MULTI-CHANNEL PROCESSING OF TURBO CODES

Pouriya Sadeghi

A Thesis

in

The Department

of

Electrical and Computer Engineering

Presented in Partial Fulfillment of the Requirements

for the Degree of Master of Applied Science at

Concordia University

Montreal, Quebec, Canada

August 2003

Your file Votre référence
ISBN: 0-612-83875-7
Our file Notre référence
ISBN: 0-612-83875-7

ABSTRACT

MULTI-CHANNEL PROCESSING OF TURBO CODES

Pouriya Sadeghi

A turbo decoder uses an iterative procedure and its performance is dependent on the number of iterations it performs. In a turbo decoder, the number of iterations required to achieve a certain FER is a function of the channel SNR. An example of this is a base station in a mobile communication system, where turbo decoder receives data frames from different users each with a given level of fading and as a result, with different SNR. If a fixed number of iterations is allocated to each frame, it is possible that certain frames remain erroneous while decoder is idle part of the time for other frames. In this thesis the idea of sharing the processing power among different frames, possibly belonging to different channels, is used. This time-sharing approach can increase the processing throughput and/or the performance of the turbo decoding scheme. We present two schemes and give their performance for the 3GPP and DVB-RCS codes. The first one is serial processing algorithm, which decodes the input frames one by one and the other one is parallel processing scheme that tries to decode all of those received channels simultaneously.

We investigated the implementation of these schemes using a Digital Signal Processor (DSP) chip (TI TMS320C6416 DSP CPU). We have developed an efficient implementation algorithm by combining those two multi-channel processing schemes. All the presented schemes can also be used for a single-channel communication system.

ACKNOWLEDGMENTS

Firstly, I would like to express my sincere gratitude and thanks to my supervisor Dr. Mohammad Reza Soleymani and my co-supervisor Dr. Asim J. Al-Khalili for their generous help and constant support. I appreciate the way Dr. Soleymani devoted his precious time to discuss with me the details of my research, which was both encouraging and inspiring. His goal has always been in place to keep the research environment friendly and stress free.

My heartfelt thanks to my dearest mother, sister and brother, who supported me with their love and understanding. Special thanks to my beloved wife who has been involved in every aspect of this research, without her support none of this would have been possible.

I would like to thank Mr. Mohsen Ghotbi, Ms. Yingzi Gao, Ms. Shirin Esfandiari and all of my friends at Wireless and Communication Laboratory for their valuable advice and suggestions.

I also extend my thanks to the faculty and staff of Electrical and Computer Engineering department, and to my defense committee Dr. Shayan, Dr. Ghrayeb and Dr. Ganesan.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1

INTRODUCTION

More than fifty years ago Shannon discovered the limits on transmission rates in digital communication systems. In "A Mathematical Theory of Communication" [1], he defined the capacity of the Adaptive White Gaussian Noise (AWGN) channel as:

$$C = W \log_2\left(1 + \frac{S}{N}\right) \tag{1.1}$$

where W is the bandwidth of the channel in Hertz, S is the signal power and $N = WN_0$ is the variance of the Gaussian noise.

Shannon's channel coding theorems indicate that for an Additive White Gaussian Noise (AWGN) channel with capacity C, there exist codes that using them the information

can be sent across the channel at a rate less than C, with an arbitrarily low bit error rate [2] (for a proof refer to [3]).

For example, for a Binary Phase Shift Keying (BPSK) modulation system, which has a spectral efficiency of 1 bps/Hz, over an AWGN channel, the lower limit for Signal to Noise Ratio (SNR) to have an arbitrarily small Bit Error Rate (BER), is 0.0 dB. However, an uncoded BPSK system requires 9.5 dB [4] for a BER of 1×10^{-5}. So the maximum possible coding gain over an uncoded BPSK modulation scheme is $(9.5-0.0)=9.5$ dB.

The Consultative Committee for Space Data Systems (CCSDS) issued the NASA/ESA standard in 1983, which has a coding gain of 7.4 dB for BER of 1×10^{-5} [5]. This standard is still 2.1 dB away from the theoretical Shannon limit. There were too many efforts in 1980's and early 1990's to narrow this gap. There were just a few improvements of a few tenths of dB by utilizing serial concatenated systems, Reed Solomon code [6] and extremely complex Viterbi decoders [7].

In 1993 turbo coding was introduced. It provided a performance that was only a few tenths of dB away from the Shannon limit. Turbo codes used decoders much less complex than those used in NASA/ESA standard. The effect of this 2 dB extra coding gain, is even stronger in the wireless communication systems, where the bandwidth demand is growing very fast.

Turbo coding has two principle parts: parallel concatenated encoders and iterative decoding system. Parallel concatenated encoder, which can be designed for convolutional codes or block codes, works as follow: assume that there are two convolutional encoders (it is also possible to have more than two encoders). The first encoder encodes the data sequence (m) as (c_1) (Fig. 1.1). Next, the original data sequence (m) is interleaved by the

interleaver resulting in $\tilde{m}$ that is used as the input data sequence for the second encoder. The output of the second encoder (c_2), the output of the first encoder (c_1) and the original data sequence (m) are multiplexed, punctured and sent through an AWGN channel (the channel model that has been used in this thesis) to the turbo decoder receiver.

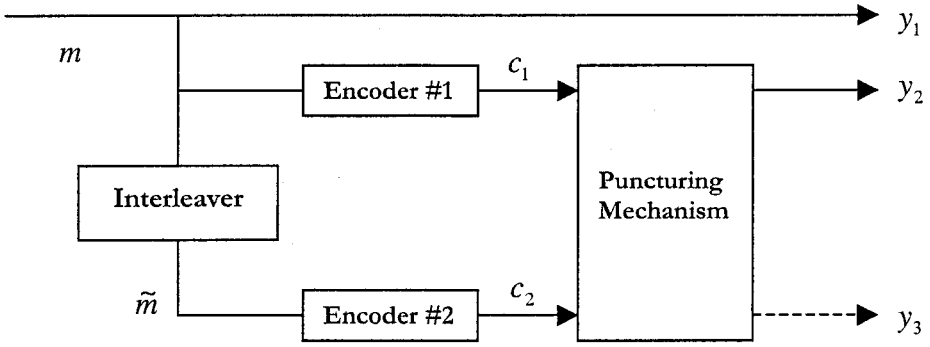

Fig. 1.1: Typical turbo encoder.

The excellent performance of turbo code is achieved at moderate decoding complexity. A turbo decoding system consists of two iterative Soft-Input/Soft-Output (SISO) decoders. The Soft-Input/Soft-Output property makes turbo coding perfect for iterative decoding. These two decoders are serially concatenated trough an interleaver identical to that used in the encoder side. In turbo coding system, the interleaver is one of the most important components: producing good weight codewords and decrease the correlation between the outputs of those two encoders. Producing good (high weight) codewords reduces the probability that the turbo decoder decodes the received sequence to a wrong codeword instead of the original one. Having less correlation between two output parity sequences is the main concept of the iterative decoding. It causes the soft outputs of the turbo decoder be relatively independent of each other, so the soft output of one component decoder can be used as the soft input of the other and vice versa. This means that for decoding a frame, the turbo decoder performs several iterations. For the first

iteration, it assumes that the probabilities of 0 and 1 are equal and based on the channel information, produces a soft decision output for each data bit. For other iterations, the decoder will use the soft output of the other decoder as a priori probability to perform the next iteration.

The performance of the turbo decoder is directly related to the number of the iterations that it performs to decode a received frame. If the turbo decoder can perform more iterations to decode each input frame, it can achieve a better BER. The number of iterations required to decode each input sequence can be assumed to be a random variable dependent on the variance of the AWGN. In traditional turbo coding schemes, the number of iterations is usually set to a fixed value, and the turbo decoder performs this fixed number of iterations to decode each input frame. Another option for the turbo decoder is to perform different number of iterations for each frame. In this case, to stop the decoding procedure on each frame, it is possible to use some rules and conditions to check whether the uncoded sequence is corrected or not. These rules and conditions are called "stopping criteria". There are several proposed stopping criteria so far. In [8], it has been shown that using some stopping criteria, will add, on the average, less than one iteration to the number of required iterations to correct each input channel (in comparison with the ideal case where the decoder knows when it has to stop).

In several parts of a communication system, we need to decode more than one channel simultaneously. These include the hub in a non-regenerative satellite communication system, the on-board processor in a regenerative satellite payload and the mobile base station where several channels are received from different terminals and the base station has to decode all of them at the same time. The ideal situation for the decoder is to have all information about all channels and the knowledge that how many iterations each frame needs to be corrected. So the decoder decodes first the channel that needs less

iteration than others, and so on. This ideal method is optimum, but in the practice, the turbo decoder does not know the original data sequence at the transmitter so it is not possible to determine whether decoded sequence is erroneous or not .So this is not a practical method and it is only for comparison and measuring the performance of other schemes.

In a traditional scheme, for each channel there is one decoder, and the procedure of decoding each channel is exactly the same as the decoding of a one-channel system (Fig. 1.2). For implementing a turbo code system, this traditional method is inefficient in practice, because in this method, each decoder has a fixed number of iterations, and if a frame needs more iterations than this limit to be decoded correctly, the decoder is not capable of correcting it. On the other hand, there are some other decoders that have decoded their assigned frames in less number of iterations than this limit, so they are idle at the same time. The basic idea in this thesis is to share these idle times for other sequences and have an efficient turbo decoder with a good throughput and performance.

In this thesis, we investigate mainly two multi-channel processing turbo decoding schemes in order to increase the performance and/or capacity of the turbo decoders. The presented schemes can be applied to any other kind of iterative decoder.

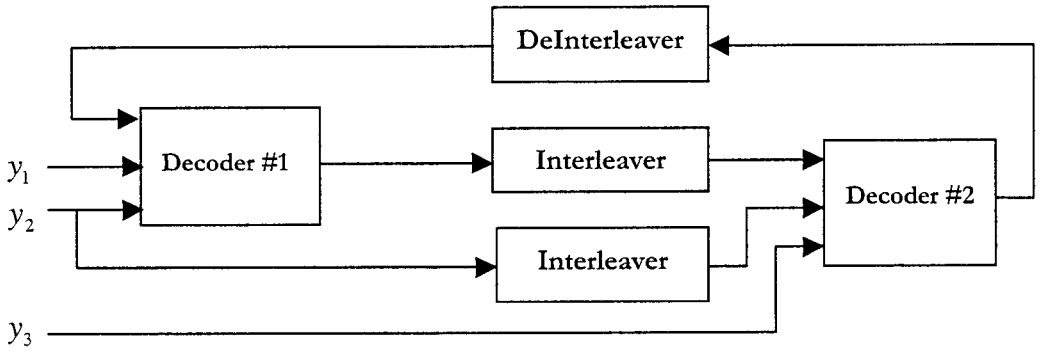

Fig. 1.2: Typical turbo decoder.

Chapter 2 deals with the subject of turbo decoding schemes and among these schemes; Bahl-Coke-Jelinek-Raviv (BCJR) algorithm is mentioned in more detail. The BCJR algorithm is being used widely in the iterative decoding systems. BCJR algorithm is an optimal algorithm over an AWGN channel and it has the minimum symbol error rate (or bit error rate). Next, two practical variants of this algorithm are described in detail (Log-MAP, Max-Log-MAP). Interleaver (permuter), which is one of the basic components of the turbo decoding, is also explained in Chapter 2. After that, the principles of iterative decoding are discussed. Then, the iterative nature of turbo decoder that is one of the main factors in performance of the turbo decoder is explained by means of some computer-based simulation examples. Chapter 2 ends with the introduction of the stopping criterion and describing several stopping criteria for turbo decoding.

Chapter 3 provides a detailed investigation of multi-channel turbo coding systems and proposes several multi-channel processing schemes for these systems. And at the end, it describes how to use these multi-channel processing schemes for a single-channel turbo decoding systems with some restrictions.

Chapter 4 begins with introducing Third Generation Partnership Project (3GPP) and Digital Video Broadcasting for Return Channel via Satellite (DVB/RCS) turbo coding standards. Then the results of computer-based simulations of 3GPP and DVB/RCS using AWGN channel and BCJR algorithm for several SNRs are presented. Based on the simulation results, the performance of each of the proposed parallel processing schemes in Chapter 3 are analyzed. It is shown that the performance of the proposed parallel processing scheme is close enough to the ideal case where prior to the decoding process, the turbo decoder knows how many iterations each input frame requires to be corrected, and it decodes the input frame that needs smaller number of iterations first.

In Chapter 5, implementation issues of the multi-channel processing schemes proposed in Chapter 3 on a Digital Signal Processing (DSP) chip, "TMS320C6416", for 3GPP turbo coding standard are investigated. This chip is one of the highest-performance DSP CPU so far, and has also two embedded coprocessor; Viterbi coprocessor and turbo decoder coprocessor (designed for 3GPP and CDMA2000 coding standards). The chapter closes with introducing an efficient multi-channel processing implementation algorithm, which can be considered as a combination of those proposed multi-channel processing schemes. This efficient multi-channel processing algorithm is not restricted to only this DSP CPU and can be used in any other hardware implementation of turbo codes.

Chapter 2

TURBO DECODING

A turbo decoder consists of two soft-input/soft-output (SISO) decoders concatenated serially via an interleaver (permuter) identical to the one used in turbo encoder [4]. BCJR (or MAP) decoder is an appropriate choice for those two decoders; because it is SISO compatible and also an optimal algorithm in terms of Symbol Error Rate (SER) or BER [9].

In this chapter, first, three different versions of BCJR algorithm are described in detail. Then, turbo code interleaver is discussed. Next, the structure of turbo decoder is explained and its most important property, the iterative nature, is mentioned in more detail. At the end of this chapter, some issues concerning parallel processing and early stopping criteria are described.

2.1 BCJR Algorithm

The Bahl, Cocke, Jelinek, and Raviv (BCJR) algorithm [10] which is also called forward-backward or *maximum a posteriori* (MAP) algorithm, is one of the turbo decoding algorithms that minimizes the SER or BER based on the information sequence received from an AWGN channel [11]. This algorithm is an optimal algorithm in terms of the SER (or BER).

Large number of calculations, complicated and non-linear mathematical functions and huge amount of memory requirement, make the implementation of the MAP algorithm in hardware or even in computer-based simulations tedious and time consuming. As a result, some variants of this algorithm such as Log-MAP algorithm and Max-Log-MAP algorithm are used in practice. The performance of the Log-MAP algorithm is the same as original MAP algorithm. Although Max-Log-MAP algorithm is a sub optimal algorithm, in most cases its performances is close enough to the original MAP algorithm, and it can be utilized instead of MAP decoder [12].

2.1.1 Maximum A Posteriori Probability (MAP) Algorithm

In this section, we use a $(n,1,v)$ binary convolutional code whose encoder has one bit input, n bits output and a v-bit shift register (v bits of memory). Therefore, the encoder has $M_s = 2^v$ different states. The state sequence from time 0 to t will be presented by:

$$\mathbf{S}_0^t = (S_0, S_1, ..., S_t) \tag{2.1}$$

It is seen that the next state is only dependent on current state and the next input bit. So, the state transition is a Markov process. The system block diagram is shown in Fig. 2.1.

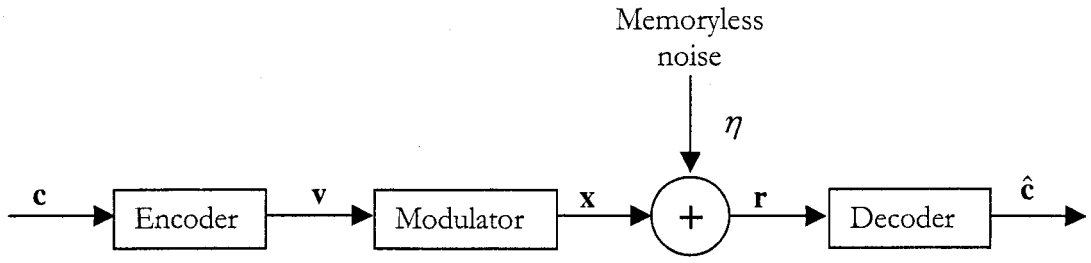

Fig. 2.1: Model of the communication system

In the following, each data sequence is defined in detail. The input binary message sequence is denoted by:

$$\mathbf{c} = (c_1, c_2, ..., c_t, ..., c_N) \tag{2.2}$$

where N is the length of the input sequence. The output sequence of the encoder is represented by:

$$\mathbf{v}_1^N = (\mathbf{v}_1, \mathbf{v}_2, ..., \mathbf{v}_N) \quad , \quad \mathbf{v}_t = (v_{t,0}, v_{t,1}, ..., v_{t,n-1}) \tag{2.3}$$

$\mathbf{v}_1^N$ is modulated by a BPSK modulator and the output sequence of it, is defined by:

$$\mathbf{x}_1^N = (\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_N) \quad , \quad \mathbf{x}_t = (x_{t,0}, x_{t,1}, ..., x_{t,n-1}) \tag{2.4}$$

$$x_{t,i} = 2v_{t,i} - 1 \quad , \quad i = 0,1,...,n-1$$

The output of the noisy channel is represented by:

$$\mathbf{r}_1^N = (\mathbf{r}_1, \mathbf{r}_2, ..., \mathbf{r}_N) \quad , \quad \mathbf{r}_t = (r_{t,0}, r_{t,1}, ..., r_{t,n-1}) \tag{2.5}$$

$$r_{t,i} = x_{t,i} + \eta_i \quad , \quad i = 0,1,...,n-1$$

where η_i is a zero-mean Gaussian random variable with variance σ^2. The channel is assumed to be an AWGN channel so each noise sample is independent from other noise samples.

The output of the decoder (estimated message sequence) is denoted by (we use "^" for representing the estimated variables):

$$\hat{\mathbf{c}} = (\hat{c}_1, \hat{c}_2, ..., \hat{c}_t, ..., \hat{c}_N) \tag{2.6}$$

In the ideal case $\hat{\mathbf{c}}$ must be identical to $\mathbf{c}$.

As a general rule, the output of the MAP decoder is in log-likelihood ratio format defined as:

$$\Lambda(c_t) = \log \frac{\Pr\{c_t = 1 | \mathbf{r}\}}{\Pr\{c_t = 0 | \mathbf{r}\}} \tag{2.7}$$

The above soft output can be converted into a hard decision output by using the following criterion:

$$c_t = \begin{cases} 1 & \text{if} \quad \Lambda(c_t) \geq 0 \\ 0 & \text{otherwise} \end{cases} \tag{2.8}$$

The MAP decoder starts decoding process from $S_0 = 0$ producing output sequence $\mathbf{x}_1^\tau$ and it ends in $S_\tau = 0$ by using trellis termination technique. It is obvious that τ must be equal to $N+v$.

The output sequence of the noisy channel is:

$$\mathbf{r}_1^{\tau} = (\mathbf{r}_1, \mathbf{r}_2, ..., \mathbf{r}_{\tau}) \tag{2.9}$$

For each received sequence $(\mathbf{r}_1^{\tau})$ using an AWGN we have:

$$\Pr\{\ \mathbf{r}_1^{\tau} \mid \mathbf{x}_1^{\tau}\ \} = \prod_{j=1}^{\tau} R\ (\mathbf{r}_j \mid \mathbf{x}_j) \tag{2.10}$$

where

$$R\ (\mathbf{r}_j \mid \mathbf{x}_j) = \prod_{i=0}^{n-1} \Pr\{\ r_{j,i} \mid x_{j,i}\ \}$$

and

$$\Pr\{\ r_{j,i} \mid x_{j,i} = -1\ \} = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(r_{j,i}+1)^2}{2\sigma^2}} \tag{2.11}$$

$$\Pr\{\ r_{j,i} \mid x_{j,i} = +1\ \} = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(r_{j,i}-1)^2}{2\sigma^2}} \tag{2.12}$$

Let c_t be the information bit input to the encoder at time t, associated to transition from S_{t-1} to S_t producing $\mathbf{v}_t$. The MAP decoder estimates c_t based on the received sequence $(\mathbf{r}_1^{\tau})$ as a soft output $(\Lambda(c_t))$:

$$\begin{aligned} \Pr\{\ c_t = 0 \mid \mathbf{r}_1^{\tau}\ \} &= \sum_{(l',l)\in B_t^0} \Pr\{\ S_{t-1} = l', S_t = l \mid \mathbf{r}_1^{\tau}\ \} \\ &= \sum_{(l',l)\in B_t^0} \frac{\Pr\{\ S_{t-1} = l', S_t = l, \mathbf{r}_1^{\tau}\ \}}{\Pr\{\ \mathbf{r}_1^{\tau}\ \}} \end{aligned} \tag{2.13}$$

where B_t^0 is the set of all possible transitions form S_{t-1} to S_t for which c_t is equal to 0.

$$\begin{aligned}\Pr\{\, c_t = 1 \mid \mathbf{r}_1^\tau \,\} &= \sum_{(l',l)\in B_t^1} \Pr\{\, S_{t-1} = l', S_t = l \mid \mathbf{r}_1^\tau \,\} \\ &= \sum_{(l',l)\in B_t^1} \frac{\Pr\{\, S_{t-1} = l', S_t = l, \mathbf{r}_1^\tau \,\}}{\Pr\{\, \mathbf{r}_1^\tau \,\}}\end{aligned} \tag{2.14}$$

where B_t^1 is the set of all possible transitions form S_{t-1} to S_t with $c_t = 1$.

It is possible to define the log-likelihood ratio based on the following joint probability function:

$$\rho_t(l',l) = \Pr\{\, S_{t-1} = l', S_t = l, \mathbf{r}_1^\tau \,\} \quad , \quad l', l = 0,1,\ldots,(M_s - 1) \tag{2.15}$$

$$\Pr\{\, c_t = 0 \mid \mathbf{r}_1^\tau \,\} = \sum_{(l',l)\in B_t^0} \frac{\rho_t(l',l)}{\Pr\{\, \mathbf{r}_1^\tau \,\}} \tag{2.16}$$

$$\Pr\{\, c_t = 1 \mid \mathbf{r}_1^\tau \,\} = \sum_{(l',l)\in B_t^1} \frac{\rho_t(l',l)}{\Pr\{\, \mathbf{r}_1^\tau \,\}} \tag{2.17}$$

and

$$\Lambda(c_t) = \log \frac{\sum_{(l',l)\in B_t^1} \rho_t(l',l)}{\sum_{(l',l)\in B_t^0} \rho_t(l',l)} \tag{2.18}$$

In order to calculate the joint probability function ($\rho_t(l',l)$), it is useful to define it as a function of 3 other probability functions as follows:

$$\alpha_t(l) = \Pr\{\, S_t = l, \mathbf{r}_1^t \,\} \quad , \quad \beta_t(l) = \Pr\{\, \mathbf{r}_{t+1}^\tau \mid S_t = l \,\} \tag{2.19}$$

$$\gamma_t^i(l',l) = \Pr\{\, c_t = i, S_t = l \mid S_{t-1} = l' \,\} \quad , \quad i = 0,1 \tag{2.20}$$

The joint probability can be rewritten as:

$$\begin{aligned}
\rho_t(l',l) &= \Pr\{S_{t-1}=l', S_t=l, \mathbf{r}_1^{\tau}\} \\
&= \Pr\{\mathbf{r}_{t+1}^{\tau}, \mathbf{r}_t, \mathbf{r}_1^{t-1}, S_{t-1}=l', S_t=l\} \\
&= \Pr\{\mathbf{r}_{t+1}^{\tau} \mid \mathbf{r}_t, \mathbf{r}_1^{t-1}, S_{t-1}=l', S_t=l\}.\Pr\{\mathbf{r}_t, \mathbf{r}_1^{t-1}, S_{t-1}=l', S_t=l\} \\
&= \Pr\{\mathbf{r}_{t+1}^{\tau} \mid S_t=l\}.\Pr\{\mathbf{r}_t, \mathbf{r}_1^{t-1}, S_{t-1}=l', S_t=l\} \\
&= \beta_t(l).\Pr\{\mathbf{r}_t, \mathbf{r}_1^{t-1}, S_{t-1}=l', S_t=l\} \\
&= \beta_t(l).\Pr\{S_t=l, \mathbf{r}_t \mid \mathbf{r}_1^{t-1}, S_{t-1}=l'\}.\Pr\{\mathbf{r}_1^{t-1}, S_{t-1}=l'\} \\
&= \Pr\{\mathbf{r}_1^{t-1}, S_{t-1}=l'\}.\beta_t(l).\Pr\{S_t=l, \mathbf{r}_t \mid S_{t-1}=l'\} \\
&= \alpha_{t-1}(l').\beta_t(l).\sum_{i\in(0,1)} \gamma_t^i(l',l)
\end{aligned} \tag{2.21}$$

so

$$\Lambda(c_t) = log \frac{\sum_{(l',l)\in B_t^1} \alpha_{t-1}(l').\beta_t(l).\gamma_t^1(l',l)}{\sum_{(l',l)\in B_t^0} \alpha_{t-1}(l').\beta_t(l).\gamma_t^0(l',l)} \tag{2.22}$$

$\alpha_t(l)$ can be calculated recursively using boundary conditions $\alpha_0(0)=1$ and $\alpha_0(l)=0$ for $l \neq 0$.

$$\begin{aligned}
\alpha_t(l) &= \Pr\{S_t=l, \mathbf{r}_1^t\} \\
&= \sum_{l'=0}^{M_s-1} \Pr\{S_{t-1}=l', S_t=l, \mathbf{r}_1^t\} \\
&= \sum_{l'=0}^{M_s-1} \Pr\{S_{t-1}=l', S_t=l, \mathbf{r}_1^{t-1}, \mathbf{r}_t\} \\
&= \sum_{l'=0}^{M_s-1} \Pr\{S_{t-1}=l', \mathbf{r}_1^{t-1}\}.\Pr\{\mathbf{r}_t, S_t=l \mid S_{t-1}=l', \mathbf{r}_1^{t-1}\} \\
&= \sum_{l'=0}^{M_s-1} \alpha_{t-1}(l').\Pr\{\mathbf{r}_t, S_t=l \mid S_{t-1}=l'\} \\
&= \sum_{l'=0}^{M_s-1} \alpha_{t-1}(l').\sum_{i\in(0,1)} \Pr\{\mathbf{r}_t, c_t=i, S_t=l \mid S_{t-1}=l'\} \\
&= \sum_{l'=0}^{M_s-1} \alpha_{t-1}(l').\sum_{i\in(0,1)} \gamma_t^i(l',l)
\end{aligned} \tag{2.23}$$

$\beta_t(l)$ can be calculated recursively using boundary conditions of $\beta_\tau(0)=1$ and $\beta_\tau(l)=0$ for $l\neq 0$.

$$
\begin{aligned}
\beta_t(l) &= \Pr\{\mathbf{r}_{t+1}^{\tau} \mid S_t = l\} \\
&= \sum_{l'=0}^{M_s-1} \Pr\{S_{t+1}=l', \mathbf{r}_{t+1}^{\tau} \mid S_t = l\} \\
&= \sum_{l'=0}^{M_s-1} \frac{\Pr\{S_{t+1}=l', \mathbf{r}_{t+1}^{\tau}, S_t = l\}}{\Pr\{S_t = l\}} \\
&= \sum_{l'=0}^{M_s-1} \frac{\Pr\{S_{t+1}=l', \mathbf{r}_{t+1}, \mathbf{r}_{t+2}^{\tau}, S_t = l\}}{\Pr\{S_t = l\}} \\
&= \sum_{l'=0}^{M_s-1} \frac{\Pr\{\mathbf{r}_{t+1} \mid S_{t+1}=l', \mathbf{r}_{t+2}^{\tau}, S_t = l\}.\Pr\{S_{t+1}=l', \mathbf{r}_{t+1}, S_t = l\}}{\Pr\{S_t = l\}} \\
&= \sum_{l'=0}^{M_s-1} \frac{\Pr\{\mathbf{r}_{t+1} \mid S_{t+1}=l'\}.\Pr\{S_{t+1}=l', \mathbf{r}_{t+1}, S_t = l\}}{\Pr\{S_t = l\}} \\
&= \sum_{l'=0}^{M_s-1} \frac{\beta_{t+1}(l').\Pr\{S_{t+1}=l', \mathbf{r}_{t+1} \mid S_t = l\}.\Pr\{S_t = l\}}{\Pr\{S_t = l\}} \\
&= \sum_{l'=0}^{M_s-1} \beta_{t+1}(l').\Pr\{S_{t+1}=l', \mathbf{r}_{t+1} \mid S_t = l\} \\
&= \sum_{l'=0}^{M_s-1} \beta_{t+1}(l'). \sum_{i\in(0,1)} \Pr\{c_{t+1}=i, S_{t+1}=l', \mathbf{r}_{t+1} \mid S_t = l\} \\
&= \sum_{l'=0}^{M_s-1} \beta_{t+1}(l'). \sum_{i\in(0,1)} \gamma_{t+1}^{i}(l', l)
\end{aligned}
\tag{2.24}
$$

also $\gamma_t^i(l', l)$ can be calculated as follows:

$$
\begin{aligned}
\gamma_t^i(l', l) &= \Pr\{S_t = l, \mathbf{r}_t, c_t = i \mid S_{t-1} = l'\} \\
&= \frac{\Pr\{S_t = l, \mathbf{r}_t, c_t = i, S_{t-1} = l'\}}{\Pr\{S_{t-1} = l'\}} \\
&= \frac{\Pr\{\mathbf{r}_t \mid S_t = l, c_t = i, S_{t-1} = l'\}.\Pr\{S_t = l, c_t = i, S_{t-1} = l'\}}{\Pr\{S_{t-1} = l'\}}
\end{aligned}
\tag{2.25}
$$

$$= \frac{\Pr\{\mathbf{r}_t \mid \mathbf{x}_t\}.\Pr\{S_t = l, c_t = i, S_{t-1} = l'\}}{\Pr\{S_{t-1} = l'\}}$$
$$= \frac{\Pr\{\mathbf{r}_t \mid \mathbf{x}_t\}.\Pr\{c_t = i \mid S_t = l, S_{t-1} = l'\}.\Pr\{S_t = l, S_{t-1} = l'\}}{\Pr\{S_{t-1} = l'\}}$$
$$= \Pr\{\mathbf{r}_t \mid \mathbf{x}_t\}.\Pr\{\mathbf{x}_t \mid S_t = l, S_{t-1} = l'\}.\Pr\{S_t = l \mid S_{t-1} = l'\}$$

$$\Pr\{\mathbf{r}_t \mid \mathbf{x}_t\} = \frac{1}{\sqrt{2\pi}\sigma}\exp\left(-\frac{\sum_{j=0}^{1}\left(r_{t,j}^i - x_{t,j}^i(l)\right)^2}{2\sigma^2}\right) \tag{2.26}$$

With further expression:

$$\gamma_t^i(l',l) = \begin{cases} p_t(i)\exp\left(-\dfrac{\sum_{j=0}^{1}\left(r_{t,j}^i - x_{t,j}^i(l)\right)^2}{2\sigma^2}\right) & \text{for } (l,l') \in B_t^i \\ 0 & \text{otherwise} \end{cases} \tag{2.27}$$

where $p_t(i)$ is a priori probability of $c_t = i$, B_t^i is a set of all transitions from $S_{t-1} = l'$ to $S_t = l$ with $c_t = i$ and $x_{t,j}^i(l)$ is the encoder output associated with transitions of B_t^i.

2.1.2 Max-Log-MAP Algorithm

The standard MAP algorithm needs huge amount of memory and also uses too many non-linear mathematical operations and multiplications. Max-Log-MAP algorithm is an approximation of MAP algorithm that is easier in implementation. However, this algorithm

is not an optimal algorithm as MAP algorithm (about 0.2dB difference in the performance for 3GPP turbo codes).

Max-Log-MAP Algorithm utilizes logarithm format of $\alpha_t(l)$, $\beta_t(l)$ and $\gamma_t^i(l',l)$ which are denoted as $\bar{\alpha}_t(l)$, $\bar{\beta}_t(l)$ and $\bar{\gamma}_t^i(l',l)$ respectively.

$$\bar{\gamma}_t^i(l',l) = \log \gamma_t^i(l',l) \tag{2.28}$$

$$\begin{aligned} \bar{\alpha}_t(l) &= \log \alpha_t(l) \\ &= \log \sum_{l'=0}^{M_s-1} \cdot \sum_{i\in(0,1)} e^{\bar{\alpha}_{t-1}(l')+\bar{\gamma}_t^i(l',l)} \end{aligned} \tag{2.29}$$

$$\begin{aligned} \bar{\beta}_t(l) &= \log \beta_t(l) \\ &= \log \sum_{l'=0}^{M_s-1} \cdot \sum_{i\in(0,1)} e^{\bar{\beta}_{t+1}(l')+\bar{\gamma}_{t+1}^i(l',l)} \end{aligned} \tag{2.30}$$

and boundary conditions become:

$$\bar{\alpha}_0(0) = 0 \quad \text{and} \quad \bar{\alpha}_0(l) = -\infty \qquad \text{for } l \neq 0 \tag{2.31}$$

$$\bar{\beta}_\tau(0) = 0 \quad \text{and} \quad \bar{\beta}_\tau(l) = -\infty \qquad \text{for } l \neq 0 \tag{2.32}$$

Using these new definitions, the log-likelihood ratio (LRR) can be rewritten as follows:

$$\Lambda(c_t) = \log \frac{\sum_{l=0}^{M_s-1} e^{\bar{\alpha}_{t-1}(l')+\bar{\gamma}_t^1(l',l)+\bar{\beta}_t(l)}}{\sum_{l=0}^{M_s-1} e^{\bar{\alpha}_{t-1}(l')+\bar{\gamma}_t^0(l',l)+\bar{\beta}_t(l)}} \tag{2.33}$$

This log-likelihood ratio can be simplified by using the following approximation:

$$\log(e^{\delta_1} + e^{\delta_2} + \ldots + e^{\delta_n}) \approx \max_{i\in\{1,2,\ldots,n\}} \delta_i \tag{2.34}$$

This maximum value can be calculated by applying *(n-1) max(x,y)* function recursively over each two values. So the LLR can be written as:

$$\Lambda(c_t) \approx \max_l [\overline{\alpha}_{t-1}(l') + \overline{\gamma}_t^1(l',l) + \overline{\beta}_t(l)] - \max_l [\overline{\alpha}_{t-1}(l') + \overline{\gamma}_t^0(l',l) + \overline{\beta}_t(l)] \quad (2.35)$$

where,

$$\overline{\alpha}_t(l') \approx \max_l \{ \overline{\alpha}_{t-1}(l') + \overline{\gamma}_t^i(l',l) \} \text{ for } 0 \le l' \le M_s - 1 \ , \ i = 0,1 \quad (2.36)$$

and

$$\overline{\beta}_t(l) \approx \max_l \{ \overline{\beta}_{t+1}(l') + \overline{\gamma}_{t+1}^i(l',l) \} \text{ for } 0 \le l' \le M_s - 1 \ , \ i = 0,1 \quad (2.37)$$

2.1.3 Max*-Log-MAP Algorithm

Because of using the *max(x,y)* approximation in calculating the maximum-likelihood ratio in the previous section, Max-Log-MAP algorithm is a sub optimal algorithm. It is possible to improve the performance of the decoder by using the Jacobian algorithm [13]:

$$\begin{aligned} \log(e^{\delta_1} + e^{\delta_2}) &= \max(\delta_1, \delta_2) + \log(1 + e^{-|\delta_1 - \delta_2|}) \\ &= \max(\delta_1, \delta_2) + f_c(|\delta_1 - \delta_2|) \end{aligned} \quad (2.38)$$

where $f_c(.)$ is correction function that can be implemented as a simple look-up table. The performance of this Log-MAP algorithm is dependent on the number of the levels of the look-up table, but on the other hand multi-level look-up table can reduce the speed of the decoder. In most cases using a 3-level or 4-level look-up table, the performance of the Log-Max*-MAP algorithm can be identical to the performance of the optimum MAP algorithm.

2.2 Interleaver (Permuter)

The function of the interleaver is to receive the N-bit sequence and rearrange it to construct a new N-bit sequence, which is applied to the input of the second encoder. Since the interleaver decorrelates the inputs of the two encoders [14], at the receiver side, after correcting some of the errors by means of parity bits of first encoder, we will be able to correct some remaining errors by using parity bits of the second encoder. There are two major points in choosing the interleaver; choosing an interleaver to obtain an output as uncorrelated as possible (with consideration of decoding constraints), and choosing the length of the output block code, N.

2.3 Iterative Decoding of Turbo Codes

Iterative decoding consists of two serially concatenated MAP-based decoders, separated by an interleaver identical to that one at the encoder (Fig. 2.2). The inputs of the first MAP decoder are the received noisy information sequence $\mathbf{r}_0$ and the parity sequence produced by the first encoder from the AWGN channel. The second MAP decoder uses the soft output sequence of the first MAP decoder as one of its inputs. Other inputs of the second decoder are an interleaved version of the received information sequence ($\tilde{\mathbf{r}}_0$) and the received encoded sequence produced by the second encoder. The second decoder also produces a soft output that will be used as one of the inputs of the first decoder, and so on. This feedback-loop is the distinguishing feature of turbo decoding algorithm. The name of "Turbo Code" is derived from the principle of the turbo engine that is using some kind of feedback of the engine output at its input.

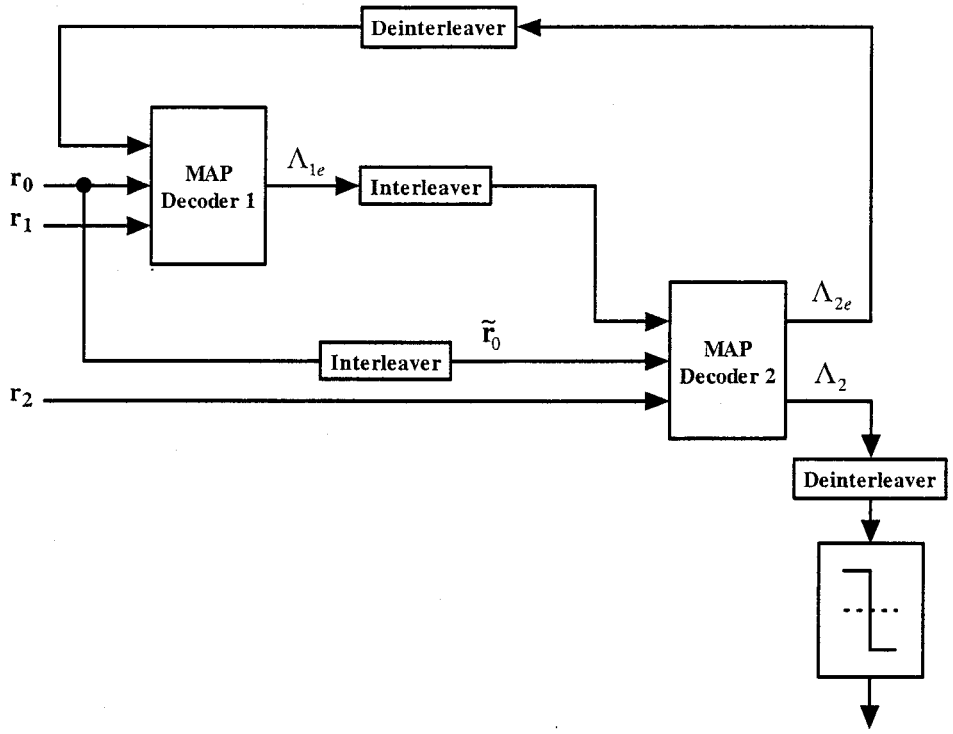

Fig. 2.2: An iterative turbo decoding system using MAP decoder

After performing a certain number of iterations, there will be no further improvement in decoding performance, so the turbo decoder stops the decoding process and makes a hard decision on the information bits based on the last iteration soft output sequence [15,16].

It is possible to write the soft output of each of those MAP decoders as follows:

$$\Lambda_1(c_t) = \log \frac{\sum_{l',l=0}^{M_s-1} \alpha_{t-1}(l').\beta_t(l).p_t^1(1)e^{\left(-\sum_{j=1}^{1}\left(r_{t,j}-x_{t,j}^1(l)\right)^2 \Big/ 2\sigma^2\right)}}{\sum_{l',l=0}^{M_s-1} \alpha_{t-1}(l').\beta_t(l).p_t^1(0)e^{\left(-\sum_{j=1}^{1}\left(r_{t,j}-x_{t,j}^0(l)\right)^2 \Big/ 2\sigma^2\right)}} \tag{2.39}$$

where $p_t^1(i)$ is the a priori probability for $c_t = i$. In the first iteration of the first decoder, it is reasonable to assume that $p_t^1(0) = p_t^1(1) = 1/2$. $\Lambda_1(c_t)$ can be written as:

$$\Lambda_1(c_t) = \log \frac{p_t^1(1)}{p_t^1(0)} + \log \frac{\sum_{l',l=0}^{M_s-1} \alpha_{t-1}(l').\beta_t(l).e^{\left(-\left((r_{t,0}-x_{t,0}^1(l))^2 + \sum_{j=1}^{1}(r_{t,j}-x_{t,j}^1(l))^2\right)/2\sigma^2\right)}}{\sum_{l',l=0}^{M_s-1} \alpha_{t-1}(l').\beta_t(l).e^{\left(-\left((r_{t,0}-x_{t,0}^0(l))^2 + \sum_{j=1}^{1}(r_{t,j}-x_{t,j}^0(l))^2\right)/2\sigma^2\right)}} \tag{2.40}$$

Since this code is systematic, $x_{t,0}^i$ is independent of trellis diagram and state (l). Using BPSK modulation $x_{t,0}^0$ is equal to -1 and $x_{t,0}^1$ is equal to $+1$. LLR becomes:

$$\Lambda_1(c_t) = \log \frac{p_t^1(1)}{p_t^1(0)} + \frac{2}{\sigma^2} r_{t,0} + \Lambda_{1e}(c_t) \tag{2.41}$$

where

$$\Lambda_{1e}(c_t) = \log \frac{\sum_{l',l=0}^{M_s-1} \alpha_{t-1}(l').\beta_t(l).e^{\left(-\sum_{j=1}^{1}(r_{t,j}-x_{t,j}^1(l))^2/2\sigma^2\right)}}{\sum_{l',l=0}^{M_s-1} \alpha_{t-1}(l').\beta_t(l).e^{\left(-\sum_{j=1}^{1}(r_{t,j}-x_{t,j}^0(l))^2/2\sigma^2\right)}} \tag{2.42}$$

$\Lambda_{1e}(c_t)$ is called *extrinsic information*. This value does not contain the information of the received data bit at time t ($r_{t,0}$). This is the information that is produced by one MAP decoder constituent and is used by the other one as the input.

The interleaved version of the first decoder's extrinsic information ($\tilde{\Lambda}_{1e}(c_t)$) is used as the a priori probability for the second decoder. The $p_t^2(i)$ is the a priori probability for $c_t = i$ of the input information bits for the second decoder

$$\tilde{\Lambda}_{1e}(c_t) = \log \frac{p_t^2(1)}{p_t^2(0)} \quad , \quad p_t^2(1) = 1 - p_t^2(0) \tag{2.43}$$

It is straight forward to write:

$$p_t^2(1) = \frac{e^{\tilde{\Lambda}_{1e}(c_t)}}{1 + e^{\tilde{\Lambda}_{1e}(c_t)}} \tag{2.44}$$

$$p_t^2(0) = \frac{1}{1 + e^{\tilde{\Lambda}_{1e}(c_t)}} \tag{2.45}$$

and for the second decoder, the LLR can be written:

$$\Lambda_2(c_t) = \log \frac{p_t^2(1)}{p_t^2(0)} + \frac{2}{\sigma^2}\tilde{r}_{t,0} + \Lambda_{2e}(c_t) \tag{2.46}$$

$$\Lambda_2(c_t) = \tilde{\Lambda}_{1e}(c_t) + \frac{2}{\sigma^2}\tilde{r}_{t,0} + \Lambda_{2e}(c_t) \tag{2.47}$$

The extrinsic information of the second MAP decoder is interleaved and used as the a priori probability for the first MAP decoder. By initializing $\Lambda_{2e}^{(0)}(c_t) = 0$ in the first iteration, and performing a definite maximum number of iterations, turbo decoder stops the decoding process and makes a hard decision based on the last iteration's soft output.

2.4 Iterative Nature of Turbo Decoder

The most important property of the turbo decoder is its iterative nature [16,17]. This means that for decoding a frame, the turbo decoder performs several iterations. For the first iteration, it assumes that the probabilities of 0 and 1 are equal and based on the channel information bits it produces a soft decision output for each data bit. For other iterations, the decoder will use the soft output of the other decoder as the a priori probability.

In each iteration, the turbo decoder improves the SER or BER performance in comparison with the previous iterations, and it can correct more number of erroneous bits. As a result, the more number of iterations the turbo decoder can perform for each frame, the better BER it can achieve. But the number of required iterations per each frame is a random variable, and it is not possible to set the turbo decoder to perform a fixed number of iterations per each frame and have an optimum receiver in terms of SER or BER.

As mentioned before, the performance of these kinds of iterative decoders is directly related to the number of iterations that they perform to decode a frame. Here, we have simulated the 3GPP turbo code for several values of signal-to-noise ratio (SNR) [18], in order to use them as practical examples in the next chapters (the details of these computer-based simulations will be given later). Table 1 shows the percentage of the 3GPP packets being corrected at any given iteration.

For instance, for SNR = 1.5 dB in Table 1, if there is a turbo decoder capable of performing a maximum of 3 iterations per frame, it will be able to correct 98.308% (0.004+59.453+38.851) of the frames. As a result, the FER is equal to 0.01692, but if this decoder was capable of performing a maximum of 4 iterations per frame (one more iteration), the FER would be improved more than 10 times (0.00104).

Table 2.1: Probabilities (%) of correction of a 3GPP frame vs. number of iterations for several SNRs on an AWGN channel (k=1440, r=1/3).

Number of iterations	SNR = 1.0 dB	SNR = 1.5 dB	SNR = 2.0 dB
1	0.000	0.004	1.467408
2	4.615	59.453	95.48462
3	47.825	38.851	3.028626
4	31.920	1.588	0.016247
5	9.946	0.077	0.001644
6	3.059	0.007	0.000483
7	1.162	0.002	0.000000
8	0.528	0.002	0.000000
9	0.258	0.002	0.000000
10	0.159	0.000	0.000000
11	0.094	0.000	0.000000
12	0.085	0.000	0.000000
13	0.056	0.000	0.000000
14	0.034	0.000	0.000000
Not-correctable	0.259	0.014	0.000967

From Table 2.1, it can be seen that for SNR of 1.5 dB, there is no frame that can be corrected in 10, 11, 12, 13 and 14 iterations. As a result, setting the maximum number of iterations to more than 9 (this parameter will be described in detail in Chapter 3) for this value of SNR, will not change the performance of the turbo decoder, and also in multi-channel decoding systems, it will make it worse.

2.5 Early Stopping Criteria

Some of the turbo decoders are capable of performing various number of decoding iterations for different input frames in order to achieve the desired performance (BER or FER), while others only perform a fixed number of iterations for each frame. As indicated in the previous sections, the required number of iterations for each frame is a random

variable with a specific distribution function based on the channel characteristic, coding specifications, etc.

The ideal case for the turbo decoder is that the decoder has the source information and can compare it with the decoded sequence in order to determine if there is any symbols (bits) in error or not. In this case, if the sequence is decoded correctly, the turbo decoder stops decoding process and makes a hard decision based on the last soft output. However, since the source information is not accessible to the receiver (had it been available, the whole communication process would have been futile), this is not a realizable method.

One of the techniques that can give the turbo decoder an estimation of how correct a frame is, is stopping criterion. At the end of each iteration for each frame, the decoder checks the condition of the stopping criterion; if the condition is satisfied the turbo decoder stops decoding and assumes that the output of the last iteration is reliable enough to make a hard decision based on it and there will not exist any improvement in SER or BER by performing more iterations on the current frame; otherwise the turbo decoder continues decoding process and performs another iteration on the current frame. To prevent an endless loop of performing iterations (when the stopping criterion is never satisfied or the input frame is not a decodable one), it is possible to define a maximum number of iterations per frame. After performing this number of iterations for each frame, if the stopping criterion is not yet satisfied, the turbo decoder stops decoding process and makes a hard decision based on the output of the last iteration, or in some cases turbo decoder assumes that this frame is in error and it is not possible to make a decision based on this frame (this depends on the application).

There are several proposed stopping criteria so far [19,8] such as soft-decision rules, hard-decision rules and Cyclic Redundancy Check (CRC) rule. Besides all of the realizable

stopping criteria, there is an unrealizable stopping rule that can be used as a limit for the performance of other stopping criteria. We call this unrealizable stopping criterion "Genie rule". Theoretically, it is assumed that genie rule can recognize whether the decoded sequence is corrected or not based on the knowledge of the original transmitted sequence, and it can immediately stop the decoding process as soon as that specific frame becomes corrected. In the following, some of the realizable stopping criteria are described in detail.

In one of the soft-decision rules, at the end of each iteration, the average value of the extrinsic information of the second decoder component ($\tilde{\Lambda}_{2e}(c_i)$) is compared to a given threshold (θ). This stopping rule is satisfied at the earliest iteration that:

$$\frac{1}{N}\sum_{i=1}^{N} |\tilde{\Lambda}_{2e}(c_i)| \geq \theta \tag{2.48}$$

In [8], it has been shown that using the above stopping criterion, will add on the average less than one iteration to the number of required iterations to correct each input channel (in comparison with the genie rule).

In another soft-decision rule, the minimum of the extrinsic value of the second constituent decoder ($\tilde{\Lambda}_{2e}(c_i)$) for all of the symbols (bits) is compared to a threshold (θ). The decoding procedure will stop at the moment that the following condition is satisfied:

$$\min_{1\leq i\leq N} [\ |\tilde{\Lambda}_{2e}(c_i)|\ ] \geq \theta \tag{2.49}$$

In one of the hard-decision rules, the decoder stops the decoding process as soon as the hard decision at the end of the current iteration is identical to the hard decision of the previous iteration.

$$\hat{c}_i^n = \hat{c}_i^{n-1} \quad , \quad 1 \le i \le N \tag{2.50}$$

In CRC stopping criterion, a specific number of CRC bits is appended to the end of each information sequence at the input of the turbo encoder. At the decoder, after performing each iteration, the decoder makes hard decision of the sequence and checks the CRC bits for error. If decoder detects no error in those CRC bits, it assumes that the output of the last iteration is reliable enough to make the final decision and it stops the decoding process.

2.6 Summary

In this chapter, BCJR algorithm was presented and three versions of this algorithm were described. Then, the turbo interleaver and turbo code structure were discussed. After that, the iterative nature of turbo decoder was explained using some computer-based simulations. And finally, the concept of stopping criteria and some of those rules were presented in detail.

Chapter 3

MULTI-CHANNEL PROCESSING

In some communication systems, the turbo decoder receives several input frames at the same time, either from one transmitter (like another mobile base station) or from several transmitters (like mobile handset terminals). These input frames are usually different and have to be decoded separately. To obtain a better performance in these kinds of systems, multi-channel processing techniques are being used widely.

In order to process all of the frames in real time, the maximum turbo decoding delay for each frame is set equal to the period of each frame (T), so there is no need for data buffering in the receiver. As it was mentioned in the previous chapter, each frame needs a specific number of iterations to be decoded correctly. This number of iterations is different from one frame to another.

If the turbo decoder takes (τ) seconds to perform one iteration, so the total number of iterations that the turbo decoder can perform to decode all of (n_{ch}) frames in one time slot (T) will be:

$$n_{it} = \lfloor T / \tau \rfloor \tag{3.1}$$

3.1 Ideal Scheme

When using one decoder to decode all the channels, the ideal situation is that the decoder has the information about all frames (channels) and knows how many iterations each frame needs to be decoded correctly. So, the decoder first decodes the frame that needs less number of iterations, and after decoding it, it decodes the frame that needs also less number of iterations among all of the remaining frames and so on. This ideal method is optimum.

But, the main problem is that the decoder neither has the capability to determine that a specific channel has been corrected nor knows how many iterations a frame needs to be corrected. As explained in the previous chapter, early stopping criteria can approximate these parameters, but still to use the stopping criteria the turbo decoder has to perform a few iterations for each frame, and it does not have access to this information prior to the decoding process. This pre-decoding process will decrease the number of available iterations for the rest of the process.

No other schemes can have better performance than this ideal (unrealizable) scheme. In this thesis, this scheme will be used as the theoretical limit for analyzing the performance of other schemes.

3.2 Fixed-Iteration Scheme

Fixed-Iteration scheme is the traditional turbo processing scheme. It is designed based on the decoding algorithm for a single input channel (single frame processing) system. In this method, for each input channel, there is one turbo decoder capable of performing a definite number of iterations (n_f). So, for an n_{ch}-channel system, there will be n_{ch} completely independent turbo decoders (Fig. 3.1).

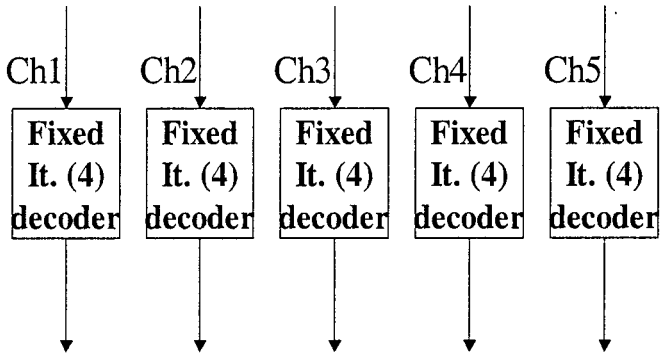

Fig. 3.1: An example of the fixed-iteration method of multi-channel decoding, consisting of 5 fixed-iteration turbo decoders with 4 iterations per channel (n_f=4), i.e., there is a total of 20 iterations for all channels (n_{it}=20).

This decoding scheme is exactly equivalent to the scheme using a single turbo decoder capable of performing a total of n_{it} iterations in each time slot (frame period). The n_{it} is determined as follows:

$$n_f = \lfloor n_{it} / n_{ch} \rfloor \tag{3.2}$$

This single turbo decoder performs a fixed number of iterations (n_f) for decoding each frame.

The implementation of this scheme is very simple and straightforward. The main disadvantage of this scheme is its performance. This disadvantage can be explained by means of an example; assume that n_f is equal to 4. There will be some input frames that need less number of iterations to be decoded. For instance, if a frame needs 2 iterations to be decoded, the turbo decoder decodes it in 2 iterations and spends the remaining 2 iterations on this frame without any gain or improvement. On the other hand, there may exist some other frames that need more than n_f iterations (e.g., 5 iterations) to be decoded. So, the decoder performs 4 iterations on them and leaves these frames in error. In summary, the turbo decoder is idle for early-decoded frames while it does not have enough iterations to perform for the late decodable frames.

3.3 Serial Processing Scheme

The multi-channel processing algorithm that we call *serial processing scheme* is based on the decoding method for a single-channel system using the stopping criteria technique explained in Chapter 2. The decoding procedure is as follows: the decoder starts decoding the first input frame and performs some iterations on this frame until the stopping criterion is satisfied; at this point the decoder assumes that the frame has been decoded correctly and it makes the hard estimation on the frame. After decoding the first channel, the decoder repeats the same procedure for the second frame and other frames one after the other (Fig. 3.2).

On the average, the decoder is supposed to decode each channel in n_f iterations and if some of the channels are corrected in less than n_f iterations, the decoder can save these extra iterations and use them for decoding other frames that need more than n_f iterations.

This method is much better than the fixed-iteration scheme, because the decoder can use the idle time left after decoding the easier frames to correct the remaining frames. As a result, with the same total number of iterations as the fixed-iteration scheme, we can achieve a better BER or with the same performance, the decoder can decode more channels (increasing the throughput of the decoder).

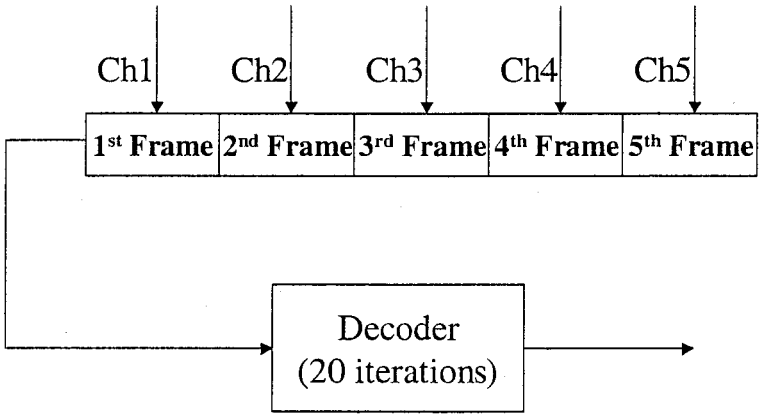

Fig. 3.2: An example of the serial processing scheme of multi-channel decoding, consisting of a single turbo decoder.

In this method, there are some cases where the decoder cannot perform even a single iteration on some of the channels. As the worst-case scenario, assume that the first frame is not correctable in less than n_{it} iterations (the total number of available iterations for one time slot), so the decoder will spend all its time to decode this frame, and cannot do any iterations on other frames. Therefore all of the frames will be in error. To avoid this problem, we define *Mipch* as the maximum number of iterations per channel (*Mipch* must be less than n_{it}). It means that if the decoder cannot decode a frame after *Mipch* iterations, it will stop processing current frame and will go to the next frame. This *Mipch* is one of the design parameters, which has to be set by the designer. In the next chapter, several

computer-based simulations results are presented with different values of *Mipch* for different SNRs.

Since this scheme -even when using the maximum number of iterations per channel- is a serial processing technique, there are some cases where the turbo decoder cannot decode the frames in the best way. We will explain these situations using an example: assume that there are 5 input channels (n_{ch}=5) and the turbo decoder is capable of performing 20 iterations (n_{it}=20) and the maximum number of iterations has been set to 7 (*Mipch* =7). Suppose that the input frames need 7, 6, 8, 4 and 2 iterations to be corrected. Using the described scheme, the turbo decoder performs the maximum number of iterations (*Mipch* =7) to correct the first frame, after that, it performs 6 iterations to decode the second frame, and next, it performs the maximum number of iterations (*Mipch* =7) to decode the third frame but this frame cannot be corrected with 7 iterations, and at the end there is not any iterations left to decode the forth and fifth frames. As a result, the turbo decoder corrects only 2 frames out of 5. If the decoder could decode the forth and fifth frame prior to third frame, 4 frames instead of 2, would be corrected. But since this is a serial processing technique, it always starts decoding process from the first received frame to the last and it is not possible to change the order of the frames in the decoding process.

3.4 Parallel Processing Scheme

In the previous method, the decoder decodes the channels serially (it decodes the first one completely, and goes to the next one and so on.), but in this new method, the decoder decodes channels in parallel. This means that the decoder performs one iteration for each frame, and checks whether that frame satisfies the stopping criterion or not. For the first

round, it performs one iteration for each frame, and for the second and other rounds it will do exactly the same procedure but only for the uncorrected frames (Fig. 3.3).

In the serial processing scheme, despite the introduction of a maximum number of iterations per channel, there are some cases where the first frames need more than the average number of iterations as explained before. So probably, the decoder cannot decode the last frames (which may be correctable even in one or two iterations), due to the lack of required iterations. In this new scheme the turbo decoder decodes the frame that needs least number of iterations first.

Although the performance of this scheme is much better than the serial processing algorithm (it will be discussed in detail in the next chapter), in the DSP implementation chapter a new algorithm based on a combination of these two schemes will be suggested which is optimum in terms of number of memory accesses.

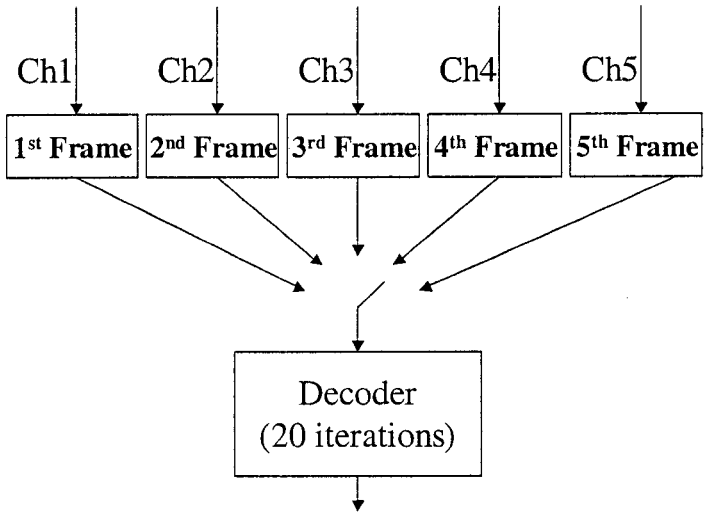

Fig. 3.3: An example of the parallel processing scheme for multi-channel decoding, consisting of a single turbo decoder.

3.5 Multi-Channel Schemes for Single Channel Systems

The above parallel processing schemes not only can improve the performance and/or throughput of the multi-channel systems, but also can be very useful for single channel turbo decoding systems with the same benefits.

With a very simple buffering technique, it is possible to convert a single channel system to a multi-channel one and apply all of the discussed schemes to it. For example, a channel with a data rate of 2 Mbps can be assumed as a 10-channel system with the data rates of 200 Kbps for each channel (Fig. 3.4). As a result, one turbo decoder can be utilized to decode all of the 10 channels at the same time. The serial to parallel converter part consists of 10 buffers each capable of storing a given number of frames (for example two) at a time. The input frames will be placed in order from the first buffer to the last. This process will be repeated at each time slot. When the frame at the top of a specific buffer unit is processed, this frame will be erased and the frames in that buffer will be shifted one place up and one empty place will be add to the end of the buffer. Each of these buffers is considered as one channel and a multi-channel processing scheme will be applied to these simulated channels.

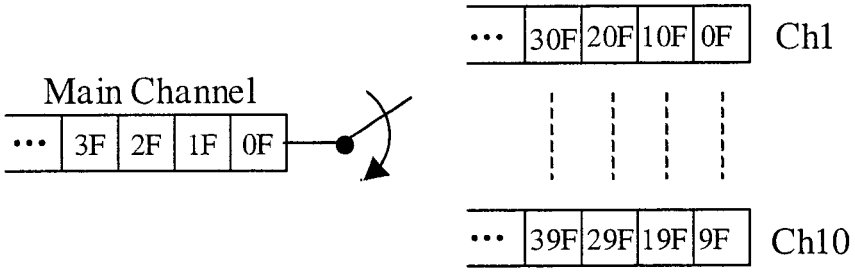

Fig. 3.4: An example of converting a one-channel system to a multi-channel one by using 10 buffers.

The only critical issue in this case, is the latency of the decoder, because the first input frame (frame 0F in Fig. 3.4) should be buffered until other 9 frames (1F...9F in Fig. 3.4) are received in order to start the decoding process. So the maximum acceptable latency of the system determines the maximum number of channels that can be assumed:

$$Maximum\ Number\ of\ channels = {Maximum\ Latency}/{The\ Frame\ Period} \qquad (3.3)$$

3.6 Summary

At the beginning of this chapter, the multi-channel turbo decoding system was described, and then in order to improve the performance and/or throughput of this system some multi-channel processing schemes were introduced. At the end, it was explained how to use these schemes even for a single channel system.

Chapter 4

COMPUTER SIMULATIONS

In this chapter, we will talk about two different turbo coding standards; 3GPP and DVB/RCS. Next, we will simulate a simple turbo decoder for each of these standards for several SNRs and code rates in order to obtain their FER. We will describe the computer algorithms that we use to simulate the proposed multi-channel processing schemes. At the end, we will simulate all those multi-channel processing schemes and will discuss their performance in detail.

4.1 3GPP Standard

Second Generation (2G) mobile communication systems, which have been designed for mobile digital telephony, have a very low data capacity [20]. For example, GSM supports only data rates up to 9.6 Kbps. In order to transmit video and image on a mobile

communication system, third generation (3G) standard has been proposed, which is capable of transmitting data rates up to 2 Mbps.

The most important step in utilizing this new technology was its standardization process between various regional and global operators. International Telecommunication Union (ITU), which is the final organization responsible for adopting standards in telecommunication area, set December 1998 as the deadline for submitting the 3G standard proposals for all of organizations and countries around the world. The family of 3G standards, which was approved later by ITU, is called IMT-2000 (International Mobile Telecommunications for the year of 2000).

European Telecommunication Standards Institute (ETSI) is the organization responsible for developing telecommunication standards (e.g. GSM) in Europe. ETSI proposed UMTS (Universal Mobile Telecommunications System) standard as the 3G candidate for the IMT-2000 (Table 4.1). UMTS is an evolutionary path form the old 2G (e.g. GSM) to the new 3G and allows the 2G operators to keep their infrastructures and investments during the upgrade process from 2G to 3G systems.

Table 4.1: UMTS key parameters

UMTS parameters	FDD	TDD
Multiple access scheme	W-CDMA	TD-CDMA
Crrier spacing	4.4-5.2 MHz	5 MHz
Chip rate	3.84 Mchip/s (Mcps)	
Spreading factor range	4 – 512	1 – 16
Modulation	QPSK	
Pulse shaping	Root raised cosine, roll-off = 0.22	
Frame length	10 ms	
Timeslots per frame	15	

Unlike Japan and Europe, there is not any nation-wide 2G standard in USA. In the USA there are three main 2G standards; TDMA-based standard (IS-136), CDMA-based standard (IS-91) and GSM derivative standard (PSC1900). CDMA2000 (Code Division Multiple Access for the year of 2000) and UWC-136 were U.S. proposals to the ITU as the evolutions of IS-95 and IS-136, respectively.

In 1998, 3rd Generation Partnership Project (3GPP) was established to harmonize the various 3G W-CDMA proposals based on the GSM core network. This organization was founded by almost all of the major countries in the telecommunication field (Japan, Europe, USA, ...). In order to merge Japanese 3G and UMTS standards into a single standard, this organization proposed 3GPP and since that time, it has been developing and refining this standard. The 3GPP2 organization was established around the CDMA2000 and it proposed 3GPP2 standard. In 1999, Operator Harmonization Groupe (OHG) proposed a harmonization between 3GPP and 3GPP2, called G3G, in order to let CDMA2000 and UMTS, work together.

4.1.1 3GPP Channel Coding

Information blocks, sent as the input data to the channel encoder, are denoted by $O_{ir1}, O_{ir2}, ..., O_{irK_i}$; where i is the transport channel number, r is the code block number, and K_i is the number of information bits in each code block and the number of code blocks on transport channel i, is denoted by C_i [18]. After encoding, the bits are denoted by $y_{ir1}, y_{ir2}, ..., y_{irY_i}$, where Y_i is the number of encoded bits. The relationship between O_{irk} and y_{irk} and between K_i and Y_i is dependent on the channel coding scheme. The following channel coding schemes can be applied to transport channels; convolutional

coding, turbo coding and no coding. Use of coding scheme and coding rate for different types of transport channel is shown in Table 4.2. The value of Y_i for each coding scheme is:

- Convolutional coding with rate 1/2: $Y_i = 2 \times K_i + 16$; rate 1/3 $Y_i = 3 \times K_i + 24$;
- Turbo coding with rate 1/3: $Y_i = 3 \times K_i + 12$;
- No coding $Y_i = K_i$.

Table 4.2: Usage of channel coding scheme and coding rate.

Type of transport channel	Coding scheme	Coding rate
Broadcat channel	Convolutional coding	1/2
Paging channel		
Random Access channel		
Dedicated Channel, Downlink Shared Channel, Forward Access Channel		1/3, 1/2
	Turbo coding	1/3
	No Coding	

4.1.2 Turbo Coding Option of 3GPP

The turbo decoding scheme uses a Parallel Concatenated Convolutional Code (PCCC) with two 8-state constituent encoders and one turbo code internal interleaver. The coding rate of the turbo encoder is 1/3. The structure of turbo coder is illustrated in Fig. 4.1. the generating function is,

$$G(D) = \left[1, \frac{g_1(D)}{g_0(D)}\right] \tag{4.1}$$

where $g_0(D) = 1 + D^2 + D^3$, $g_1(D) = 1 + D + D^3$

the initial value of the shift registers of the 8-state constituent encoders shall be all zeros when starting to encode the input bits.

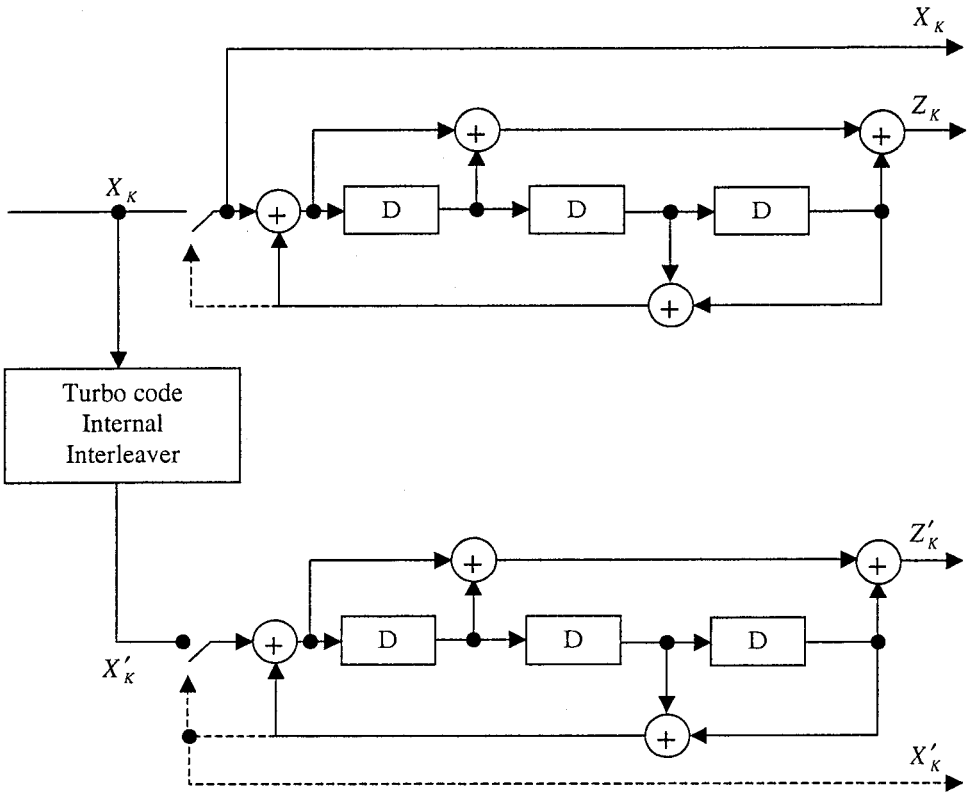

Fig. 4.1: Structure of rate 1/3 Turbo coder (dotted lines apply for trellis termination only)

The turbo encoder output sequence is:

$$x_1, z_1, z'_1, x_2, z_2, z'_2, ..., x_K, z_K, z'_K \tag{4.2}$$

where $x_1, x_2, ..., x_K$ are the input bits to the first constituent encoder and the interleaver, K is the number of bits, and $z_1, z_2, ..., z_K$ and $z'_1, z'_2, ..., z'_K$ are the output bits from the first and the second 8-state constituent encoders, respectively.

The output bits from the interleaver are denoted by $x'_1, x'_2, ..., x'_K$, and these bits are applied to the second 8-state constituent encoder.

The 3GPP turbo decoder consists of two MAP decoders as described in Chapter 2.

4.1.3 Trellis Termination for 3GPP Turbo Codes

In order to have a better performance, trellis termination can be used in 3GPP turbo coding. Trellis termination is performed by taking the tail bits from the shift registers feedback after all information bits are encoded. Tail bits are padded after the encoding of information bits. The first three tail bits shall be used to terminate the first constituent encoder (upper switch of Fig. 4.1. in lower position) while the second constituent encoder is disabled. The last three tail bits shall be used to terminate the second constituent encoder (lower switch of Fig. 4.1. in lower position) while the first constituent encoder is disabled. So the transmitted bits for trellis termination shall be:

$$x_{K+1}, z_{K+1}, x_{K+2}, z_{K+2}, x_{K+3}, z_{K+3}, x'_{K+1}, z'_{K+1}, x'_{K+2}, z'_{K+2}, x'_{K+3}, z'_{K+3} \tag{4.3}$$

4.1.4 3GPP Turbo Code Interleaver

The 3GPP turbo code internal interleaver functions as follows: applying input bits to a rectangular matrix with padding, performing intra-row and inter-row permutations of the rectangular matrix, and producing the output bits from the rectangular matrix with pruning.

The input bits to the turbo code internal interleaver are denoted by $x_1, x_2, x_3, ..., x_K$, where K is the number of information bits and takes a value between 40 and 5114. The relationship between the input bits to the turbo code internal interleaver and the input bits to the encoder is defined by $x_K = o_{irk}$ and $K = K_i$ [18].

4.1.5 3GPP Simulation Results

These simulation results are based on the 3GPP turbo codes using a simple 3GPP decoder. These simulations were run on a "Sun-Blade-1000 (dual-processor)" with a "SUNOS 5.8" operating system and "GCC 2.95" C++ compiler.

In Fig. 4.2, the performance (FER) of 3GPP turbo code versus the number of iterations per frame has been shown. The frame length is equal to 1440, code rate is 1/3 and we have chosen an AWGN channel model. We will use the results of these simulations as the inputs of the multi-channel processing simulations.

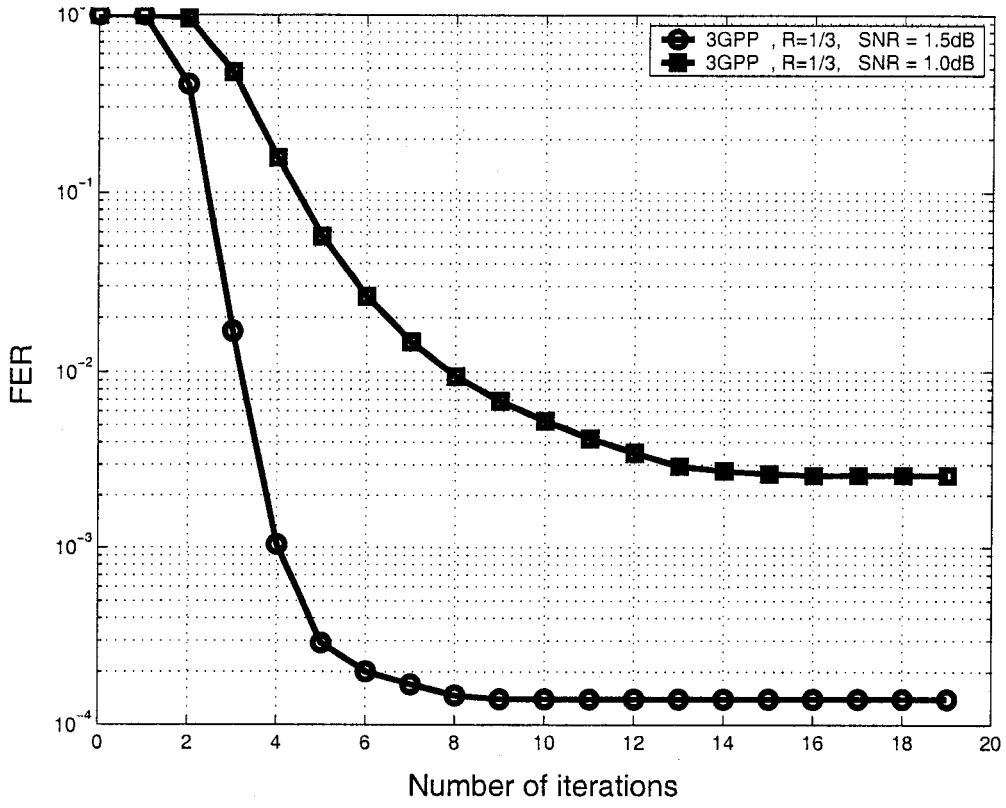

Fig. 4.2: 3GPP frame error rate (FER) vs. number of iterations per frame for several SNRs on an AWGN channel (k=1440, r=1/3).

4.2 DVB/RCS Standard

Digital Video Broadcasting for Return Channel via Satellite (DVB/RSC) has been proposed by the DVB Committee for the satellite communication systems [21]. ETSI (European Telecommunication Standards Institute) has also a DVB/RCS standard [22] to provide two-way, full-IP and asymmetric communications via satellite in order to be used in the countries where Asymmetric Digital Subscriber Line (ADSL) and cable modem are not economically capable of covering more than 25% of the demands. By using this standard, a large number of small terminals will be capable of sending *return* signals to a

central gateway and at the same time, receiving IP data from that gateway on the *forward* link in the DVB/MPEG2 (Digital Video Broadcasting/Moving Picture Expert Group-2) format. In DVB/RCS systems, the quality of service and the cost of terminals are almost independent of the distance of the central gateway and terminals. This advantage puts satellites in a favorable position. The speed of the return channel for DVB/RCS standard can range from 144 Kbps to 2 Mbps and it is shared between all terminals sending small packets using MF-TDMA (Multi-Frequency - Time Division Multiple Access)/DAMA (Demand-assigned Multiple Access) techniques. Use of Turbo Codes is also an option in DVB/RCS.

4.2.1 Turbo Coding Option of DVB/RCS

The DVB/RCS turbo encoder is depicted in Fig. 4.3. It uses double binary Circular Recursive Systematic Convolutional (CRSC) codes. The most significant bit (MSB) of the first byte, after the burst preamble, is assigned to A, the next bit to B and so on for the remaining of the burst content.

The encoder is fed by blocks of k bits or N couples ($k = 2 * N$ bits). N is a multiple of 4 (k is a multiple of 8). The polynomials defining the connections are described in octal and symbolic notations as follows:

- For the feedback branch: 15 (in octal), equivalently, $1 + D + D^3$ (in symbolic notation);
- For the Y parity bits: 13, equivalently, $1 + D^2 + D^3$;
- For the W parity bits: 11, equivalently, $1 + D^3$.

The input bit A is connected to tap "1" of the shift register and the input bit B is connected to the taps "1", D and D^2. First, the encoder (after initialization by the circulation state S_{C_1}) is fed by the sequence in the natural order (switch on position 1) with incremental address *i=0,...,N-1*. This first encoding is called C_1. Then, the encoder (after initialization by the circulation state S_{C_2}) is fed by the interleaved sequence (switch in position 2) with incremental address *j=0,...,N-1*. This second encoding is called C_2. The $\Pi(j)$ gives the natural address i of the considered couple, when reading it at place j for the second encoding.

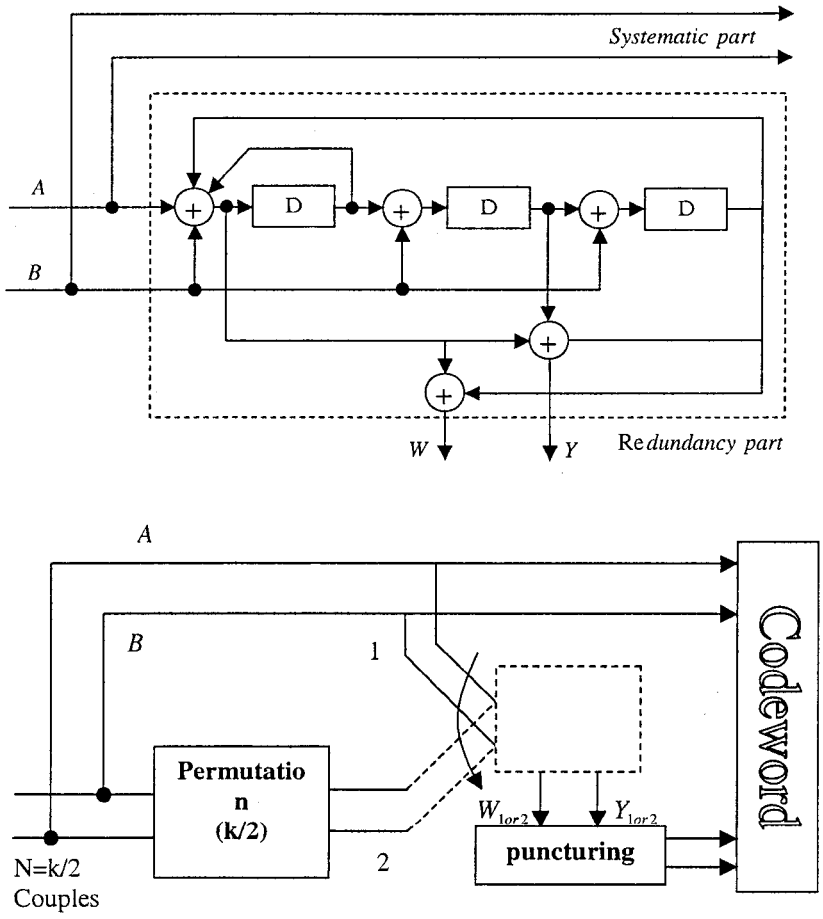

Fig. 4.3: DVB/RCS turbo encoder block diagram.

The DVB/RCS permutation is done on two levels, the first one inside the couples, and the second one between couples.

4.2.2 Determination of DVB/RCS Circulation States

The DVB/RCS turbo decoder consists of two MAP decoders as described in the Chapter 2.

The DVB/RCS turbo code is a Circular recursive systematic (CRSC) convolutional code. Using circular coding technique will enable each encoder to retrieve the initial state at the end of encoding procedure, as a result, data encoding process can be represented by a circular trellis structure, and there is no need to add extra bits for trellis termination. To determine the value of the circulation state (S_C), DVB/RCS uses a pre-encoding operation.

The state of the encoder is denoted by S $(0 \leq S \leq 7)$ with $S = 4 \times s_1 + 2 \times s_2 + s_3$. The encoding procedure is as follows:

1. Initialize the encoder with state 0. Precode the sequence in the natural order for the determination of S_{C1} or in the interleaved order for the determination of S_{C2}. In both cases, the final state of the encoder is denoted S_{N-1}^{0}.

2. According to the length N of the sequence, use the following correspondence to find S_{C1} or S_{C2} (Table 4.3).

3. Encode for the second time starting from states S_{C1} and S_{C2}. The circular recursive property of the constituent codes ensures that the two encoders will end in states S_{C1} and S_{C2}.

Table 4.3: Circulation state correspondence table

$S_{N-1}^{0} \rightarrow$ $\downarrow$N mod.7	0	1	2	3	4	5	6	7
1	S_C=0	S_C=6	S_C=4	S_C=2	S_C=7	S_C=1	S_C=3	S_C=5
2	S_C=0	S_C=3	S_C=7	S_C=4	S_C=5	S_C=6	S_C=2	S_C=1
3	S_C=0	S_C=5	S_C=3	S_C=6	S_C=2	S_C=7	S_C=1	S_C=4
4	S_C=0	S_C=4	S_C=1	S_C=5	S_C=6	S_C=2	S_C=7	S_C=3
5	S_C=0	S_C=2	S_C=5	S_C=7	S_C=1	S_C=3	S_C=4	S_C=6
6	S_C=0	S_C=7	S_C=6	S_C=1	S_C=3	S_C=4	S_C=5	S_C=2

4.2.3 DVB/RCS Code Rates

Seven code rates are defined for the DVB/RCS turbo code mode: R=1/3, 2/5, 1/2, 2/3, 3/4, 4/5, 6/7. This is achieved through selectively deleting the parity bits (puncturing). The puncturing patterns are identical for both the first encoder (C_1) and the second encoder (C_2) (deleting is always done in couples).

Rates 1/3, 2/5, 1/2, 2/3 and 4/5 are exact ones, independently of the block size. Rates 3/4 and 6/7 are exact ones only if N is a multiple of 3. In other cases, the actual rates are slightly lower than the nominal ones.

Depending on the code rate, the length of the encoded block is:

2N+M for R<1/2, with:

- M=N for R=1/3;
- M=N/2 for R=2/5.

N+M for R>1/2, with:

- M=N for R=1/2;
- M=N/2 for R=2/3;
- For R=3/4:
    - M=N/3 (if N mod. 3=0); or
    - M=(N-4)/3+2 (if N mod. 3=1); or
    - M=(N-8)/3+3 (if N mod. 3=2).
- M=N/4 for R=4/5;
- For R=6/7:
    - M=N/6 (if N mod. 3=0); or
    - M=(N-4)/6+1 (if N mod. 3=1); or
    - M=(N-8)/6+2 (if N mod. 3=2).

4.2.4 DVB/RCS Simulation Results

These simulation results are based on the DVB/RCS turbo codes using a simple DVB/RCS decoder. These simulations were run on a "Sun-Blade-1000 (dual-processor)" with a "SUNOS 5.8" operating system and "GCC 2.95" C++ compiler.

In Fig. 4.4, the performance (FER) of the DVB/RCS turbo code versus the number of iterations per frame has been shown. The frame length is equal to 212 (424 bits), code rate is 1/2 and we have chosen an AWGN channel model. We will use all results from these simulations as the inputs of the multi-channel processing simulations.

4.3 Simulation algorithms

In this section we describe the computer-based algorithms for the proposed multi-channel processing schemes. We use "C++" programming language to analyze their performance based on the probability statistics from computer-based DVB/RCS and 3GPP simulations.

Assume that the probability of decoding a frame in exactly i iteration is:

$$P(i) \quad , \quad 0 \le i < M \tag{4.4}$$

M is bigger than the maximum number of iterations that turbo decoder can perform in each time slot ($M > n_{it}$), as the result, the decoder is never capable of performing M iterations on any frames. By defining P_M as the probability that a frame is not corrected in less than M iteration, we will have:

$$1 - \sum_{i=0}^{M-1} P(i) = P_M \tag{4.5}$$

By these assumptions, the frame error rate (FER) is always greater than P_M.

These $P(i)$ values are obtained from running DVB/RCS and 3GPP computer-based simulations for different SNRs and code rates that were described earlier in detail and at this point, we assume that we have those statistics.

It is important to note that in all of the following parts we will use the ideal stopping criterion (Genie rule) that stops the decoding procedure exactly after correcting all of the symbols of the frame.

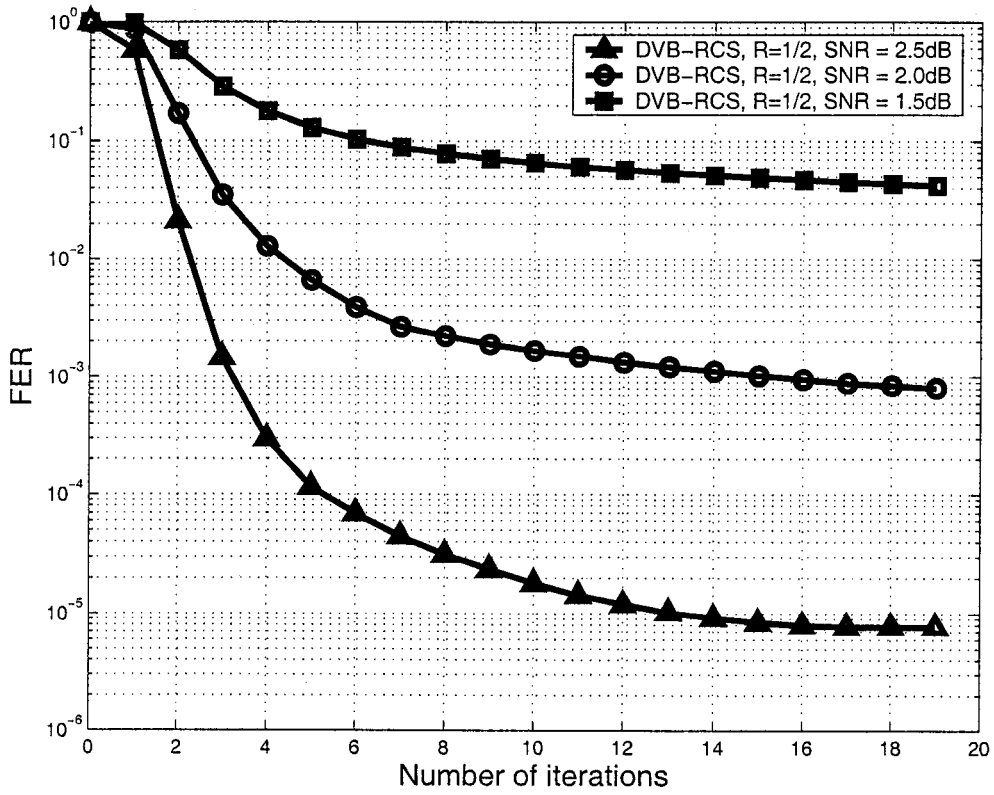

Fig. 4.4: DVB/RCS frame error rate (FER) vs. number of iterations per frame for several SNRs on an AWGN channel (k=212, r=1/2).

4.3.1 Fixed-Iteration Scheme

In practice, the performance of a multi-channel processing system using fixed-iteration (traditional) scheme that receives n_{ch} channels at the same time and has n_{it} iterations available each time slot, is the same as the performance of a single channel system performing $\lfloor n_{it} / n_{ch} \rfloor$ iterations on that channel. We will have:

$$\text{FER} = 1 - \sum_{i=0}^{\lfloor n_{it}/n_{ch} \rfloor} P(i) = 1 - (P(0) + P(1) + \cdots + P(\lfloor n_{it} / n_{ch} \rfloor)) \tag{4.6}$$

4.3.2 Ideal Processing Scheme

In order to compute the performance of the ideal processing scheme applying to those turbo coding standards, the mentioned C++ program creates a tree of n_{ch} level of branches, and using a recursive function (walk()) goes through those branches. At the end of each branch, it performs a function that calculates the frame error rate (FER) of that specific branch called Calculate_Brach_Error(x_i, x_j, x_k). Using a practical example can illuminate this process.

Assume n_{ch} is equal to 3, so the program will create a tree of three levels of branches as shown in Fig. 4.5, and the walk() procedure (the recursive procedure),walks through this tree. As is shown in Fig. 4.5, the walk() procedure selects one branch at each time:

$$\text{Probability of selecting this branch} = P_{br}(x_i, x_j, x_k)$$

Since the required number of iterations for each frame is independent from other frames:

$$P_{br}(x_i, x_j, x_k) = P_{br}(x_j, x_k, x_i) = P_{br}(x_k, x_i, x_j) \tag{4.7}$$

$$P_{br}(x_i, x_j, x_k) = P(x_i) \times P(x_j) \times P(x_k) \tag{4.8}$$

The probability of receiving a frame that requires x number of iterations to be corrected ($P(x)$), is calculated from simulation results in section 4.1.5 and 4.2.4.

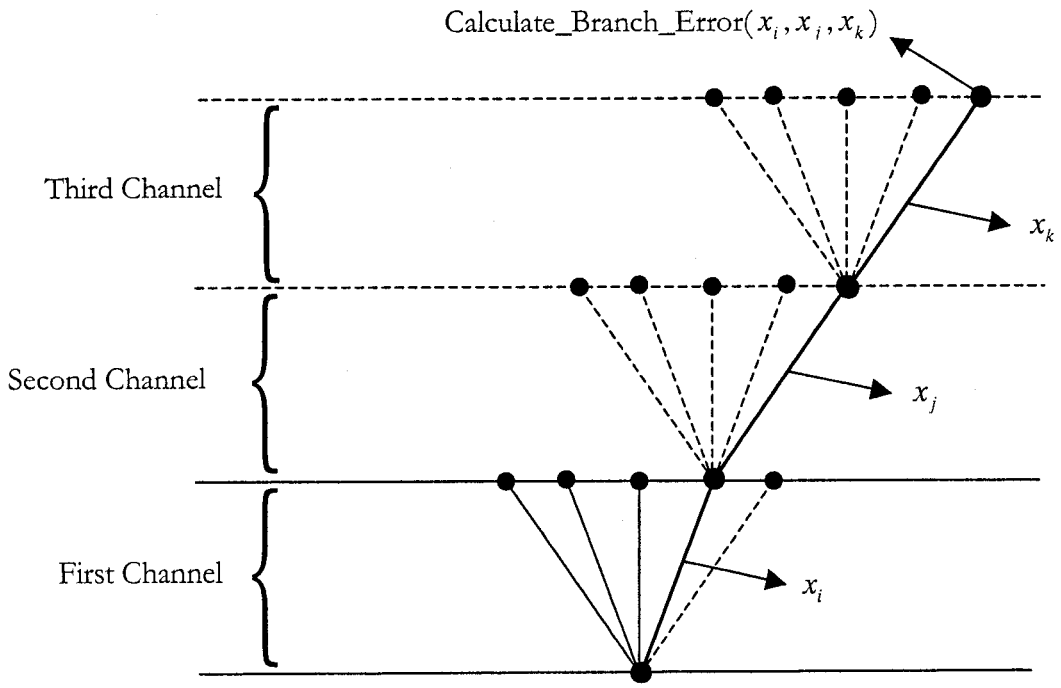

Fig. 4.5: Tree of n_{ch} level of branches, used for spanning all of the possibilities.

The inputs of the Calculate_Branch_Error(x_i, x_j, x_k) are the required number of iterations for each frame, and it will return the number of frames that remain in error. For the ideal processing scheme, Calculate_Branch_Error(x_i, x_j, x_k) functions as follows:

Initialization:

available_#_of_iterations = n_{it};

#_of_ corrected_ frame = 0;

Procedure:

For n_{ch} times do

x = find the minimum non-zero frame value between all frames (x_i);

if ($x \leq$ available_#_of_iterations)

available_#_of_iterations = available_#_of_iterations – x;

$x = 0$;

#_of_ corrected_ frame = #_of_ corrected_ frame + 1;

Output:

Return (n_{ch} - #_of_ corrected_ frame).

So this program will calculate the FER as follows:

$$\text{FER} = \sum_{i=0}^{M-1} \cdots \sum_{k=o}^{M-1} (P_{br}(x_i, \cdots, x_k) \times \text{Calculate\_Branch\_Error}(x_i, \cdots, x_k) / n_{ch}) \quad (4.9)$$

4.3.3 Serial Processing Scheme without MIPCH

In the following, a formula will be derived for the serial processing scheme without defining the maximum number of iteration per channel (*Mipch*) parameter. This formula can be easily implemented in Matlab.

If we have a communication system with n_{ch} input channels, for each channel, it is possible to define a polynomial:

$$G(x) = P(0)\times 1 + P(1)\times x + P(2)\times x^2 + \ldots + P(M-1)\times x^{M-1} \tag{4.10}$$

and for the whole system we will have:

$$H(x) = G(x)\times G(x)\times \cdots \times G(x) = (G(x))^{n_{ch}} \tag{4.11}$$

If the maximum number of iterations that the decoder can perform in each time slot is denoted be n_{it}, then:

$$R(x) = H(x) \mod . \; x^{n_{it}+1} \tag{4.12}$$

where *R(x=1)* is the probability of correcting a frame. As a result, frame error rate can be obtained as follows:

$$FER = 1 - R(x=1) \tag{4.13}$$

Another approach for serial processing scheme is using recursive programming method in "C++" which can take a long time for each run.

4.3.4 Serial Processing Scheme with MIPCH

The program written for this scheme is very similar to that one for ideal processing scheme and the structure is the same, but it uses a different Calculate_Brach_Error() function. The procedure of Calculate_Brach_Error() function is described as follows:

Initialization:

available_#_of_iterations = n_{it} ;

Take one frame as x;

#_of_ corrected_ frame = 0;

Procedure:

If x > Maximum_#_of_iteration_per_frame

If Maximum_#_of_iteration_per_frame > available_#_of_iterations

Go to Output;

Else

available_#_of_iterations= available_#_of_iterations – Maximum_#_of_iteration_per_frame;

Else

If x > available_#_of_iterations

Go to Output;

Else

available_#_of_iterations = available_#_of_iterations – x;

#_of_ corrected_ frame = #_of_ corrected_ frame + 1;

If there is any frame unprocessed

Take next frame as x

Else

Go to Output;

Go to Procedure

Output:

Return ((n_{ch} - #_of_ corrected_ frame) / n_{ch}).

And FER will be calculated as follows:

$$\text{FER} = \sum_{i=0}^{M-1} \cdots \sum_{k=o}^{M-1} (P_{br}(x_i, \cdots, x_k) \times \text{Calculate\_Branch\_Error}(x_i, \cdots, x_k) / n_{ch}) \quad (4.14)$$

4.3.5 Parallel Processing Scheme

The program written for this scheme is very similar to that one for ideal processing scheme and the structure is the same, but it uses a different Calculate_Brach_Error() function. The procedure of Calculate_Brach_Error() function is described as follows:

Initialization:

available_#_of_iterations = n_{it};

#_of_ corrected_ frame = 0;

Procedure:

While (available_#_of_iterations ≠ 0)and(#_of_ corrected_ frame ≠ n_{ch}) do

For (each input frame(x_i) ≠ 0) do

If (available_#_of_iterations ≠ 0)

$x_i = x_i - 1$;

available_#_of_iterations = available_#_of_iterations - 1

if ($x_i = 0$)

#_of_ corrected_ frame = #_of_ corrected_frame + 1;

Output:

Return (n_{ch} - #_of_ corrected_ frame).

And FER will be calculated as follows:

$$\text{FER} = \sum_{i=0}^{M-1} \cdots \sum_{k=o}^{M-1} (P_{br}(x_i, \cdots, x_k) \times \text{Calculate\_Branch\_Error}(x_i, \cdots, x_k) / n_{ch}) \quad (4.15)$$

4.4 Multi-Channel Processing Simulation Results

The simulation results for the above multi-channel processing schemes, are shown in Fig. 4.6 and Fig. 4.7 for SNR=1.0 dB and SNR=1.5 dB, respectively, for the 3GPP standard and in Fig. 4.8 and Fig. 4.9 for SNR=2.0 dB (R=1/2) and SNR=3.0 dB (R=2/3), respectively, for the DVB/RCS turbo code. These simulation results are based on the 3GPP and DVB/RCS turbo codes, and we use the probabilities of frame error from computer-based simulations for a single channel presented earlier in this chapter. These simulations were run on a "Sun-Blade-1000 (dual-processor)" with a "SUNOS 5.8" operating system and "GCC 2.95" C++ compiler.

It is shown in Fig. 4.6 and Fig. 4.7 for 3GPP, and in Fig. 4.8 and Fig. 4.9 for DVB/RCS turbo codes that the fixed-iteration scheme (or the traditional scheme) has the worst performance among the multi-channel processing schemes.

According to the 3GPP simulation graphs in Fig. 4.6 and Fig. 4.7, if the maximum number of iterations is less than 20 and 14 for SNR=1.0 dB (R=1/3) and SNR=1.5 dB (R=1/3), respectively, then the performance of the serial processing scheme is better than that of the parallel processing. But, FER in this area is too high and, as a result, designing a system operating within this range of FERs is not of practical interest. Therefore, we will compare the performances of those schemes only in the practical region (Fig. 4.6 and Fig. 4.7). For the maximum number of iterations more than 20 and 14 for SNR=1.0 dB (R=1/3)

and SNR=1.5 dB (R=1/3), respectively, the performance of the parallel processing scheme is better than that of the serial processing and it is also close enough to the ideal scheme where the decoder has the knowledge of the computational requirements of each channel before starting the decoding process.

With the same reason explained for the 3GPP simulations, we compare the performances of the proposed multi-channel processing schemes for DVB/RCS standard, only in the practical region. As shown in Fig. 4.8 and Fig. 4.9, for the maximum number of iterations more than 11 and 9 for SNR=2.0 dB (R1/2) and SNR=3.0 dB (R=2/3), respectively, the performance of the parallel processing scheme is better than that of the serial processing one and also close enough to the ideal scheme.

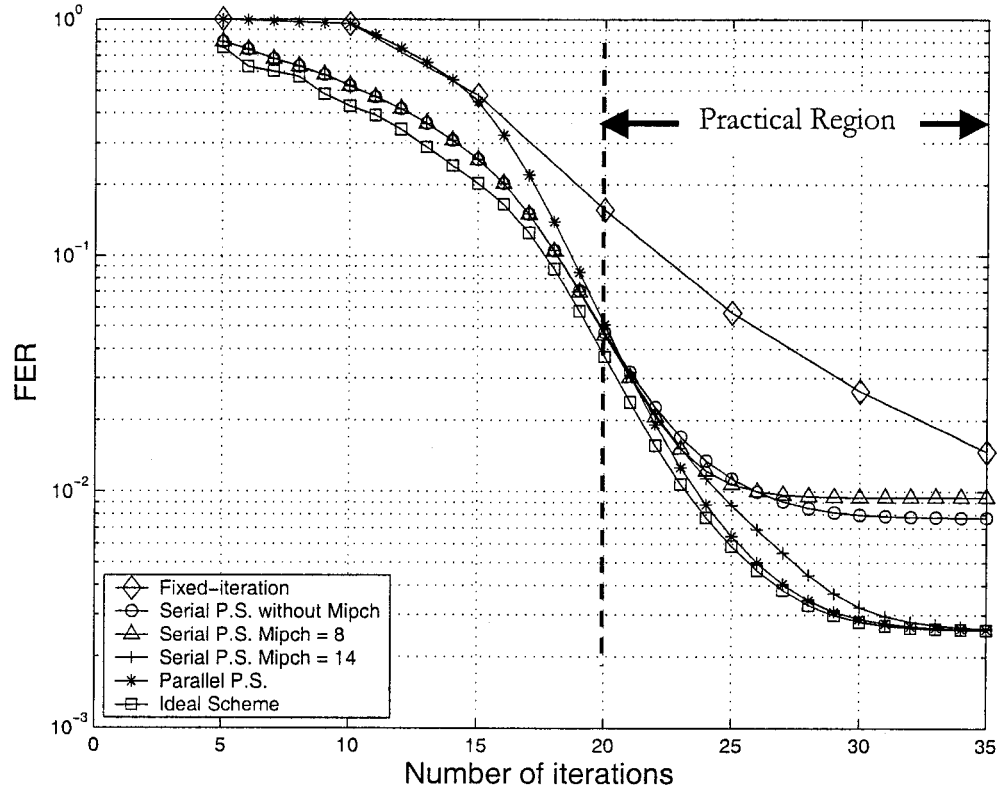

Fig. 4.6: FER performance of all multi-channel processing schemes for 3GPP turbo codes at SNR=1.0 dB, Frame length=1440. (Mipch=Maximum number of iterations per channel)

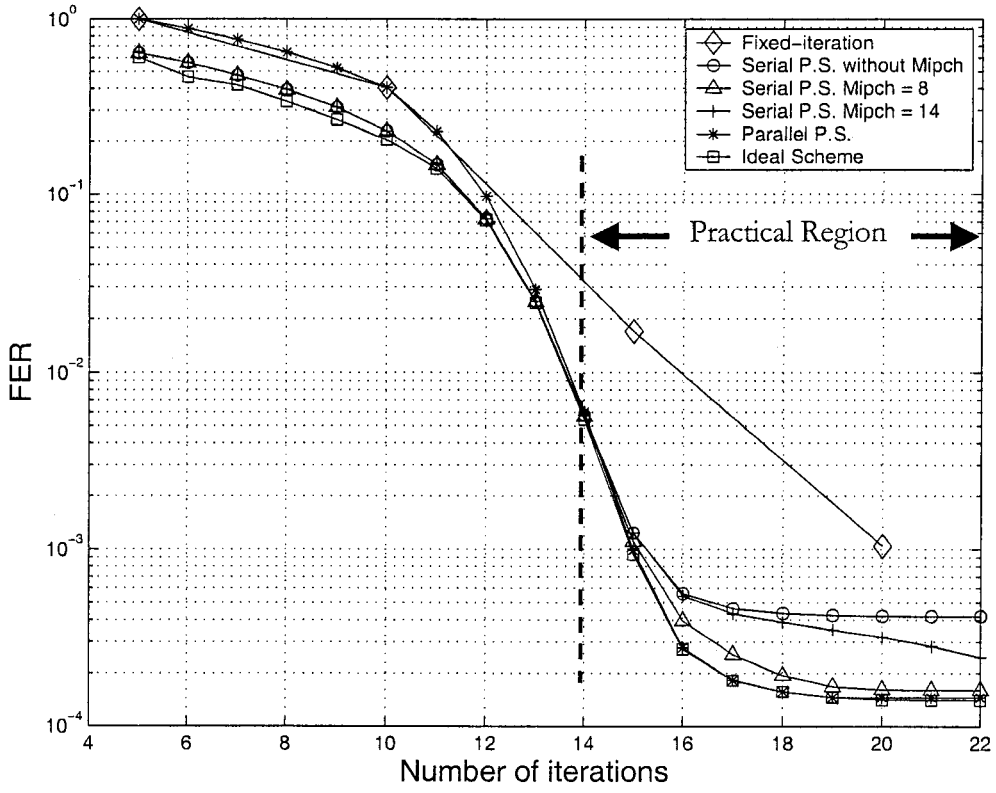

Fig. 4.7: FER performance of all multi-channel processing schemes for 3GPP turbo codes at SNR=1.5 dB, Frame length=1440. (Mipch=Maximum number of iterations per channel)

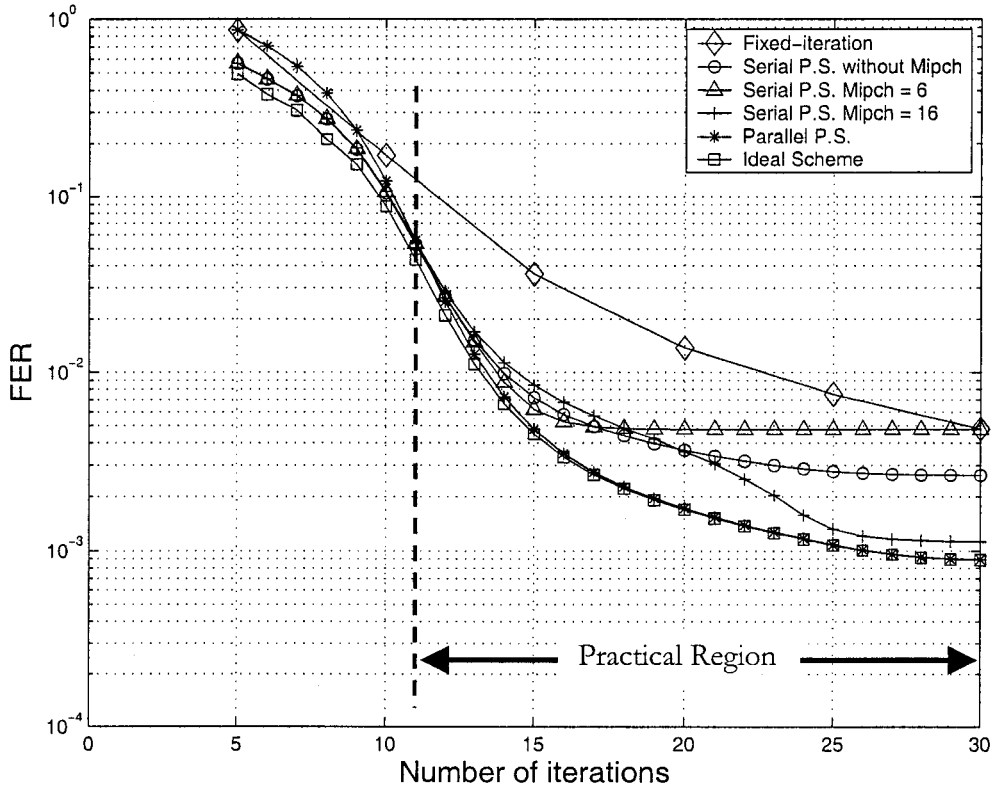

Fig. 4.8: FER of all multi-channel processing schemes for DVB-RCS turbo codes at SNR=2.0 dB (R=1/2), Frame length=212.
(Mipch=Maximum number of iterations per channel)

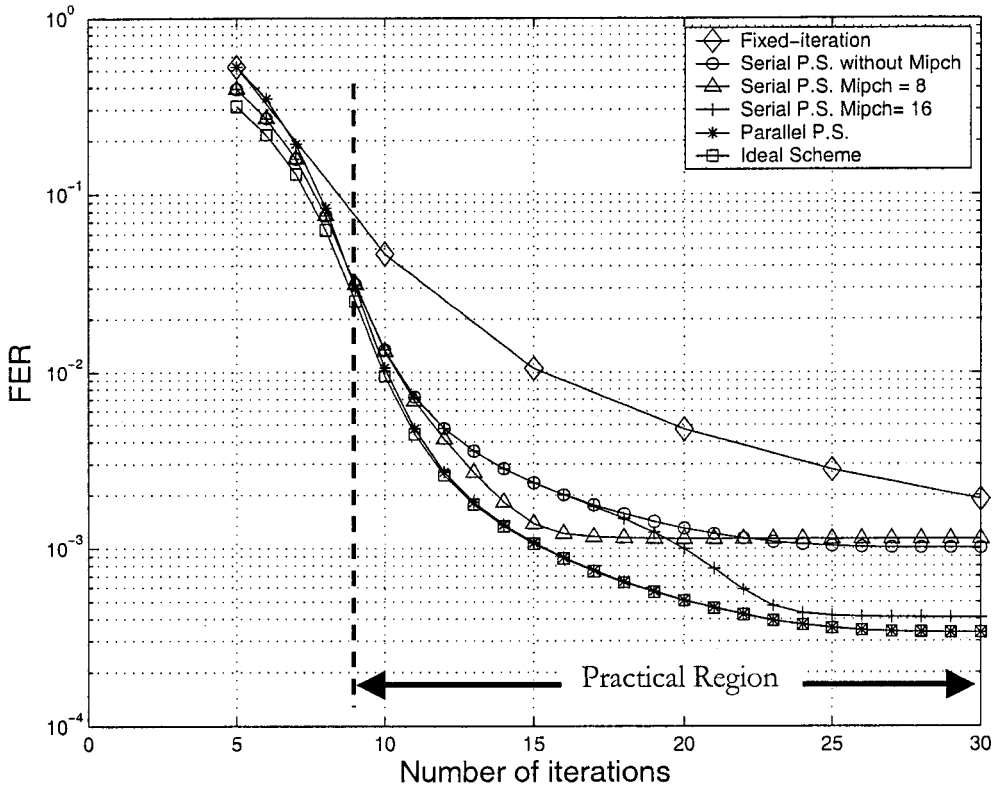

Fig. 4.9: FER of all multi-channel processing schemes for DVB-RCS turbo codes at SNR=3.0 dB (R=2/3), Frame length=212.
(Mipch=Maximum number of iterations per channel)

For any given SNR, it is possible to design a serial processing scheme decoder with a specific maximum number of iterations per channel having a performance near the performance of the parallel processing scheme (not better). But the main problem with this decoder is that its performance is very sensitive to the maximum number of iterations per channel. For example, it can be seen that the best maximum number of iterations per channel in Fig. 4.7, is 8, so if we design a decoder for this case, and for any reason the SNR changes during the data transmission from 1.5 dB to 1.0 dB, according to Fig. 4.6, the performance of the decoder is not appropriate at all for this new SNR, and the best performance at SNR=1.0 dB, is obtained by 14 as the maximum number of iterations per channel.

As a result, another advantage of the parallel processing scheme in comparison to the serial processing scheme is its robustness with respect to the SNR of the input channels and it always has the best performance for any given value of SNR. But in serial processing scheme, as explained before, MIPCH parameter must be chosen carefully based on the SNR of the channels in order to have a better performance, and since in a multi-channel processing system, the decoder is receiving more than one channel at a same time and these input channels are usually independent and have different SNRs, it is not possible to design a single serial processing turbo decoder with an optimum MIPCH for all of the channels. This disadvantage makes the serial processing scheme not a reliable scheme in practice.

It is possible to apply some modifications to the parallel processing algorithm to increase its performance. For example, after the first step and performing one iteration for each of the input frames, by using the stopping criterion concept, the decoder may decide which channel might needs less number of iterations to be corrected and it can decode that one first. The performance of this new scheme will be better than other proposed schemes. But, the implementation of this method is too complicated and on the other hand, it was

shown that the performance of the parallel processing algorithm is close enough to the theoretical limit. So there is no need to try to improve its performance in practice.

4.5 Summary

In this chapter, 3GPP and DVB/RCS turbo codes were introduced. Then the results of computer-based simulations of these turbo codes were presented. After that, based on the simulation results the performance of all those multi-channel processing schemes introduced in Chapter 3, were discussed and analyzed. And at end, it was shown that the performance of the parallel processing scheme is close enough to the performance of the ideal scheme.

Chapter 5

DSP IMPLEMENTATION

The TMS320C6000 Digital Signal Processor (DSP) platform is a part of Texas Instrument TMS320 DSP family, which has an architecture designed specifically for real-time signal processing. TMS320C62x DSP and TMS320C64x DSP generations comprise fixed-point devices and TMS320C67x DSP generation comprises floating point devices in the C6000 DSP platform. The C6000 family uses an advanced Very Long Instruction Word (VLIW) architecture. The architecture contains multiple execution units running in parallel, which allows it to perform multiple instructions in a single clock cycle. This parallelism is the key to high performance.

The newest member of the C6000 family, the C64x, brings a very high level of performance for data processing. At clock rates of 500-720 MHz, the C64x can process information at a rate of 4000-5760 Million Instructions Per Second (MIPS). Besides these

advantages, TMS320C6416, one of the members of C64x family, has also two embedded coprocessors: Viterbi coprocessor and Turbo Decoder coprocessor (which will be described in more detail later).

In this chapter, we will perform a feasibility study of the implementation of all those proposed multi-channel processing schemes on the TMS320C6416 DSP.

5.1 TMS320C6416 Architecture

Fig. 5.1 shows the block diagram for the TMS320C6416 DSP CPU [23]. The C6000 devices come with program memory and various sizes of data memory. Peripherals such as a Direct Memory Access (DMA) controller, power-down logic and External Memory Interface (EMIF) usually come with the CPU, while peripherals such as serial ports and host ports are only on certain devices.

5.1.1 Central Processing Unit (CPU)

As it is shown in Fig. 5.1, the TMS320C6416 CPU contains [24]:

- Program fetch unit.
- Instruction dispatch unit, advanced instruction packing.
- Instruction decode unit.
- Two data paths, each with four functional units.

- 64 32-bit registers.
- Control registers and control logics
- Test, emulation and interrupt logic

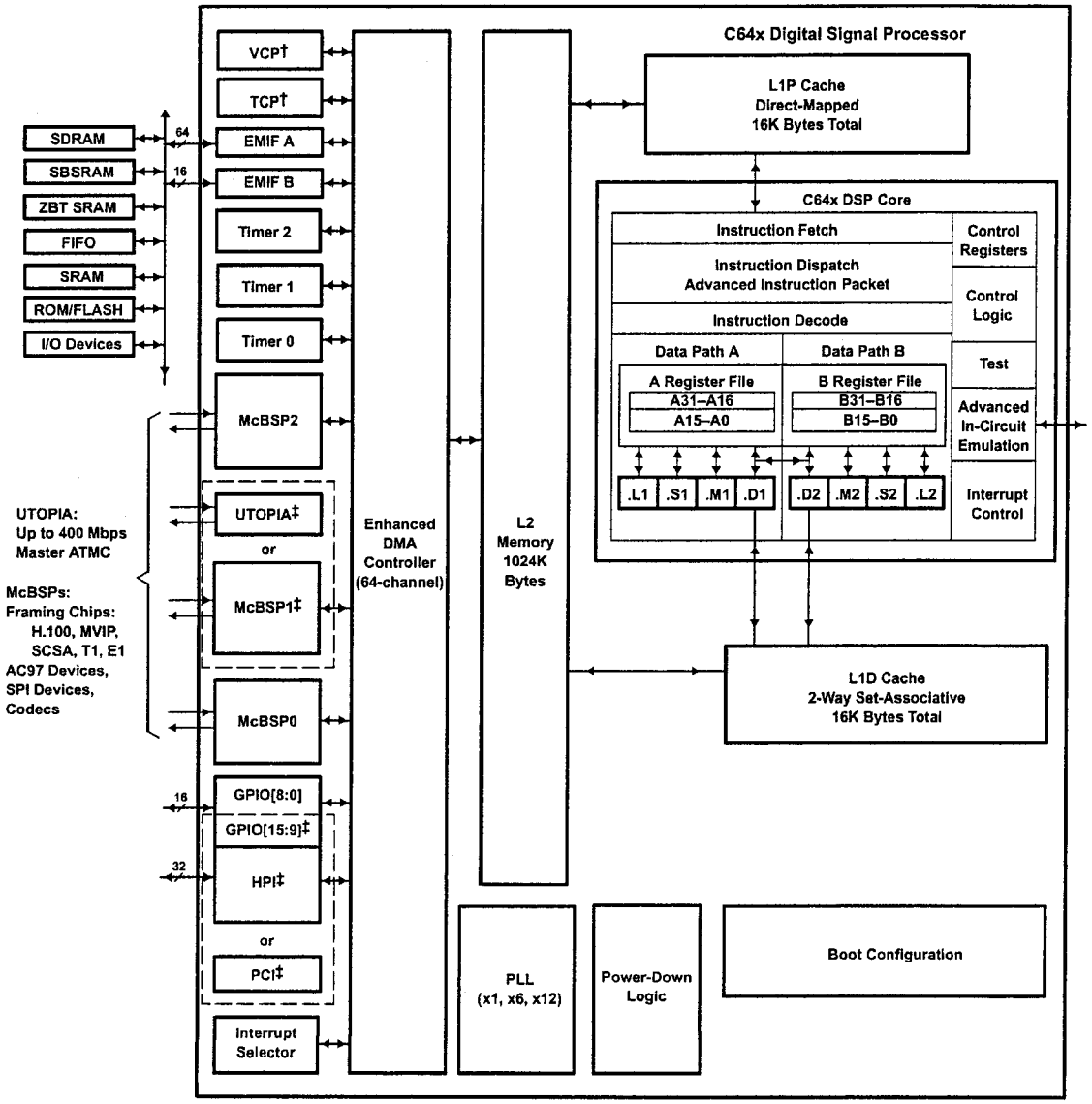

Fig. 5.1: Block diagram of TMS320C6416 DSP.

The program fetch, instruction dispatch and instruction decode units can deliver up to eight 32-bit instructions to the functional units every CPU clock cycle. The processing of instructions occurs in each of the two data paths (A and B) and each, contains four functional units (.L, .S, .M, and .D) and 32 32-bit general purpose registers. As a result, this DSP CPU is capable of performing up to 8 instructions in parallel.

5.1.2 Memory

The memory configuration of TMS320C6416 includes: internal data/program memory, internal peripherals and external memory accessed through the External Memory Interface (EMIF)[25].

TMS320C6416 DSP CPU has a cache-based architecture, with separate level-one program and data caches. These cache spaces are not included in the memory map and are enabled at all times. The level-one caches are only accessible by the CPU. The level-one program cache (L1P) controller interfaces the CPU to the L1P. A 256-bit wide path is provided from L1P to the CPU to allow a continuous stream of 8 32-bit instructions for maximum performance. The level-one data cache (L1D) controller provides the interface between the CPU and the L1D. The L1D allows simultaneous access by both sides of the CPU.

On a miss to either L1D or L1P, the request is passed to the L2 controller. This CPU can address up to 1280 Mbytes through L2 controller.

5.1.3 Peripheral Options

The user-accessible peripherals are configured via a set of memory-mapped control registers [26]. The main functions of the devices shown in Fig. 5.1 are summarized as follows:

Enhanced Direct Memory Access (EDMA) Controller: The EDMA controller transfers data between address ranges in the memory map without intervention by the CPU. The EDMA has 64 programmable channels, as well as a RAM space to hold multiple configurations for future transfers.

Multichannel Buffered Serial Port (McBSP): these three McBSPs interface to a variety of standards. Each C6416 McBSP supports independent channel selection at any given time for up to 128 channels. The 128 channels represent a full Serial Telecom (ST) Bus span. ST-Bus in combination with the flexible asynchronous interface provides a glueless connection to a variety of multi-channel telecommunication interface products.

Turbo Decoder and Viterbi Decoder Coprocessor: The C6416 device has two high-performance embedded coprocessors: Viterbi Decoder Coprocessor (VCP) and Turbo Decoder Coprocessor (TCP), that significantly speed up channel-decoding operations on-chip. The VCP operating at CPU clock divided-by-4 can decode over 500 7.95-Kbps adaptive multi-rate voice channels. The VCP supports constraint lengths K = 5, 6, 7, 8, and 9, rates R = 1/2, 1/3 and 1/4, and flexible polynomials, while generating hard decisions or soft decisions [27]. The Turbo Decoder Coprocessor will be described in next section in details.

5.2 Turbo Decoder Coprocessor (TCP)

The Turbo Coprocessor (TCP) is a programmable peripheral for decoding of 3GPP turbo codes, integrated into Texas Instruments TMS320C6416 DSP device. The TCP, operating at CPU clock divided by two, can decode up to thirty-six 384 Kbps or six 2 Mbps turbo encoded channels (assuming 6 iterations). This turbo decoder implements the Max-Log-MAP algorithm and is designed to support all polynomials and rates required by the Third-Generation Partnership Projects (3GPP and 3GPP2). Communications between the TCP and the CPU are carried out through the EDMA controller [28].

The TCP Programmable parameters are: code rate (1/2, 1/3 or 1/4), frame length, maximum number of iterations, threshold for early stopping criterion, turbo interleaver, number of iterations.

5.2.1 Interleaving/De-interleaving

In the standalone processing mode, the interleaving and de-interleaving are performed in the TCP, for frames up to 5114 bits. The look up table is generated by the DSP CPU and transferred to the TCP interleaver memory as a part of the TCP initialization process. This transfer is optional and determined by the TCP from the value of one of the TCP flags. If the current input frame uses the same turbo interleaver as the frame previously decoded, the transfer of interleaver table can be omitted. In the shared processing mode, the interleaving and the de-interleaving are done by the DSP CPU.

5.2.2 Stopping Criterion

The stopping criterion implemented in the TCP is used in standalone processing only. It is based on the measurement of the extrinsic SNR, comparison against a user-defined threshold and stopping the iterative processing if the threshold has been reached or exceeded. The SNR of extrinsic information is computed over the entire frame. The user-defined threshold is input into the TCP for each frame and can range from 1 to 100.

Setting SNR to zero disables the stopping criterion, so in this case, the maximum number of iterations (Mipch) specified for that particular frame will always be executed.

5.2.3 Data Format

Systematic and parity information from the channel, expected at the input of the TCP, are 8-bit signed quantities in (5,3) or 5Q3 fixed point representation. This means that there are 5 integer bits, followed by a binary point, followed by 3 fractional bits, which results in a range from –32.000 to +31.875. The TCP assumes that the mapping from unsigned to signed binary is 1 to –1 and 0 to +1.

5.2.4 MAP Processing Unit

The MAP decoder implements Max-Log-MAP algorithm with a small lookup table. The TCP processing unit (MAP unit) is shown in Fig. 5.2. It is using also a sliding window technique that allows parallel processing and reduces memory requirements [28].

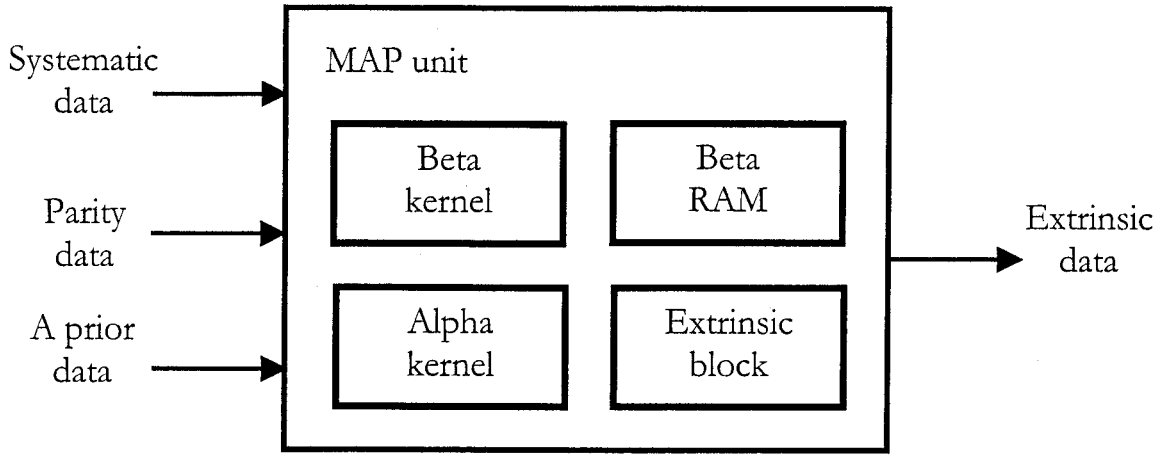

Fig. 5.2: MAP processing unit block diagram

The computation starts with β computation, with β's for all states and all stages stored to the TCP internal RAM, followed by the computation of α and the extrinsic information. The α values are used immediately for computation of extrinsic information. The computation of β is performed traversing the trellis from the end.

5.2.5 Sliding Windows Decoding

One of the drawbacks of the MAP decoding algorithm is its large storage requirement. Forward metrics α and backward metrics β need to be computed and stored for all states and all stages in the trellis, since they are required for the last step in the algorithm, which is extrinsic information computation. It is possible to combine α or β accumulation with extrinsic information, such that only β or α needs to be stored. This represents $(N+K-1)\times 2^{(K-1)}$ values, which need to be stored, for a frame containing N information bits and code constraint length K. In order to reduce memory requirement, large frames can be split into sub-blocks or windows, and MAP decoding can be performed on each window independently.

When the frame is split into independent sliding windows, the initial and final states for each window are not known. Therefore, equal probability is given to all α's at the first stage in the window, and all β's at the last stage of the window. In order to achieve reliable decoding, the first segment of α's as well as the last segment of β's should not be used in the final computation of extrinsic information. These initial and final segments of the window are called *header prolog* and *tail prolog*, respectively, and both are denoted by *Pro*. The extrinsic information is only computed over the middle segment of the sliding window, also called *reliability length* denoted by *Re*. This concept is illustrated in Fig. 5.3.

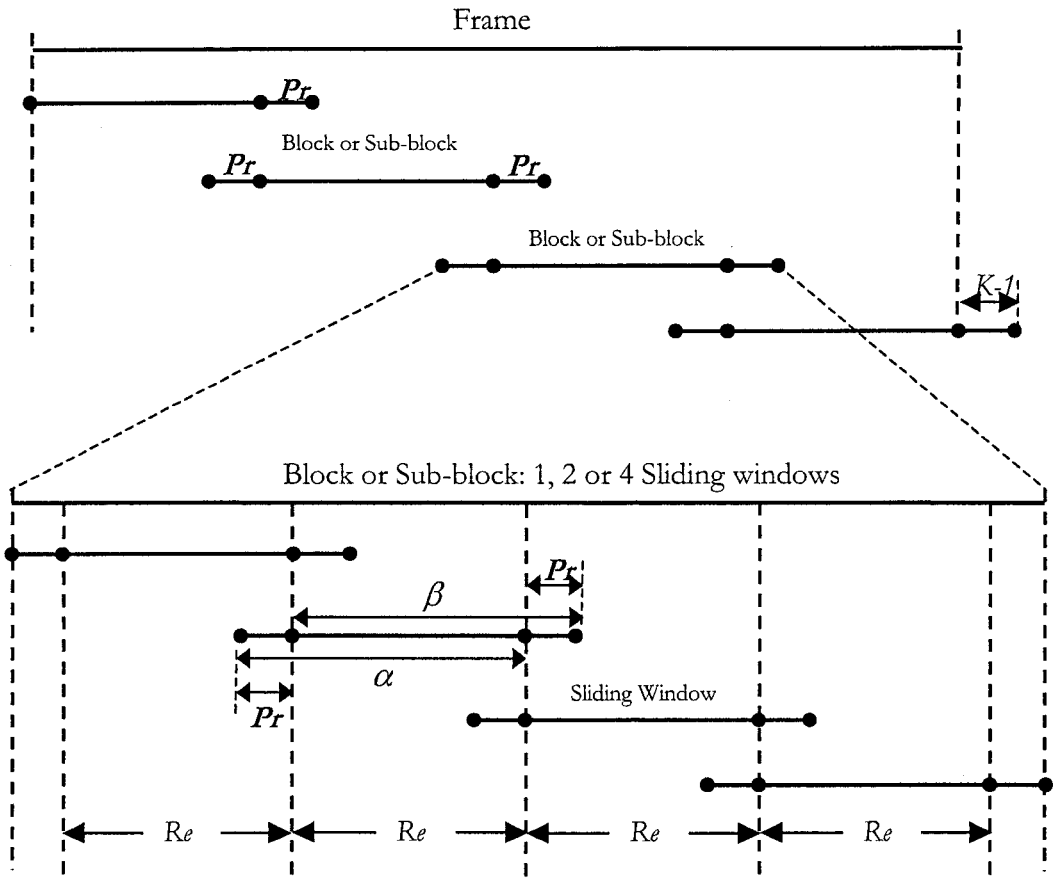

Fig. 5.3: Sub-blocks and sliding windows segmentation.

With sufficiently large values for *Pro* and *Re*, this approach does not result in any BER degradation. Usually *Pro* is chosen equal to 4 to 6 times the constraint length (K) of the code (This DSP CPU, is using *Pro* equal to 3 to 5 times the number of states (2^{K-1}) [29]).

The alpha and beta have usually equal length of prolog. The reliability length and the prolog length are user-defined. The frame length and number of sub-blocks are also user-defined. The number of sliding windows will be decided by the MAP unit based on the frame length (Table 5.1).

Table 5.1: Number of sliding windows per sub-block.

Frame Length	Number of Sliding Windows per Sub-block
From 40 to 128	1
From 129 to 256	2
From 257 to 5114	4

The MAP unit supports reliability sizes ranging from 40 to 128, therefore sub-blocks size ranges from 40 to 512 as there is a maximum of 4 sliding windows per sub-block. Prolog size ranges from 24 to 48 [28].

5.2.6 TCP Processing Delay

The processing blocks shown in Fig. 5.2 (beta, alpha and extrinsic blocks) have a latency of four TCP clock cycles (the TCP is running at a frequency of CPU clock divided by 2). Because each cycle is independent, this allows each block to process 4 sliding windows in parallel [28].

The sliding window processing is being pipelined in each of the blocks, so the performance of the MAP Unit will then depend on the number of sliding windows per sub-block:

- 4 sliding windows per sub-block $\Rightarrow$ 4 TCP cycles are required to process 4 symbols.
- 2 sliding windows per sub-block $\Rightarrow$ 4 TCP cycles are required to process 2 symbols.
- 1 sliding windows per sub-block $\Rightarrow$ 4 TCP cycles are required to process 1 symbols.

Fig. 5.4 shows the number of TCP cycles per symbol in the best-case scenario (*Pro*=24) and Fig. 5.5 the worst-case scenario (*Pro*=48).

EDMA transfer cycles should also be added to the total number of processing cycles for an estimation of the overall processing delay, which depend greatly on the system partitioning and loading. The EDMA transfers are at a maximum rate of 64 bits at a frequency of CPU clock divided by 4. It should be noted that the TCP will start processing as soon as all systematic and parities have been transferred.

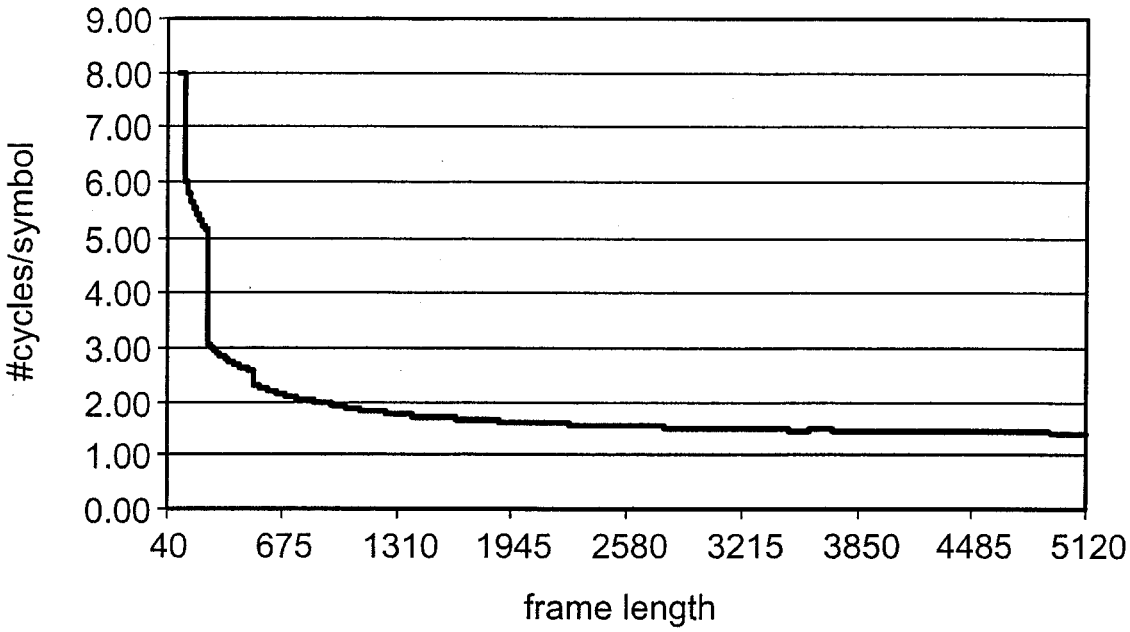

Fig. 5.4: Processing unit performance for Pro=24 (with 4 sliding windows in each sub-block).

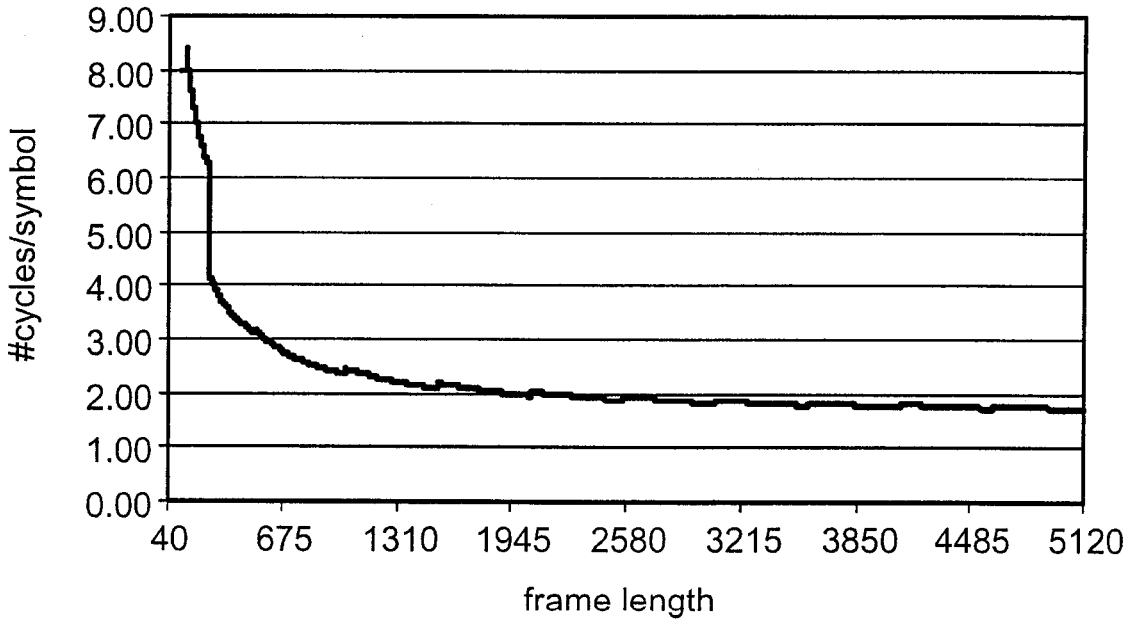

Fig. 5.5: Processing unit performance for Pro=48 (with 4 sliding windows in each sub-block).

5.3 TCP Processing Modes

The TCP has two different processing modes. The TCP operates either as a complete turbo decoder including the iterative structure (standalone processing mode), or it can operate as a single MAP decoder (shared processing mode). In the following, these two processing modes are described in details.

5.3.1 TCP Standalone Mode (SA Mode)

In SA mode, the DSP sends the systematic data, parity data, and the interleaver table. The TCP then works independently of the DSP (standalone), iterates a user-defined maximum number of iterations and outputs the hard decision data. In this mode, minimum DSP processing is required.

A stopping criterion can be enabled to reduce the processing delay, and if the stopping criterion is not satisfied after performing the user defined maximum number of iterations, the TCP will discard this input frame and will start the decoding process on the next frame. The SA block diagram is shown Fig. 5.6.

Systematic and parity information have to be quantized using 8 bits as (5,3) numbers, i.e., *SIIII.FFF* (*S* = sign bit, *I* = integer, *F* = fractional bit). As a result, for every systematic and parity information, the DSP CPU has to send one byte to the TCP. The outputs of the TCP are the hard decision bits and TCP sends eight decoded symbols (bits) in each one byte, to the DSP CPU.

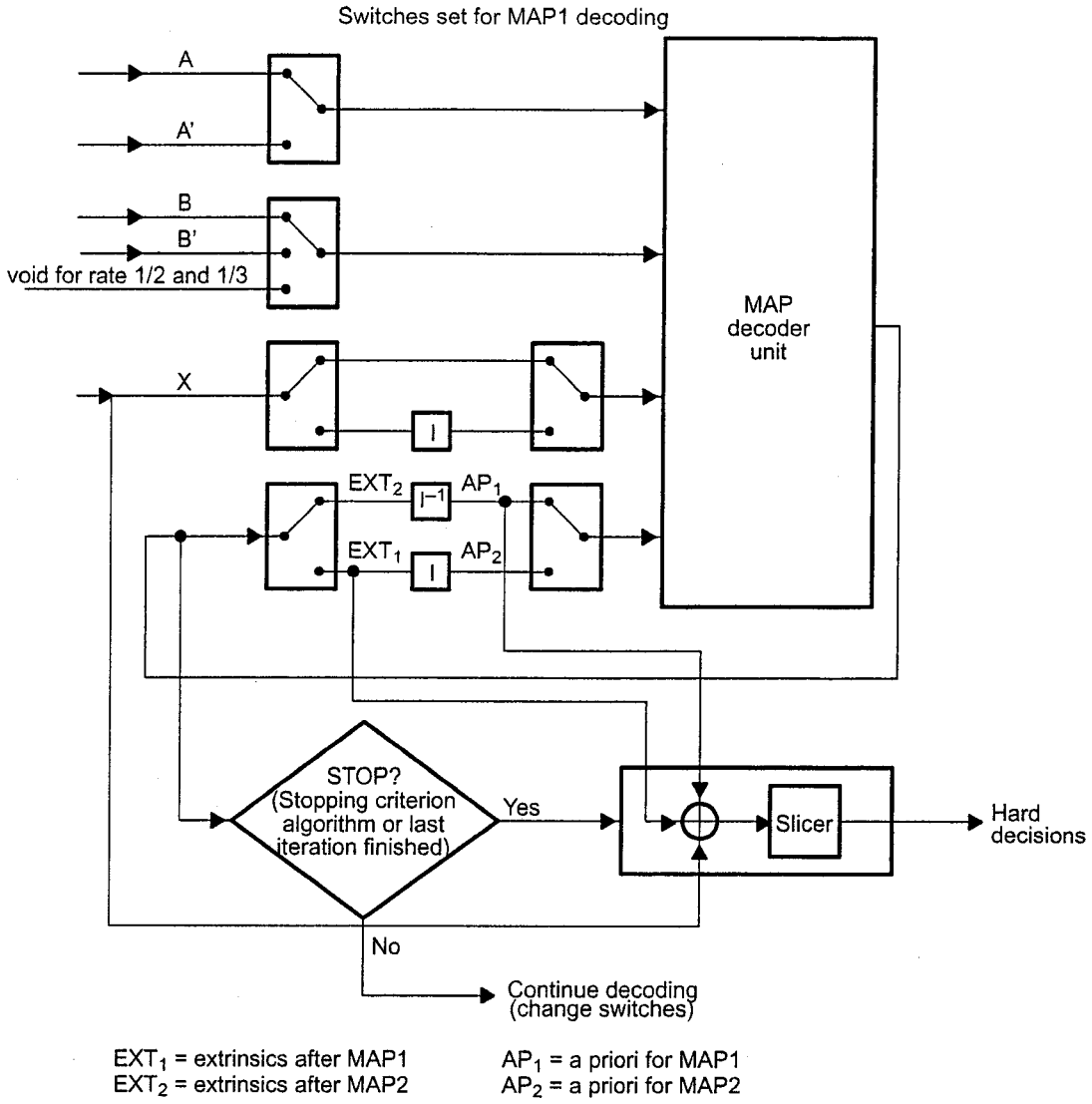

Fig. 5.6: TCP standalone mode block diagram.

5.3.2 TCP Shared-Processing Mode (SP Mode)

In SP mode, the DSP will send systematic, parity and a priori data. The TCP will perform one single MAP decoding and output the extrinsic data. A priori information for MAP1 decoding is obtained by de-interleaving the extrinsic information from the MAP2 decoder, and a priori data for MAP2 decoding is obtained by interleaving the extrinsic information from MAP1 decoder. An overview of the shared processing is shown in Fig. 5.7.

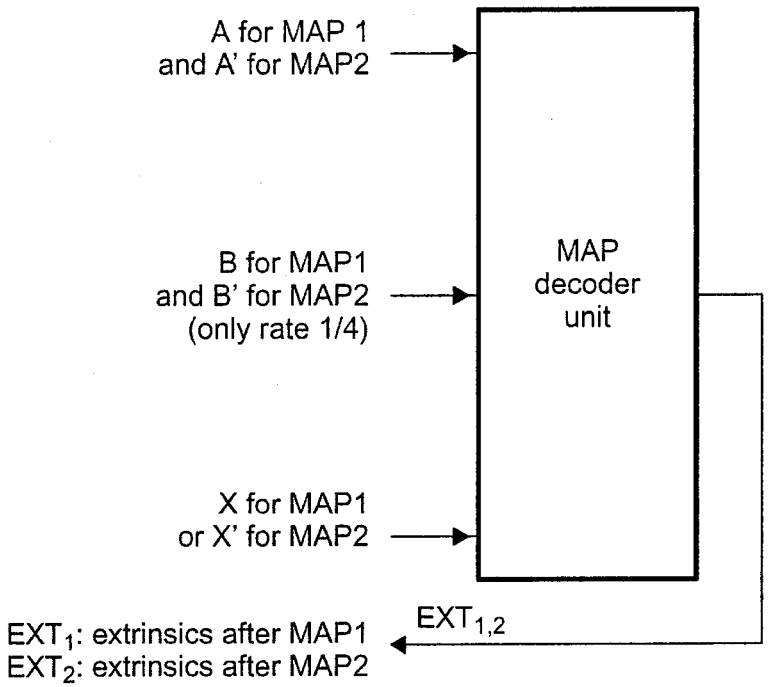

Fig. 5.7: TCP shared-processing mode block diagram.

The DSP CPU is responsible for the interleaving and de-interleaving operations, hard decision calculations and any stopping criterion algorithm. As a result, in this mode, there is too much load on the DSP CPU compared to the standalone mode.

In order to perform one complete iteration, the TCP has to perform two MAP decodings. Two MAP decodings require two complete sets of data transfers including sending information to the TCP and receiving its output. So, using the TCP in the shared processing mode also requires much more data transfers than the standalone mode.

The systematic and the parity information quantization is the same as in the standalone mode, and for every systematic and parity information, the DSP CPU has to send one byte to the TCP. Each input a priori and output extrinsic information is in 8-bit format as (5,2), i.e., *SIIII.FF* (*S* = sign bit, *I* = integer, *F* = fractional bit) and will be right justified with 0 in the most significant bit position. As a result, the DSP CPU has to send

one byte for each a priori information to the TCP and also receive one byte for each extrinsic information output of the TCP.

5.4 Applying Multi-Channel Processing Schemes

In this section, we investigate the implementation issues of all of our multi-channel processing schemes on TMS320C6416 DSP.

5.4.1 Fixed-Iteration Scheme

Based on the specifications of TMS320C6416 DSP, it is very straightforward to implement the fixed-iteration scheme on this CPU. For this purpose, the TCP can be used in the standalone processing mode. The user can define the number of iterations performed on each frame and the stopping criterion must be disabled (by choosing zero as its value). So the DSP CPU performs a fixed number of iterations on each frame and will send the decoded symbols to the DSP CPU.

In the following we will investigate the maximum throughput of the TMS320C6416 DSP implementing this scheme and performing n_{it} iterations in each time slot for an n_{ch}-channel system.

According to Fig. 5.5, in the worst case scenario, each symbol needs approximately two TCP cycles for one MAP decoding, and since each complete iteration includes two MAP decodings, and TCP performs $n_f = \lfloor n_{it} / n_{ch} \rfloor$ iterations on each input frame, each

symbol requires $4\times n_f$ TCP cycles or $8\times n_f$ DSP CPU cycles to be processed (TCP cycle is DSP CPU cycle divided by two).

As explained before, the other important factor for the performance of the TCP is EDMA link, which is capable of transferring 16 bits in each cycle of DSP CPU. In the standalone processing mode, the DSP CPU is sending 8 bits for each systematic information and 16 bits for parity information (two 8-bit information for R=1/3) and it is receiving 1 bit (decoded binary symbol) for each symbol of the input frame.

$$\frac{8+16+1}{16}=1.56 \quad \text{DSP CPU cycle} \tag{5.1}$$

So the EDMA link takes 1.56 DSP CPU cycles for transferring required processing information for each binary symbol. Assuming the maximum throughput of the system is R_b, we will have (DSP CPU clock rate is 600 MHz):

$$\frac{600\times10^6-1.56\times R_b}{8\times\lfloor n_{it}/n_{ch}\rfloor}>R_b \tag{5.2}$$

For example, for a system receiving 5 2.4-Mbps channels ($R_b=5\times2.4=12\,\text{Mbps}$), we will have:

$$\frac{600\times10^6-1.56\times12\times10^6}{8\times\lfloor n_{it}/5\rfloor}>12\times10^6 \tag{5.3}$$

$$\lfloor n_{it}/n_{ch}\rfloor=n_f=6 \quad , \quad n_{it}=30 \tag{5.4}$$

As a result, this system will be capable of performing 30 iterations for 5 input frames in each time slot. If 3GPP turbo code is used for this system, it will have a FER of 3×10^{-2} at SNR=1.0 *dB* (Fig. 4.6).

5.4.2 Serial Processing Scheme

The standalone mode of the TCP is suitable for implementing the serial-processing scheme. As explained before, in this mode, the user can define the threshold of the stopping criterion and also the maximum number of iterations performed on each channel. So in this mode, after processing one channel (either correcting that frame by using the stopping criterion or discarding it after performing maximum number of iterations), the TCP sends the output to the DSP CPU, it receives the next frame form DSP CPU and starts decoding this frame and so on.

Similar to the implementation of the fixed iteration method in the previous section, there is only one set of data transfer by EDMA for processing each frame.

In the following we will investigate the maximum throughput of the TMS320C6416 DSP implementing serial processing scheme and performing n_{it} iterations in each time slot for an n_{ch}-channel system (the same system we used in the previous section).

Here, we assume that the serial processing parameters have been chosen carefully in a way, that the system is using all power of the TCP. So on the average, in each time slot the TCP is performing all its n_{it} iterations to decode the input frames (input symbols) and there is no idle time for the TCP. As a result, on the average, TCP is performing n_f equal to (n_{it}/n_{ch}) iterations on each input frame. Each symbols requires $4\times n_f$ TCP cycles or $8\times n_f$ DSP CPU cycles to be processed (TCP cycle is DSP CPU cycle divided by two).

As explained for the fixed iteration scheme in the previous section, the EDMA link takes 1.56 DSP CPU cycles for transferring required processing information for each

binary symbol. Assuming the maximum throughput of the system is R_b (DSP CPU clock rate is 600 MHz), we will have:

$$\frac{600\times10^6 - 1.56\times R_b}{8\times(n_{it}/n_{ch})} > R_b \tag{5.5}$$

For example, for a system receiving 5 2.4-Mbps channels ($R_b = 5\times2.4 = 12$ Mbps), we will have:

$$\frac{600\times10^6 - 1.56\times12\times10^6}{8\times(n_{it}/n_{ch})} > 12\times10^6 \tag{5.6}$$

$$(n_{it}/n_{ch}) = n_f = 6 \quad , \quad n_{it} = 30 \tag{5.7}$$

As a result, this system will be capable of performing 30 iterations for 5 input frames in each time slot. If 3GPP turbo code is used for this system, it will have a FER of 8×10^{-3} at SNR=1.0 *dB* (Fig. 4.6, serial processing scheme without Mipch).

5.3.4 Parallel Processing Scheme

As discussed in the previous chapters, for a parallel processing system, the TCP must be capable of performing one iteration on each input frame and return the output extrinsic information for the following iterations, which are usually performed after some processing on other frames. Since, in the standalone processing mode, the only outputs of the TCP are hard decision bits, it is not possible to implement the parallel processing scheme in this mode as it is (later we will discuss about some possible hardware modifications).

The parallel processing scheme can be implemented on this DSP using the TCP in shared processing mode. For performing each iteration, the DSP core sends the systematic, parities and input extrinsic information to the TCP, TCP performs one MAP decoding (half iteration) on this input frame and sends back the output extrinsic information to the DSP core, DSP core interleaves the required data for the next half iteration and sends them back to the TCP, TCP performs another MAP decoding (the second half of an iteration) and sends back the output extrinsic information to the DSP core, DSP core de-interleaves the received data from TCP, calculates the likelihood ratio, checks the stopping criterion and if that criterion is satisfied, the DSP core makes the hard decision output and if not, it stores the last received extrinsic information from the TCP for the next iteration on this input frame.

Using the TCP in the shared processing mode has some disadvantages. First of all, there is too much load on the DSP core in this mode (explained in the previous sections), and secondly, there are too many EDMA accesses by TCP and DSP core for even one iteration. In the following we will investigate the throughput of the TMS320C6416 DSP implementing this scheme and performing n_{it} iterations in each time slot for an n_{ch}-channel system.

As we did for serial processing system, we assume that the parallel processing parameters have been chosen carefully in a way that the system is using all the power of the TCP, so on the average, in each time slot, the TCP is performing all its n_{It} iterations to decode the input frames (input symbols) and there is no idle time for the TCP. As a result, on the average, TCP is performing $n_f = (n_{it} / n_{ch})$ iterations on each input frame, each symbol requires $4 \times n_f$ TCP cycles or $8 \times n_f$ DSP CPU cycles to be processed.

In the shared processing mode, the DSP CPU sends 8 bits for each systematic information, 16 bits for parity information (two 8-bit information for R=1/3), 8 bits for the a priori probabilities (or input extrinsic information) and it receives 8 bits for the output extrinsic information for performing one MAP decoding on each symbol of the input frame. Since each iteration includes two MAP decoding, we will have:

$$\frac{2\times(8+16+8)}{16}=4.0 \quad \text{DSP CPU cycle} \tag{5.8}$$

So the EDMA link takes 4 DSP CPU cycles to transfer the required processing information for each binary symbol for each iteration. The TCP performs (n_{it}/n_{ch}) iterations on each input frame, therefore, the EDMA takes $(n_{it}/n_{ch})\times 4$ DSP CPU cycles to transfer necessary data for complete processing of one symbol. Assuming the maximum throughput of the system is bit_{rate}, we will have (DSP CPU clock rate is 600 MHz):

$$\frac{600\times10^6-((n_{it}/n_{ch})\times4\times R_b)}{8\times(n_{it}/n_{ch})}>R_b \tag{5.9}$$

For example, for a system receiving 5 2.4-Mbps channels ($R_b=5\times2.4=12$ Mbps), we will have:

$$\frac{600\times10^6-((n_{it}/n_{ch})\times4\times12\times10^6)}{8\times(n_{it}/n_{ch})}>12\times10^6 \tag{5.10}$$

$$(n_{it}/n_{ch})=n_f=4 \quad , \quad n_{it}=20 \tag{5.11}$$

As a result, this system will be capable of performing 20 iterations for 5 input frames in each time slot. If 3GPP turbo code is used for this system, it will have a FER of 5×10^{-2} at

SNR=1.0 *dB* (Fig. 4.6). The performance of this system has been decreased in comparison to the serial processing scheme because of too many memory accesses.

5.5 TMS320C6416 Hardware Modification

Another approach to apply the parallel processing scheme to this TMS320C6416 DSP is to try to use the TCP in the standalone mode. But, the main problem is that, in this mode, the only output of the TCP is the hard decision.

At this point, we suggest a minor hardware modification easily implementable with current technology. The modification consists in the addition of a programmable parameter in this mode through which, the user can select the hard decision or the soft decision output. If this parameter is enabled, there is no need for the TCP to calculate the hard decisions, therefore, the load on this modified DSP is less than that without hardware modification. To implement the parallel processing scheme on this modified DSP, for each iteration there will be only one set data transfer by EDMA instead of two in the shared processing mode (previous section).

Next, we calculate the maximum throughput of this modified TMS320C6416 DSP implementing the parallel processing scheme and performing n_{it} iterations in each time slot for an n_{ch}-channel system. The steps will be the same as that for the previous section, but the only difference is that instead of two sets of data transfers there is only one set of data transfer for each iteration:

$$\frac{(8+16+8)}{16} = 2.0 \quad \text{DSP CPU cycle} \tag{5.12}$$

So the EDMA link takes 2 DSP CPU cycles to transfer required processing information for one iteration on each binary symbol. We will have:

$$\frac{600\times10^6 - ((n_{it}/n_{ch})\times 2\times R_b)}{8\times(n_{it}/n_{ch})} > R_b \tag{5.13}$$

For example, for a system receiving 5 2.4-Mbps channels ($R_b = 5\times 2.4 = 12$ Mbps), we will have:

$$\frac{600\times10^6 - ((n_{it}/n_{ch})\times 2\times 12\times10^6)}{8\times(n_{it}/n_{ch})} > 12\times10^6 \tag{5.14}$$

$$(n_{it}/n_{ch}) = n_f = 5 \quad , \quad n_{it} = 25 \tag{5.15}$$

As a result, this system will be capable of performing 25 iterations for 5 input frames in each time slot. If 3GPP turbo code is used for this system, it will have a FER of 6×10^{-3} at SNR=1.0 *dB* (Fig. 4.6).

5.6 The New Implementation Scheme

In the previous section, we described how to implement the parallel processing scheme on that modified DSP, but there are still too many sets of memory access (one set for each iteration).

To decrease the number of direct memory accesses on that modified DSP, we introduce a new algorithm which is a combination of parallel processing and serial processing schemes, therefore it has the advantages of both schemes; less number of memory accesses (similar to serial processing scheme) and the same performance as that of

parallel processing scheme. This algorithm is based on the fact that in the parallel processing scheme, decoder always performs more than one iteration (on the average (n_{it}/n_{ch}) iterations) on each input frame, so most of the times, there is no need to send and receive all the required data for each iteration, and those iterations on each frame can be performed serially. In this new scheme, we use a combination of the serial and parallel processing schemes. In the original parallel processing scheme, at each step, turbo decoder performs one iteration on each input frame, but in this new scheme, turbo decoder performs $\lfloor n_{it-a}/n_{ch-u} \rfloor$ iterations on each input frame (the maximum possible iterations at each step), where n_{it-a} is number of available (unused) iterations and n_{ch-u} is the number of undecoded frames in that time slot.

This new implementation scheme is as follows:

1. Initializing: n_{it-a} = *maximum number of iterations* (n_{it}), n_{ch-u} = *number of input frames* (n_{ch});

2. Setting: *Maximum number of iterations per channel* (*Mipch*)=$\lfloor n_{it-a}/n_{ch-u} \rfloor$;

3. Decode one processing step for each not-corrected channel with the modified *Mipch*;

4. Reinitializing: n_{it-a} = *number of unused iterations*, n_{ch-u} = *number of uncorrected channels*;

5. If $n_{it-a} \neq 0$ and $n_{ch-u} \neq 0$, go to (2);

An example of multi-channel processing using this new scheme is shown in Table 5.2. To process the input frames, the implementation of the parallel processing scheme in

shared processing mode of the TCP, requires 30 sets of memory access and on the modified DSP, it takes 30 sets of memory access. But by using this new implementation scheme on the modified DSP, it takes only 8 sets of memory access.

Table 5.2: Example of a multi-channel processing system by using the new implementation scheme.

Round	Ch1	Ch2	Ch3	Ch4	Ch5	It./Round
-	4	8	6	3	9	-
1	0	2	0	0	3	25
2	-	0	-	-	1	4
3	-	-	-	-	0	1

Exact calculation of the maximum throughput of this new system is highly related to the SNR and statistic specifications of the turbo coding standard, which is used in the system. In the following we will investigate the maximum throughput of the modified DSP implementing this new implementation scheme in a special case that on the average, each frame needs two iterations to be decoded, for a system capable of performing n_{it} iterations in each time slot for an n_{ch} -channel system.

The steps are the same as that for the parallel processing applied to the modified DSP CPU. The only difference is the number of data transfers. For example, for a system receiving 5 2.4-Mbps channels performing up to 30 iterations in each time slot, by using computer simulations, there are only 11 sets of memory access in the worst case scenario (11/5=2.2 sets of memory access per frame), we will have:

$$\frac{600\times10^6-(2\times2.2\times R_b)}{8\times(n_{it}/n_{ch})}>R_b \tag{5.16}$$

For example, for a system receiving 5 2.4-Mbps channels ($R_b = 5 \times 2.4 = 12$ Mbps), we will have:

$$\frac{600 \times 10^6 - 2 \times 2.2 \times 12 \times 10^6}{8 \times (n_{it} / n_{ch})} > 12 \times 10^6 \tag{5.17}$$

$$(n_{it} / n_{ch}) = n_f = 5.6 \quad , \quad n_{it} = 28 \tag{5.18}$$

As a result, this system will be capable of performing 28 iterations for 5 input frames in each time slot. If 3GPP turbo code is used for this system, it will have a FER of 3.2×10^{-3} at SNR=1.0 *dB* (Fig. 4.6).

5.7 Summary

In this chapter, we discussed about the implementation issues of the proposed multi-channel processing schemes on a Digital Signal Processing (DSP) chip, "TMS320C6416", for 3GPP turbo coding standard. This DSP CPU has also two embedded coprocessor; Viterbi coprocessor and turbo decoder coprocessor (designed for 3GPP and CDMA2000 coding standards). Then, we introduced a novel and efficient multi-channel processing implementation algorithm, which can be considered as a combination of those proposed multi-channel processing schemes, and we also explained that this scheme can be used in any other hardware implementation of turbo codes.

Chapter 6

CONCLUSION

In a multi-channel processing turbo coding system, turbo decoder is receiving several input channels at the same time, and it has to decode all of them simultaneously. In this thesis, several multi-channel processing schemes were presented including ideal processing schemes, fixed-iteration scheme, serial processing scheme and parallel processing scheme in order to improve the throughput and/or performance of that multi-channel processing system. The main idea in these schemes is to use the turbo decoder idle gained from easily corrected input frames, to decode those frames that require more iterations to be corrected.

The proposed multi-channel processing schemes were applied to 3GPP and DVB/RCS turbo coding standards. Several computer-based simulations for different code rates and SNRs were performed to compare the performance and throughput of the

schemes. It was shown that the parallel processing scheme has a better performance than other schemes and that is also close enough to the ideal processing scheme.

A feasibility study of implementation of all those proposed multi-channel processing schemes was performed on TMS320C6416 DSP. This Texas Instruments (TI) DSP CPU is one of the fastest DSP CPUs in the market, which has also two embedded coprocessors: viterbi coprocessor and turbo decoder coprocessor. Its turbo decoder is capable of processing any 3GPP or 3GPP2 turbo coding standard. The performance and system throughput of this DSP CPU using our processing schemes were investigated and discussed in detail.

It was shown that the implementation of the parallel processing scheme requires too many memory accesses. In order to solve this problem, a minor hardware modification, which can easily be implemented based on the current technology, was suggested.

At the end, based on that modified DSP CPU, a new multi-channel processing implementation scheme was presented. This new scheme is a combination of the parallel processing scheme and the serial processing scheme. This new scheme has the same performance as the parallel processing scheme. However, it has the least number of memory accesses. It was shown that there is only a small loss in throughput because of the data transferring between DSP core and the embedded turbo decoder.

For the future work, it is possible to implement all of those multi-channel processing schemes on the real DSP CPU and FPGA chips.

BIBLIOGRAPHY

[1] C.E. Shannon, "A Mathematical Theory of Communication", *Bell System Technical Journal*, 27:379-423, 623-656, Oct. 1948.

[2] R. G. Gallager, *Information Theory and Reliable Communication*, John Wiley, 1968.

[3] Richard B. Blahut, *Principles and Practice of Information Theory*, Addison-Wesley Publishing Company, 1978.

[4] M. R. Soleymani, Y. Gao and U. Vilaipornsawai, *Turbo Coding for Satellite and Wireless Communications,* Kluwer Academic Publishers, 2002.

[5] C. Heegard and S. B. Wicker, *Turbo Coding*, Kluwer Academic Publishers, 1999.

[6] J. P. Odenwalder, *Optimal Decoding of Convolutional Codes*, Ph.D. Thesis, University of California, Los Angeles, 1970.

[7] S. B. Wicker, *Deep Space Applications*, in *Coding Theory Handbook.* (V. Pless and W. Huffman), Amsterdam, 1998.

[8] M.C. Valenti and J. Sun, "The UMTS turbo code and an efficient decoder implementation suitable for software defined radios", *Int. J. Wireless Info. Networks*, Vol. 8, pp. 203-216, Oct. 2001.

[9] B. Vucetic and J. Yuan, *Turbo codes: Principles and Applications, Boston*, Kluwer Academic Publishers, 2000.

[10] L. 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.

[11] S. Benedetto, G. Montorsi, D. Divsalar and F. Pollara, "A soft-input soft-output maximum a posteriori (MAP) module to decode parallel and serial concatenated codes", *TDA Progress report*, pp. 42-127, 1996.

[12] P. Robertson, E. Villebrun and P. Hoeher, "A comparison of optimal and sub-optimal MAP decoding algorithms operating in the log domain", *Proc. IEEE ICC'95*, Seattle,WA, pp. 1009-1013, June 1995.

[13] J.A. Erfanian, S. Pasupathy and G. Gulot, "Reduced complexity symbol detectors with parallel structures for ISI channels", *IEEE Trans. Commun.* , Vol. 42, pp. 1661-1671, April 1994.

[14] S. Dolinar and D. Divsalar, "Weight distribution for turbo codes using random and nonrandom permutations", *TDA Progress report*, pp. 42-122, Aug. 1995.

[15] J. Hagenauer, P. Robertson and L. Papke, "Iterative Turbo decoding of systematic convolutional codes with the MAP and SOVA algorithms", *ITG Conference on Source and Channel Coding Proc.*, Munich, Oct. 1994.

[16] J. Hagenauer, E. Offer and L. Papke, "Iterative decoding of binary block and convolutional codes", *IEEE Trans. Inform. Theory*, Vol. 42, pp. 429-444, Mar. 1996.

[17] B. Vucetic, "Iterative decoding algorithms", *PIMRC'97*, Helsinki, Finland, pp. 99-120, Sept. 1997.

[18] *Universal Mobile Telecommunicaions system (UMTS): Multiplexing and channel coding (FDD), 3GPP TS 125.212 version 4.0.3*, European Telecommunications Standards Instituite, pp. 14-20, Dec. 2001.

[19] A. Matache, S. Dolinar, and F. Pollara, "Stopping rules for turbo decoders", *JPL TMO Progress Report,*pp. 42-142, August 2000.

[20] K. W. Richardson, "UMTS Overview", *Electronics and Communications Engineering Journal*, June 2000.

[21] C. Douillard, M. Jesequel, C. Berrou, N. Brengarth, J. Tousch and N. Pham, "The Turbo Code Standard for DVB/RCS", *Second International Symposium on Turbo Codes*, Brest, France, 2000.

[22] *Digital Video Broadcasting (DVB); Interaction channel for satellite distribution systems, ETSI EN 301.790 version 1.2.2*, European Telecommunications Standards Instituite,pp. 20-26, Dec. 2000.

[23] *TMS320C6416: Fixed-Point Digital Signal Processor Data Sheet*, Texas Instruments Inc., Feb. 2001.

[24] *TMS320C6000 CPU and Instruction Set Reference Guide*, Texas Instruments Inc., Oct. 2000.

[25] *TMS320C6000 Peripherals Reference Guide*, Texas Instruments Inc., Feb. 2001.

[26] *TMS320C64x Technical Overview*, Texas Instruments Inc., Jan. 2001.

[27] *Using TMS320C6416 Coprocessors:Viterbi Coprocessor (VCP)* , Texas Instruments Inc., Mar. 2003.

[28] *Turbo Decoder Coprocessor User's Guide*, Texas Instruments Inc., Aug. 2001.

[29] *Using TMS320C6416 Coprocessors: Turbo Coprocessor (TCP)*, Texas Instruments Inc., June 2001.