$ is illustrated in Figure 3.20. Whenever a vector $B_k(s)$ is available, use it together with the forward path metrics $A_{k-1}(s)$ to produce the soft

output. Notice that when $N_b = 1$, this method degenerates to be the conventional MAP algorithm. A factor of $(N_b - 1)/N_b$ storage space for $B_k(s)$ will be saved with this method. Apparently, the computation complexity is compromised. The computation of $B_k(s)$ is increased to be $(1 + (N_b - 1)/N_b)$ times what it was in the conventional way.

3.7.3 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 of $g(D) = [1, 1 + D^2/1 + D + D^2]$, if the Equations 3.164 and 3.165 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:

$$\begin{bmatrix} B_i(0) \\ B_i(1) \\ B_i(2) \\ B_i(3) \end{bmatrix} = G^B_{i,i+l} \begin{bmatrix} B_{i+l}(0) \\ B_{i+l}(1) \\ B_{i+l}(2) \\ B_{i+l}(3) \end{bmatrix} \tag{3.168}$$

where $G^B_{i,i+l}$ is a 4×4 matrix. Therefore, to compute $B_{i+l}(s)$ from $B_i(s)$, we have the following:

$$\begin{bmatrix} B_{i+l}(0) \\ B_{i+l}(1) \\ B_{i+l}(2) \\ B_{i+l}(3) \end{bmatrix} = (G^B_{i,i+l})^{-1} \begin{bmatrix} B_i(0) \\ B_i(1) \\ B_i(2) \\ B_i(3) \end{bmatrix} \tag{3.169}$$

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

$$G^B_{i,i+l}(p, q) = P(s_{i+l} = q, Y_{i+1}^{i+l} | s_i = p).$$

Consequently, if l is larger than the time the trellis needs to converge, the matrix $G^B_{i,i+l}$ will have an entry much larger than the other entries, which corresponds to the most likely path in the trellis section between $k = i$ and $k = i + l$. Mathematically, $G^B_{i,i+l}$ becomes nearly singular. As a result, the inverse matrix $(G^B_{i,i+l})^{-1}$ 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.

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 $(G^B_{i,i+l})^{-1}$ is shown in Figure 3.21 for one MAP operation. The depth l ranges

from 1 to $7K$, where $K = m + 1 = 3$ is the constraint length. As seen in the plot, the condition number of the forward matrix $(G^B_{i,i+l})^{-1}$ increases exponentially with the distance l.

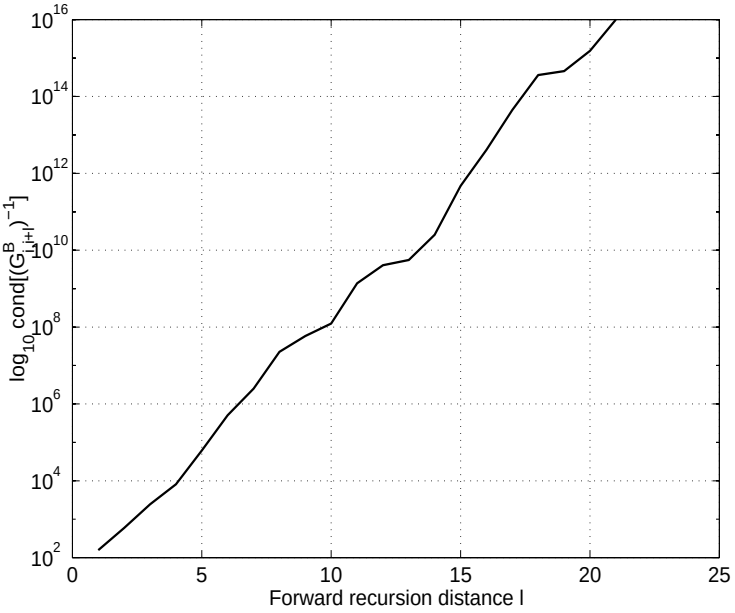


Figure 3.21: Logarithm of the condition number of $(G^B_{i,i+l})^{-1}$.

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 new method with block size N_b. The relative error is defined to be

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

where $E(x)$ is the mean value of x. In Figure 3.22, the maximum $\Delta^{N_b}_r$ for a frame of size 600 is shown with N_b ranging from $3K$ to $10K$. This graph shows that the relative error is negligible when $N_b \leq 5K$. When $N_b \geq 6K$, the computation error accumulates to an unacceptable value.

Curves are shown in Figure 3.23 for BER vs. iteration number. When $E_b/N_0 = 0$ dB, $N_b \in [1, 12]$ gives the same BER. When $E_b/N_0 = 1$ dB, 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 becomes more and more confident about the estimation, and the trellis converges with fewer stages. At medium to high E_b/N_0, N_b needs to be set smaller 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

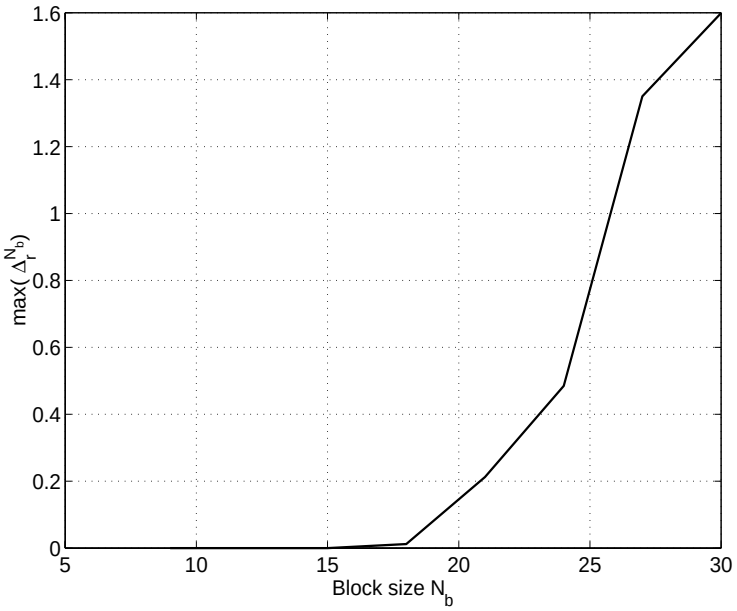


Figure 3.22: Maximum relative error of LLR output.

to set $N_b = 2$, which saves 50% of the storage for $B_k(s)$.

3.7.4 Conclusion

In this section, the existence of butterfly pairs for the RSC trellis is shown. Based on this structure, a new method is shown 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 MAP decoder for the turbo codes, especially when the frame size is large. This provides a 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 speed up the process by reducing the clock cycles required to fetch data. Since the computation load is increased, this method is useful when the computation speed is not a major concern, but the storage is.

3.8 Summary

In this chapter, the maximum *a posteriori* 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 [51] [49]. In comparison to the original algorithm, the branch metric computation was reduced by about 90%

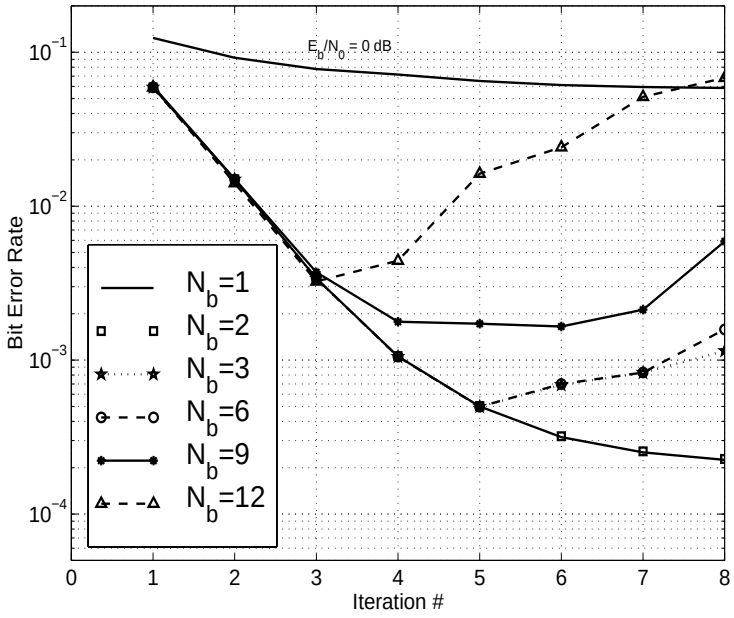


Figure 3.23: N_b choice for multiple iterations.

and the extrinsic information computation was reduced by 50%. Then a general soft-input soft-output decoding algorithm was presented for decoding both PCCC and SCCC. The decoding procedures with SISO module for both PCCC and SCCC were shown in detail. Afterwards, the sliding window SISO was introduced to reduce the decoding delay. Finally, a new method was proposed to compute the backward metrics of MAP algorithm in the forward direction so as to reduce storage requirement.

This chapter presents the procedures and notation associated with the iterative decoding of PCCC and SCCC. In subsequent chapters, we will adopt these procedures for practical implementation.

Chapter 4

SPW Design of PCCC and SCCC Systems

4.1 Introduction

The Signal Processing WorkSystem (SPW) is a powerful software package for developing, simulating, debugging, evaluating and implementing digital communication system designs. Using the SISO algorithm developed in Chapter 3, the encoding-decoding systems are built and validated in SPW for both PCCC and SCCC. In this chapter, the structure and parameters of both PCCC and SCCC systems are documented.

The systems are created in block diagram form using the Block Diagram Editor (BDE). The systems are hierarchical with multiple nested levels. The top-level block diagram representing a signal processing system is the "system" model. This model contains hierarchical block symbols, which represent lower-level block diagram models. The lower-level block diagrams may also contain hierarchical block symbols, which represent other block diagrams at still lower levels. The simulator "flattens" the whole design into a single-level design for simulation.

At each level, there may be some parameters, global or local, which the blocks refer to. Each hierarchical block symbol usually has one or more inputs and/or outputs, through which the signal flows. In Section 4.2, the top-level design of a rate 1/2 and a rate 1/3 PCCC systems are explained; in Section 4.3, the top-level design of a rate 1/4 SCCC system is explained; in Section 4.4, details of the hierarchical blocks of both PCCC and SCCC systems are documented. The parameters, input(s) and output(s) for each block diagram are explained.

4.2 Top-level Design of PCCC System

Both a rate 1/2 and a rate 1/3 PCCC system have been constructed in SPW. The rate 1/2 system is modified from the rate 1/3 system by alternatively puncturing the parity bits. Both systems have the same parameters which are listed in Table 4.1. To simulate either system, specify all the parameters except L_{total}, which will be automatically calculated as $L_{total} = L_{info} + m$.

4.2.1 Rate 1/2

In the rate 1/2 PCCC system, the parity bits of the RSC1 and RSC2 encoders are alternatively punctured. This system consists of four parts, which are shown in Figure 4.24, Figure 4.25, Figure 4.26 and Figure 4.27.

The structure in Figure 4.24 performs the encoding, antipodal modulation, addition of proper AWGN channel noise, and finally the generation of the received signal 'rec_ch'. Details of its blocks *p_enc7_5_p* and *EbN0_sigma* are shown in Figures 4.35 and 4.44, respectively. Details of blocks *rsc7_5* and *rsc7_5_init* in *p_enc7_5_p* are in turn shown in Figures 4.31 and 4.32, respectively. The *INTERLEAVE* block in *p_enc7_5_p* is a custom coded block used to interleave a data stream.

The structure in Figure 4.25 accepts the received signal and generates soft input for the decoder. It first scales 'rec_ch' by the channel reliability 'L_c'. Then it demultiplexes the product into two data streams. One is the soft input 'Lc_i1' for SISO1, and the other is 'Lc_i2' for SISO2. Block *EbN0_Lc* is the calculator of 'L_c', and is shown in detail in Figure 4.43.

Figure 4.26 shows the procedure of decoding and decision making. The soft input 'Lc_i1' and 'Lc_i2' are received from Figure 4.25. The SISO algorithm is employed to decode a frame. Hard decisions are made at the last iteration for the transmitted bits. Four iterations are shown in Figure 4.26. The other number of iterations can be achieved by linking the corresponding number of *pccc_dec_it1* blocks serially. Each *pccc_dec_it1* block performs one iteration, and its details are shown in Figure 4.38. Two of its components, *SISO* and *VECTOR INTERLEAVE*, are custom coded blocks.

Figure 4.27 shows the postprocessor, which has two inputs. 'bit_rx' is the estimation coming from the decoder in Figure 4.26, and 'bit_tx' is the actual transmitted bits coming from Figure 4.24. These signals are compared to each other to find out the bit errors and frame errors. A window will pop up when the simulation starts. The processing result

Table 4.1: Parameters for a PCCC system.

Parameter Name	**Type and Range**	**Description**
RSC encoder memory size	integer, > 0, usually 2, 3, or 4.	This is the number of shift registers in the constituent RSC encoder. It is denoted as m.
Code generator (octal) $\rightarrow$ feedback	string	This is the octal representation of the feedback polynomial g_b. For instance, if $g_b(D) = 1 + D + D^3$, then $g_b = (1101)$ in binary form, $g_b = 15$ in octal form.
Code generator (octal) $\rightarrow$ feed forward	string	It is the octal representation of the feed forward polynomial g_f. For instance, if $g_f(D) = 1 + D^2 + D^3$, then $g_b = (1011)$ in binary form, $g_b = 13$ in octal form.
Frame size not including tail bits	integer, > 0.	It specifies L_{info}, the size of an information frame.
Frame size including tail bits	integer, > 0.	This variable is set to be $L_{total} = L_{info} + m$. It will be calculated once L_{info} and m are given.
Signal to noise ratio E_b/N_0 (dB)	double, usually 0.0 (dB) $< E_b/N_0 <$ 5.0 (dB).	This gives the SNR condition that the simulation is to be run under.
1/rate	double, > 0.	This is the inverse of the code rate.
Interleaver mapping data file	string	This specifies the name of the interleaver mapping file. The file should contain scrambled integers ranging from 0 to $(L_{total} - 1)$. For example, if the information frame size is 126 and $m = 2$, then the extended frame size is 128 = 126+2. Consequently the interleaver mapping file should contain 128 integers in ASCII form ranging from 0 to 127. Let the i-th integer in the interleaver file to be $\alpha(i)$, the uninterleaved data be $x(0), ..., x(127)$, the interleaved data be $y(0), ..., y(127)$. The relationship between them is: $y(i) = x(\alpha(i)), i = 0, ..., 127$.
Number of frame errors to count	integer, > 0, usually less than 50.	This parameter specifies the number of frame errors to count before the simulation stops.

is shown dynamically inside the window while the simulation is running. The displayed information includes: the number of transmitted bits, the number of bit errors, the bit error rate, the number of transmitted frames, the number of frame errors, and the frame error rate. The details of the key block *display_bfer* are shown in three parts in Figures 4.40, 4.41 and 4.42.

4.2.2 Rate 1/3

For the rate 1/3 PCCC system, both parity bits of RSC1 and RSC2 are transmitted. Similar to the rate 1/2 system, the rate 1/3 system is the combination of four parts as described below.

As in the rate 1/2 system, the structure in Figure 4.24 is used to generate the received signal from the information bits. The only change necessary is that the encoder should be a rate 1/3 block. Figure 4.36 shows an example of rate 1/3 PCCC encoder with code generator $g(D) = [1, 1 + D^2/1 + D + D^2]$. With the replacement of other constituent RSC encoders, rate 1/3 encoders with different constituent codes can be built readily. For example, when the encoders in Figure 4.33 and 4.34 are used in Figure 4.36, it becomes a rate 1/3 PCCC encoder with code generator $g(D) = [1, 1 + D + D^2 + D^3/1 + D + D^3]$.

Instead of Figure 4.25, the structure in Figure 4.28 accepts the received signal and generates the soft input 'Lc_i1' and 'Lc_i2' for the decoder in the rate 1/3 system. The block *EbN0_Lc* calculates 'L_c' to scale the received bits 'rec_ch' as in the rate 1/2 system.

The decoding and decision making unit and the postprocessor are the same as that for the rate 1/2 system, which are shown in Figure 4.26 and Figure 4.27.

4.3 Top-level Design of SCCC System

A rate 1/4 SCCC system is built with SPW. The SCCC system uses the same top-level parameters listed in Table 4.1 as the PCCC system. Both the outer and inner codes are rate 1/2 RSC codes with a generator $g(D) = [1, 1 + D^2/1 + D + D^2]$, and both codes are terminated with two tail bits. By puncturing and depuncturing, SCCC systems of different rates can be built with the same constituent encoders and decoders. Similar to the PCCC systems, the SCCC system also consists of four parts as described below.

The part that generates the received signal ('rec_ch') from the information bit ('bit_tx') is similar to the structure in Figure 4.24 for PCCC. Only the encoder needs to be replaced. The rate 1/4 SCCC encoder, with block symbol *s_enc7_5_4*, is expanded in Figure 4.37.

The processing unit between the channel and the decoder is shown in Figure 4.29. Since

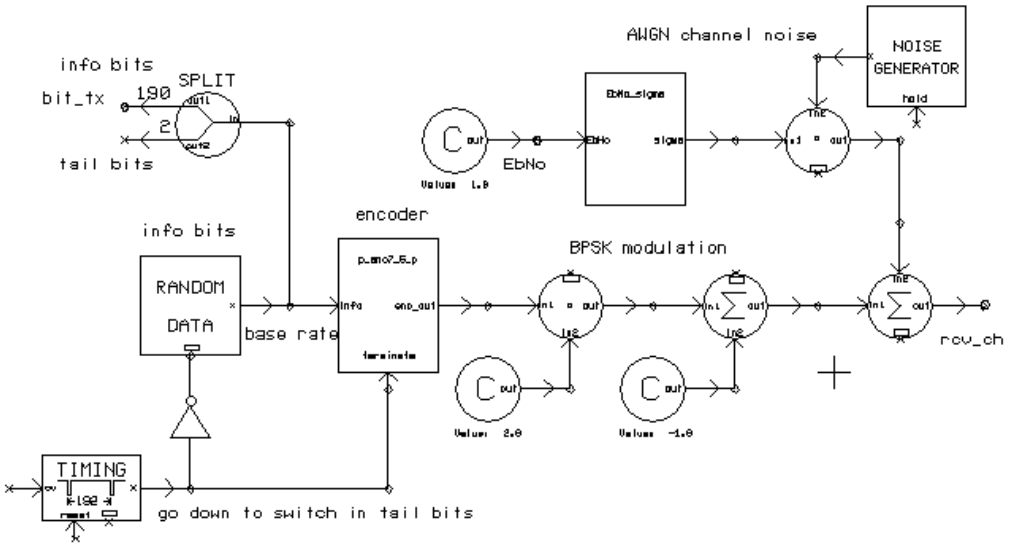


Figure 4.24: SPW model: transmitter of a rate 1/2 PCCC system.

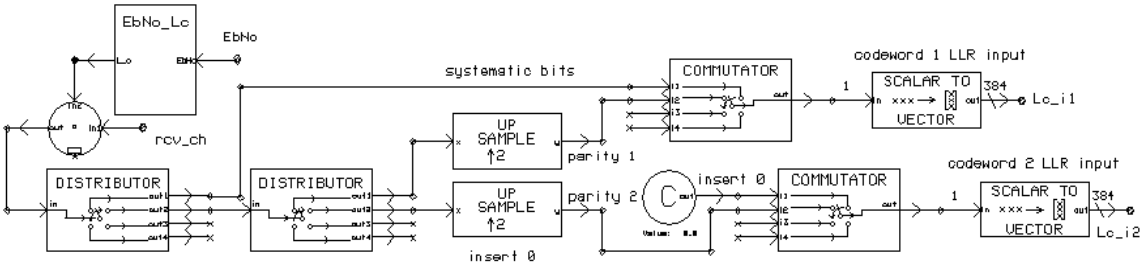


Figure 4.25: SPW model: demultiplexer of a rate 1/2 PCCC system.

no puncturing or depuncturing is used in the rate 1/4 system, the receiver only scales the received signal 'rec_ch' to be the LLR input 'Lc_i' to the decoder.

Figure 4.30 illustrates the decoding and decision making procedure for a SCCC system. Four iterations are shown. Other number of iterations can be realized by linking the corresponding number of *sccc_dec_it1* blocks in series. Each *sccc_dec_it1* block performs one iteration, and its lower-level structure is shown in Figure 4.39. The soft input 'Lc_i' of the decoder comes from the processing unit in Figure 4.29. The sign of the soft output 'Lu_o_o' is used to make the estimation ('bit_rx') about the information bits ('bit_tx').

Finally, the SCCC system shares the same postprocessor as PCCC, which computes and displays BER and FER among other results. The postprocessor is shown in Figure 4.27.

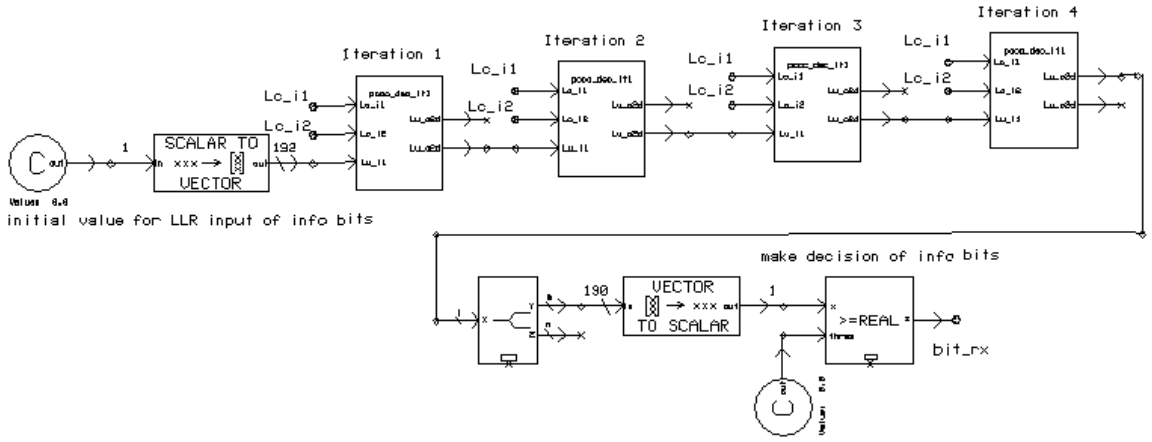


Figure 4.26: SPW model: decoder of a PCCC system.

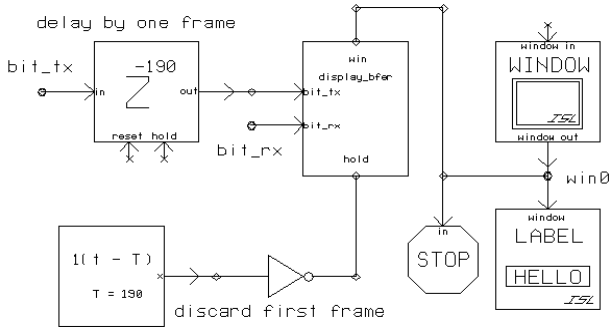


Figure 4.27: SPW model: postprocessor of a PCCC system.

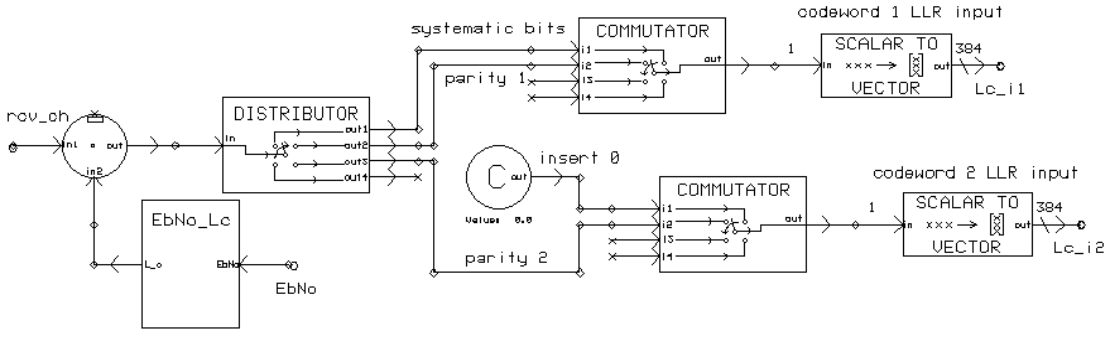


Figure 4.28: SPW model: demultiplexer of a rate 1/3 PCCC system.

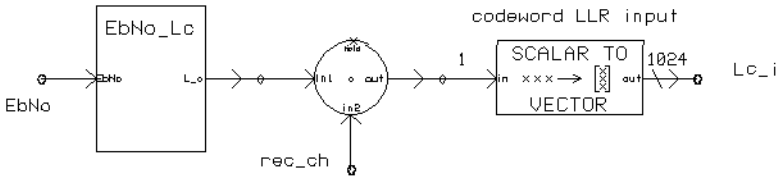


Figure 4.29: SPW model: demultiplexer of rate 1/4 SCCC system.

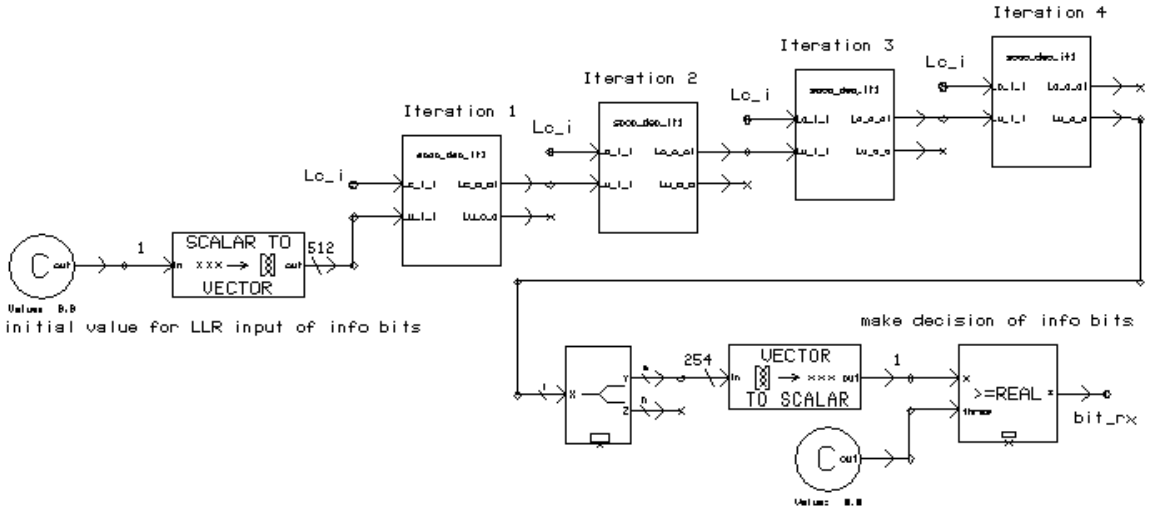


Figure 4.30: SPW model: decoder of SCCC system.

Table 4.2: Input and output of block *rsc7_5*.

Input Name	**Description**
info	The binary (0/1) information bits to be encoded. It comes as a data stream. It is to be encoded blockwise. There are $m = 2$ cycles between frames during which the information source should be held and the tail bits are added to terminate the encoder.
terminate	When information bits are under transmission, 'terminate' should be kept high (=1), and the input of the first shift register is the information bit. At the end of each frame, 'terminate' should go low (=0) for $m = 2$ cycles to terminate the encoder, and the first shift register accepts the tail bits that come from the feedback line. In the mean time, the information source should be held for $m = 2$ cycles to avoid loss of data.
Output Name	**Description**
c_s	The systematic bit. It is equal to 'info' for most of the frame, and is equal to the tail bits for the last 2 cycles in a frame.
c_p	The parity bit.

4.4 Detailed SPW Blocks for Concatenated Systems

The following sections describe the details of the components that make up an integrated PCCC or SCCC system.

4.4.1 RSC Encoders

rsc7_5: **RSC** $g = (7,5)_{octal}$, **Tail Termination**

The structure of the *rsc7_5* block is shown in Figure 4.31, and it has no parameters. The input and output are listed in Table 4.2. The *rsc7_5* block implements the rate 1/2 RSC encoder with the best $m = 2$ code generator [55]

$$g(D) = \left[1, \frac{1 + D^2}{1 + D + D^2}\right]. \tag{4.170}$$

When the information bits are transmitted by frames, this encoder can generate tail bits at the end of each frame. The tail bits will return the shift registers to the all-zero state.

Table 4.3: Input and output of block *rsc7_5_init*.

Input Name	**Description**
info	The binary (0/1) information bits to be encoded. It comes in as a data stream, and is encoded blockwise.
reset	When 'reset' is high (=1), the shift registers accepts the input and encodes it. When 'reset' is low (=0), the shift registers will be returned to all-zero immediately. At the beginning of each frame, 'reset' should go low for one cycles to reset the encoder, and then remain high until the beginning of next frame.
Output Name	**Description**
c_s	The systematic bit. It equals to 'info' for the entire frame.
c_p	The parity bit.

rsc7_5_init: RSC $g = (7,5)_{octal}$, No Tail Termination

The structure of the *rsc7_5_init* block is shown in Figure 4.32, and it has no parameters. The input and output are listed in Table 4.3. The *rsc7_5_init* block implements the rate 1/2 RSC encoder with the best $m = 2$ code generator [55]

$$g(D) = \left[1, \frac{1+D^2}{1+D+D^2}\right]. \tag{4.171}$$

When the information bits are transmitted by frames, this encoder can directly reset the encoder to the all-zero state at the beginning of each frame, and it encodes every bit inside a frame without adding any tail bits.

rsc15_17: RSC $g = (15,17)_{octal}$, Tail Termination

The structure of the *rsc15_17* block is shown in Figure 4.33, and it has no parameters. Its input and output are the same as those listed in Table 4.2, with the substitution of $m = 3$. This block implements the rate 1/2 RSC encoder with the best memory $m = 3$ code generator [55]

$$g(D) = \left[1, \frac{1+D+D^2+D^3}{1+D+D^3}\right]. \tag{4.172}$$

It resets the shift registers to the all-zero state by padding 3 tail bits at the end of each frame.

rsc15_17_init: **RSC** $g = (15, 17)_{octal}$, **No Tail Termination**

The structure of the *rsc15_17_init* block is shown in Figure 4.34, and it has no parameters. Its input and output are the same as those listed in Table 4.3. This block implements the rate 1/2 RSC encoder with the best memory $m = 3$ code generator [55]

$$g(D) = \left[1, \frac{1 + D + D^2 + D^3}{1 + D + D^3}\right]. \tag{4.173}$$

It resets the shift registers to the all-zero state instantly at the beginning of each frame.

rsc31_37: **RSC** $g = (31, 37)_{octal}$, **Tail Termination**

The structure of *rsc31_37* block is built in the same way as those in Figures 4.31 and 4.33, and it has no parameters. The input and output are the same as those in Table 4.2, with the substitution of $m = 4$. This block implements the rate 1/2 RSC encoder with the best memory 4 code generator [55]

$$g(D) = \left[1, \frac{1 + D + D^2 + D^3 + D^4}{1 + D + D^4}\right]. \tag{4.174}$$

It resets the shift registers to the all-zero state by padding 4 tail bits at the end of each frame.

rsc31_37_init: **RSC** $g = (31, 37)_{octal}$, **No Tail Termination**

The structure of *rsc31_37_init* block is built in the same way as those in Figures 4.32 and 4.34, and it has no parameters. The input and output are the same as those in Table 4.3. This block implements the rate 1/2 RSC encoder with the best memory 4 code generator [55]

$$g(D) = \left[1, \frac{1 + D + D^2 + D^3 + D^4}{1 + D + D^4}\right]. \tag{4.175}$$

It resets the shift registers to the all-zero state directly.

4.4.2 PCCC Encoders

p_enc7_5_p: **Rate 1/2,** $g = (7, 5)_{octal}$

The block symbol *p_enc7_5_p* is expanded in Figure 4.35. The parameters are listed in Table 4.4. The input and output are tabulated in Table 4.5. This block implements the rate 1/2 PCCC encoder. It contains two constituent RSC encoders with code generator $g(D) = [1, 1 + D^2/1 + D + D^2]$. RSC1 is terminated at the end of each frame with two

Table 4.4: Parameters for rate 1/2 PCCC encoder *p_enc7_5_p*.

Parameter Name	**Type and Range**	**Description**
RSC encoder memory size (fixed)	integer, equal to 2 (uneditable)	This indicates that the constituent RSC encoders have memory size $m = 2$.
Frame size not including tail bits	integer, ≥ 1.	This is the information frame size.
Interleaver mapping data file	string	This specifies the path and name of the interleaver mapping file. The file should contain scrambled integers ranging from 0 to $(L_{total} - 1)$. For example, if the information frame size is 126, then the expanded frame size is 128 = 126+2. Consequently the interleaver mapping file should contain 128 integers ranging from 0 to 127 in a pseudo-random order. Let the i-th integer in the interleaver file to be $\alpha(i)$, the uninterleaved data be $x(0), ..., x(127)$, the interleaved data be $y(0), ..., y(127)$. Then the relationship between them is: $y(i) = x(\alpha(i)), i = 0, ..., 127$.

tail bits (Figure 4.31), while RSC2 is left open (Figure 4.32). The interleaver scrambles the expanded frame, (info bits + tail bits for RSC1), and passes it to the input of RSC2. Alternative puncturing of the parity bits is used to increase the code rate to 1/2. Denote the k-th systematic and parity bits out of RSC1 as (c_{s1k}, c_{p1k}), and denote those out of RSC2 as (c_{s2k}, c_{p2k}). Then, the overall encoder output is

$$..., \underbrace{c_{s1,2k}, c_{p1,2k}}_{\text{for 'info' bit } 2k}, \underbrace{c_{s1,2k+1}, c_{p2,2k+1}}_{\text{for 'info' bit } 2k+1}, \underbrace{c_{s1,2k+2}, c_{p1,2k+2}}_{\text{for 'info' bit } 2k+2}, ...$$

There are two encoder output bits for each 'info' bit. One is the systematic bit from RSC1, and the other is selected alternatively from the parity bit of either RSC1 or RSC2.

p_enc7_5: Rate 1/3, $g = (7, 5)_{octal}$

The lower-level structure of *p_enc7_5* block is shown in Figure 4.36. The parameters are the same as those for the rate 1/2 PCCC encoder, which are listed in Table 4.4. The input and output are similar to those in Table 4.5, except that the output 'enc_out' runs three times as fast as 'info' instead of twice as fast.

Table 4.5: Input and output for rate 1/2 PCCC encoder *p_enc7_5_p*.

Input Name	**Description**
info	This is the binary (0/1) information bit stream to be encoded block-wise by the PCCC encoder. It comes as a data stream. There are $m = 2$ cycles between frames when the information source should be held, during which the tail bits are added to terminate RSC1.
terminate	This is a timing signal. It remains high when the information frame is under transmission, and goes low for $m = 2$ cycles to terminate RSC1 at the end of a frame.
Output Name	**Description**
enc_out	This is the binary (0/1) encoder output stream. It runs two times as faster as the 'info' stream. For each information bit and each tail bit, there are two output bits.

RSC1 is terminated at the end of each frame with two tail bits (Figure 4.31), while RSC2 is left open (Figure 4.32). The interleaver scrambles the expanded frame, (info bits + tail bits for RSC1), and passes it to the input of RSC2. No puncturing is used. Let the k-th systematic and parity bits out of RSC1 be (c_{s1k}, c_{p1k}), and those out of RSC2 be (c_{s2k}, c_{p2k}). Then the encoder output is a stream of

$$\cdots, \underbrace{c_{s1k}, c_{p1k}, c_{p2k}}_{\text{for 'info' bit } k}, \quad \underbrace{c_{s1,k+1}, c_{p1,k+1}, c_{p2,k+1}}_{\text{for 'info' bit } k+1}, \cdots$$

There are three encoder output bits for each 'info' bit. The first is the systematic bit from RSC1, the second is the parity bit from RSC1, and the third is the parity bit from RSC2.

p_enc15_17_p: **Rate 1/2,** $g = (15, 17)_{octal}$

The *p_enc15_17_p* block implements the rate 1/2 PCCC encoder in the same way as the *p_enc7_5_p* block expanded in Figure 4.35. Instead of $m = 2$ RSC encoders, it contains two $m = 3$ RSC encoders with code generator $g(D) = [1, 1 + D + D^2 + D^3/1 + D + D^3]$. They are the *rsc15_17* and *rsc15_17_init* blocks shown in Figures 4.33 and 4.34, respectively.

p_enc15_17: **Rate 1/3,** $g = (15, 17)_{octal}$

The *p_enc15_17* block implements the rate 1/3 PCCC encoder in the same way as the *p_enc7_5* block expanded in Figure 4.36. Instead of $m = 2$ RSC encoders, it contains two $m = 3$ RSC encoders with code generator $g(D) = [1, 1 + D + D^2 + D^3/1 + D + D^3]$. They are the *rsc15_17* and *rsc15_17_init* blocks shown in Figures 4.33 and 4.34, respectively.

p_enc31_37_p: **Rate 1/2,** $g = (31, 37)_{octal}$

The *p_enc31_37_p* block implements the rate 1/2 PCCC encoder in the same way as the *p_enc7_5_p* block expanded in Figure 4.35. Instead of $m = 2$ RSC encoders, it contains two $m = 4$ RSC encoders with code generator $g(D) = [1, 1 + D + D^2 + D^3 + D^4/1 + D + D^4]$. They are the *rsc31_37* and *rsc31_37_init* blocks explained in Section 4.4.1.

p_enc31_37: **Rate 1/3,** $g = (31, 37)_{octal}$

The *p_enc31_37* block implements the rate 1/3 PCCC encoder in the same way as the *p_enc7_5* block expanded in Figure 4.36. Instead of $m = 2$ RSC encoders, it contains two $m = 4$ RSC encoders with code generator $g(D) = [1, 1 + D + D^2 + D^3 + D^4/1 + D + D^4]$. They are the *rsc31_37* and *rsc31_37_init* blocks explained in Section 4.4.1.

4.4.3 SCCC Encoder

The structure of a rate 1/4 SCCC encoder, block *s_enc7_5_4*, is shown in Figure 4.37. It contains two constituent RSC encoders with code generator $g(D) = [1, 1 + D^2/1 + D + D^2]$. Both RSCs are terminated by 2 tail bits. A bit interleaver is used between the outer and inner code. The parameters for this block are similar to those in Table 4.4, except that the interleaver's range should be 0 to $2L_{info} + 3$. The input and output parameters are similar to those in Table 4.5, except that 'enc_out' runs four times as fast as 'info'.

4.4.4 Decoder Components

SISO: **General SISO Module**

The *SISO* module is a custom coded block built with C code. It accepts LLR input of the codeword bits ('Lc_i') and that of information bits ('Lu_i'), processes them with additive SISO algorithm, and produces the extrinsic information of the codeword bits ('Lc_o') and that of the information bits ('Lu_o'). It is the fundamental decoding unit for both PCCC and SCCC systems. Its parameters are listed in Table 4.6, while its input and output are explained in Table 4.7.

pccc_dec_it1: **One Decoding Iteration for PCCC**

The hierarchical block symbol *pccc_dec_it1* performs one decoding iteration for a PCCC system, and its expanded structure is shown in Figure 4.38. It contains two SISO modules, an

Table 4.6: Parameters for SISO module *SISO*.

Parameter Name	**Type and Range**	**Description**
Size of the frame including tail bits	integer, $\geq m+1$.	This is the size of the expanded frame. For instance, if the information is transmitted in frames of 126 bits, and the encoder contains RSC of memory 2, then the whole frame size is $126+2=128$.
Code generator (in octal form) $\rightarrow$ feedback	string	It is the octal representation of the feedback polynomial g_b. For instance, if $g_b(D)=1+D+D^3$, then $g_b=(1101)$ in binary form, $g_b=15$ in octal form.
Code generator (in octal form) $\rightarrow$ feed forward	string	It is the octal representation of the feed forward polynomial g_f. For instance, if $g_f(D)=1+D^2+D^3$, then $g_f=(1011)$ in binary form, $g_f=13$ in octal form.
Memory size of the encoder	integer	It is equal to the number of shift registers in the constituent RSC encoder.
Termination of trellis ('yes' or 'no')	string	'yes' indicates that the corresponding RSC encoder is terminated to the all-zero state at the end of a frame, e.g., RSC1 of PCCC; 'no' indicates that the corresponding RSC encoder is left open at the end of a frame, e.g., RSC2 of PCCC.

Table 4.7: Input and output for SISO module *SISO*.

Input Name	**Description**
Lc_i	This is the LLR input for the codeword bits. For both SISOs of a PCCC decoder and the inner SISO of a SCCC decoder, when the signal is contaminated by AWGN noise, 'Lc_i'$(k) = L_c y_k = (4rE_b/N_0)y_k$. Here E_b/N_0 is the signal to noise ratio, r is the code rate, and y_k is the k-th received signal. For the outer SISO of a SCCC decoder, it is the deinterleaved version of 'Lu_o' from the inner SISO.
Lu_i	This is the LLR input for the information bit. It stands for the *a priori* knowledge. For both SISOs for a PCCC decoder, it is the (de)interleaved version of 'Lu_o' from the other SISO. For inner SISO of a SCCC decoder, it is the interleaved version of 'Lc_o' from the outer SISO. For outer SISO of a SISO decoder, it is set to be zero for all iterations.
Output Name	**Description**
Lc_o	This is the LLR output for the codeword bits. It is not used in PCCC. In SCCC, 'Lc_o' of the outer SISO will be interleaved and used as 'Lu_i' to the inner SISO, while 'Lc_o' of the inner SISO is abandoned.
Lu_o	This is the LLR output for the information bits. For both SISOs of a PCCC decoder, it will be (de)interleaved and used as 'Lu_i' to the other SISO. For inner SISO of a SCCC decoder, it will be deinterleaved and used as 'Lc_i' to the outer SISO. For outer SISO of a SCCC decoder, it will be used to make hard decisions at the last iteration.

Table 4.8: Parameters for the block performing one iteration of PCCC, *pccc_dec_it1*.

Parameter Name	**Type and Range**	**description**
Size of the frame including tail bits	integer, $\geq m+1$.	This is the size of the expanded frame. For instance, the information is transmitted in frames of 126 bits, and the encoder contains RSC of $m=2$. Then the expanded frame size is $L_{total} = L_{info} + m = 126 + 2 = 128$.
Code generator (in octal form) → feedback)	string	It is the octal representation of the feedback polynomial g_b. For instance, if $g_b(D) = 1 + D + D^3$, then $g_b = (1101)$ in binary form, $g_b = 15$ in octal form.
Code generator (in octal form) → feed forward)	string	It is the octal representation of the feed forward polynomial g_f. For instance, if $g_f(D) = 1 + D^2 + D^3$, then $g_f = (1011)$ in binary form, $g_f = 13$ in octal form.
Memory size of the encoder	integer	It is equal to the number of shift registers in the constituent RSC encoder.
interleaver mapping data file	string	This specifies the path and name of the interleaver file. The interleaver file should contain integers in ASCII form ranging from 0 to $(L_{total} - 1)$ in a pseudorandom order.

interleaver (the first *VECTOR INTERLEAVE*), and a deinterleaver (the second *VECTOR INTERLEAVE*). The output of the deinterleaver, 'Lu_o2d', can be used as 'Lu_i1' for next iteration. There is an extra adder and deinterleaver (the *VECTOR INTERLEAVE* after the adder), which are used to produce the deinterleaved complete information, 'Lu_c2d', for the last iteration. Hard decision will be made according to the sign of 'Lu_c2d'.

The parameters are explained in Table 4.8, while the input and output are listed in Table 4.9.

sccc_dec_it1: One Decoding Iteration for SCCC

The lower-level structure of the *sccc_dec_it1* block, which is shown in Figure 4.39, performs one decoding iteration for a SCCC system. There are two SISOs. The first one, the inner SISO, is for inner code; the second one, the outer SISO, is for outer code. A deinterleaver, the first *VECTOR INTERLEAVE* block, is used between them to pass the soft information.

Table 4.9: Input and output for the block performing one iteration of PCCC, *pccc_dec_it1*.

Input Name	**Description**
Lc_i1	This is the LLR input for the codeword bits of RSC1. Its length is $2L_{total}$. When the signal is contaminated by AWGN noise, 'Lc_i1'(k)= $L_c y_{1k} = 2rE_b/N_0 y_{1k}$. E_b/N_0 gives SNR, r is the code rate, and y_{1k} is the k-th received signal of code one.
Lc_i2	Similar to 'Lc_i1', this is the LLR input for the codeword bits of RSC2. Its length is $2L_{total}$. However, the systematic bits of RSC2 are not transmitted. Thus 0 is used in the position of the systematic bits. The 'Lc_i2' stream appears as: 0, 'Lc_i2'(1), 0, 'Lc_i2'(3), ...
Lu_i1	This is the LLR input for the information bits of RSC1. Its length is L_{total}. It is also called the *a priori* information.
Output Name	**Description**
Lu_c2d	This is the complete information out of SISO2. Its length is $2L_{total}$. It is in the deinterleaved order. The estimation of the information bits k can be made based on the sign of 'Lu_c2d'(k). When 'Lu_c2d'(k) $> 0, \hat{u}_k = 1$; when 'Lu_c2d'(k) $< 0, \hat{u}_k = 0$.
Lu_o2d	This is the extrinsic information for the information bits out of SISO2. It is a vector of length L_{total} in deinterleaved order. It can be used as 'Lu_i1' for the next iteration.

Table 4.10: Input and output for the block performing one iteration of SCCC, *sccc_dec_it1*.

Input Name	**Description**
Lc_i_i	This is the LLR input for the codeword bits of inner SISO. Its length is $4L_{info}+12$. When the signal is contaminated by AWGN noise, 'Lc_i_i'(k)= $L_c y_k = (0.5E_b/N_0)y_{1k}$, since code rate is 1/4. E_b/N_0 gives SNR, and y_k is the k-th received signal.
Lu_i_i	This is the LLR input for the information bits of inner SISO. Its length is $2L_{info}+6$. It is also called the *a priori* information.
Output Name	**Description**
Lc_o_oi	This is the extrinsic information for the codeword bits of outer SISO. It is in the interleaved order. It can be used as the *a priori* information 'Lu_i_i' for the next iteration.
Lu_o_o	This is the complete information for the information bits. Its length is $2L_{info}+4$. It is in the deinterleaved order. The estimation of the information bits can be made based on the sigh of 'Lu_o_o'(k). When 'Lu_o_o'(k) $> 0, \hat{u}_k = 1$; when 'Lu_o_o'(k) $< 0, \hat{u}_k = 0$.

An interleaver, the second *VECTOR INTERLEAVE* block, is used to scramble 'Lc_o' of outer SISO to be the output 'Lc_o_oi', which can be the input 'Lu_i_i' for the next *sccc_dec_it1* block. The soft output 'Lu_o_o' can be used to make the hard decision at last iteration.

The parameters for this block are the same as those in Table 4.8. The input and output are explained in Table 4.10.

4.4.5 Miscellaneous Components

INTERLEAVE: **Interleaver/Deinterleaver for a Data Stream**

The *INTERLEAVE* block has two modes, one for interleaving, the other for deinterleaving. The mode is set in the parameter window. When it is set as an interleaver, $y(i) = x(\alpha(i))$; when it is set as a deinterleaver, $y(\alpha(i)) = x(i), i = 0, ..., N-1$. Here x is the input, α is the interleaver mapping, y is the output, and N is the (de)interleaver size.

This block will wait until it has collected N input data before it sends out the scrambled data using the specified mapping. Two registers of size N are used to store data. Register one is dedicated to store the input data stream. When it is full, the vector of size N will be passed to register two, which will hold the vector until it is (de)interleaved and sent out in a stream. In this way, when a vector is under transmission, register one is able to continue

Table 4.11: Parameters for the stream (de)interleaver *INTERLEAVE*.

Parameter Name	**Type and Range**	**description**
Mode (interleaver or deinterleaver)	string, 'interleave' or 'deinterleave'.	This specifies the function of the block. Let the interleaver mapping be $\alpha(i), 0 \leq i \leq N-1$. If 'interleave' is specified, $y(i) = x(\alpha(i))$. If 'deinterleave' is specified, $y(\alpha(i)) = x(i)$.
Size of the frame to be (de)interleaved	integer, > 1.	It is equal to the frame size.
Interleaver mapping data file	string	This specifies the path and name of the interleaver file. The interleaver file should contain integers in ASCII form ranging from 0 to $N-1$ in a pseudorandom order.

Table 4.12: Input and output for the stream (de)interleaver *INTERLEAVE*.

Input Name	**Description**
x	The input data stream.
Output Name	**Description**
y	The output data stream.

reading in data of next vector without any pause. Apparently the output data stream has a delay of N with respect to the input data stream.

This block is a custom coded block built with C code. The parameters are explained in Table 4.11. The input and output are tabulated in Table 4.12.

VECTOR INTERLEAVE: Interleaver/Deinterleaver for a Vector

The *VECTOR INTERLEAVE* block accepts a vector of size N, then (de)interleaves it. Its output is a vector of the scrambled data produced at the same clock cycle. Since in SPW the operation is done instantaneously, no register is necessary. Just as for the *INTERLEAVE* block, it has two modes so that it can act as an interleaver or a deinterleaver depending on the setting.

This block is a custom coded block built with C code, and its parameters are listed in Table 4.13. The input and output are in Table 4.14.

Table 4.13: Parameters for the vector (de)interleaver *VECTOR INTERLEAVE*.

Parameter Name	**Type and Range**	**description**
Mode (interleaver or deinterleaver)	string, 'interleave' or 'deinterleave'.	This specifies the function of the block. Let the interleaver mapping be $\alpha(i), 0 \leq i \leq N-1$. If 'interleave' is specified, $y(i) = x(\alpha(i))$. If 'deinterleave' is specified, $y(\alpha(i)) = x(i)$.
Size of vectors	integer	The size of input and output vectors. It is denoted by N.
Interleaver mapping data file	string	It specifies the path and name of the interleaver file. The file should contain integers in ASCII form ranging from 0 to $N-1$, with their order scrambled.

Table 4.14: Input and output for the vector (de)interleaver *VECTOR INTERLEAVE*.

Input Name	**Description**
x	The input vector of size N.
Output Name	**Description**
y	The output vector of size N.

Table 4.15: Parameters for the postprocessor *display_bfer*.

Parameter Name	**Type and Range**	**description**
Signal to noise ratio E_b/N_0 (dB)	double, usually 0.0 (dB) $< E_b/N_0 <$ 5.0 (dB)	This is the signal to noise ratio that the simulation is running at. This parameter is included just for displaying purpose.
Frame size not including tail bits	integer	This indicates the size of a frame of information bits. It is used to parse frames in a stream before the frames are counted.
Number of frame errors to count	integer, > 0, usually less than 50.	This parameter specifies the number of frame errors to count before the simulation stops.

Table 4.16: Input for the postprocessor *display_bfer*.

Input Name	**Description**
bit_tx	This is the transmitted binary data stream. Notice that it should be the delayed version of the actual transmitted bits, so that it is aligned properly with the estimation.
bit_rx	This is the estimated binary data stream. When there is no bit error, it is the same as 'bit_tx'.

display_bfer: Calculate and Display BER and FER

The *display_bfer* block accepts the tranmitted binary (0/1) bit stream ('bit_tx') and its estimation ('bit_rx'), then calculates and displays the results in a pop-up window. The displayed information includes: the number of transmitted frames, the number of frame errors, the frame error rate, the number of transmitted 'info' bits, the number of bit errors, and the bit error rate. Notice that transmitted bit stream ('bit_tx') should be aligned with the recovered bit stream ('bit_rx') before they are sent to this block. This is usually done by delaying the transmitted bits ('bit_tx') .

The block is expanded in three parts in Figures 4.40, 4.41 and 4.42. The parameters are explained in Table 4.15, and the inputs are explained in Table 4.16.

Table 4.17: Parameter for the block computing σ, *EbN0_sigma*.

Parameter Name	**Type and Range**	**description**
1/rate	double	The inverse of the code rate.

Table 4.18: Input and output for the block computing σ, *EbN0_sigma*.

Input Name	**Description**
E_b/N_o	Signal to noise ratio in dB.
Output Name	**Description**
sigma	This is the deviation of the AWGN noise. It is used to scale the $N(0,1)$ distributed random variables to become proper AWGN noise.

EbN0_sigma: **Noise Deviation** σ

The *EbN0_sigma* block computes the deviation of the AWGN noise to provide the specified E_b/N_0 for the simulation. Unit power is assumed for the transmitted bipolar symbols. This block performs the algebraic calculation with the formula:

$$\begin{aligned} \sigma &= 1/\sqrt{2rE_b/N_0} \\ &= 1/\sqrt{2r10^{0.1E_b/N_0(dB)}} \end{aligned} \tag{4.176}$$

where r is the code rate. Its parameter is shown in Table 4.17, while the input and output are displayed in Table 4.18.

EbN0_Lc: **Channel Reliability** L_c

The *EbN0_Lc* block computes L_c based on E_b/N_0. L_c is used to scale the received signal to obtain the LLR input of the codeword bits. This block basically performs an algebraic calculation with the formula:

$$\begin{aligned} L_c &= 4arE_b/N_0 \\ &= 4ar10^{0.1E_b/N_0(dB)} \end{aligned} \tag{4.177}$$

Here a is the fading amplitude, which is equal to 1 for an AWGN channel; r is the code rate; E_b/N_0 is the signal-to-noise ratio.

Table 4.19: Parameter for the block computing L_c, *EbN0_Lc*.

Parameter Name	**Type and Range**	**description**
1/rate	double	The inverse of the code rate.

Table 4.20: Input and output for the block computing L_c, *EbN0_Lc*.

Input Name	**Description**
E_b/N_o	Signal to noise ratio in dB.
Output Name	**Description**
L_c	This is the channel reliability corresponding to a given E_b/N_0.

The input of this block is shown in Table 4.19, while the input and output are explained in Table 4.20.

4.5 Summary

In this chapter, we have presented complete SPW models of PCCC and SCCC. These models represent an important intermediate towards full implementation. In Chapter 5 we make use of the models to undertake a comprehensive survey of the performance of concatenated convolutional codes. In Chapter 6, we apply the model to examine the fixed point implementation of turbo codes.

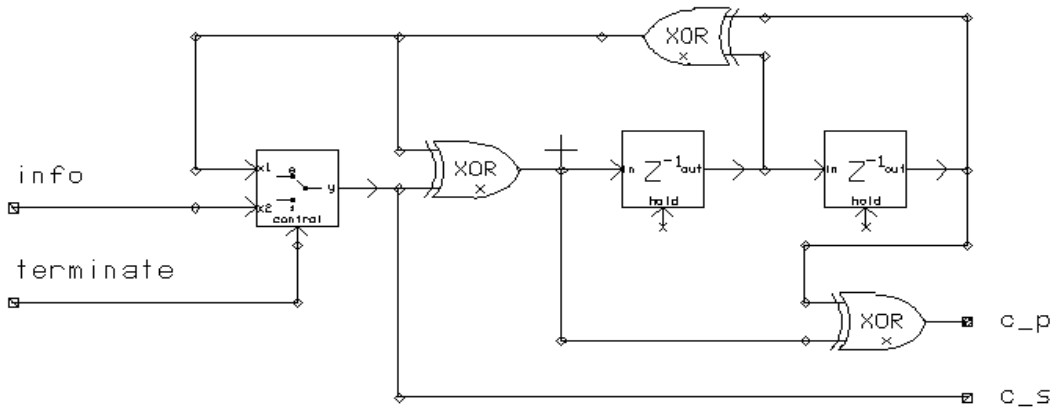


Figure 4.31: SPW model *rsc7_5*: RSC encoder $g = (7, 5)_{octal}$ using tail bits to terminate.

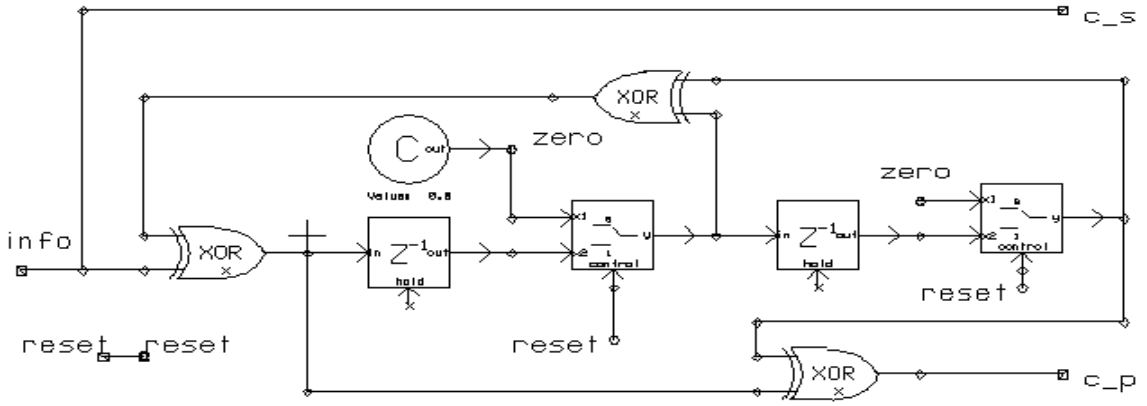


Figure 4.32: SPW model *rsc7_5_init*: RSC encoder $g = (7,5)_{octal}$ with direct reset.

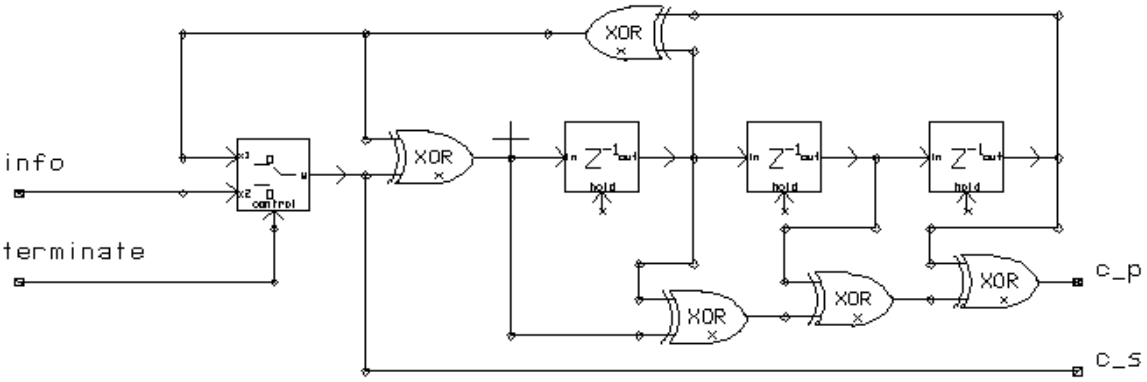


Figure 4.33: SPW model *rsc15_17*: RSC encoder $g = (15,17)_{octal}$ using tail bits to terminate.

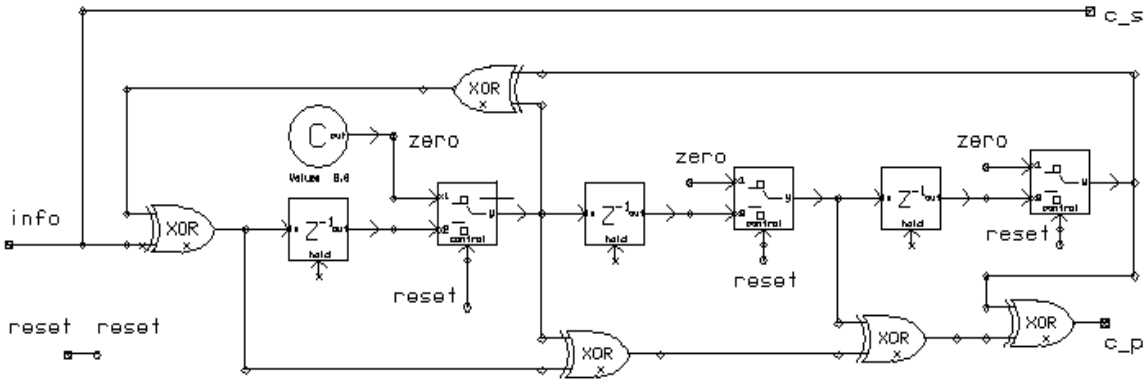


Figure 4.34: SPW model *rsc15_17_init*: RSC encoder $g = (15,17)_{octal}$ with direct reset.

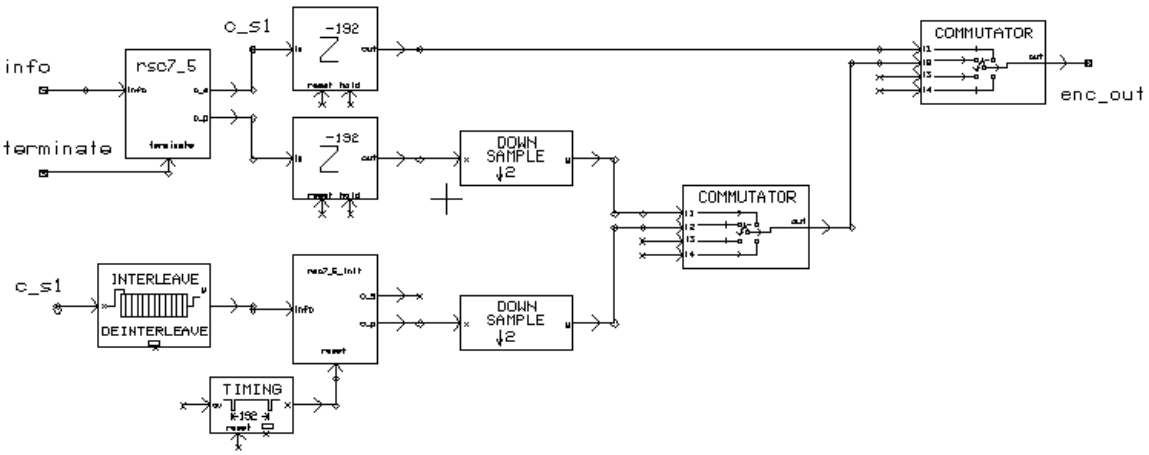


Figure 4.35: SPW model *p_enc7_5_p*: rate 1/2 PCCC encoder.

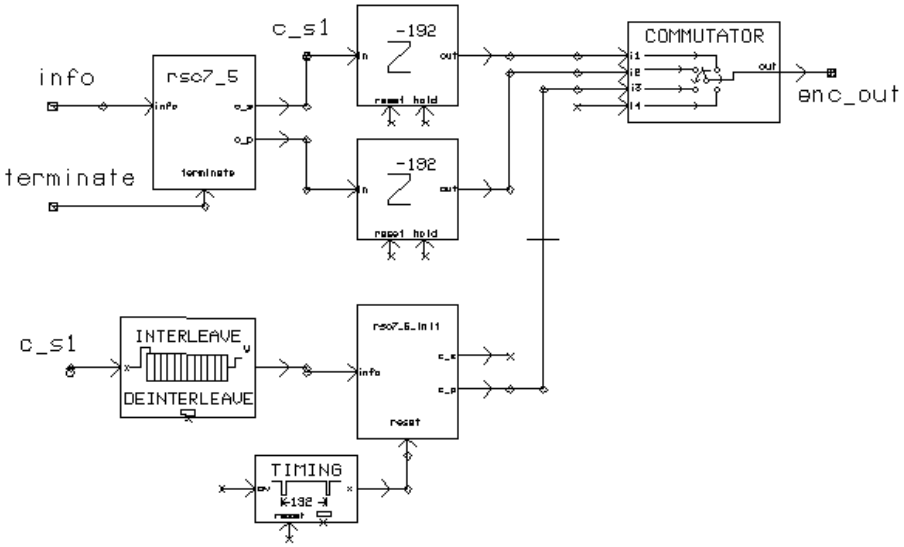


Figure 4.36: SPW model *p_enc7_5*: rate 1/3 PCCC encoder.

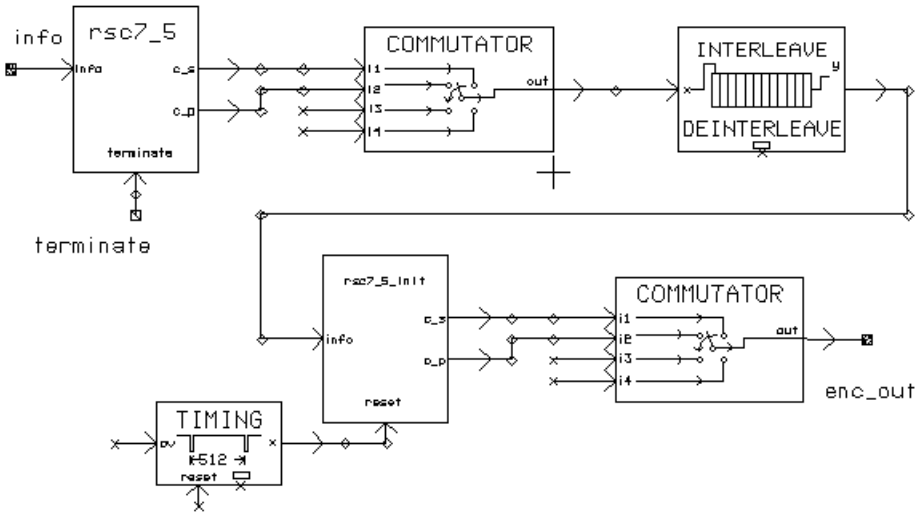


Figure 4.37: SPW model *s_enc7_5_4*: rate 1/4 SCCC encoder.

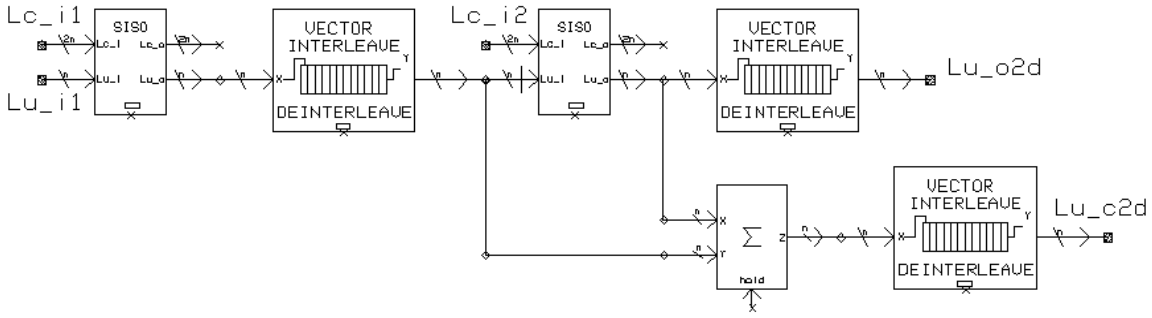


Figure 4.38: SPW model *pccc_dec_it1*: one decoding iteration of PCCC.

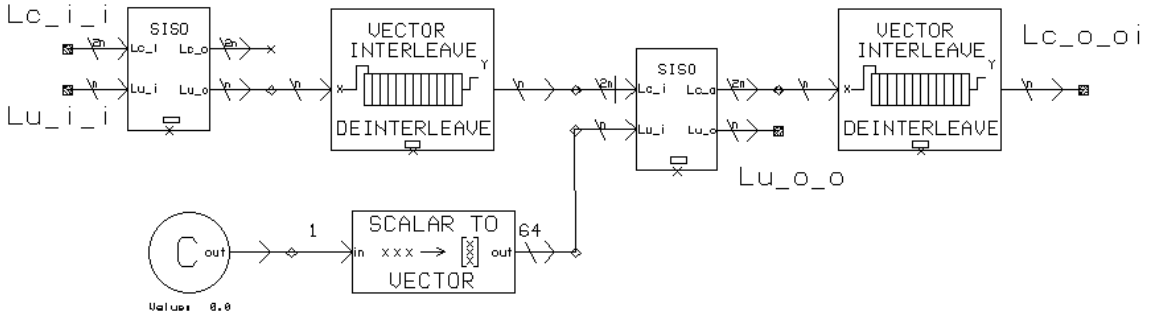


Figure 4.39: SPW model *sccc_dec_it1*: one decoding iteration of SCCC.

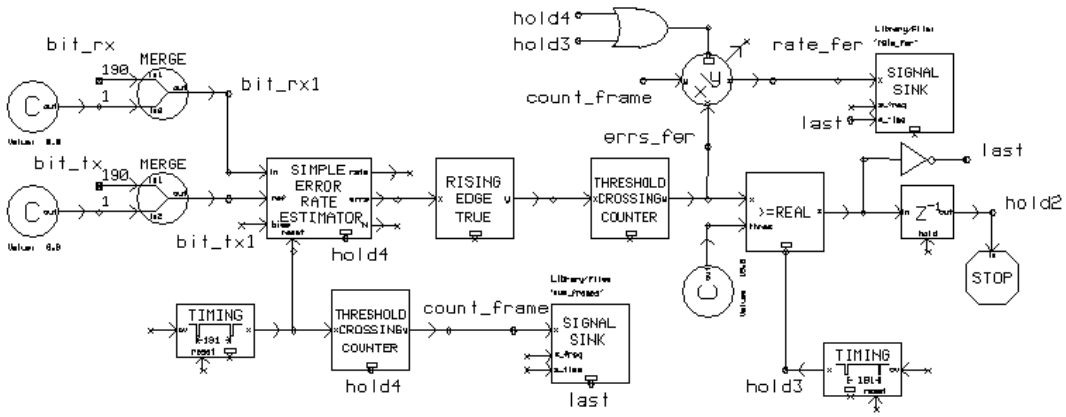


Figure 4.40: SPW model *display_bfer* part 1: FER calculator.

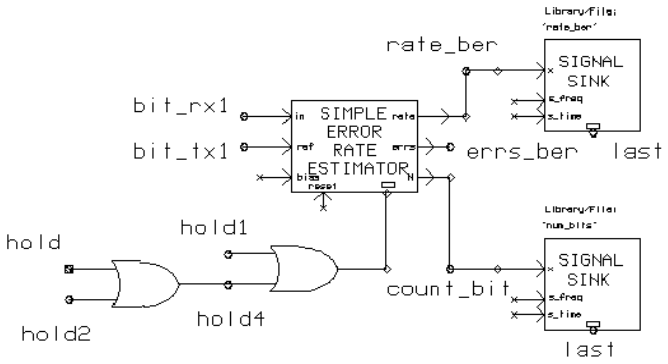


Figure 4.41: SPW model *display_bfer* part 2: BER calculator.

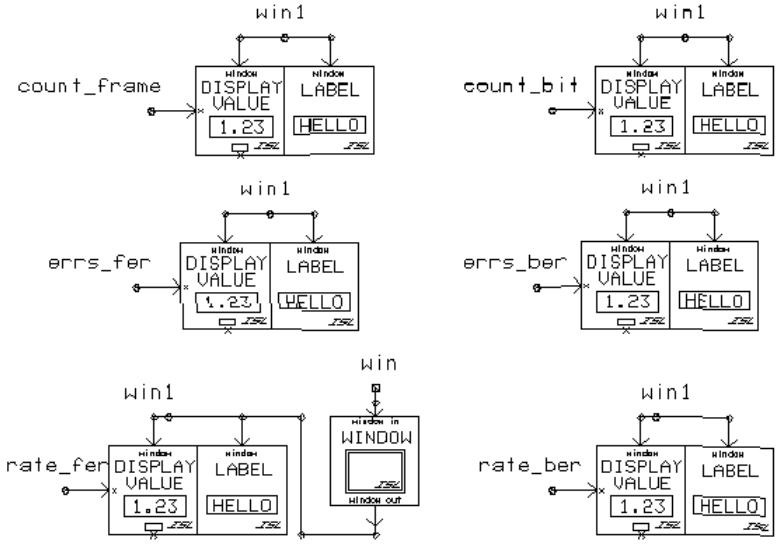


Figure 4.42: SPW model *display_bfer* part 3: window display.

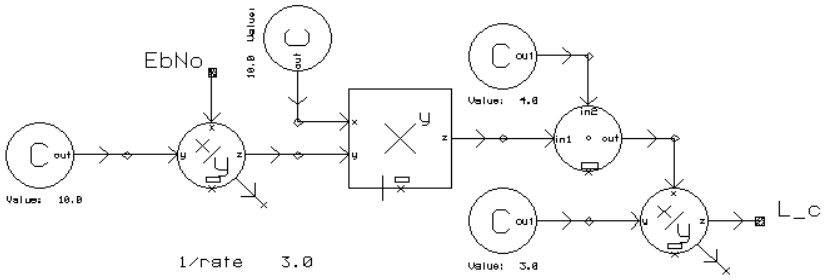


Figure 4.43: SPW model *EbN0_Lc*: L_c calculator.

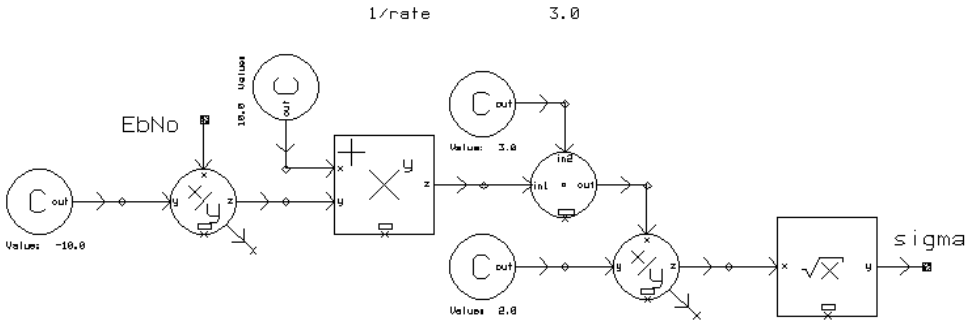


Figure 4.44: SPW model *EbN0_sigma*: σ calculator.

Chapter 5

Performance of Concatenated Codes

5.1 Introduction

With the structures and the decoding algorithms of both PCCC and SCCC systems explained in previous chapters, we now undertake a comprehensive study of the characteristics of their bit error probability performance in this chapter. This knowledge enables us to choose proper design parameters in accordance to specific performance requirements. Curves of BER vs. E_b/N_0 were plotted to show the influence of various parameters, including the number of iterations, the frame size, the code rate, and the code generator. Also a comparison of the performance of PCCC and SCCC was undertaken in Section 5.3.

5.2 Simulation Curves for PCCC

In this section, simulation curves for different cases are shown for PCCC. The properties of its performance can be observed from the plots.

5.2.1 Influence of Iteration Number

In Figures 5.45 through 5.50, BER vs. E_b/N_0 curves are shown parameterized by the number of decoding iterations. The simulation parameters are provided in Table 5.21. Rate 1/2 codes are obtained from their rate 1/3 counterparts by alternatively puncturing the parity bits of the constituent encoders.

Figures 5.45 and 5.46 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

Table 5.21: Simulation parameters for Figures 5.45 to 5.50.

Figure #	code generator	code rate	frame sizes
5.45	$g(D) = [1, \frac{1+D^2}{1+D+D^2}]$	1/2	200, 400, 1000, 2000, 4000, 16000
5.46	$g(D) = [1, \frac{1+D^2}{1+D+D^2}]$	1/3	200, 400, 1000, 2000, 4000, 16000
5.47	$g(D) = [1, \frac{1+D+D^2+D^3}{1+D+D^3}]$	1/2	200, 400, 1000, 2000, 4000, 16000
5.48	$g(D) = [1, \frac{1+D+D^2+D^3}{1+D+D^3}]$	1/3	200, 400, 1000, 2000, 4000, 16000
5.49	$g(D) = [1, \frac{1+D+D^2+D^3+D^4}{1+D+D^4}]$	1/2	200, 400, 1000, 2000, 4000, 16000
5.50	$g(D) = [1, \frac{1+D+D^2+D^3+D^4}{1+D+D^4}]$	1/3	200, 400, 1000

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 vs. E_b/N_0 curve flattens, shows up clearly. Comparing Figure 5.46 to Figure 5.45, it can be observed that a larger coding gain is achieved by decreasing the code rate from 1/2 to 1/3.

Comparing Figures 5.47, 5.48, 5.49, and 5.50 to Figures 5.45 through 5.46, it can be observed that, for the same frame size, code rate, and number of iterations, increasing the constraint length decreases BER at BER$< 10^{-3}$ region. In Figure 5.50, BER vs. 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.

5.2.2 Influence of Frame Size

In Figures 5.51 to 5.56, BER vs. E_b/N_0 curves are shown parameterized by the frame size. The simulation parameters are provided in Table 5.22. Rate 1/2 codes are obtained from the rate 1/3 counterparts by alternatively puncturing the parity bits of the constituent encoders. These figures show that frame size is a very 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.

5.2.3 Influence of Code Rate

In Figures 5.57 to 5.62, the BER vs. E_b/N_0 curves are shown parameterized by the code rate. The simulation parameters are provided in Table 5.23. Rate 1/2 codes are obtained from

Table 5.22: Simulation parameters for Figures 5.51 to 5.56.

Figure #	code generator	code rate	frame sizes	iteration #
5.51	$g(D) = [1, \frac{1+D^2}{1+D+D^2}]$	1/2	200 to 16000	10
5.52	$g(D) = [1, \frac{1+D^2}{1+D+D^2}]$	1/3	200 to 16000	10
5.53	$g(D) = [1, \frac{1+D+D^2+D^3}{1+D+D^3}]$	1/2	200 to 16000	10
5.54	$g(D) = [1, \frac{1+D+D^2+D^3}{1+D+D^3}]$	1/3	200 to 16000	10
5.55	$g(D) = [1, \frac{1+D+D^2+D^3+D^4}{1+D+D^4}]$	1/2	200 to 16000	10
5.56	$g(D) = [1, \frac{1+D+D^2+D^3+D^4}{1+D+D^4}]$	1/3	200, 400, 1000	10

their rate 1/3 counterparts by alternatively 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, there is 0.68 dB extra coding gain by decreasing the code rate from 1/2 to 1/3. This difference is approximately matched by the simulation results in Figures 5.57 to 5.62.

Table 5.23: Simulation parameters for Figures 5.57 to 5.62.

Figure #	constraint length	code rate	frame sizes	iteration #
5.57	3, 4, 5	1/2, 1/3	200	10
5.58	3, 4, 5	1/2, 1/3	400	10
5.59	3, 4, 5	1/2, 1/3	1000	10
5.60	3, 4	1/2, 1/3	2000	10
5.61	3, 4	1/2, 1/3	4000	10
5.62	3, 4	1/2, 1/3	16000	18

5.2.4 Influence of Code Generator

In Figures 5.63 and 5.64, BER vs. E_b/N_0 curves are shown for constituent codes of constraint length three, four, and five. Figure 5.63 is for code rate 1/2, and Figure 5.64 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. And the divergence is more pronounced when the frame size is large. Also, the improvement going from constraint length three to four is larger than that from four to five.

5.2.5 Conclusion

The following conclusions can be drawn from the above figures:

- 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.

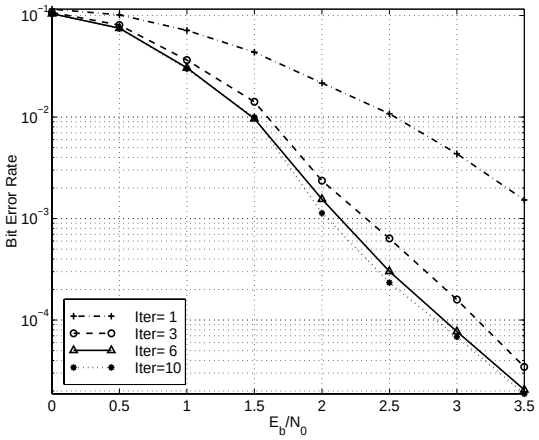
  - The second region is around $10^{-2} >$BER$> 10^{-5}$, where BER drops abruptly as E_b/N_0 increases. This is also called the "waterfall" region. This is the region where PCCC performs better in comparison to other codes. The gain comes from the characteristic that the multiplicities of low weight code sequences are small.

  - The third region is around BER$< 10^{-6}$, where BER decreases very slow with respect to E_b/N_0. This is called the "error floor" of PCCC, and it exists because of the small free distance of PCCC.

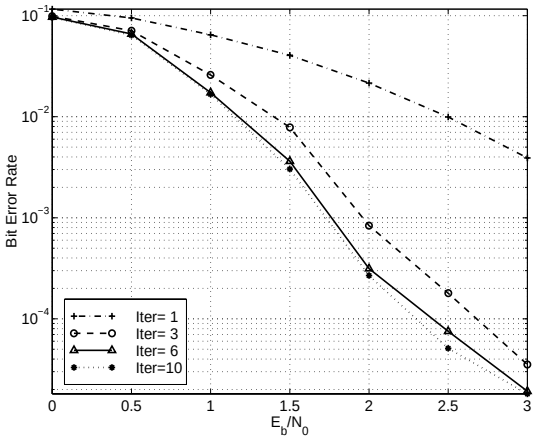
- The BER decreases approximately at the rate of N^{-1} as the frame size increases in the waterfall region. This manifests the coding gain of PCCC.

- The BER is lower for smaller code rate. Decreasing the code rate from 1/2 to 1/3 provides 0.5$\sim$0.8 dB extra coding gain around BER$= 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.

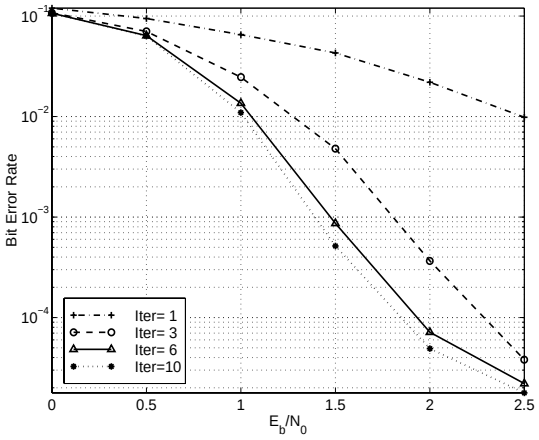
- Constraint length of the code generator does not play a significant role in the performance at BER$> 10^{-3}$ region. However, in the BER$< 10^{-4}$ region, a difference becomes evident. The lower BER is, the more significant the influence of the code generator. The error floor is lower for the schemes 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.



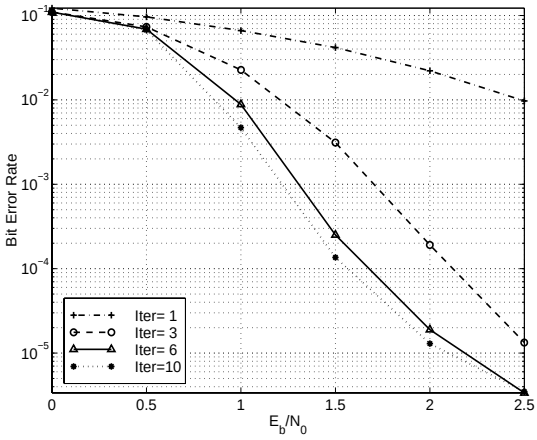
(a) N=200



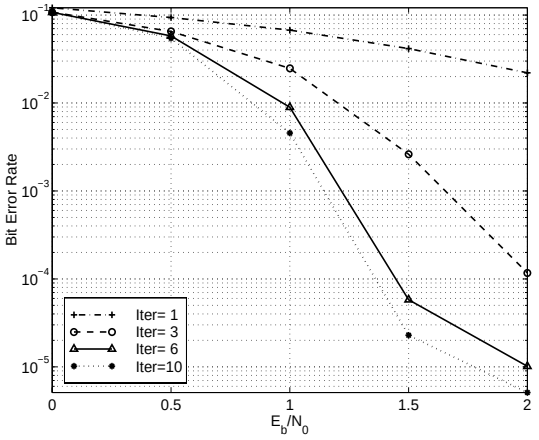
(b) N=400



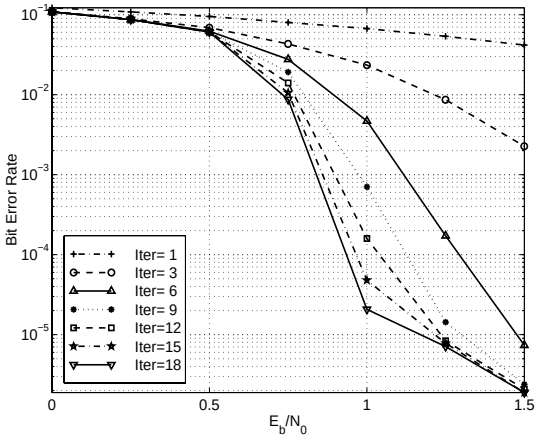
(c) N=1000



(d) N=2000

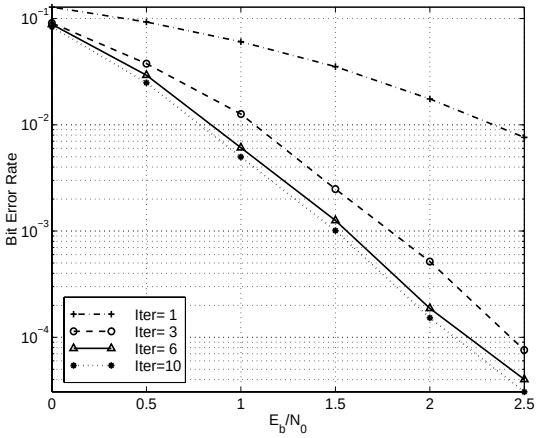


(e) N=4000

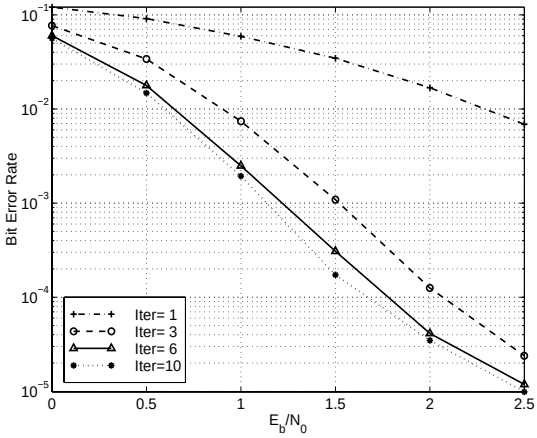


(f) N=16000

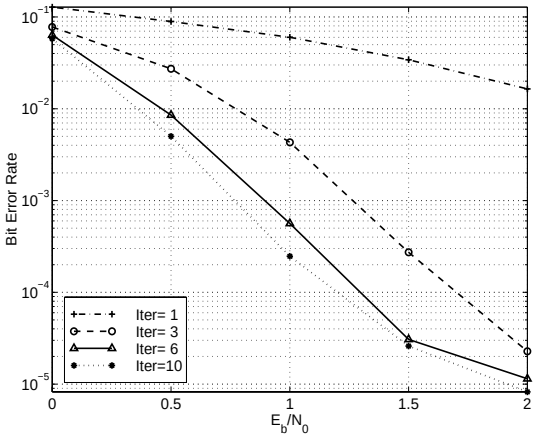
Figure 5.45: BER vs. E_b/N_0 as parameterized by the number of decoding iterations ($g = (7,5)_{octal}$, rate 1/2).



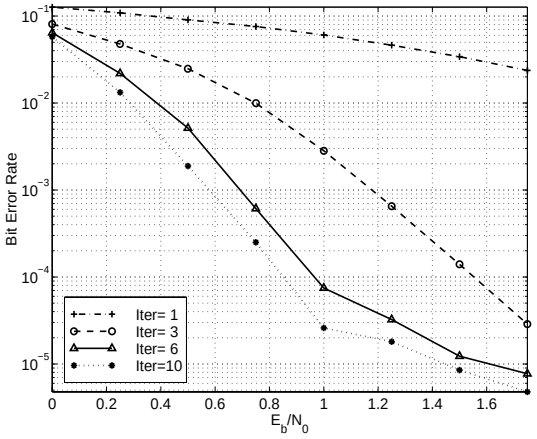
(a) N=200



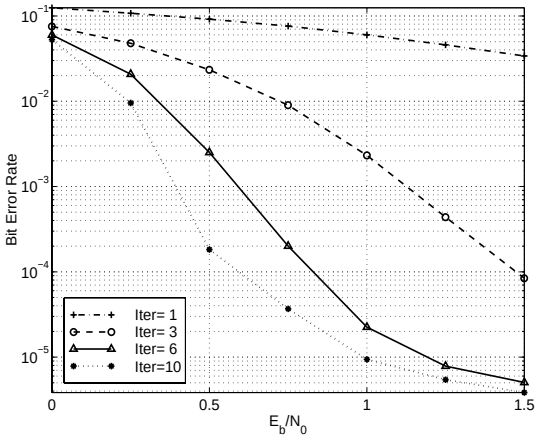
(b) N=400



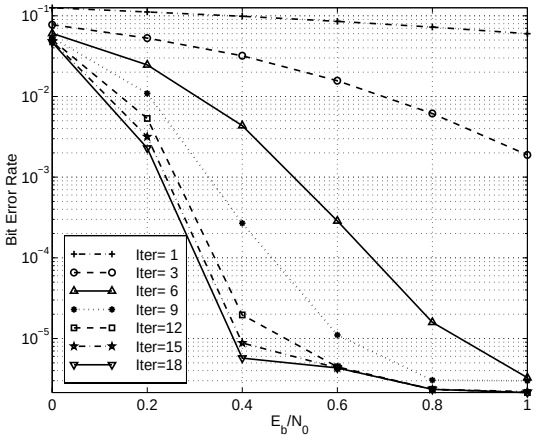
(c) N=1000



(d) N=2000

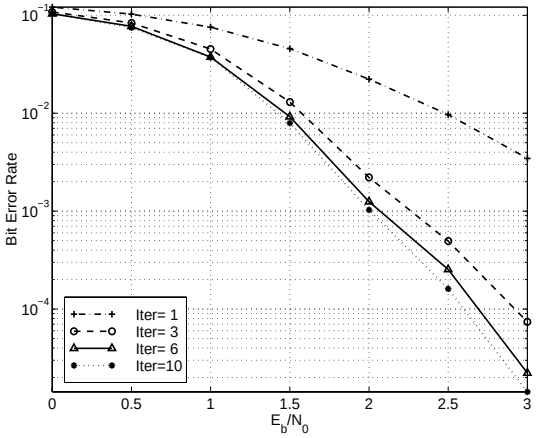


(e) N=4000

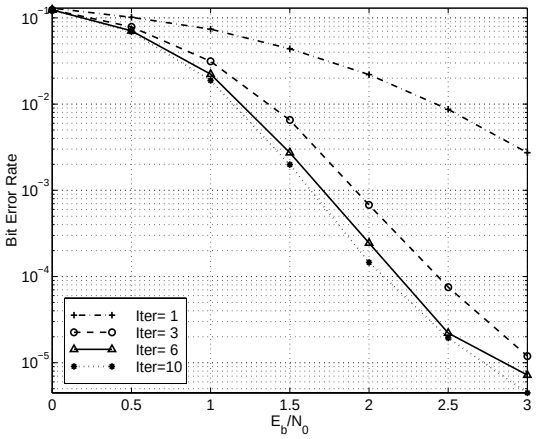


(f) N=16000

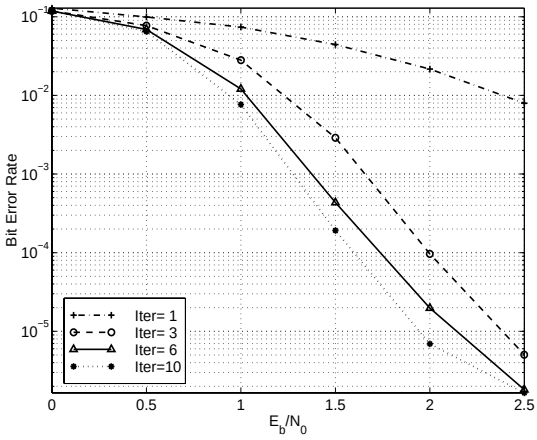
Figure 5.46: BER vs. E_b/N_0 as parameterized by the number of decoding iterations ($g = (7,5)_{octal}$, rate 1/3).



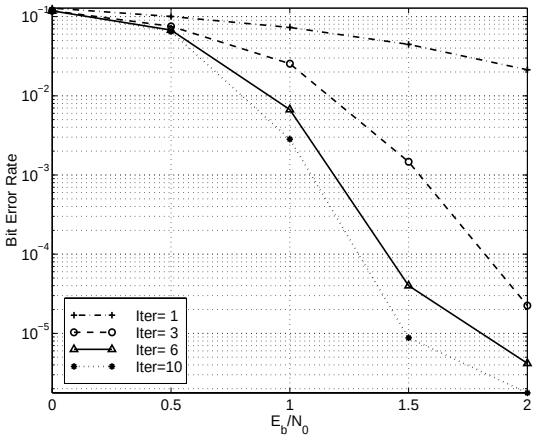
(a) N=200



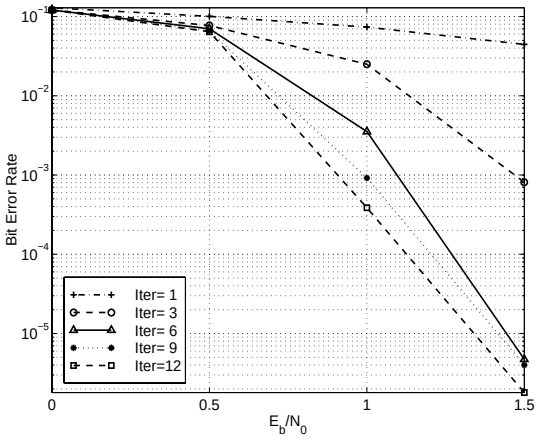
(b) N=400



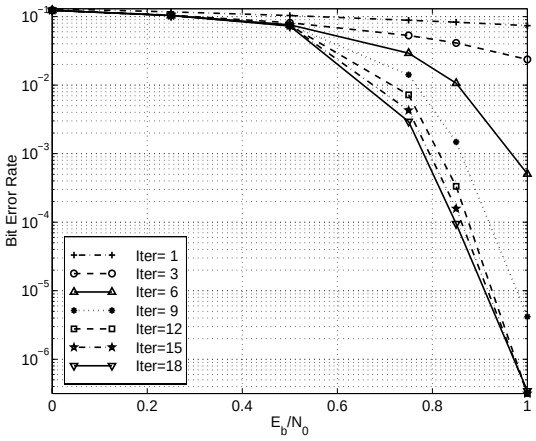
(c) N=1000



(d) N=2000

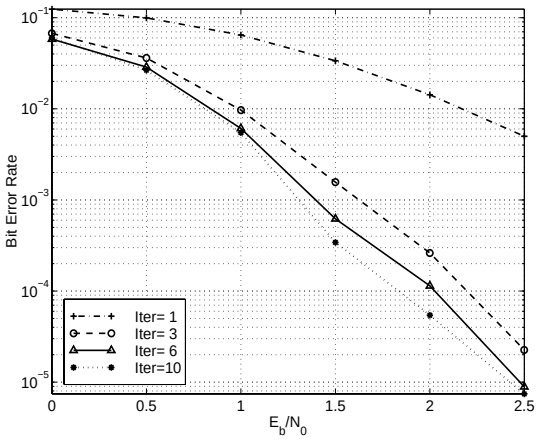


(e) N=4000

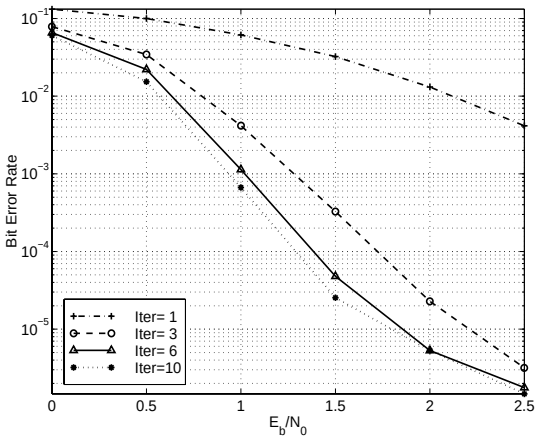


(f) N=16000

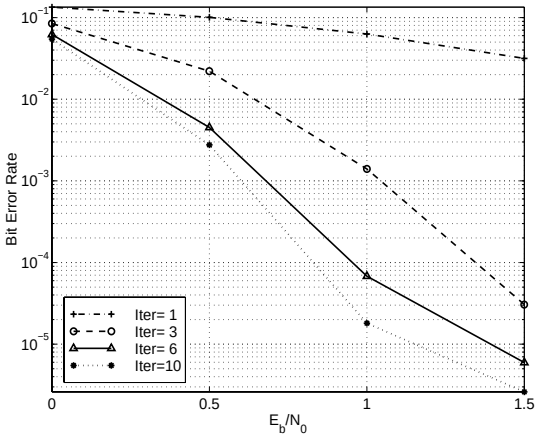
Figure 5.47: BER vs. E_b/N_0 as parameterized by the number of decoding iterations ($g = (15, 17)_{octal}$, rate 1/2).



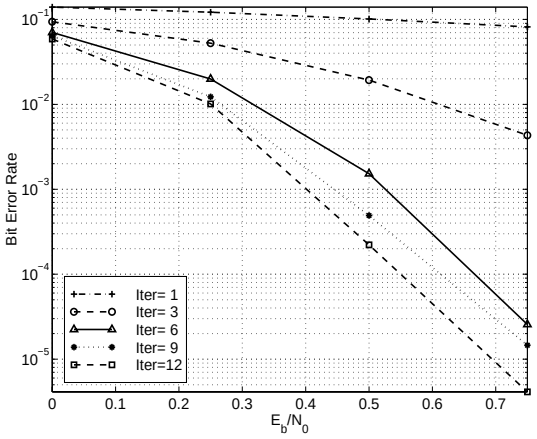
(a) N=200



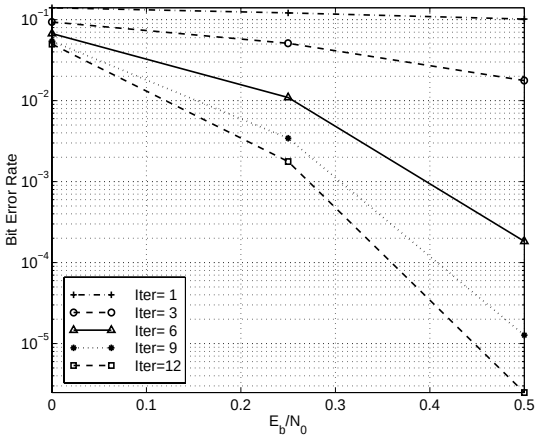
(b) N=400



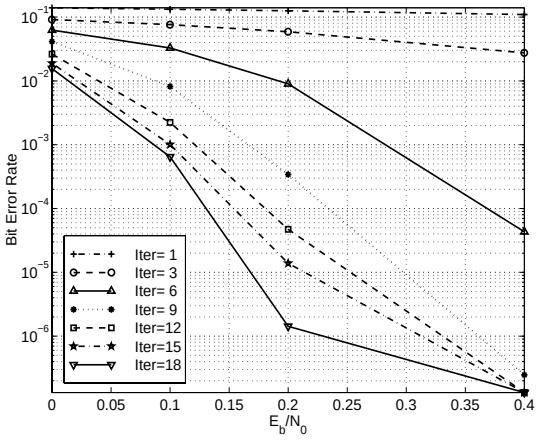
(c) N=1000



(d) N=2000

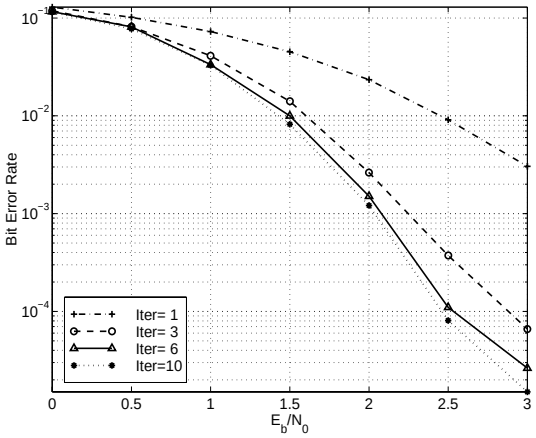


(e) N=4000

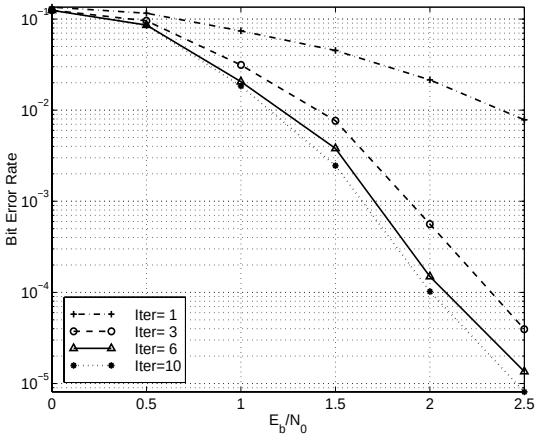


(f) N=16000

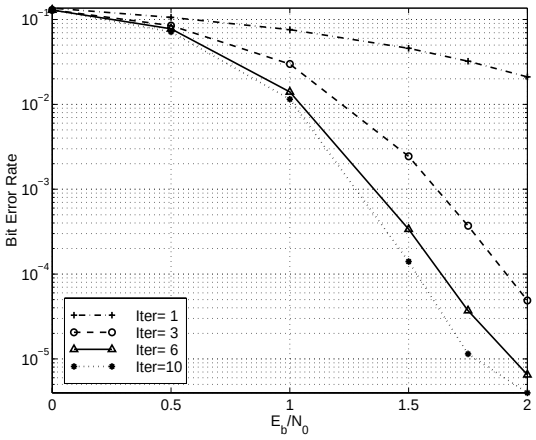
Figure 5.48: BER vs. E_b/N_0 as parameterized by the number of decoding iterations ($g = (15, 17)_{octal}$, rate 1/3).



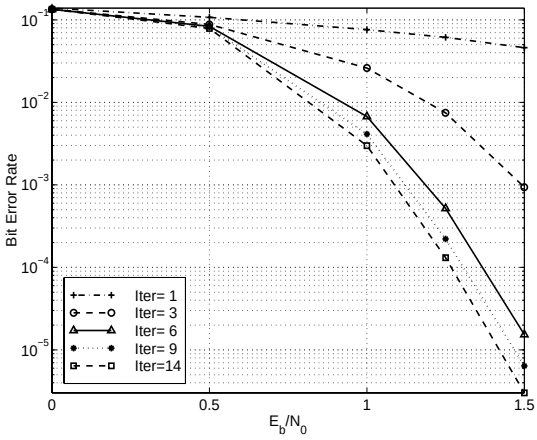
(a) N=200



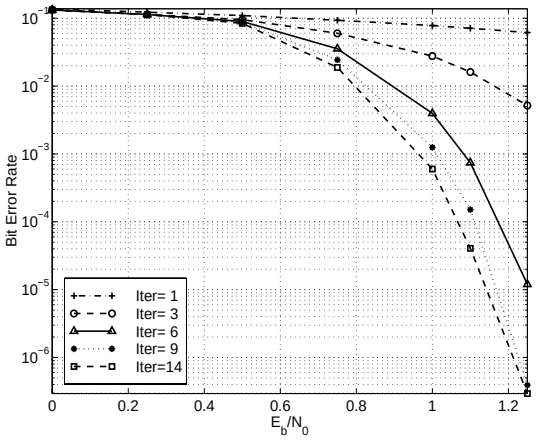
(b) N=400



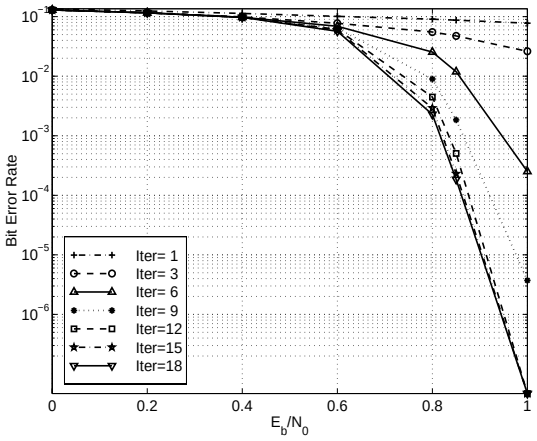
(c) N=1000



(d) N=2000

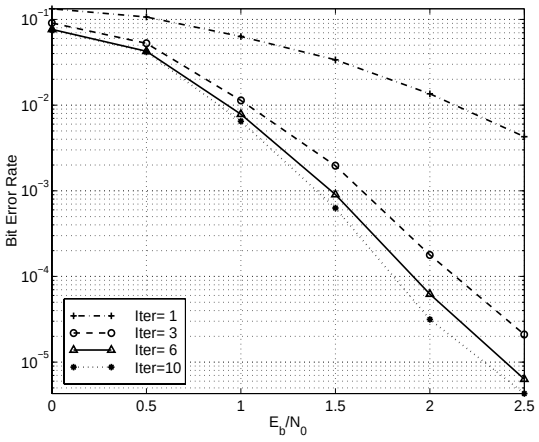


(e) N=4000

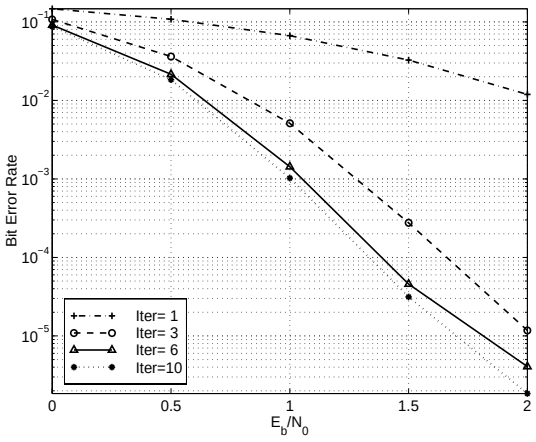


(f) N=16000

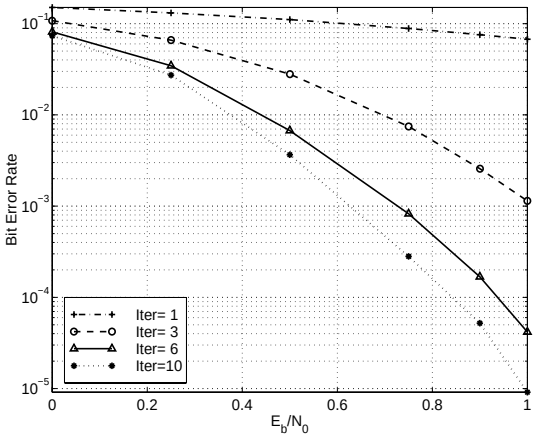
Figure 5.49: BER vs. E_b/N_0 as parameterized by the number of decoding iterations ($g = (31,37)_{octal}$, rate 1/2).



(a) N=200



(b) N=400



(c) N=1000

Figure 5.50: BER vs. E_b/N_0 as parameterized by the number of decoding iterations ($g = (31,37)_{octal}$, rate 1/3).

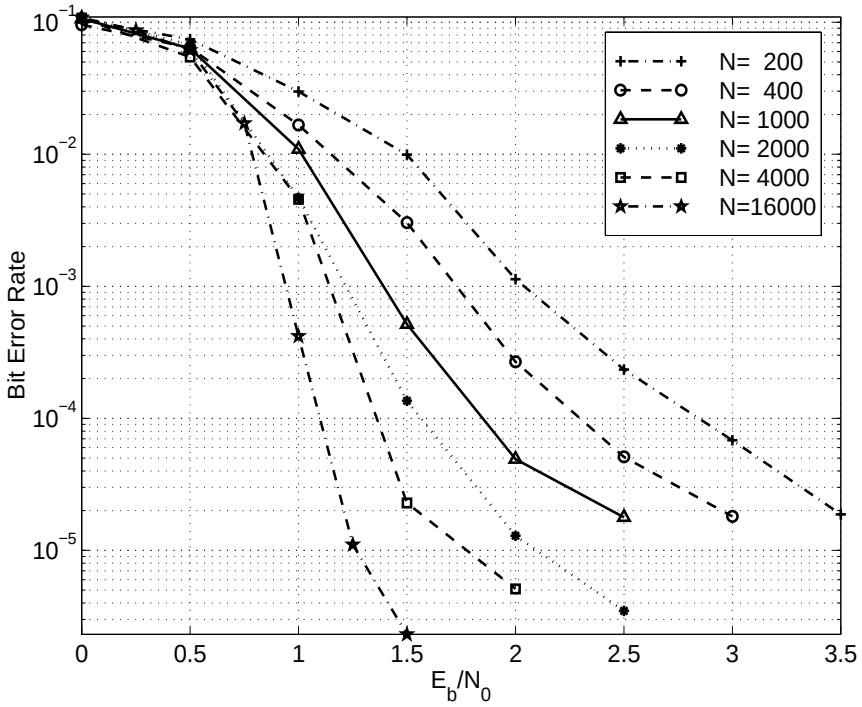


Figure 5.51: BER vs. E_b/N_0 as parameterized by frame size N ($g = (7,5)_{octal}$, rate 1/2, 10 iterations).

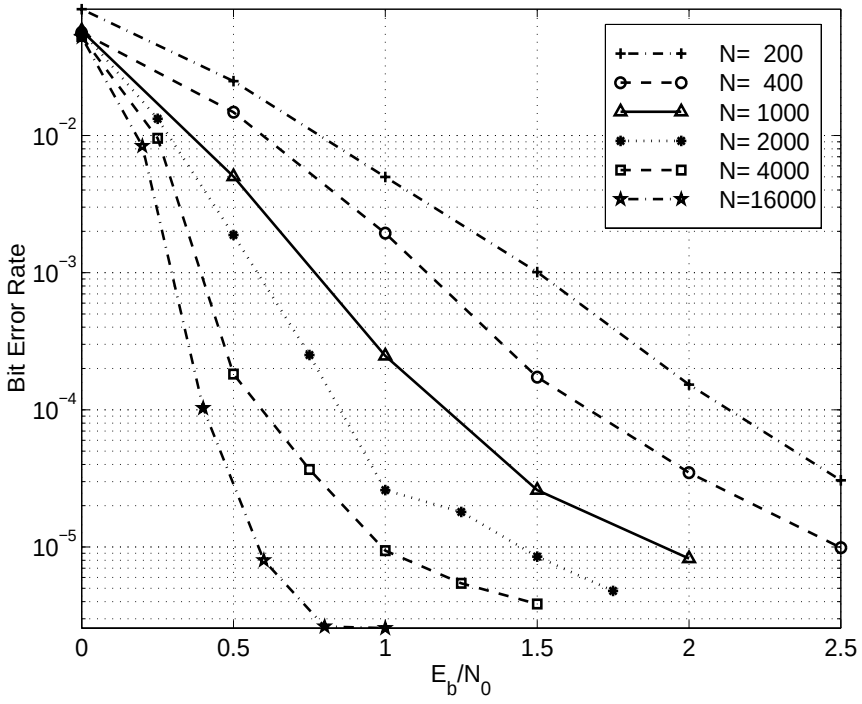


Figure 5.52: BER vs. E_b/N_0 as parameterized by frame size N ($g = (7,5)_{octal}$, rate 1/3, 10 iterations).

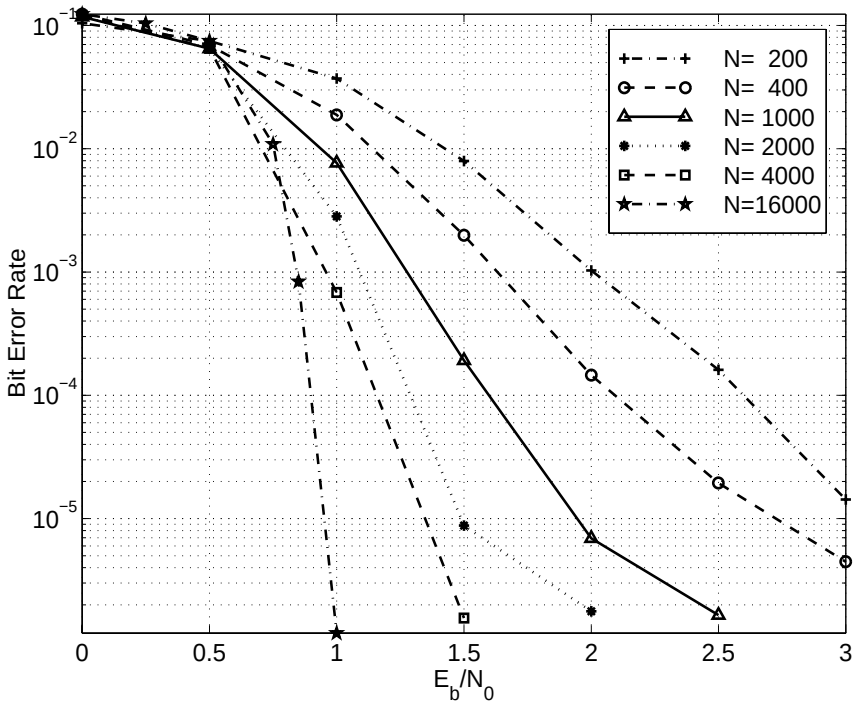


Figure 5.53: BER vs. E_b/N_0 as parameterized by frame size N ($g = (15, 17)_{octal}$, rate 1/2, 10 iterations).

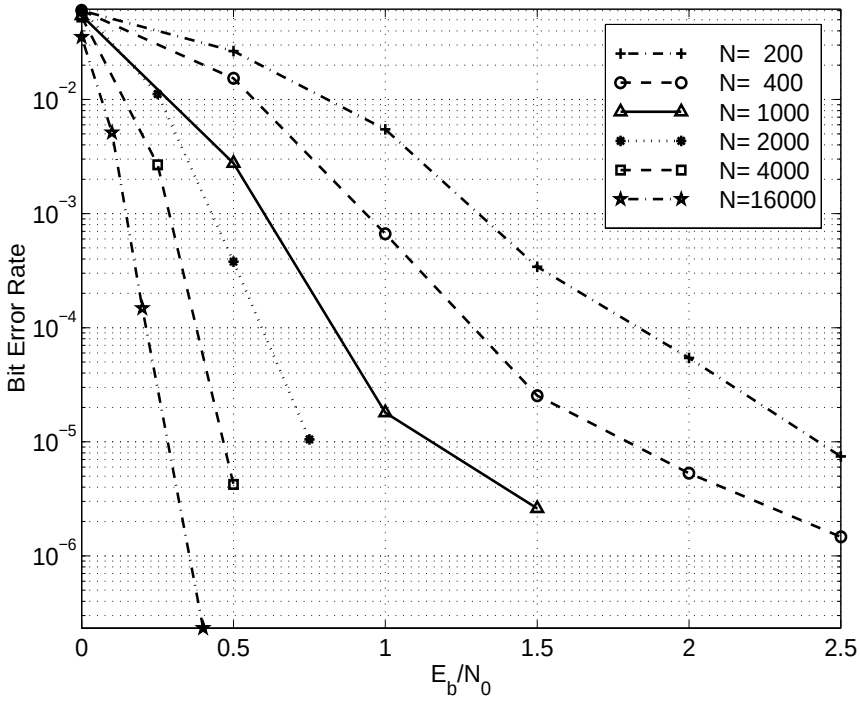


Figure 5.54: BER vs. E_b/N_0 as parameterized by frame size N ($g = (15, 17)_{octal}$, rate 1/3, 10 iterations).

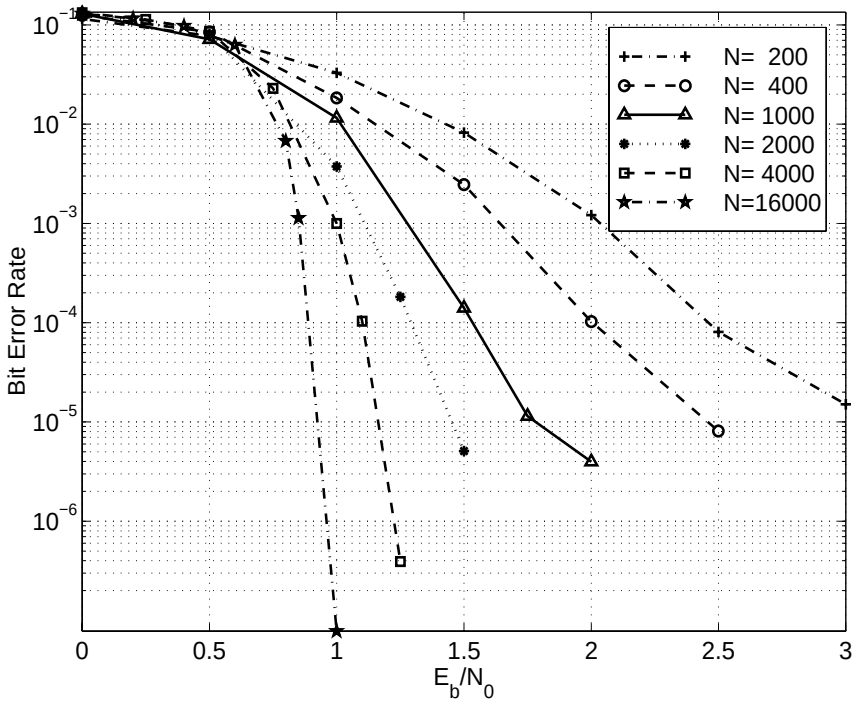


Figure 5.55: BER vs. E_b/N_0 as parameterized by frame size N ($g = (31,37)_{octal}$, rate 1/2, 10 iterations).

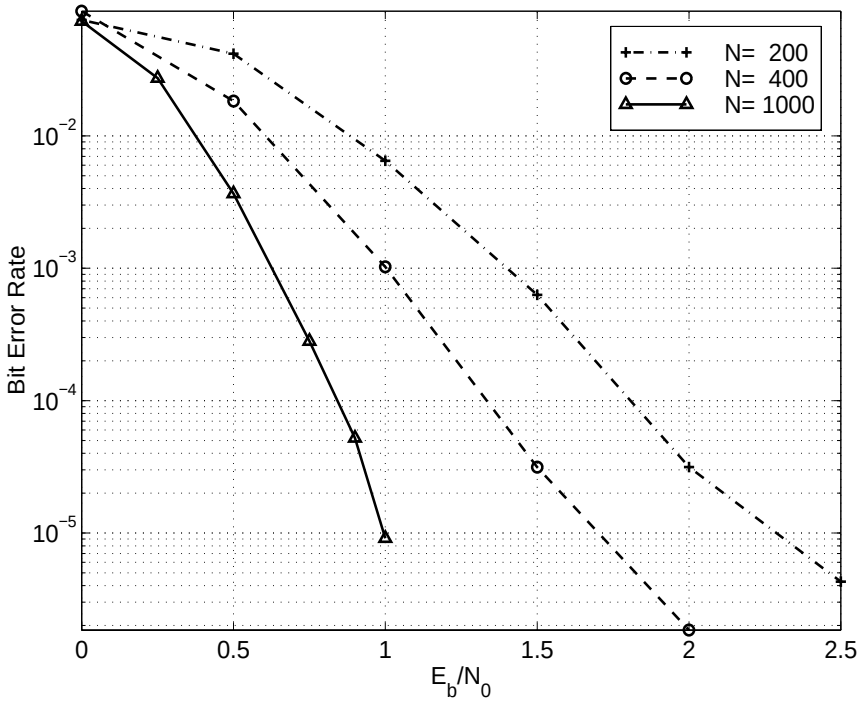


Figure 5.56: BER vs. E_b/N_0 as parameterized by frame size N ($g = (31,37)_{octal}$, rate 1/3, 10 iterations).

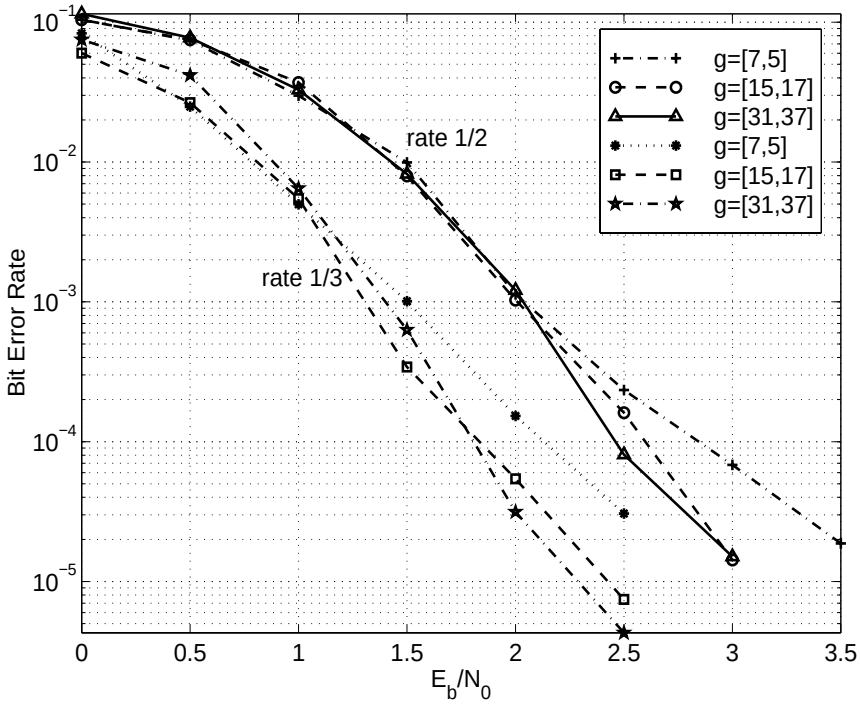


Figure 5.57: BER vs. E_b/N_0 as parameterized by code rate ($N = 200$, 10 iterations).

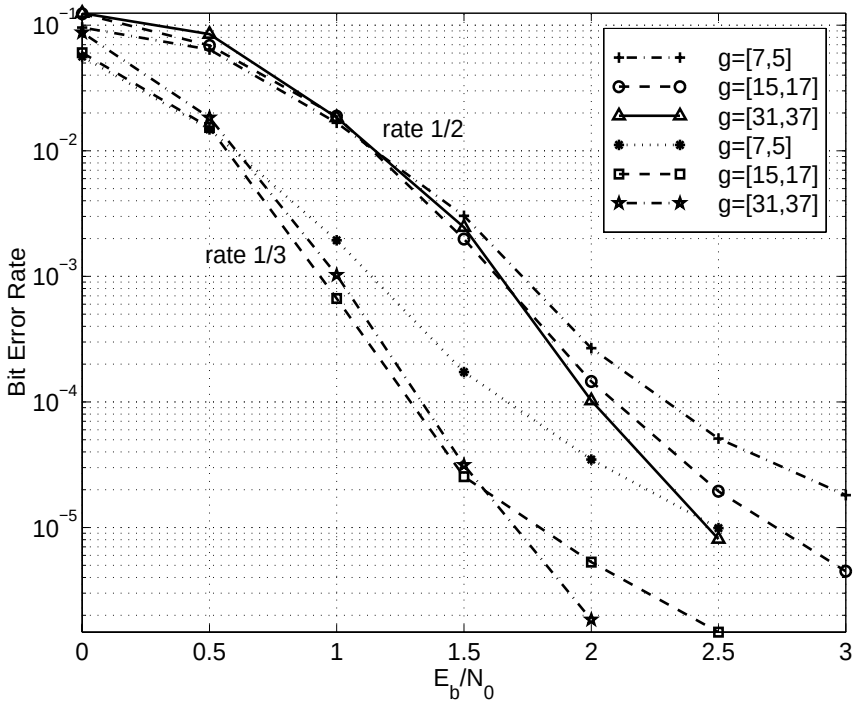


Figure 5.58: BER vs. E_b/N_0 as parameterized by code rate ($N = 400$, 10 iterations).

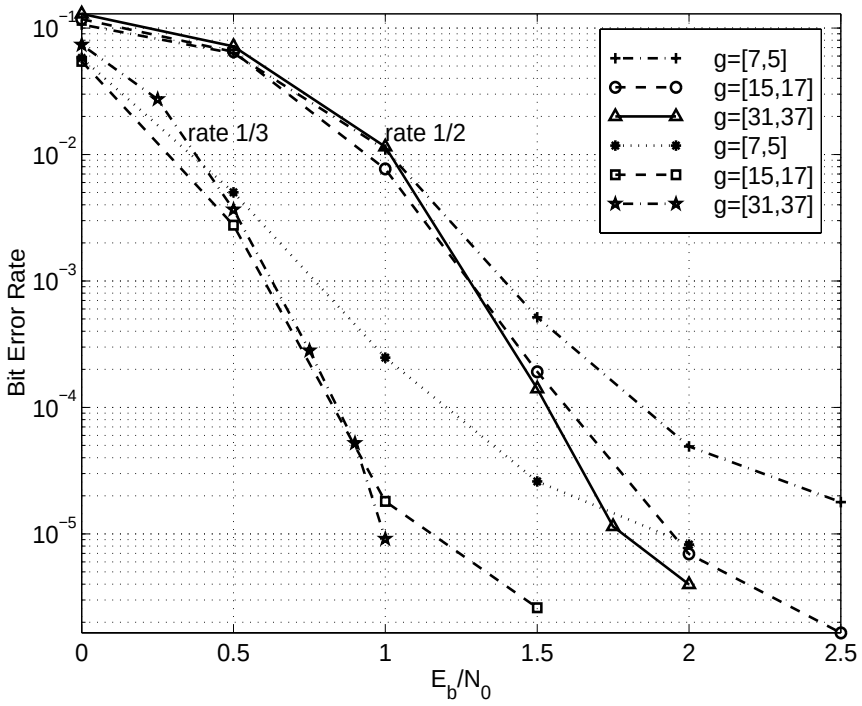


Figure 5.59: BER vs. E_b/N_0 as parameterized by code rate ($N = 1000$, 10 iterations).

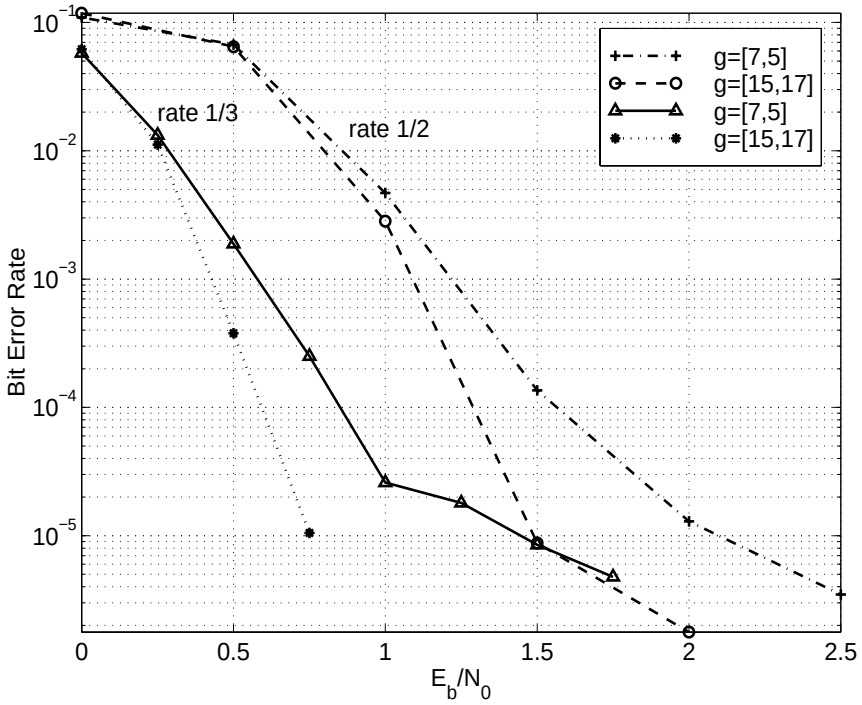


Figure 5.60: BER vs. E_b/N_0 as parameterized by code rate ($N = 2000$, 10 iterations).

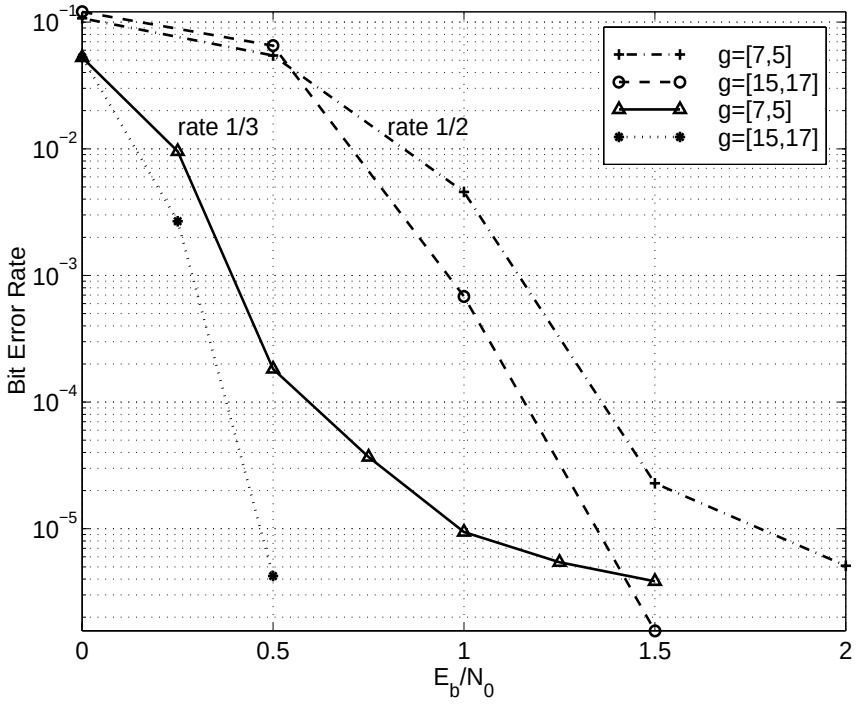


Figure 5.61: BER vs. E_b/N_0 as parameterized by code rate ($N = 4000$, 10 iterations).

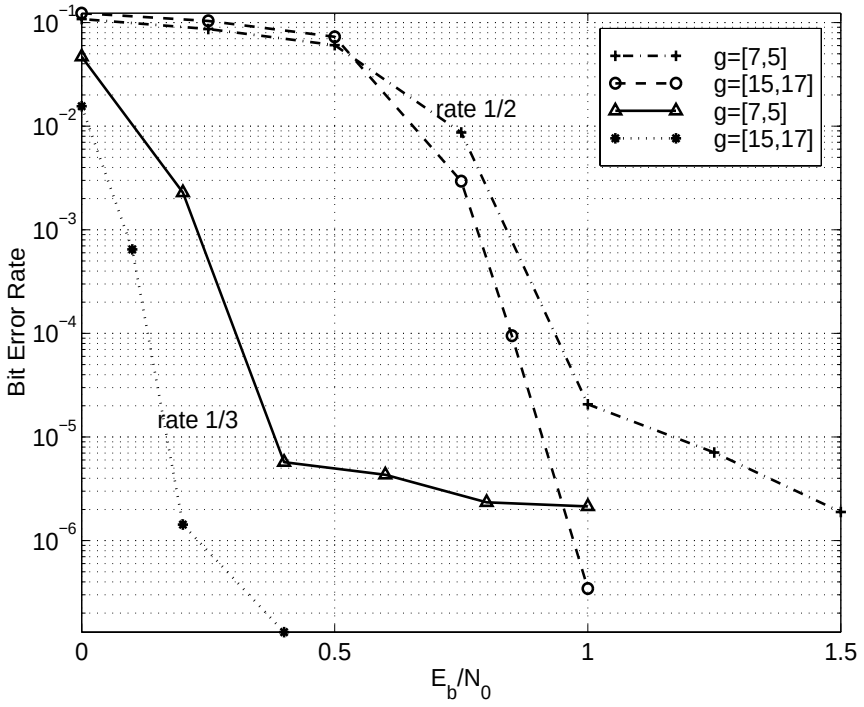


Figure 5.62: BER vs. E_b/N_0 as parameterized by code rate ($N = 1.6 \times 10^4$, 18 iterations).

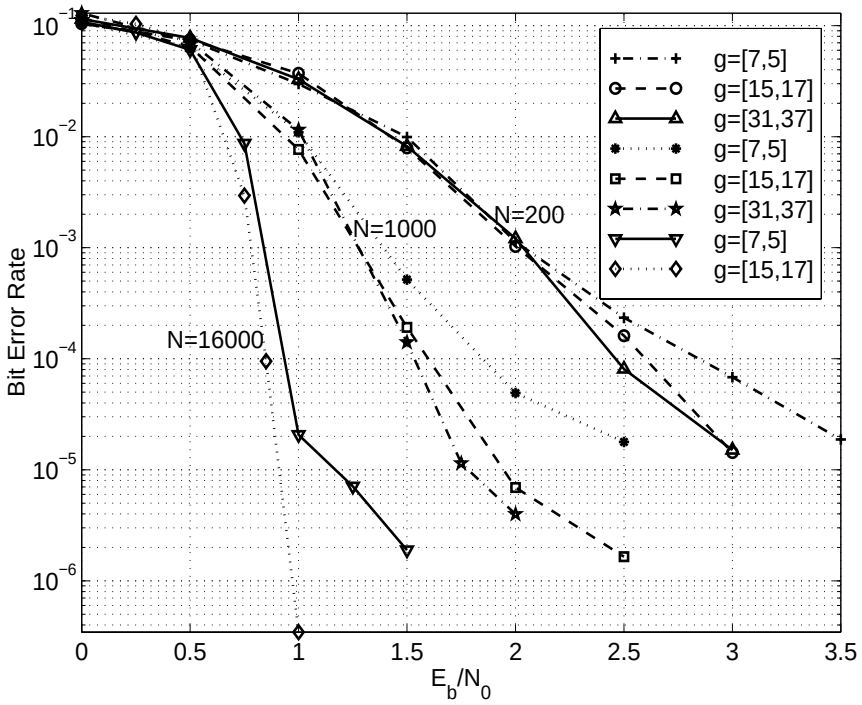


Figure 5.63: BER vs. E_b/N_0 as parameterized by code generator (code rate 1/2, 10 iterations for $N = 200$ and 1000, 18 iterations for $N = 1.6 \times 10^4$).

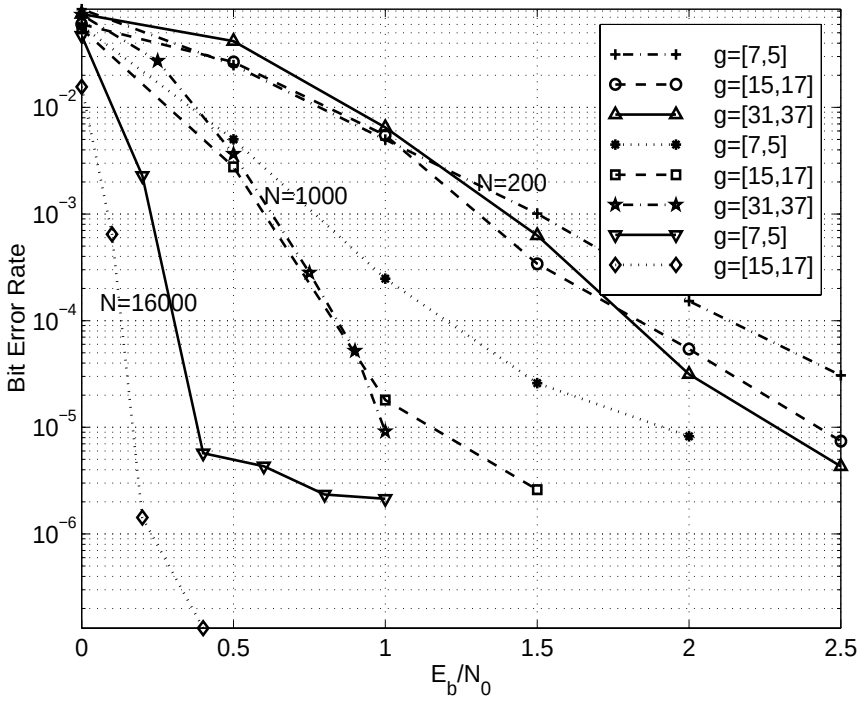


Figure 5.64: BER vs. E_b/N_0 as parameterized by code generator (code rate 1/3, 10 iterations for $N = 200$ and 1000, 18 iterations for $N = 1.6 \times 10^4$).

5.3 Simulation Curves for SCCC

In this section, the performance of SCCC is studied with Monte Carlo simulations. A spread interleaver is designed for each frame size as presented in [56].

In Figure 5.65, the curves of BER vs. E_b/N_0 are shown for SCCC of various numbers of iterations. The information frame size is 5120. The outer code rate is 1/3, the inner code rate is 2/3, and consequently, the overall code rate is 1/3. Both constituent codes are $K = 3$ RSC codes with $g(D) = [1, 1 + D^2/1 + D + D^2]$, and both are terminated with 2 tail bits. The parity bits of the inner code are alternatively punctured so that the inner code rate is increased to 2/3.

Figure 5.66 shows the curves for 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 5.67, simulation curves of BER vs. E_b/N_0 are drawn for SCCC of the same structure as that of Figure 5.65. Four different frame sizes are simulated: N=160, 320, 640, and 5120. Figure 5.68 shows the BER curves of the same frame sizes for PCCC of rate 1/3 as a comparison.

In Figure 5.69, $\Delta E_b/N_0$ is shown, which is the difference between the E_b/N_0 required by SCCC and that required by PCCC to reach a certain bit error probability. At the region where $\Delta E_b/N_0 > 0$, SCCC needs higher E_b/N_0 to achieve the same performance, while at $\Delta E_b/N_0 < 0$, SCCC requires less power to gain the same quality of performance.

The following conclusions can be drawn for SCCC in comparison to PCCC from the above curves.

- Just as for PCCC, the BER of SCCC goes down 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.
- SCCC performs better than PCCC at very low BER region.
  - At BER> 10^{-5} region, PCCC has a larger coding gain. Based on our simulations, the difference is about $0.3 \sim 0.4$ dB. The slow dropping region of SCCC extends to the waterfall region of PCCC.
  - At around BER= $10^{-5} \sim 10^{-6}$ region, PCCC and SCCC have similar coding gain.

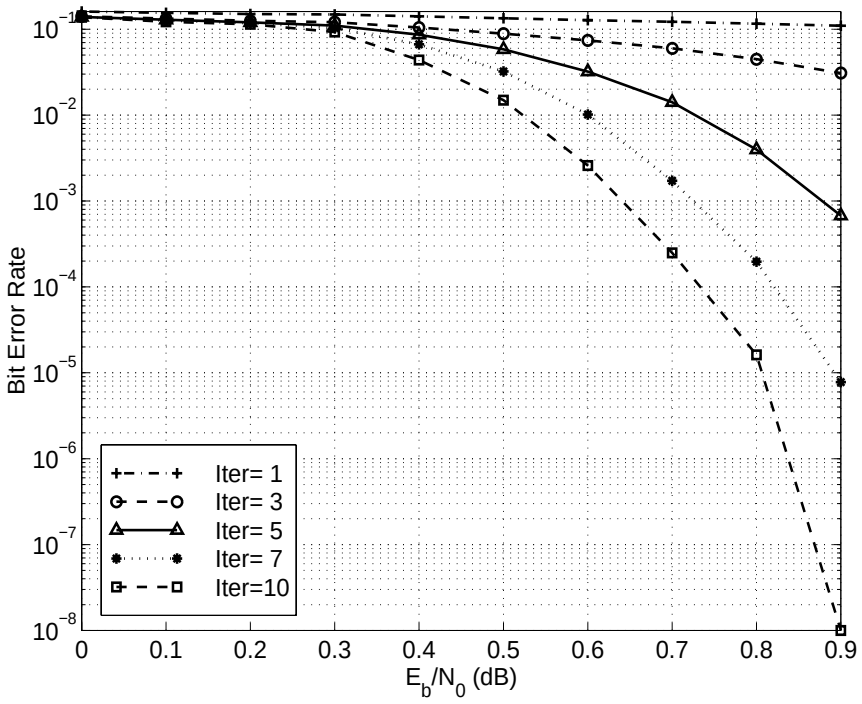


Figure 5.65: BER vs. E_b/N_0 as parameterized by number of decoding iterations (SCCC, $g = (7,5)_{octal}$, rate 1/3 ($r_o = 1/2, r_i = 2/3$), $N = 5120$).

- At BER$< 10^{-6}$, BER of SCCC falls down steeply, while PCCC reaches its error floor region. This shows that SCCC does not have a flat error floor region as PCCC does.

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 constraint length-K PCCC and that of a constraint length-$(K+1)$ PCCC.

5.4 Summary

In this chapter, a large number of simulation curves were presented to illustrate the performance of PCCC and SCCC systems with the SISO decoding algorithm introduced in Chapter 3. The waterfall and error floor regions of PCCC were displayed, while SCCC was shown to have a steep BER curve at medium-to-high E_b/N_0 region. In consequence, with comparable complexity, PCCC performs better at BER$> 10^{-5}$ region, while SCCC performs better at BER$< 10^{-6}$ region.

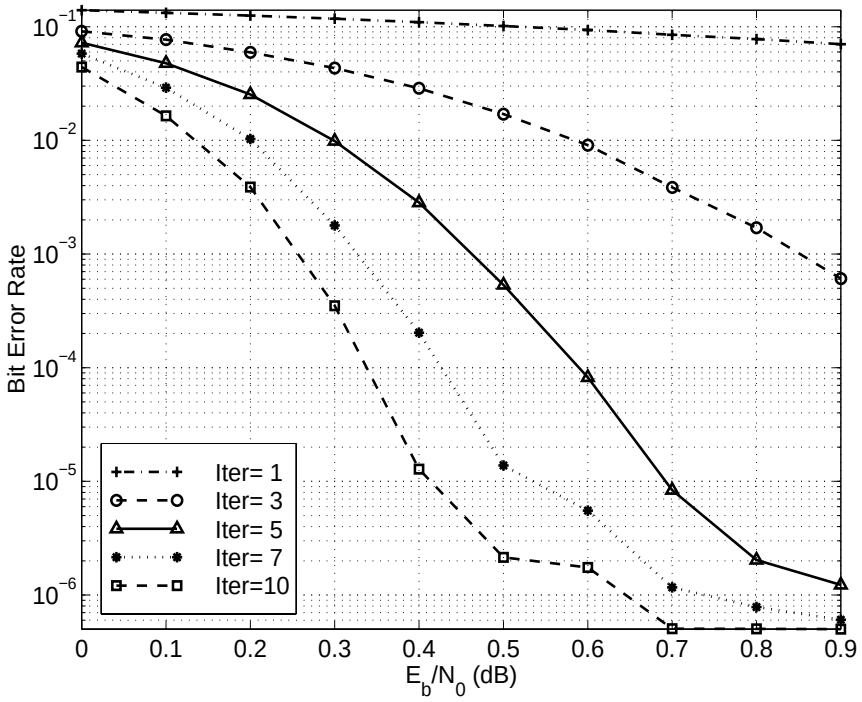


Figure 5.66: BER vs. E_b/N_0 as parameterized by number of decoding iterations (PCCC, $g = (15, 17)_{octal}$, rate 1/3, $N = 5120$).

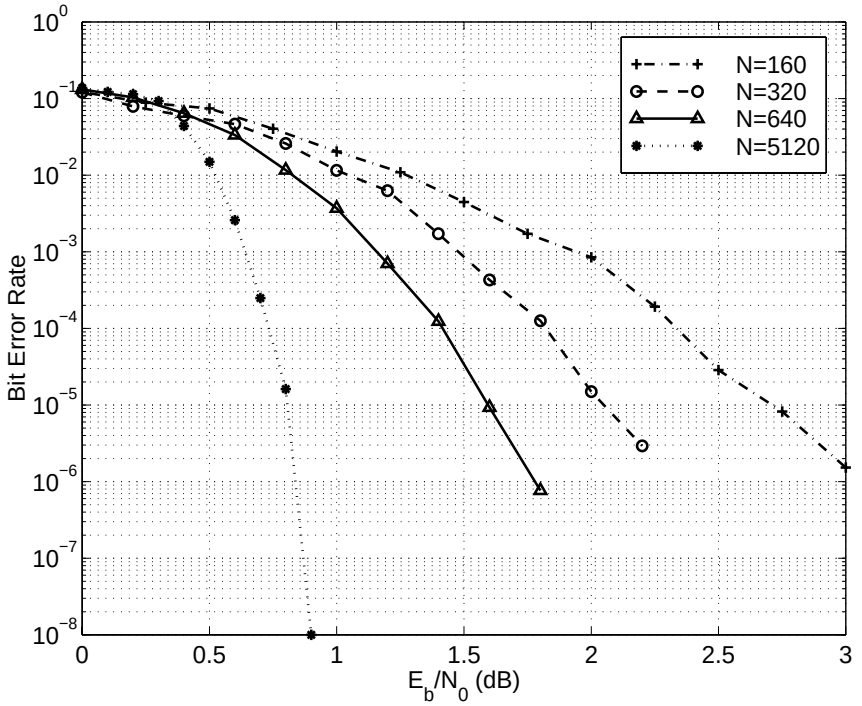


Figure 5.67: BER vs. E_b/N_0 as parameterized by frame size N (SCCC, $g = (7, 5)_{octal}$, rate 1/3 ($r_o = 1/2, r_i = 2/3$), 10 iterations, N=160, 320, 640, 5120).

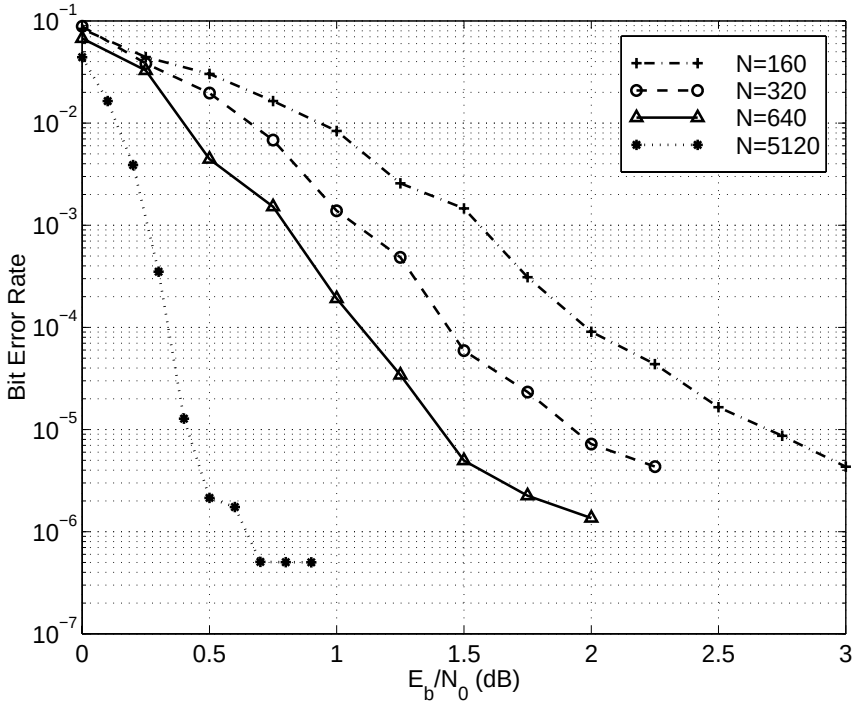


Figure 5.68: BER vs. E_b/N_0 as parameterized by frame size N (PCCC, $g = (15, 17)_{octal}$, rate 1/3, 10 iterations, N=160, 320, 640, 5120).

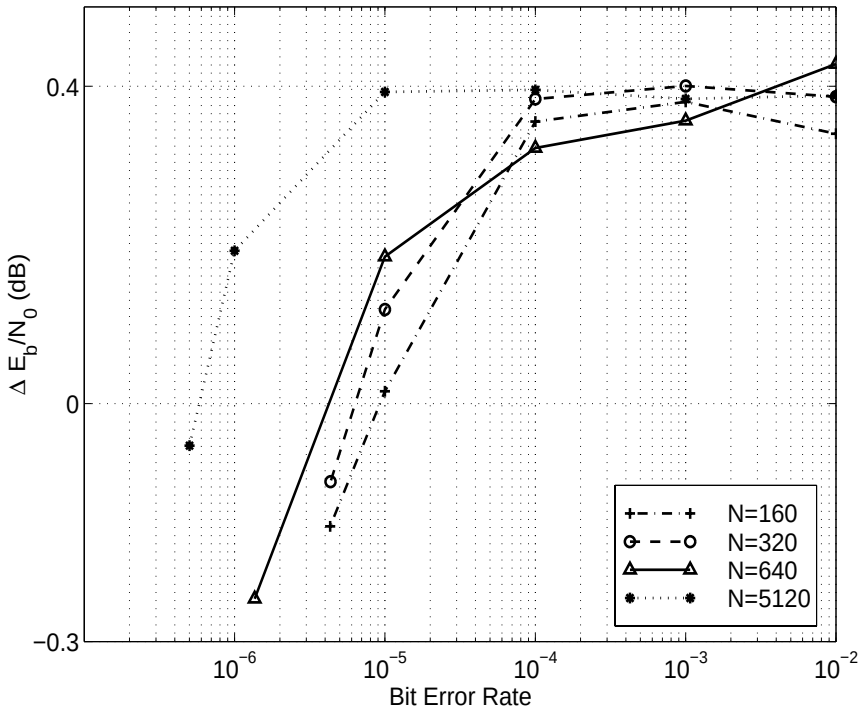


Figure 5.69: $\Delta E_b/N_0$ vs. BER as parameterized by frame size N (rate 1/3, 10 iterations, $\Delta E_b/N_0 = E_b/N_{0,\mathrm{SCCC}} - E_b/N_{0,\mathrm{PCCC}}$).

Chapter 6

Quantization Influence on Decoding Performance

6.1 Introduction

In the previous chapter, the superior performance of turbo codes has been 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. The amplitude of both signals and coefficients in the decoder is discrete. While the self-scaling of floating point arithmetic largely eliminates the quantization and rounding noise problem, it is an important issue for fixed point arithmetic. This section focuses on the problem of quantized input to the turbo decoder and the associated fixed point arithmetic.

Fixed point arithmetic and quantization 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 eliminated or 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, we investigated the dynamic range adjustment, and the influence of quantization and fixed point arithmetic for the implementation of the turbo decoder [57]. Two typical decoding algorithms were experimented with: Log-MAP

and SOVA.

6.2 System Model

In our study, we employ 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 alternatively punctured [27] to increase the overall code rate to 1/2 before BPSK modulation.

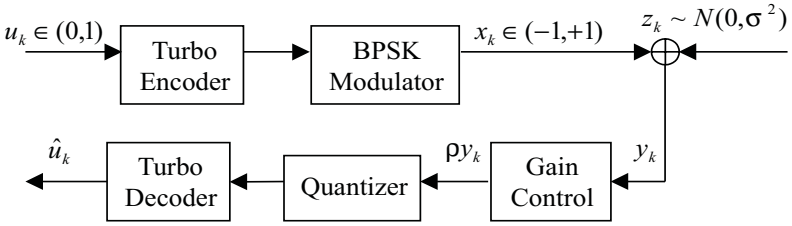


Figure 6.70: Turbo coding system model with quantization.

The coded bits are BPSK modulated into antipodal signals $(-1, +1)$ and transmitted through the channel. AWGN noise $z_k \sim N(0, \sigma^2)$ is added to the transmitted signal by the channel. After the BPSK demodulation, the received bits are scaled by a factor ρ 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 $(-2^{n-1}, 2^{n-1} - 1)$, where n is the number of bits of resolution of the quantizer [58] [59]. 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 evenly spaced bins. These bins are numbered between -2^{n-1} and $2^{n-1} - 1$, inclusive. The bin width is $\delta = (V_{hi} - V_{low})/2^n$, and the bin boundaries are at

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

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.

The phenomenon modeled here is 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 usually higher than the number of quantization bits. All the intermediate results are integers which have length less than 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 3.65 [51].

$$\begin{aligned} \max^*(\delta_1, \delta_2) &= \ln(e^{\delta_1} + e^{\delta_2}) \\ &= \max(\delta_1, \delta_2) + f_c(|\delta_1 - \delta_2|) \end{aligned} \tag{6.178}$$

where $f_c(x) = \ln(1 + e^{-x})$ is a nonlinear correction function. All the other computations with Log-MAP 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 δ_1 and δ_2 in the Log-MAP algorithm can be thought of as having been scaled from their floating point counterparts by a factor of approximately $2^n/(V_{hi} - V_{low})$. In our simulation, the floating point $f_c(\cdot)$ was implemented using the fixed point counterpart $f_c'(\cdot)$. This necessitated the quantity $|\delta_1 - \delta_2|$ to be first scaled to $|\delta_1' - \delta_2'|$ as follows:

$$|\delta_1' - \delta_2'| = \frac{|\delta_1 - \delta_2|(V_{hi} - V_{low})}{2^n \rho}, \tag{6.179}$$

where ρ is the scaling factor. The look-up table for $f_c'(\cdot)$ was constructed by computing the nonlinear correction function using the value of $|\delta_1' - \delta_2'|$ as follows:

$$f_c'(|\delta_1 - \delta_2|) = \left[\frac{2^n \rho \, f_c(|\delta_1' - \delta_2'|)}{V_{hi} - V_{low}} \right], \tag{6.180}$$

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 in order to accommodate the employment of the fixed point quantities.

6.3 Optimal Gain

For the system in Figure 6.70, to achieve the best signal-to-distortion ratio (SDR) 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 would minimize the distortion, or maximize SDR, 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

$$p_x(x) = 0.5[\delta(x+1) + \delta(x-1)] \tag{6.181}$$

The Gaussian noise z has zero mean, σ^2 variance and pdf

$$p_z(z) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{z^2}{2\sigma^2}\right) \tag{6.182}$$

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:

$$\begin{aligned} p_y(y) &= \int_{-\infty}^{\infty} p_z(y-t)p_x(t)dt \\ &= \frac{1}{2\sqrt{2\pi}\sigma}\left(\exp\left(-\frac{(y-1)^2}{2\sigma^2}\right) + \exp\left(-\frac{(y+1)^2}{2\sigma^2}\right)\right) \end{aligned} \tag{6.183}$$

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

$$\begin{aligned} p_\nu(\nu) &= \frac{1}{|\rho|}p_y\left(\frac{\nu}{\rho}\right) \qquad (\rho > 0) \\ &= \frac{1}{2\sqrt{2\pi}\rho\sigma}\left(\exp\left(-\frac{(\nu-\rho)^2}{2\rho^2\sigma^2}\right) + \exp\left(-\frac{(\nu+\rho)^2}{2\rho^2\sigma^2}\right)\right) \end{aligned} \tag{6.184}$$

Or, ν'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 that the quantization boundaries are (ν_{k-1}, ν_k), where $k = 1, ..., L$, and $L = 2^n$ is the number of quantization levels. The distortion function is thus:

$$D = \sum_{k=1}^{L}\int_{\nu_{k-1}}^{\nu_k}(\nu - \tilde{\nu}_k)^2 p_\nu(\nu)d\nu = A + \sum_{k=1}^{L}\left[-2\tilde{\nu}_k B_k + \tilde{\nu}_k^2 C_k\right] \tag{6.185}$$

where $A = \int_{-\infty}^{\infty}\nu^2 p_\nu(\nu)d\nu$, $B_k = \int_{\nu_{k-1}}^{\nu_k}\nu p_\nu(\nu)d\nu$, $C_k = \int_{\nu_{k-1}}^{\nu_k} p_\nu(\nu)d\nu$, and $p_\nu(\nu)$ is as in (6.184). The following relations are derived, where $Q(\cdot)$ is the commonly used Q function as in Equation 2.28:

$$\begin{aligned} A &= \rho^2(\sigma^2+1) \\ B_k &= \frac{\rho\sigma}{2\sqrt{2\pi}}\left[\exp\left(\frac{-(\nu_{k-1}-\rho)^2}{2\rho^2\sigma^2}\right) - \exp\left(\frac{-(\nu_k-\rho)^2}{2\rho^2\sigma^2}\right)\right. \\ &\quad \left. + \exp\left(\frac{-(\nu_{k-1}+\rho)^2}{2\rho^2\sigma^2}\right) - \exp\left(\frac{-(\nu_k+\rho)^2}{2\rho^2\sigma^2}\right)\right] \\ &\quad + \frac{\rho}{2}\left[Q\left(\frac{\nu_{k-1}-\rho}{\rho\sigma}\right) - Q\left(\frac{\nu_k-\rho}{\rho\sigma}\right) - Q\left(\frac{\nu_{k-1}+\rho}{\rho\sigma}\right) + Q\left(\frac{\nu_k+\rho}{\rho\sigma}\right)\right] \end{aligned} \tag{6.186}$$

$$C_k = \frac{1}{2}\left[Q\left(\frac{\nu_{k-1}-\rho}{\rho\sigma}\right) - Q\left(\frac{\nu_k-\rho}{\rho\sigma}\right) + Q\left(\frac{\nu_{k-1}+\rho}{\rho\sigma}\right) - Q\left(\frac{\nu_k+\rho}{\rho\sigma}\right)\right] \quad (6.187)(6.188)$$

From the above equation it is easy to see that the signal power is equal to A:

$$P = \int_{-\infty}^{\infty} \nu^2 p_\nu(\nu) d\nu = A = \rho^2(\sigma^2+1) \quad (6.189)$$

And SDR is:

$$\frac{P}{D} = \frac{A}{A + \sum_{k=1}^{L}\left[-2\tilde{\nu}_k B_k + \tilde{w}_k^2 C_k\right]} \quad (6.190)$$

To find the optimal scaling factor, first the curve of SDR versus ρ is plotted. Then the optimal ρ is found to be the value corresponding to the maximum SDR.

The SDR plot for a four-bit, range $(-1,1)$ quantizer is shown in Figure 6.71. The curves of the optimal gain when the quantizer range is $(-1,1)$ are plotted in Figure 6.72. Let the optimal gain for quantizer range $(-1,1)$ be ρ_1 at a certain E_b/N_0. When the dynamic range of the quantizer is $(-v,v)$, the optimal gain is $\rho_v = v\rho_1$.

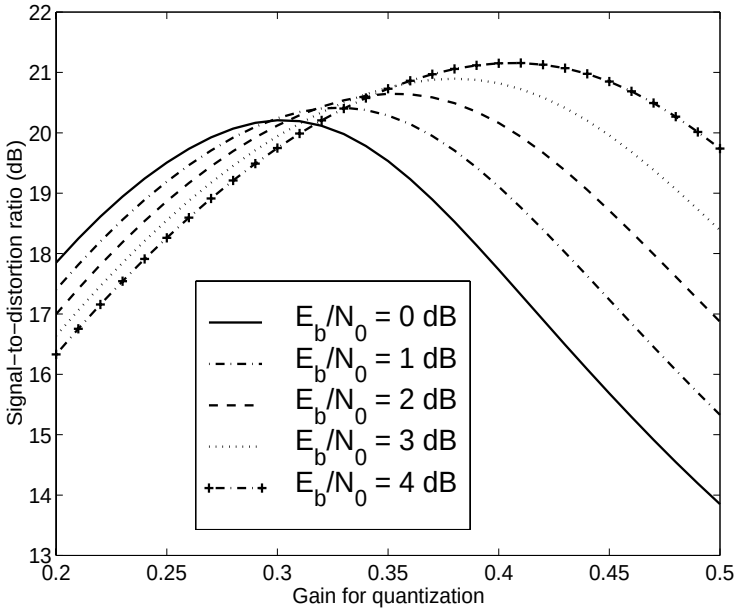


Figure 6.71: SDR for a four-bit, range $(-1,1)$ quantizer.

The following conclusions can be made about the scaling factor for an n-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 SDR.

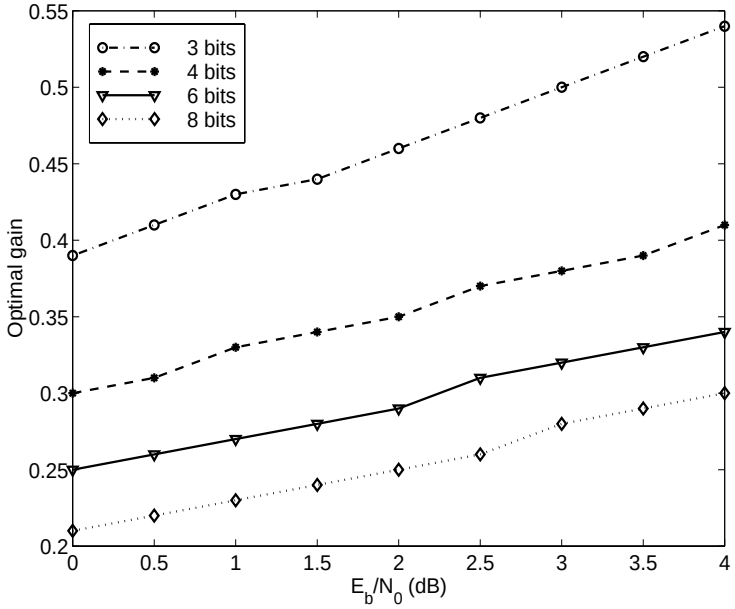


Figure 6.72: Optimal scaling factor for n-bit, range $(-1, 1)$ quantizer.

- Figure 6.71 shows that larger values of n allow for higher SDR. However, with proper scaling, a lower resolution quantizer can yield a higher SDR than that of a higher resolution quantizer, if the input of the latter is not properly scaled.
- It can be seen from Figure 6.72 that, for the given E_b/N_0 range and a given n, the optimal gain increases approximately linearly with the E_b/N_0 (dB) of the signal at the input of the quantizer.
- Figure 6.72 also shows that for a given E_b/N_0, a higher resolution (larger n) quantizer needs a smaller gain to maximize SDR.

6.4 Simulation Results

The noise affecting the fixed point arithmetic decoding procedure is composed of two parts: the quantization error of the quantizer at the input 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 is used to study the effect.

In the simulations, three different quantizer resolutions, $n = 3, 4$ 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 6.3 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 6.73 through 6.76, the BER curves are plotted versus E_b/N_0. Figures 6.73 and 6.74 are for a quantizer of range $(-0.5, 0.5)$. Figures 6.75 and 6.76 are for a quantizer of range $(-8, 8)$.

The following observations are made based on the 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 that all curves for the optimally scaled cases tend to converge at lower E_b/N_0.

- As expected, higher n provides better performance for both decoding algorithms when the distribution of the signal is fixed. The most significant performance improvement was observed when the main fixed point error was 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 since 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 exhibit marked inability to correct errors. However, the performance of SOVA did improve by increasing either n or E_b/N_0, in contrast to the Log-MAP algorithm. This result implies that the SOVA algorithm is more computationally stable.

- 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 in which the received signal is optimally scaled, in the region of $E_b/N_0 > 1$

dB, 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.

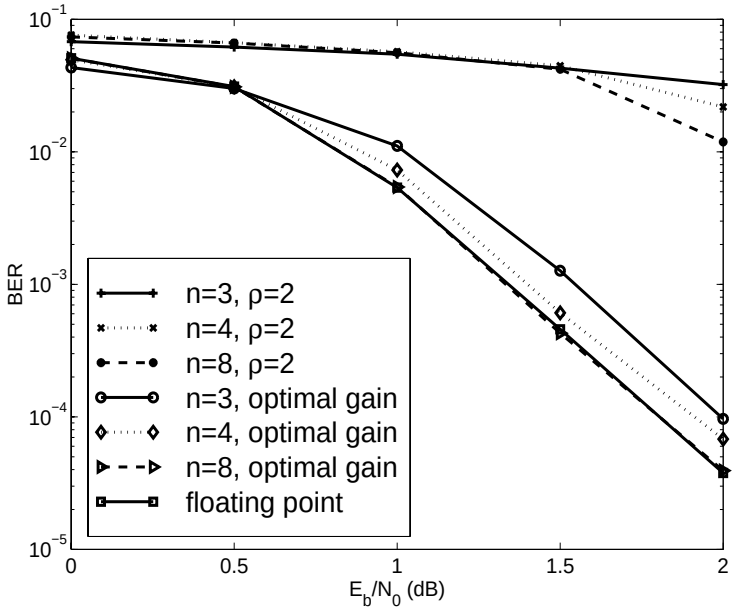


Figure 6.73: BER using Log-MAP for various resolution quantizers with range $(-0.5, 0.5)$.

6.5 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 factor by which to scale the received signal so as to suit it to the full scale range and resolution of the quantizer. Two typical decoding algorithms, Log-MAP and SOVA, were tested. The following points were shown:

- Scaling the received signal by an optimal gain leads to excellent turbo decoder performance, even with very few bits ($n = 4$).
- SOVA involves no extra computation with fixed point arithmetic because of its linearity, and it is also more computationally stable. 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.

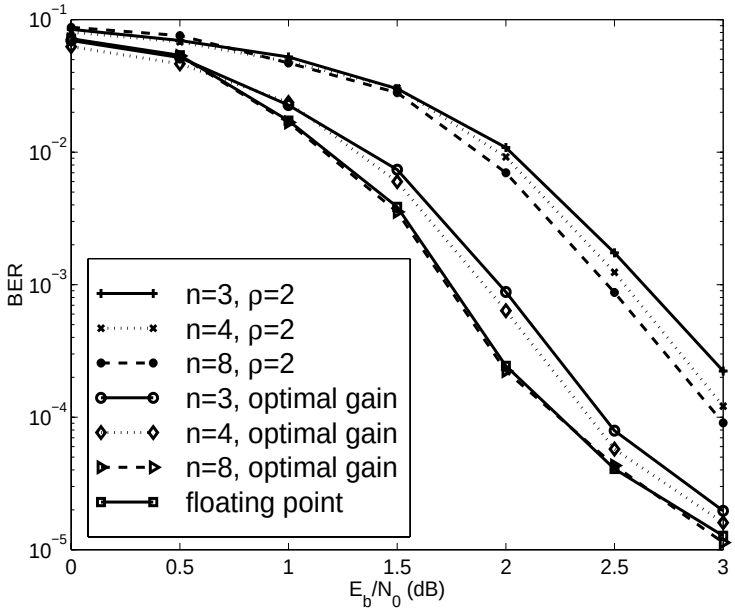


Figure 6.74: BER using SOVA for various resolution quantizers with range $(-0.5, 0.5)$.

Further investigations include finding the optimal gain for the Rayleigh fading channel, investigating nonuniform quantization, and estimating the channel characteristics.

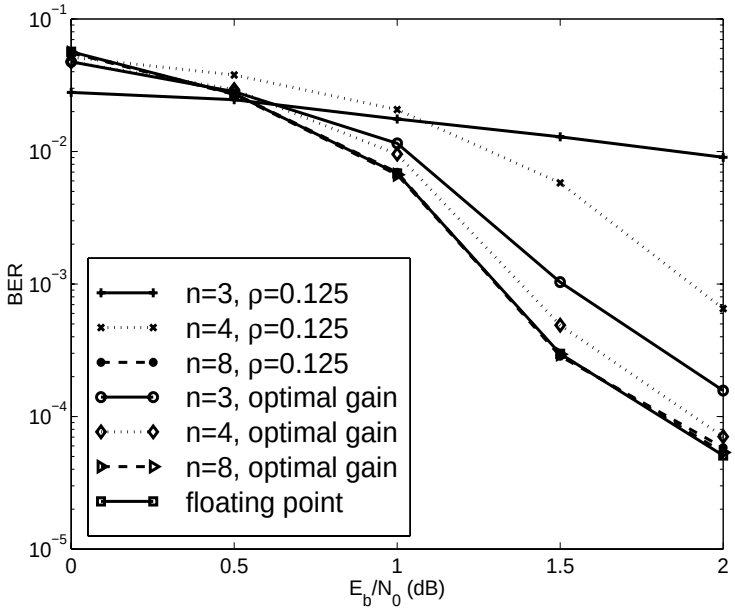


Figure 6.75: BER using Log-MAP for various resolution quantizers with range $(-8, 8)$.

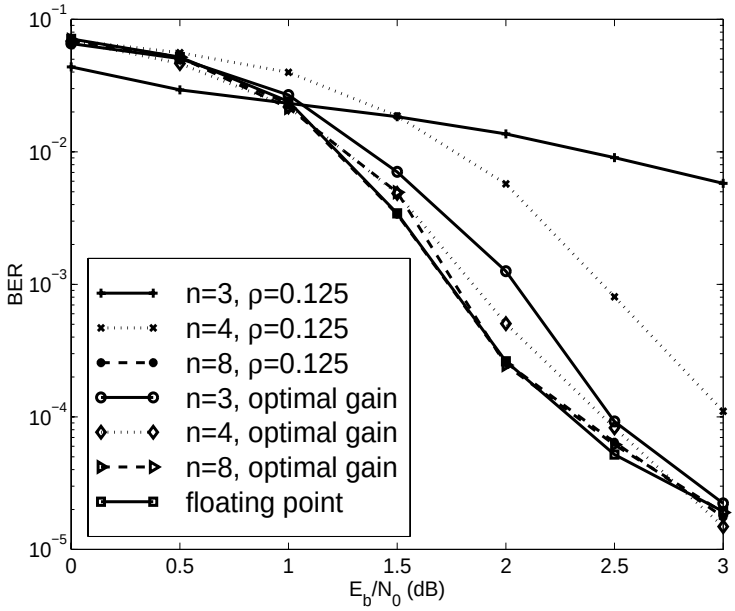


Figure 6.76: BER using SOVA for various resolution quantizers with range $(-8, 8)$.

Chapter 7

Contributions and Future Work

7.1 Contributions

This report documents the research that has been done to date. In summary, the following contributions have been made:

1. A thorough survey was done concerning turbo codes, serial concatenated codes, and other techniques that incorporate the turbo principle. The relevant techniques include turbo equalization, turbo trellis coded modulation (TTCM), source-controlled channel decoding, and multiuser detection with turbo feedback.

2. A study was carried out for the optimal and suboptimal decoding algorithms, including MAP, Log-MAP, max-Log-MAP, SOVA and the general SISO algorithm. Performance, latency, complexity, and storage requirements are compared. Chapter 3 includes a summary of the algorithm research.

3. After Log-MAP had been selected for the GloMo project, an effort was invested to simplify the computation of the algorithm. The butterfly structure of the RSC code was exploited to speed up calculation. The major results are summarized in Chapter 3. The resulting algorithm presented is a combination of our original research and other researchers' similar schemes.

4. A new method was proposed to compute the backward metrics of MAP decoder in the forward direction so as to reduce storage requirements. It is described in detail in Section 3.7 and [54].

5. Practical SPW models were constructed for both PCCC and SCCC systems to aid hardware implementation. The detailed description is presented in Chapter 4.

6. Simulations were performed comprehensively to evaluate the influence of the parameters and to compare the performance of the PCCC and SCCC. Chapter 5 demonstrates the results.

7. A study of the quantization effect on decoding algorithms are done. The scaling factor of the received signal was discovered to minimize the error. The results can be found in Chapter 6 and [57].

7.2 Future Work

7.2.1 Further Simplification of SISO for Implementation

A study will be carried out for a simple, practical way to implement the max*(·) function in Equation 3.65 without severely degrading the performance. This investigation is meaningful since the max*(·) function has to be performed as many as 2^{m+2} times for each information bit at each constituent decoder in one iteration.

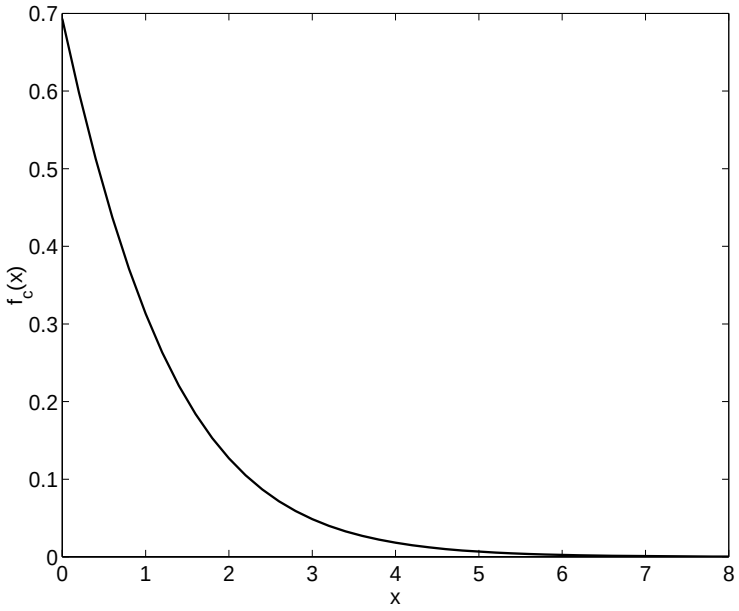


Figure 7.77: Correction function $f_c(x)$ in max*(·).

Since max(·) is a simple operation for hardware, the challenge lies in the nonlinear function $f_c(x)$ in Equation 3.65, where $x > 0$. The original proposal was to use a look-up table in place of $f_c(x)$, which requires a large number of visits to the memory. If $f_c(x)$ can be realized with some simple calculation, the hardware will run faster. Check the curve of $f_c(x)$ vs. x in Figure 7.77, we see that $f_c(x)$ decreases monotonically as x increases. When

$x \geq 5$, $f_c(x)$ is approximately 0. Thus a proper approximation only needs to be developed for the part of $f_c(x)$ where $0 < x < 5$, while simply setting $f_c(x) \equiv 0$, $\forall x > 5$.

Combine the approximation of $f_c(x)$ with the simplified SISO structure explained in Chapter 3, we believe that complexity is reduced to a reasonable level for the fast implementation of the maximum *a posteriori* decoding algorithm.

7.2.2 Turbo Synchronization

Much of the communications research over the past several years has focused on signal processing techniques for improved link level signal performance. New techniques in the areas of equalization, turbo coding, multiuser detection and adaptive antennas now allow reliable communications at extremely low signal-to-noise ratio (SNR) and signal-to-interference ratio (SIR). Most of the research on these techniques has assumed steady-state operation in which the receiver has already acquired synchronization with the desired signal. However, by reducing the SNR and SIR, these techniques make synchronization much more difficult to acquire.

This problem is particularly acute in packet communications systems because bursty packet operation necessitates rapid acquisition of frame synchronization. Most current frame synchronization techniques are based on the transmission of a recognizable pilot sequence in the header of the packet. Unfortunately, since these pilot sequences are uncoded, they are much less robust to noise and interference than data bits which may be protected by powerful error correction coding techniques.

For this thesis, we will investigate the performance of new "soft" frame synchronization techniques in which the pilot bits for synchronization are inserted into the data sequence and then encoded along with data bits. As a result of this technique, the relative delay of a data frame can be determined as part of the decoding process. We believe this technique will substantially increase the reliability of the synchronization process, enabling synchronization to take place at low SNR and SIR. Furthermore, the soft-input soft-output decoding techniques will allow the synchronization process to produce an estimate of the reliability of the synchronization. These estimates may be used by subsequent processing stages to refine the frame synchronization estimate. In much the same fashion as iterative turbo decoders operate, it should be possible to develop "turbo synchronization" techniques that allow robust synchronization of packet-oriented wireless communications.

7.2.3 Termination of decoding iteration

It is the nature of a turbo decoder to iteratively process the received signal. The decrease of BER slows down as the number of iteration increases. After a certain number of iterations, the decoder will gradually converge, where the hard decision remains static although the absolute value of the soft output keeps growing. Since the computation complexity and delay are proportional to the number of iterations, it is important to perform the proper number of iterations before terminating the decoding process.

Simulations have shown that:

- The larger the frame size, the more iterations are necessary before the decoder converges.
- The higher the code rate, the fewer iterations need to be performed.
- More iterations bring very little gain in the BER$> 10^{-2}$ region. Four iterations are usually enough. At the waterfall region where BER drops abruptly, the decoder will not converge until more iterations are done. A rough rule from our experience is that, for $N \leq 1000$, 8 iterations are a good choice; for $1000 < N \leq 10000$, 14 iterations are proper; for $N > 10000$, e.g., $N = 65536$, further iterations continue to lower BER up to 18 iterations [27].

A suitable number of iterations can be chosen beforehand based on the code parameters and previous experience. The advantage of a fixed number of iterations is that both the delay and the computation complexity remain constant from frame to frame. This is convenient for hardware implementation. The disadvantages are the waste of the computation power to perform unnecessary iterations for the "good" frames and lack of sufficient iterations when a frame is severely contaminated.

There are several ways to terminate the iteration individually for each frame. Four options are listed below.

- Compute the relative entropy as a criterion for the change in the weighted soft decisions for successive iterations. The decoding will be aborted when the current iteration does not bring about a change in the relative entropy which lies above a threshold [60]. This is a very sophisticated method, involving an extra computation of the relative entropy and a comparison.
- Set a threshold for the average absolute value of the soft output. When the threshold is exceeded, the decoder will abort. Extra computation is required to figure out the

threshold for a particular SNR, to find out the average absolute LLR for each iteration, and to compare at the end of each iteration.

- Make hard decisions at the output of each constituent decoder. Terminate when a vector of estimation is the same as the previous one. This method is called the "$n = n + \frac{1}{2}$ rule". It involves only a minimum excess computation to make the hard decisions by checking the sign of the soft output and to compare the current hard decisions with the previous ones.

- Make hard decisions at the end of each iteration. Abort decoding when a vector of estimation remains unchanged in comparison to the previous one. This method is called the "n=n+1" rule, and is analogous to the "$n = n + \frac{1}{2}$" rule.

We are convinced by simulations that a variable number of iteration can reduce the average number of iterations by as much as 50% with almost no loss in the performance. This means 50% of computation reduction in the decoder. The benefit is even more pronounced when the signal-to-noise ratio is high, where most frames can be decoded within a couple of iterations. However, the existing methods explained above are either too complicated or lack of reliability. Thus, we are motivated to search for a good terminating method.

Simulations have shown that SNR of the extrinsic information $\lambda(u;O)$, which is calculated as $\text{SNR}_{\lambda(u;O)} = \mu^2_{\lambda(u;O)}/\sigma^2_{\lambda(u;O)}$, increases rapidly as the iteration proceeds when the errors in a frame are corrected in one or two iterations. On the other hand, when decoding fails even when a large number of iterations have been performed, $\text{SNR}_{\lambda(u;O)}$ never grows large. This implies that $\text{SNR}_{\lambda(u;O)}$ can be a good criterion for terminating the decoder. In order to use it, we are challenged to decide the threshold of $\text{SNR}_{\lambda(u;O)}$.

Before determining the $\text{SNR}_{\lambda(u;O)}$ threshold for a given SNR of the received signal, we may start with finding the relationship between $\text{SNR}_{\lambda(u;O)}$ and the probability of error $P_b(e)$. Next, we can decide the relationship between SNR and $P_b(e)$ with the given code scheme by simulation or experience. With the above two relationships in hand, it is easy to establish the relationship between $\text{SNR}_{\lambda(u;O)}$ and SNR. For a turbo decoder operating on a given SNR, the corresponding $\text{SNR}_{\lambda(u;O)}$ can be used as a threshold to terminating the decoding iterations.

The difficulty of the above method lies in finding the relationship between $\text{SNR}_{\lambda(u;O)}$ and $P_b(e)$. We propose to use the transfer function of the code for the derivation.

Simulations have shown that when SNR is low, the decoder performance is not as sensitive to the threshold as when SNR is high. We anticipate that when SNR is high, the threshold should be set high and the decoder performs a couple of iterations on average

without performance degradation; when SNR is low, the threshold should be set low and the average number of decoding iterations should be kept small, with a small degradation of performance.

7.3 Timetable

Implementation of function $\max^*(\cdot)$

06/1999-07/1999	Analysis of the $f_c(x)$ function to derive the proper approximation.
07/1999-08/1999	Application of the approximation in the decoding algorithm. Simulation and comparison of the performance.

Turbo synchronization

08/1999-09/1999	Literature search and model development.
09/1999-10/1999	Derivation of the algorithm for iterative synchronization and decoding.
10/1999-12/1999	Simulation and validation of the method.

Termination of decoding iteration

12/1999-01/2000	Literature search for the up-to-date methods.
01/2000-02/2000	Theoretic analysis of the trend of soft output with respect to the iteration.
02/2000-03/2000	Development of the method for timing termination.
03/2000-04/2000	Comparison of complexity and performance among various strategies.

Summary

04/2000-05/2000	Documentation, dissertation defense and final edits.

Appendix A

Channel Capacity for 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 $y_i = x_i + z_i$. X is independent from Z, and $Z \sim N(0, N_0/2)$. The mutual information between the input and the output is

$$\begin{aligned} I(X,Y) &= h(Y) - h(Y|X) && \text{(A.191)} \\ &= h(Y) - h(X+Z|X) && \text{(A.192)} \\ &= h(Y) - h(Z) && \text{(A.193)} \end{aligned}$$

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

$$\begin{aligned} C &= \max_{p(x)} I(X,Y) && \text{(A.194)} \\ &= I(X,Y) && \text{(A.195)} \end{aligned}$$

since the pdf of X is fixed to be

$$p(x) = 0.5(\delta(x + \sqrt{E_s}) + \delta(x - \sqrt{E_s})). \quad \text{(A.196)}$$

The pdf of AWGN noise Z is

$$p(z) = \frac{1}{\sqrt{\pi N_0}} \exp\left(\frac{-z^2}{N_0}\right), \quad \text{(A.197)}$$

therefore the pdf of Y is:

$$p(y) = \frac{1}{2\sqrt{\pi N_0}} \left(\exp\left(\frac{-(y-\sqrt{E_s})^2}{N_0} \right) + \exp\left(\frac{-(y+\sqrt{E_s})^2}{N_0} \right) \right). \tag{A.198}$$

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 \\
&\quad - \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 \\
&\quad - \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
\end{aligned}$$

$$
\begin{aligned}
& +2\int \frac{1}{2\sqrt{\pi N_0}} \exp\left(\frac{-t^2}{N_0}\right) (\log_2 e) \left(\frac{2E_s}{N_0}\right) dt \\
= \quad & \frac{2E_s}{(\ln 2)N_0} - \frac{1}{\sqrt{2\pi}} \int \exp\left(\frac{-t^2}{2}\right) \cdot \log_2 \cosh\left(t\sqrt{\frac{2E_s}{N_0}} + \frac{2E_s}{N_0}\right) dt \\
= \quad & \frac{2rE_b}{(\ln 2)N_0} - \frac{1}{\sqrt{2\pi}} \int \exp\left(\frac{-t^2}{2}\right) \cdot \log_2 \cosh\left(t\sqrt{\frac{2rE_b}{N_0}} + \frac{2rE_b}{N_0}\right) dt
\end{aligned}
\tag{A.199}
$$

where r is the code rate and E_s is the energy of a symbol in a codeword. The following relationships are used:

$$E_s = rE_b \tag{A.200}$$

$$\cosh x = \frac{1}{2}\left(e^x + e^{-x}\right) \tag{A.201}$$

$$\int \exp\left(\frac{-t^2}{N_0}\right) \cdot t dt = 0 \tag{A.202}$$

$$\int \exp\left(\frac{-t^2}{N_0}\right) dt = \sqrt{\pi N_0} \tag{A.203}$$

$$\frac{1}{2}\left[1 + \exp\left(\frac{-4\sqrt{E_s}t - 4E_s}{N_0}\right)\right] = \exp\left(\frac{-2\sqrt{E_s}t - 2E_s}{N_0}\right) \cdot \cosh\left(\frac{2\sqrt{E_s}t + 2E_s}{N_0}\right) \tag{A.204}$$

$$\frac{1}{2}\left[1 + \exp\left(\frac{4\sqrt{E_s}t - 4E_s}{N_0}\right)\right] = \exp\left(\frac{2\sqrt{E_s}t - 2E_s}{N_0}\right) \cdot \cosh\left(\frac{2\sqrt{E_s}t - 2E_s}{N_0}\right) \tag{A.205}$$

$$\int \exp\left(\frac{-t^2}{N_0}\right) \cdot \log_2 \cosh\left(\frac{2\sqrt{E_s}t + 2E_s}{N_0}\right) dt = \int \exp\left(\frac{-t^2}{N_0}\right) \cdot \log_2 \cosh\left(\frac{2\sqrt{E_s}t - 2E_s}{N_0}\right) dt \tag{A.206}$$

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 Conv. Record*, vol. 4, pp. 37–47, 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. Thitimasjshima, "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] 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.

[43] S. Benedetto and G. Montorsi, "Design guidelines of parallel concatenated convolutional codes," in *Proc., IEEE GLOBECOM*, (Singapore), pp. 2273–2277, Nov. 1995.

[44] S. Benedetto and G. Montorsi, "Design of parallel concatenated convolutional codes," *IEEE Trans. Commun.*, vol. COM-44, pp. 591–600, May 1996.

[45] D. Divsalar and R. J. McEliece, "Effective free distance of turbo codes," *Electronics Letters*, vol. 32, pp. 445–446, Feb. 29th 1996.

[46] 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.

[47] J. Hagenauer, "Iterative decoding of binary block and convolutional codes," *IEEE Trans. Inform. Theory*, vol. IT-42, pp. 429–445, Mar. 1996.

[48] 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.

[49] 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.

[50] 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.

[51] 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.

[52] 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.

[53] 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.

[54] Y. Wu and B. D. Woerner, "Forward computation of backward path metrics for MAP decoder," in *Proc., IEEE GLOBECOM*, (Rio de Janeiro, Brazil), Dec. 1999. to appear.

[55] S. Benedetto, R. Garello, and G. Montorsi, "A search for good convolutional codes to be used in the construction of turbo codes," *IEEE Trans. Commun.*, vol. 46, pp. 1101–1105, Sept. 1998.

[56] 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.

[57] 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.*, (Huston, TX), May 1999.

[58] M. J. Demler, *High-Speed Analog-to-Digital Conversion*. San Diego: Academic Press, 1991.

[59] 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.

[60] J. Hagenauer and F. Burkert, "Method and arrangement for determining an adaptive abort criterion in iterative decoding of multi-dimensionally coded information." Patent US5761248, June 1998.

[61] S. A. Barbulescu, *Iterative decoding of turbo codes and other concatenated codes*. PhD thesis, Univ. South Australia, Feb. 1996.

[62] S. A. Butman and R. J. McEliece, "The ultimate limits of binary coding for a wideband Gaussian channel," *JPL TDA Progress Report 42-22*, pp. 78–80, Aug. 15, 1974.

[63] D. Divsalar and F. Pollara, "Turbo codes for PCS applications," in *Proc., IEEE Int. Conf. on Commun.*, pp. 54–59, May 1995.

[64] D. Divsalar, S. Dolinar, R. J. McEliece, and F. Pollara, "Performance analysis of turbo codes," in *Proc., IEEE MILCOM*, pp. 91–96, Nov. 1995.

[65] E. K. Hall and S. G. Wilson, "Turbo codes for noncoherent channels," in *Proc., IEEE GLOBECOM, Communication Theory Mini-Conference*, (Phoenix, AZ), pp. 66–70, Nov. 1997.

[66] P. Hoeher, "New iterative ("turbo") decoding algorithms," in *Proc., Int. Symp. on Turbo Codes and Related Topics*, (Brest, France), pp. 63–70, Sept. 1997.

[67] H. Koorapaty, Y. P. E. Wang, and K. Balachandran, "Performance of turbo codes with short frame sizes," in *Proc., IEEE Veh. Tech. Conf.*, pp. 329–333, 1997.

[68] M. Oberg, A. Vityaev, and P. H. Siegel, "The effect of puncturing in turbo encoders," in *Proc., Int. Symp. on Turbo Codes and Related Topics*, (Brest, France), pp. 184–187, Sept. 1997.

[69] M. Oberg and P. H. Siegel, "Lowering the error floror for turbo codes," in *Proc., Int. Symp. on Turbo Codes and Related Topics*, (Brest, France), pp. 204–207, Sept. 1997.

[70] S. S. Pietrobon, "Implementation and performance of a turbo/MAP decoder," *Int. J. Satell. Commun.,*, vol. 16, pp. 23–46, 1998.

[71] L. Papke, P. Robertson, and E. Villebrun, "Improved decoding with the SOVA in a parallel concatenated (turbo-code) scheme," in *Proc., IEEE Int. Conf. on Commun.*, pp. 102–106, 1996.

[72] 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.

[73] 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.

[74] P. Robertson, "Improving decoder and code structure of parallel concatenated recursive systematic (turbo) codes," in *Proc., IEEE Int. Conf. on Universal Personal Commun.*, pp. 183–187, 1994.

[75] P. Robertson and T. Wörz, "A novel bandwidth efficient coding scheme employing turbo codes," in *Proc., IEEE Int. Conf. on Commun.*, pp. 962–967, 1996.

[76] P. Robertson, "Illuminating the structure of parallel concatenated recursive systematic (turbo) codes," in *Proc., IEEE GLOBECOM*, pp. 1298–1303, 1994.

[77] S. Shamai and S. Verdu, "Capacity of channels with uncoded side information," *European Trans. on Telecommun.*, vol. 6, pp. 587–600, Sept./Oct. 1995.

[78] S. Shami, S. Verdu, and R. Zamir, "Information theoretic aspects of systematic coding," in *Proc., Int. Symp. on Turbo Codes and Related Topics*, (Brest, France), pp. 40–46, Sept. 1997.

[79] Y. V. Svirid, "Weight distributions and bounds for turbo-codes," *European Trans. on Telecommun.*, vol. 6, pp. 543–555, Sept./Oct. 1995.

[80] T. A. Summers and S. G. Wilson, "SNR mismatch and online estimation in turbo decoding," *IEEE Trans. Commun.*, vol. COM-46, pp. 421–423, April 1998.

[81] A. J. Viterbi, *CDMA: Principles of Spread Spectrum Communication.* Addison Wesley Longman, Inc., 1995.

[82] M. C. Valenti and B. D. Woerner, "Variable latency turbo codes for wireless multimedia applications," in *Proc., Int. Symp. on Turbo Codes and Related Topics*, (Brest, France), pp. 216–219, Sept. 1997.

[83] M. Valenti and B. Woerner, "Performance of turbo codes in interleaved flat fading channels with estimated channel state information," in *Proc., IEEE Veh. Tech. Conf.*, (Ottawa Canada), pp. 66–70, May 1998.

[84] 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.

[85] M. C. Jeruchim, P. Balaban, and K. S. Shanmugan, *Simulation of Communication Systems.* New York: Plenum Press, 1992.

[86] C. Schlegel, *Trellis Coding.* IEEE Press, 1997.

[87] R. A. Cameron, *Fixed-Point Implementation of a Multistage Receiver.* PhD thesis, Virginia Polytechnic Institute and State University, Jan. 1997.

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. During her stay at NPU, she published two papers on system identification in Chinese journals.

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.

Her primary research interests are in wireless communications, spread spectrum, and channel coding.