TELECOM PARIS
École Nationale Supérieure des Télécommunications

Thèse

**présentée pour obtenir le grade de docteur
de l'École Nationale Supérieure
des Télécommunications**

Spécialité : Électronique et Communication

Luis Fernando González Pérez

Architectures VLSI pour le Codage Conjoint Source-Canal en Treillis

ENST
A soutenir le 26 Octobre devant le jury composé de

Benoît Macq	Rapporteurs
Bruno Rouzeyre	
Pierre Duhamel	Examinateurs
Jorge Rogríguez Guisantes	
Catherine Lambert-Nebout	
Roger Reynaud	
Emmanuel Boutillon	Directeur de Thèse

École Nationale Supérieure des Télécommunications

To my family

Remerciements

Abstract

Conventionally, a digital communication system is composed of a source coder (reduction of the information to be transmitted) and a channel coder (protection against channel errors). This approach allows on the one hand to divide the transmission of information into two independent tasks, and on the other hand, to operate close to the theoretical limits for a given channel SNR. Nevertheless, this tandem system exhibits a high overall complexity and in addition, it suffers from a dramatic performance degradation when the channel conditions are poor. In this thesis, we focus on a joint source-channel trellis coding technique from both a theoretical and hardware standpoints. Our goal is to implement a simple and robust coding system for a large range of channel SNR and which could replace more efficiently the tandem system in certain applications.

The main goal of this joint source-channel trellis coding technique is to find a representation of the source sequence which minimizes the expectation of the distortion between the source sequence and the reproduction sequence decoded at the receiver end. This minimization is accomplished by a codebook design algorithm which takes into account the channel distribution during the generation of the reproduction codebook, and by the fidelity criterion employed during the quantization of the source.

In the first part of this work, we show that the type of computations required for the codebook design operation and the branch metrics of the Viterbi algorithm are quite similar. Upon proposing a simplification in the computations of the distortion measure, a reconfigurable architecture is presented which allows to implement both the codebook design algorithm and the quantization process within the same architecture.

In the next stage of this work, complexity reductions were investigated by replacing the Viterbi algorithm with a suboptimum trellis search. We showed that the M algorithm, when used in the context of joint source-channel trellis coding, presented excellent complexity-performance trade offs. Then, we focused on the design of VLSI architectures for the M algorithm, and two new ideas which are potentially more advantageous than previously reported work are proposed. The first one profits from the trellis structure to reduce up to 50% the hardware complexity of the sorting networks required by this algorithm. The second idea consists in the adaptation of the trace-back technique employed in Viterbi decoders to the M algorithm with the use of pointer tables.

Finally, in the last part of this work, a joint source-channel coding technique is presented consisting in the joint optimization of the trellis quantizer described above and a convolutional code. The pairwise error probabilities required to perform the codebook design operation and the source quantization are derived from the generator polynomials of the convolutional code. Then, at the decoder end, we propose to decode the received channel sequence by means of the MAP algorithm. This algorithm provides us with the a posteriori probabilities of each trellis branch, or equivalently, with the probabilities of decoding a given codeword from the reproduction codebook. We called this a “soft” source decoding.

Glossary of Abbreviations

JSCC	Joint Source-Channel Coding
JSCTC	Joint Source-Channel Trellis Coding
OPTA	Optimum Performance Theoretically Achievable
TQ	Trellis Quantization
VQ	Vector Quantization
LBG	Linde-Buzo-Gray
MAP	Maximum A Posteriori
APP	A Posteriori Probabilities
BCJR	Bahl-Cocke-Jelinek-Raviv
VA	Viterbi Algorithm
VA	M Algorithm
MA	M Algorithm
TCQ	Trellis Coded Quantization
TCM	Trellis Coded Modulation
CELP	Code Excited Linear Prediction
MSE	Mean Square Error
SNR	channel Signal to Noise Ratio
SQR	Signal to Quantization noise Ratio
BSC	Binary Symmetric Channel
AWGN	Additive White Gaussian Noise
BPSK	Binary Phase Shift Keying
FSM	Finite State Machine
LUT	Look-Up Table
CL	Computational Load
MSB	Most Significant Bit
LSB	Least Significant Bit

RE	Register Exchange
TB	Trace-Back
PM	Path Metric
BM	Branch Metric
ACS	Add-Compare-Select
BMU	Branch Metric Unit
ACSU	Add-Compare-Select Unit
SMU	Survivor Memory Unit
LIFO	Last In First Out
FIFO	First In First Out
VLSI	Very Large Scale Integration
FPGA	Field Programmable Gate Array
DSP	Digital Signal Processor
TTL	Transistor Transistor Logic
CER	Comparison-Exchange-Rejection
MRO	Merging Rejection Operator

Glossary of Symbols

k	time index
K	trellis constraint length
$\mathcal{X}$	source alphabet
$\mathbf{x}$	source sequence
x_k	source symbols
$\mathcal{C}$	reproduction codebook
$\mathbf{y}$	reproduction sequence
y_i	reproduction codeword
$\hat{\mathbf{x}}$	decoded source sequence
$\hat{x}_k$	decoded source symbol
l_i	binary label associated to codeword y_i
$\mathbf{u}$	binary sequence
u_k	binary symbol
$\mathbf{v}$	transmitted channel sequence
v_k	transmitted channel symbol
$\hat{\mathbf{v}}$	received channel sequence
$\hat{v}_k$	received channel symbol
$p(\cdot)$	probability density function
R_s	source code rate
R_c	channel code rate
C	channel capacity
W	bandwidth
P_{av}	average signal power
N_o	noise spectral density
p	channel transition error probability
$R(D)$	rate-distortion function
$R(C)$	OPTA function
$I(X,Y)$	mutual information
D	distortion

$d(x,y)$	per-letter distortion measure
$d(i,j)$	hamming distance
$Q(x)$	quantization function
$Q(x)^{-1}$	decoding function
Q_i	set of source symbols (vectors) coded by y_i
$\|\|Q_i\|\|$	cardinality of Q_i
σ^2	variance
w	additive white gaussian noise variable
$Pr(y_i \vert y_j)$	probability of decoding codeword y_j given that codeword y_i was sent
L_{TS}	length of the training sequence
L	decoding depth in trellis search algorithms
S_i	*ith* trellis state
M	number of surviving paths in a trellis search algorithm

Contents

List of Figures

List of Tables

Introduction

The main goal of a digital communication system consists in both looking for efficient ways to represent the information to be transmitted and to render this representation robust to the corrupting effects of the transmission channel. The usual approach to accomplish these two goals is the well known “divide et impera” principle: the information transmission problem is divided into two tasks, source coding and channel coding. The former seeks to represent the information to be transmitted with the least possible number of bits whereas the latter must guarantee a reliable communication between the transmitter and the receiver. This is achieved by introducing additional information to the compressed source so that the receiver can recover the original information from a compressed and noise-corrupted version of it.

This task separation is due to Shannon’s pioneering work which stablished the foundations of what is now known as Information Theory. C. E. Shannon demonstrated that, under certain assumptions, this separation allows the design of optimum-performance communication systems. Nowadays, we witness the optimality of these assumptions. Indeed, today’s advances in both information theory and semiconductor technology permit the design of communication systems which allow in turn to transmit information with performances that approach the theoretical limits predicted by Shannon’s work. These systems have reached a certain maturity that has dramatically increased the services that can be provided by them. Satellite and mobile communications, internet and mutimedia applications have had an amazing developement in the last five years thanks to the great advances in source and channel coding. Source coders such as vocoders, CELP coders, JPEG, MPEG2 and MPEG4 standards; and channel coding tecniques such as coded modulation, turbo codes, turbo-equalization and code division multiple access, constitute few examples of the potential techniques which facilitate the approach towards the Shannon limits.

Nevertheless, this apogee in the services provided by present time communication systems causes in turn a strong demand to make data transmission more efficient and provide greater performance. In particular, environments such as mobile satellite systems require minimal power usage and minimal bandwidth usage to maximize the number of users, which restricts the use of channel coding. On the other hand, its time-varying nature causes bursty errors which, if mitigated by interleaving, would induce lengthy delays. As a consequence, it seems that communication systems design is close to its upper limit.

This is because Shannon's separability theorem has been used as a design principle, motivating always the concatenation of separately optimized source and channel coders. However, the theorem assumes that the source coder is an optimal one that removes all source redundancy. Moreover, it assumes that for rates below channel capacity, the channel coder corrects all errors. Such optimal systems can only be achieved in general by allowing limitless encoding/decoding complexity and delay. Practical systems must limit complexity and delay, and thus sacrifice performance. In many cases it is not reasonable to assume then that the conditions for the separation theorem hold even approximately. For instance, for several noisy channels, channel codes may fail to reduce errors and may in fact, increase the bit error rate. Moreover, the output of practical source encoders may contain a significant amount of redundancy, specially for sources with memory such as speech and images. In this circumstances, we can potentially improve performance by considering the souce and channel designs jointly.

Since the late seventies, researchers started to realized this last problem and wondered if a joint design could not lead to more efficient and less complex systems. This question was taken into account more seriously in the following years and Joint Source-Channel Coding was born. Since the last decade, JSCC has attracted so much attention that today, we can find a wide range of applications where the concepts and underlying principles of JSCC are proposed. Nonetheless, these systems are still confined to the pure research stage and no or very little hardware implementations of JSCC systems have been proposed. This thesis is intended to start contemplating the possibility of designing hardware architectures for JSCC.

In particular, this thesis is concerned with the design of hardware architectures for Joint Source-Channel Trellis Coding. This technique consists in the design of trellis quantization algorithms that are robust to channel errors. The main characteristic of this technique is that source and channel protection are indeed merged into a single operation. The goal of this system is the complete elimination of the channel coding scheme or at least to reduce its hardware complexity. With this approach we seek new alternatives to the conventional tandem scheme, presenting a reduced hardware complexity system, with a smooth degradation of its overall performance, as opposed to the tandem system where the optimality is obtained for a very small range of channel conditions.

The methodology employed in this thesis consisted in three main parts. First, a study on the hardware requirements of the JSCTC technique was peformed, and a comparison to conventional systems was done from both performance and complexity standpoints. Then, suboptimal architectures were studied which allow a significant reduction in the hardware complexity while maintaining the overall performance as close as possible to the optimum system. This way, its hardware implementation in medium- to low-complexity circuits such as digital signal processors, field programmable gate arrays, etc. can be accomplished. Finally, the third step consisted in further studying this technique, this time from a theoretical point of view, in order to seek for other means to improve its overall performance.

This work is a continuation of a series of research activities at the COMELEC department focused on JSCC.

Thesis Outline

The thesis is organized as follows. In Chapter 1, the underlying principles of a conventional communication system are presented. The theoretical limits given by the rate-distortion theory and channel coding theory are highlighted. Then, their relationship to the underlying theory of JSCC is described.

Chapter 2 presents the theory behind Joint Source-Channel Trellis Coding. The working operation of this technique is presented and compared to both non-joint trellis quantization techniques over noisy channels and a tandem system comprising a non-joint trellis quantizer and a convolutional code.

Every quantization technique uses a reproduction codebook to represent the input source. This codebook needs to be optimized to account for the statistical distribution of the input source. In addition, when lossy compression techniques are considered, a distortion measure is required to assess the performance of the source coding technique. In Chapter 3, hardware architectures are presented for the implementation of both the codebook design algorithm and the distortion measure employed to encode the input source in the context of joint source-channel trellis coding.

As its name suggests, the core of trellis quantizers is a trellis search algorithm. In Chapter 4, a study of suboptimal trellis search algorithms is presented so as to reduce the hardware requirements of the joint source-channel trellis coding tecnique. The goal of this study is to replace the optimum trellis search (Viterbi algorithm) by a suboptimal search with the best trade off performance-hardware complexity.

Chapter 5 presents hardware architectures for the selected suboptimum trellis search algorithm. The trellis search algorithm that we have selected is the M algorithm. This algorithm is very similar to the Viterbi algorithm except for the number of surviving paths retained. In this algorithm, only the best M paths are retained at each trellis stage. This process of selecting the best M paths implies the use of sorting circuits. In this chapter, new methods to perform the selection of the M best paths are presented which take advantage of the trellis structure in order to reduce the hardware requirements of the sorting architecture. In addition, a new method for survivor memory management based on the Trace-Back algorithm is described. We will show that, as in the case of Viterbi decoders, the Trace-Back approach results in more efficient architectures than the register exchange procedure.

After the architectural study of the joint source-channel trellis coding technique, we propose in chapter 6 to concatenate this robust quantization method with a convolutional code. The robust trellis source coder takes into account the error probability of the convolutional code to optmized its reproduction codebook and to perform the quantization of the source. On the other hand, a new approach for computing the error probaility needed during the robust source quantization is presented which avoids the extensive Monte-carlo simulations of previous reported work. Nevertheless, this channel estimation is not adapted to channels with time varying statistics such as the rayleigh fading channel.

In addition, at the decoder end, an exploration of Maximum A Posteriori decoding in the context of source reconstruction is considered. With source distortion being the performance criterion, the MAP algorithm is used not to correct transmission errors but to provide channel a posteriori information for decoding the source. This "soft" source decoding allows to improve the performance of the overall system.

Finally, concluding remarks are given and future work to be explored is hightlighted.

Published Work

* Luis Gonzalez and Emmanuel Boutillon, *"VLSI Architecture for Joint Source-Channel Trellis Coding"*, 42nd Midwest Symposium on Circuits and Systems, Las Cruces, New Mexico, USA, August 1999.

* Luis Gonzalez and Emmanuel Boutillon, *"Architecture VLSI pour le Codage Source-Canal Conjoint en Trellis"*, 5th Workshop on Algorithm-Architecture Adequation, Rocquencourt, Frace. January 2000.

* Luis Gonzalez and Emmanuel Boutillon, *"A Study of a Suboptimal Architecture for Joint Source-Channel Trellis Coding"*, 2000 International Symposium on Circuits and Systems (ISCAS2000), Geneva, Switzerland, May 20000.

* Luis Gonzalez and Emmanuel Boutillon, *"Simplified Path Metric Updating in the M Algorithm for VLSI Implementation"*, 2000 International Conference on Acoustics, Speech and Signal Processing (ICASSP2000),Istambul, Turkey, June 2000

* Emmanuel Boutillon and Luis Gonzalez *"Trace Back Techniques Adapted to the Surviving Memory Management in the M Algorithm"*, 2000 International Conference on Acoustics, Speech and Signal Processing (ICASSP2000),Istambul, Turkey, June 2000

* Emmanuel Boutillon and Luis Gonzalez *"Joint Source-Channel Trellis Coding using Convolutional Codes' Partial Transfer Function and the BCJR Algorithm"*, 2nd International Symposium on Turbo Codes & Related Topics, Brest, France, September 2000

Chapter 1

Joint Source-Channel Coding: Fundamentals and State of the Art

In this Chapter, the theoretical background on digital communication systems related to Joint Source-Channel Coding (JSCC) is presented. The notations and associated terminology that will be used throughout this thesis is outlined and the fundamental results leading to source and channel coding are presented. Specifically, upper bounds such as the rate-distortion function, the channel capacity and the Optimum Performance Theoretically Attainable (OPTA) [52] are unveiled. Finally, a brief overview on the state of the art on joint source and channel coding is given.

1.1 Digital Communication System

A classical digital communication systems is illustrated in figure 1.1. The purpose of every communication system is the transmission of the information generated by the *source* to a *destination*. The source may be speech, audio or video signals. In a digital communication system, these kind of signals are converted into binary sequences and the aim is to represent the source signals by as few bits as possible. We say in this case that the goal is to obtain a binary sequence representing the source with little or no redundancy. This operation is accomplished by the *source encoder*.

The binary sequence issued from the source encoder is passed to the *channel encoder*. The channel encoder introduces redundancy in a controlled and deterministic way so that the receiver may overcome the disturbing effects of the transmission channel onto the binary sequence generated by the source encoder.

The coded binary sequence is passed to the *modulator* which maps the binary sequence into signal waveforms suited to be transmitted through the channel. The *transmission channel* is the physical medium that is used to send the signal from the source to the destination. The transmission channel can be either free space, wire lines, optical fiber, etc. The main feature of the channel is that of corrupting the transmitted signal in a random manner.

At the receiver, the *demodulator* detects the corrupted transmitted waveforms and maps them into a sequence of bits which are estimates of the binary sequence at the input of the modulator. Then, the *channel decoder* takes this sequence and attempts to reconstruct the original binary sequence from knowledge of the controlled redundancy that was introduced at the encoder end.

Finally, the binary sequence found by the channel decoder is passed to the *source decoder* which converts the binary sequence into the signal generated by the information source. This reconstruction of the source is made with knowledge of the algorithm used to encode the source. Due to the channel decoding errors, the signal at the output of the source decoder is an approximation of the original source.

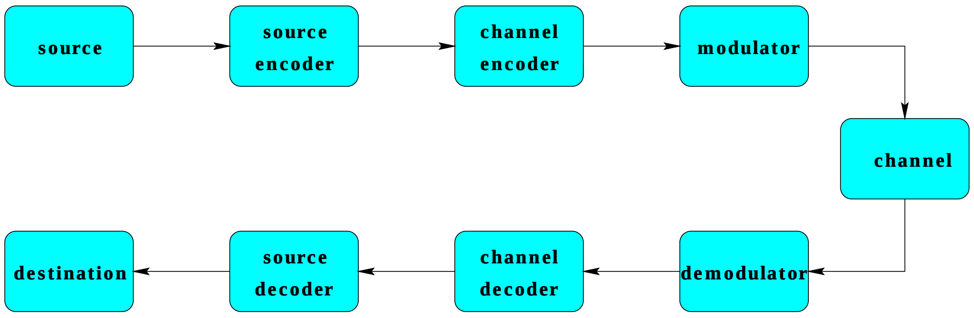


Figure 1.1: Conventional communication system.

The source and channel encoder-decoder pairs need some measures of performance so as to know the efficiency of their operation. In the following, these performance criteria and theoretical limits are outlined for both coders. It must be pointed out that the goal of the next sections is not to provide a detailed description of source and channel coding but to recall the fundamental results. For a detailed derivation of these results, the reader is referred to [14, 51, 52, 69, 76, 89, 116].

1.2 Source Coding

Source coding or signal compression can be divided into two major areas: lossless compression and lossy compression. Lossless compression techniques involve no loss of information, that is, the original source can be recovered **exactly** from the compressed data. Lossless compression is generally used for discrete data, such as text, computer data and some kinds of image and video.

Since lossless compression implies the exact reconstruction of the source, Shannon showed that the best that a lossless compression scheme can do is to encode the output of a source with an average number of bits equal to its entropy. Remember that the entrooy of a source is defined as the average number of bits needed to code the source output [51].

Lossy compression, on the other hand, involves some loss of information, meaning that the original source **cannot** be recovered exactly from the compressed data. Nonetheless, by accepting this loss of information in the reconstructed source, it is possible to obtain much higher compression ratios than is possible with lossless compression. In this thesis, only lossy compression is considered.

The basic definitions and notations of a source coder are described in figure 1.2. The source to be encoded can be modeled either as continuous random variable X with a probability density function $p(x)$ or as a discrete random variable X which delivers source symbols x from an alphabet $\mathcal{X} = \{x_1, x_2, \cdots x_M\}$ with M possible symbols. Each symbol has a probability of occurrence $p(x_i)$.

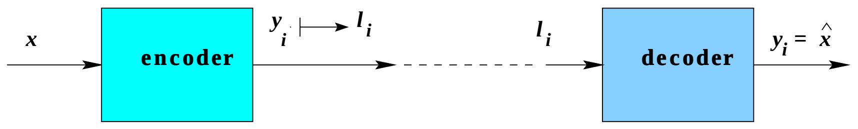


Figure 1.2: Source coder-decoder pair.

The source encoder takes the source symbol(s) and produces the compressed representation Y. To do so, the encoder uses its own alphabet or **codebook** $\mathcal{C} = \{y_0, y_1, \cdots y_{N-1}\}$ and selects the letter or **codeword** $y_i = \hat{x}$ which represents in the best manner the source symbol x. In addition, a function $\mathcal{F} : y_i \mapsto l_i$ is used to map the reproduction codeword y_i to a binary label l_i which is used to represent the reproduction codebook y_i. Since the function $\mathcal{F}$ is a one-to-one mapping, there is no loss of information by representing codeword y_i by the binary label l_i. This binary label is the information that will be transmitted to the receiver. The length of the binary label is $n = \log_2 N$ bits.

A measure of the compression achieved by a given source coder can be defined as the number of bits required to represent a source symbol. This measure of compression is known as the **code rate** R_s. For the encoder described above, $R_s = \log_2 N$ bits.

At the receiving end and assuming error-free transmission, the transmitted and received binary labels are the same ($\hat{l}_i = l_i$). Hence, the source decoder takes this binary label and reconstitutes the codeword $y_i = \hat{x}$ which is an approximation of the source symbol x.

1.2.1 Distortion Measures

As indicated earlier, since lossy compression implies a loss of information during source encoding, a measure of the closeness or fidelity of the reconstructed source sequence with regard to the original source sequence is necessary. These measures are called distortion measures. A natural thing to do when looking at the fidelity of the reconstructed source is to measure the difference between the original source symbol and the reconstructed source symbol. The measure of the distortion introduced by the reconstructed source in a symbol-by-symbol basis is called **per-letter distortion**. The most common per letter distortion measures are the squared error or **euclidean distance** and the absolute difference. The squared error measure is given by

$$d(x, y) = (x - y)^2 \tag{1.1}$$

and the absolute difference measure is given by

$$d(x, y) = |x - y| \tag{1.2}$$

In general, it is difficult to examine the difference on a symbol-by-symbol basis. Therefore, average measures are used to summarize the information in the difference sequence. The most popular average measure is the average of the square error measure and is known as the **Mean Squared Error** (MSE). The MSE is defined as

$$MSE = E\{(x - y)^2\} \tag{1.3}$$

Assuming that the source X verifies the hypotheses of stationarity and ergodicity [52, 91], the MSE can be rewritten as

$$MSE = \frac{1}{L} \sum_{k=1}^{L} (x_k - y_k)^2 \tag{1.4}$$

where L is the length of the source sequence.

When the distortion introduced by the encoding process is measured with regard to the original source, the ratio of the average squared value of the source and the MSE are used. This is called the **Signal to Quantization noise Ratio** (SQR) and defined as

$$SQR = \frac{\sigma_x^2}{MSE} \tag{1.5}$$

where σ_x^2 is the average squared value of the source. Expressed in decibels, the SQR is given by

$$SQR(dB) = 10\log_{10}\frac{\sigma_x^2}{MSE} \tag{1.6}$$

1.2.2 Rate-Distortion Function

Rate-distortion theory [14] stablishes the relationship between the source coding rate R_s of a given source coder and the minimum distortion D that can be achieved with this rate. The function relating these two parameters is called **rate-distortion function**.

Consider a source X that has a probability density function $p(x)$, a reconstructed source Y belonging to a reproduction codebook $\mathcal{C}$ and a per-letter distortion $d(x,y)$ where $x \in \mathcal{X}$ and $y \in \mathcal{C}$. The rate-distortion function is defined as the minimum rate in bits per source symbol that is required to represent the source X with a distortion less than or equal to D. Mathematically, this can be expressed as

$$R(D) = \min_{p(y|x):E\{d(x,y)\}\leq D} I(X;Y) \tag{1.7}$$

where $p(y|x)$ is the probability of encoding source symbol x by the reproduction codeword y and $I(X,Y)$ is the average mutual information between X and Y. The mutual information for discrete sources is defined as [14]

$$I(X;Y) = \sum_{i=0}^{M-1}\sum_{j=0}^{N-1} P(x_i,y_j)\log\frac{P(x_i|y_j)}{P(x_i)} \tag{1.8}$$

whereas that for continuous sources is

$$I(X;Y) = \int\int p(x,y)log_2\frac{p(x,y)}{p(x)p(y)}dxdy \tag{1.9}$$

From this result, the rate-distortion function of a continuous gaussian source with variance σ_x^2 is given by [14, 89, 99]

$$R_s(D) = \max\left\{0, \frac{1}{2}\log\frac{\sigma_x^2}{D}\right\} \tag{1.10}$$

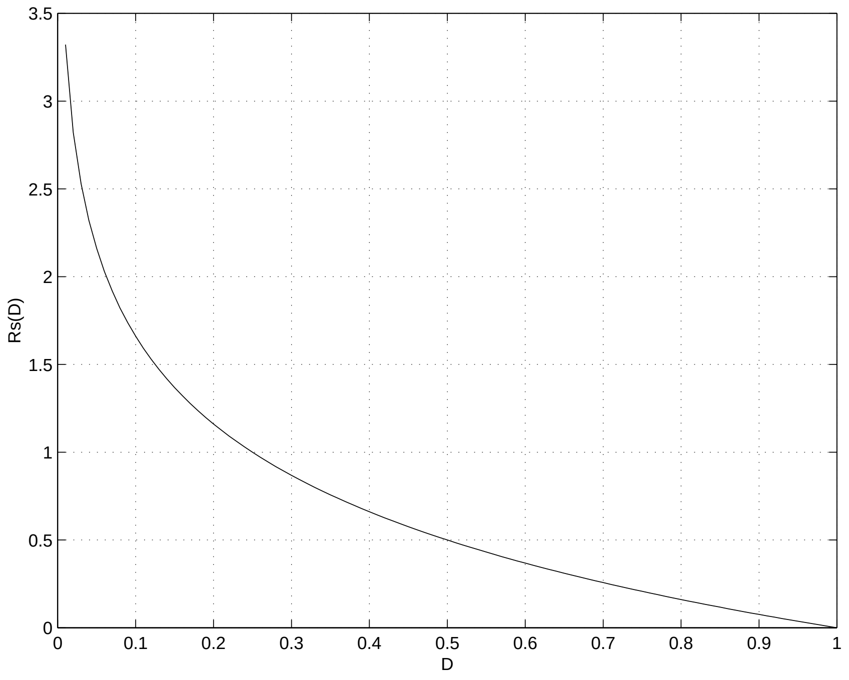


Figure 1.3: Rate-distortion function of a gaussian source.

Figure 1.3 plots the rate-distortion function of a unit variance gaussian source.

The rate distortion function of a source is associated to the source coding theorem stated by Shannon [89]. This theorem states that:

> *"Under certain hypothesis, there exists and encoding scheme that maps the source output into reproduction codewords such that for any given distortion D, the minimum rate $R_s(D)$ bits per symbol is sufficient to reconstruct the source output with an average distortion that is arbitrarily close to D."*

The rate-distortion function gives us, thus, a lower bound on the source rate that is possible for a given level of distortion.

In practical situations, there exist source coding procedures that attempt to reach the theoretical limit given by the rate distortion function. These procedures are called quantization algorithms. Quantization algorithms can be divided into two main classes: scalar quantization and vector quantization. In the following, these two clases are treated.

1.2.3 Scalar Quantization

In scalar quantization, the source sequence X is quantized in a symbol-by-symbol basis. The quantizer consists of two mappings: an encoder mapping $Q(x)$ and a decoder mapping

$Q^{-1}(x)$. Suppose the input sequence takes values from the real numbers ($X \in \Re$) and the quantizer outputs are drawn from the reproduction codebook $\mathcal{C}$ with N reproduction codewords y_i. The encoder divides the real line into N intervals or **partitions** Q_i ($i = 0 \cdots N$). Each partition is represented by a distinct codeword y_i. The encoder represents all the source symbols that fall into a particular interval by the reproduction codeword representing that interval, as shown in figure 1.4. As indicated above, the reproduction codewords can be represented in turn by a binary label l_i. This way, all the source symbols represented by the same reproduction codeword y_i will have the same binary label l_i.

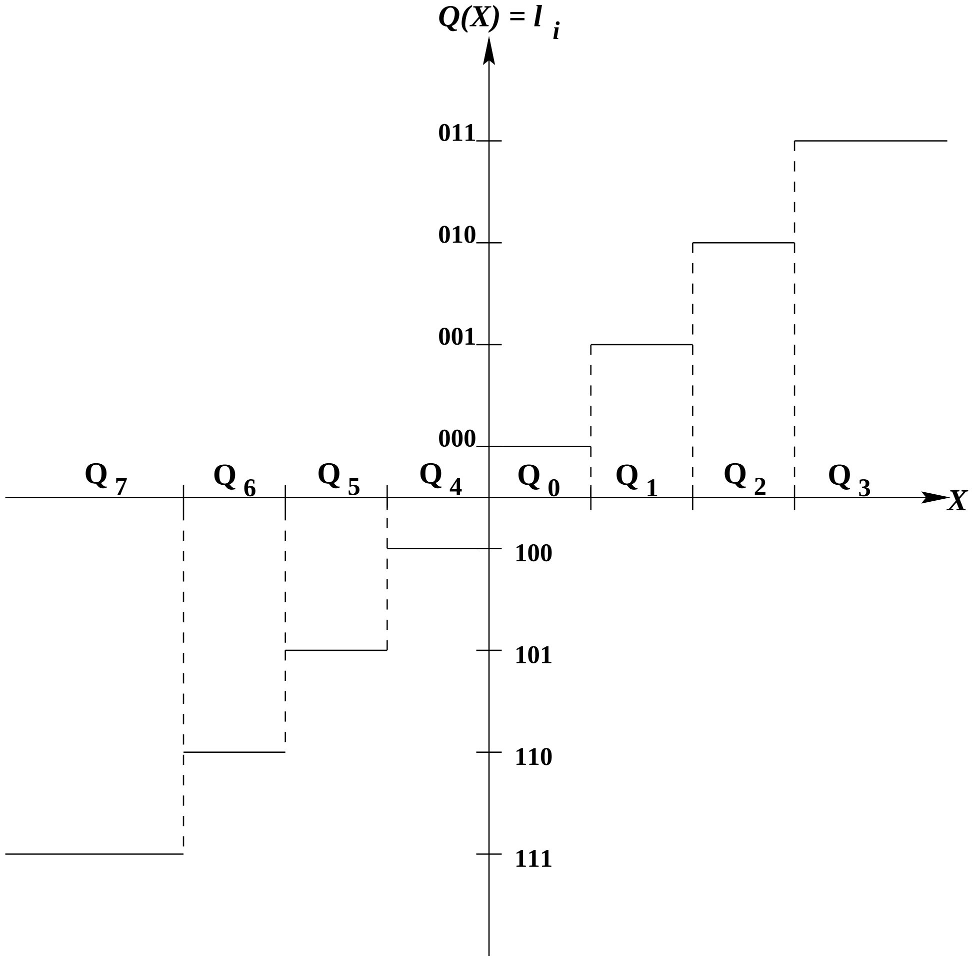


Figure 1.4: Scalar quantizer with 8 reproduction levels.

In order to reconstruct the input symbol, the decoder mapping retrieves from the reproduction codebook $\mathcal{C}$, the reproduction codeword y_i given by the binary label l_i. Since that codeword represents an entire interval, there is no way to know the exact value of the source symbols x_k of time k. As a result, the loss of information is irreversible. Consequently, the codeword y_i must be chosen in such a way that the distortion introduced by this codeword in representing the source is minimized.

The problem of finding the optimum values of the reproduction codebook can be solved by optimizing the encoder and decoder mappings. This problem was considered independently by Lloyd [70] and Max [72] and the resulting optimum quantizer is called the Lloyd-Max quantizer. They proposed an iterative algorithm for optimizing the reproduction codewords either from the knowledge of the source statistical distribution or from a training sequence which is considered of being representative of the source to be quantized. This algorithm consists of two steps:

1. **nearest neighbor condition**: each source symbol x_k falls into the encoding interval Q_i represented by codeword y_i according to the condition

$$d(x_k, y_i) < d(x_k, y_j), \quad \forall j \neq i \tag{1.11}$$

2. **centroid computation**: the optimum reproduction codeword y_i for a given partition Q_i is the centroid of all the input symbols that fell into that interval during the encoding process. This centroid is given by

$$y_i = E\{\mathbf{x} | \mathbf{x} \in Q_i\} \tag{1.12}$$

   For equally likely source symbols and squared error per-letter distortion measure, the centroid is given by

$$y_i = \frac{1}{||Q_i||} \sum_{j:x_j \in Q_i} x_j \tag{1.13}$$

   that is, it is simply the average of all the source symbols belonging to a given partition Q_i.

These steps are performed alternately until the distortion introduced by the reproduction codebook is acceptable. The process of optimizing the values of the reproduction codewords is called **codebook design**.

1.2.4 Vector Quantization

Unlike scalar quantization where the source sequence is encoded one symbol at a time, in vector quantization the source is divided into vectors $\mathbf{x}$ of M input symbols and each block is encoded into a single binary label l_i. Each binary label l_i is assigned to a reproduction **codevector** $\mathbf{y}_i$ of dimension M. Notice that the reproduction codebook is still constituted by N reproduction codewords. The difference is that now, each reproduction codevector has dimension M. The encoder and decoder mappings work in the same manner as the encoder and decoder mapping of scalar quantizers but translated to the vector domain. That is, the source sequence takes values from $\Re^M$ and the encoder divides this M-dimensional space into N cells or **Voronoi regions**. Each Voronoi region is represented by a codevector $\mathbf{y_i}$ and by a binary label l_i, as shown in figure 1.5. The rate of the vector quantizer is now given as $R_s = \frac{\log_2 N}{M}$ bits per source symbol.

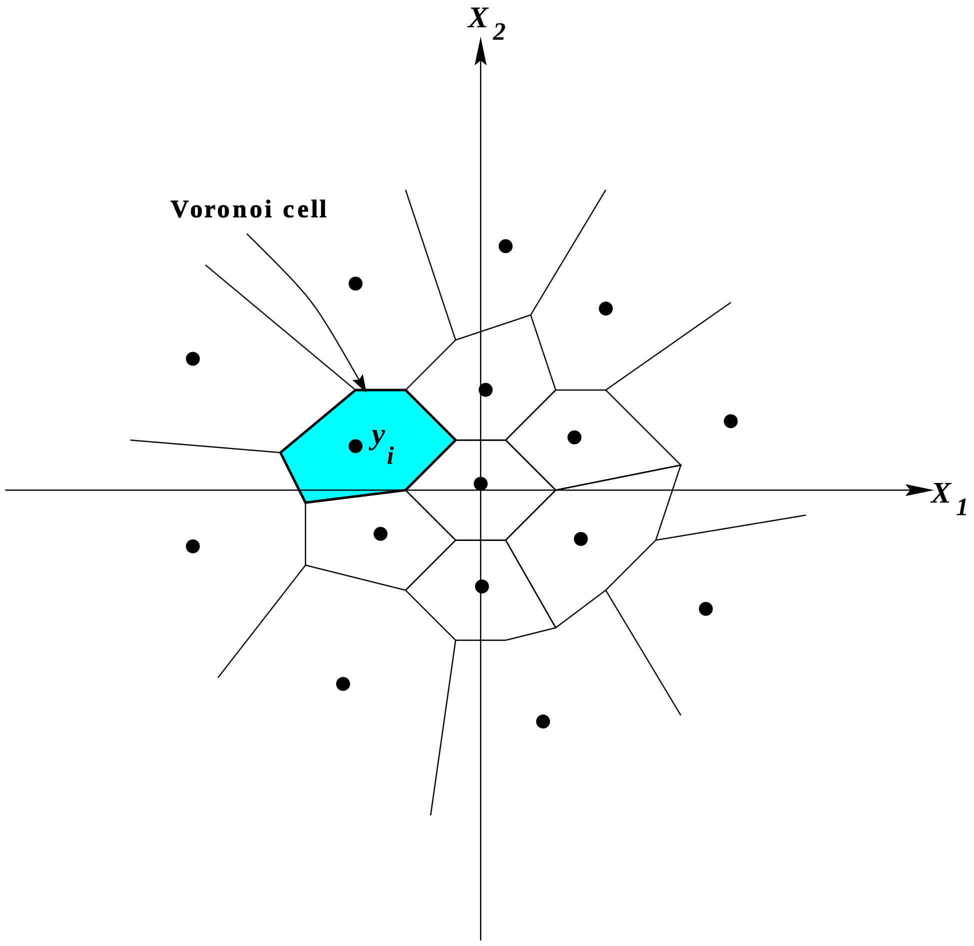


Figure 1.5: Bidimensional Vector Quantizer with 16 reproduction vectors.

At the decoder, the binary label l_i is used to retrieve the codevector $\mathbf{y_i}$ which represents the source vector $\mathbf{x_k}$ at the destination.

As in scalar quantization, vector quantizers also need to optimize their reproduction codebook according to the statistics of the input source. With this goal in mind, Linde, Buzo and Gray [67] designed a codebook optimization algorithm for vector quantizers. This algorithm is the M dimensional version of the Lloy-Max algorithm. In practical applications, this algorithm is based on a training sequence to design the codebook.

The algorithm consists in the following steps:

1. **Initialization**: Start with an initial codebook $\mathcal{C}^0$ and a training sequence X. Set $m = 0$ and $D^{-1} = \infty$. Select a convergence threshold ϵ.

2. **encoding**: encode the training sequence according to the nearest neighbor condition. This encoding partitions the training sequence into N sets Q. Each set Q_i contains the source vectors $\mathbf{x}$ that are closer to the reproduction codevector $\mathbf{y_i}$ than any other codevector $\mathbf{y_j}$.

3. **halt test**: Compute the average distortion D^m between the training vectors and the representative codevectors. If

$$\frac{D^{m-1} - D^m}{D^{m-1}} < \epsilon \tag{1.14}$$

   then stop with codebook $\mathcal{C}^m$ as the final codebook. Otherwise go to step 4.

4. **codebook update**: find the new reproduction codebook $\mathcal{C}^{m+1}$ according to the centroid equation 1.12. Set $m = m + 1$ and go to step 2.

Under certain assumptions about the source, the LBG algorithm guarantees that the distortion from one iteration to the next will not increase. However, the algorithm does not guarantee that the optimized codebook is the optimum one. This is because the algorithm is heavily dependent on the initial codebook $\mathcal{C}^0$. Linde, Buzo and Gray also proposed a codebook initialization technique called the **splitting** algorithm. In this technique, the codebook design procedure begins by designing a vector quantizer with a single codevector, that is, a codebook of size one. Then, this codebook is used as the initial codebook for a two-codeword codebook by using the codeword of the one-size codebook and an additional codeword which is obtained by adding a fixed perturbation vector ϵ to the optimized codeword. Then, the LBG algorithm is used to obtain an optimized two-codewords codebook. Once this new codebook is optimized, a new four-codewords codebook is generated by adding the perturbation vector ϵ to the optimized codewords of the current two-codewords codebook. The LBG algorithm is then used to optimize this four-codeword codebook and so on. In this manner, the algorithm keeps doubling the number of codewords until the desired codebook is reached.

1.3 Channel Coding

Figure 1.6 presents the definitions and notations of a channel coder [29, 69]. As indicated before, channel coding adds some redundancy to the encoded source so that it can be protected against possible transmission errors. Usually, the channel encoding process implies taking m information bits at a time from a binary sequence $\mathbf{u}$, and mapping each sequence of m bits into a unique n-bit sequence v called a **codeword** or **channel symbol** belonging to an alphabet $\mathcal{V}$. The ratio m/n is called the **code rate** R_c.

The sequence of channel symbols $\mathbf{v}$ is transmitted through the channel and corrupted by noise. At the receiving end, the channel decoder is fed with the channel-corrupted symbols $\hat{\mathbf{v}}$ beloging to an alphabet $\hat{\mathcal{V}}$ and exploits the available redundancy to correct the transmission errors and to recover the original information sequence $\mathbf{u}$.

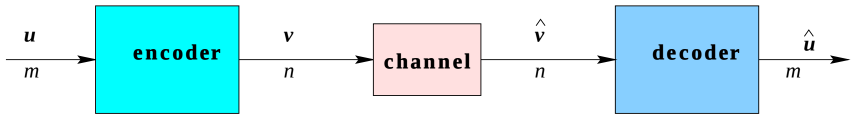


Figure 1.6: Channel coder-decoder pair.

1.3.1 Channel Models

As we can see, the transmission channel plays a crucial role in determining the performance of a given channel coding technique. As a consequence, channel models are required to design high-performance channel codes.

Channels can be characterized by a probabilistic model relating the channel input v and its output $\hat{v}$. The simplest channel model is the Binary Symmetric Channel (BSC). This model is characterized by binary input and output alphabets $\mathcal{V} = \hat{\mathcal{V}} = \{0, 1\}$, and a set of conditional probabilities $p(\hat{v}|v)$, for the output symbols given each of the input symbols. These probablities are defined as

$$p(1|0) = p(0|1) = p \tag{1.15}$$
$$p(0|0) = p(1|1) = 1 - p \tag{1.16}$$

where parameter p is known as the *transition error probability*. The BSC is depicted in figure 1.7.

If a n-bit channel symbol v is transmitted through a BSC, the conditional probability of receiving $\hat{v}$ given that v was transmitted is

$$p(\hat{v}|v) = p^{d(v,\hat{v})} \cdot (1 - p)^{n - d(v,\hat{v})} \tag{1.17}$$

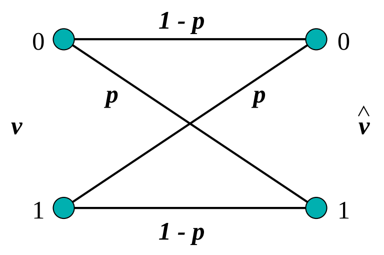


Figure 1.7: Binary Symmetric Channel Model.

where $d(v, \hat{v})$ is the Hamming distance between the channel input and its output.

Another channel that plays an important role in communication systems is the Additive White Gaussian Noise channel (AWGN). This is a memoryless channel with discrete input alphabet $\mathcal{V} = \{v_1, v_2, \cdots v_M\}$ and continuous output alphabet $\hat{\mathcal{V}} = (-\infty, \infty)$. The channel is assumed to corrupt the channel input v by the addition of white gaussian noise, as illustrated in figure 1.8. Thus, the received signal can be expressed as

$$\hat{v} = v + w \tag{1.18}$$

where w denotes additive white gaussian noise of zero mean and variance σ_w^2. The probability of receiving the channel output $\hat{v}$ given that the channel input was v is defined as

$$p(\hat{v}|v) = \left(\frac{1}{\sqrt{2\pi\sigma_w^2}}\right) \exp\left(-\frac{||\hat{v} - v||^2}{2\sigma_w^2}\right) \tag{1.19}$$

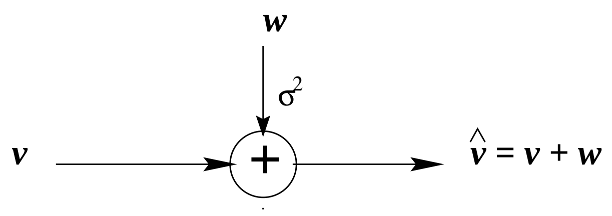


Figure 1.8: Additive White Gaussian Noise channel.

1.3.2 Channel Capacity

The channel capacity is defined as the maximum average mutual information between its input and its output, where the maximization is over all possible input probability distributions. Mathematically this can be expressed as

$$C = \max_{p(v)} I(V;\hat{V}) \tag{1.20}$$

The usefulness of channel capacity is that it serves as an upper limit on the transmission rate for reliable communication over noisy channels. This important result was stated by Shannon in his noisy channel coding theorem which states that [89, 98]

> *"Under certain hypothesis, there exist channel codes that make it possible to achieve reliable communication with as small an error probabilities as desired, if the transmission rate $R_c < C$, where C is the channel capacity. If $R_c > C$, it is not possible to make the probability of error tend toward zero with any code"*

In a BSC with transition error probability p, the channel capacity is defined as [89]

$$C = p\ \log_2 2p + (1-p)\log_2 2(1-p) \tag{1.21}$$

Figure 1.9 sketches the channel capacity of the BSC channel.

Regarding an AWGN channel, its channel capacity is defined as [98]

$$C = W \log_2 \left(1 + \frac{P_{av}}{W N_o}\right) \tag{1.22}$$

where W, P_{av} and N_o is the channel bandwidth, the average signal power and the spectral density of the AWG noise, respectively. Figure 1.10 illustrates the channel capacity of a AWGN channel.

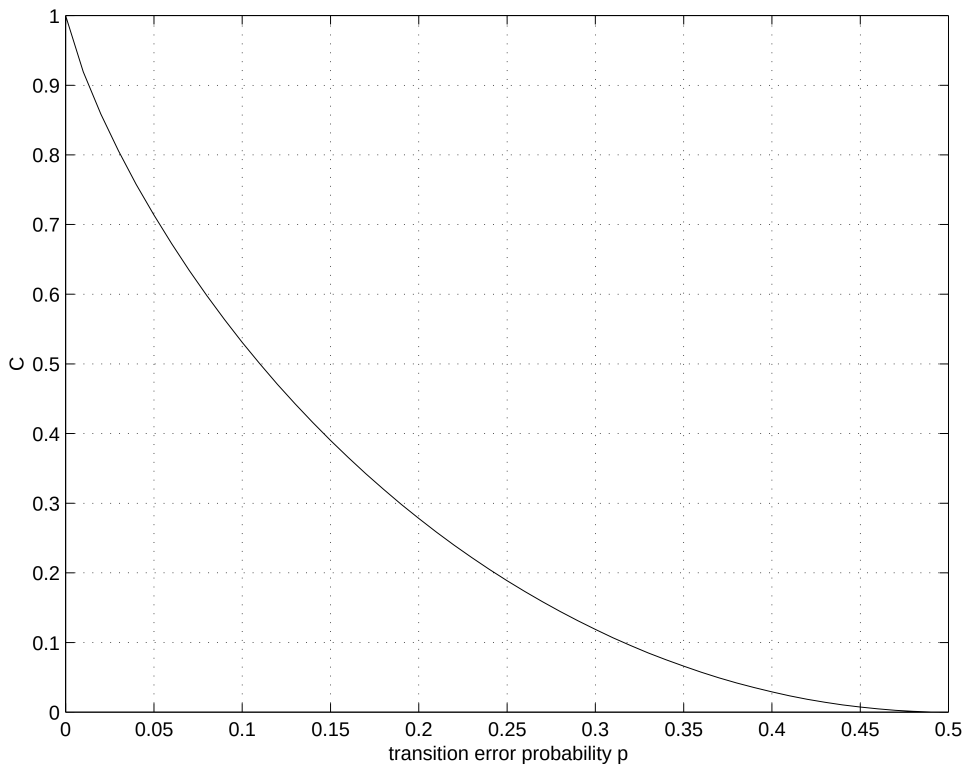


Figure 1.9: Channel Capacity of a BSC channel.

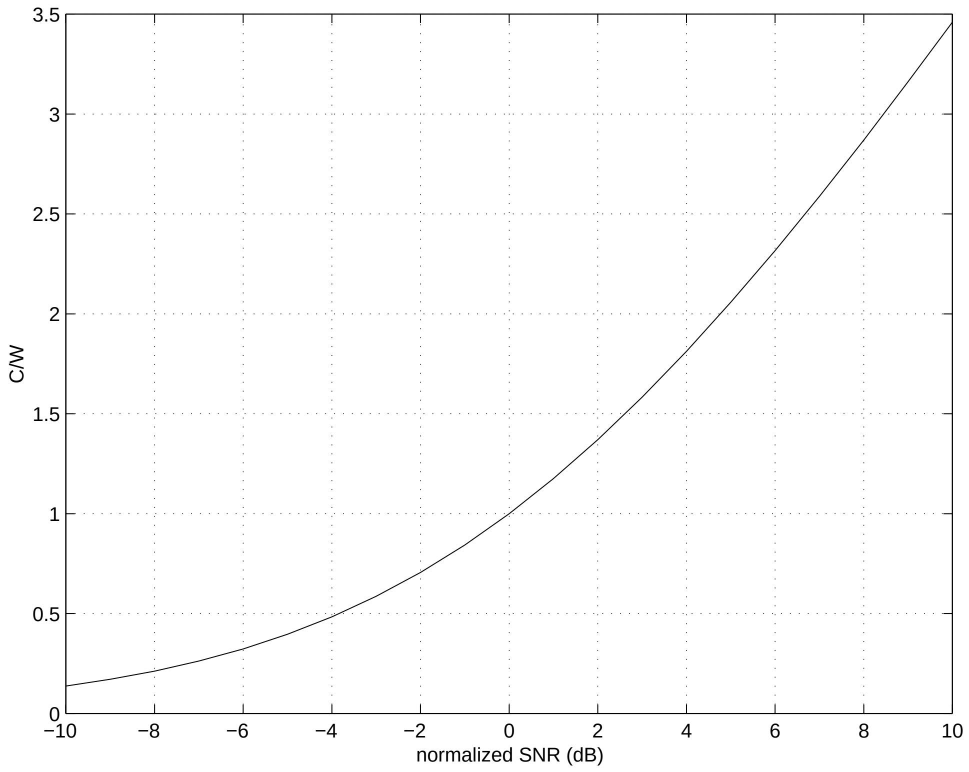


Figure 1.10: Channel Capacity of an AWGN channel.

1.4 Joint Source-Channel Coding

When the source and channel coders of the previous sections are considered as a whole, the noisy channel coding theorem and the source coding theorem can be merged into a single statement [91]:

> *"A source sequence which satisfies the source coding theorem can be reconstructed at the receiving end with a distortion arbitrarily close to D if the transmission channel has a capacity C which is higher than $R(D)$"*

Conversely, it is not possible to reconstruct a source sequence with distortion D if $C < R(D)$. This theorem allows to define an ideal communication system as that for which a distortion D can be obtained when the source coding rate R_s equals the channel capacity C. That is, the distortion introduced to the decoded source depends only on the distortion introduced by the source coder. The distortion D that can be attained with this system is called **Optimum Performance Theoretically Achievable** (OPTA) and denoted as $D(C)$ [52].

In the case of a gaussian source with variance σ_x^2, transmitted through a BSC channel, the OPTA function is given by

$$D(C) = 2^{-2(1+(1-\epsilon)\log_2(1-\epsilon)+\epsilon\log_2\epsilon)} \tag{1.23}$$

This function is illustrated in figure 1.11.

For an AWGN channel, the OPTA function can be expressed as

$$D(C) = \frac{\sigma_x^2}{D} = \left(1 + \frac{P_{av}}{2N_o}\right)^2 \tag{1.24}$$

A sketch of this functions is shown in figure 1.12.

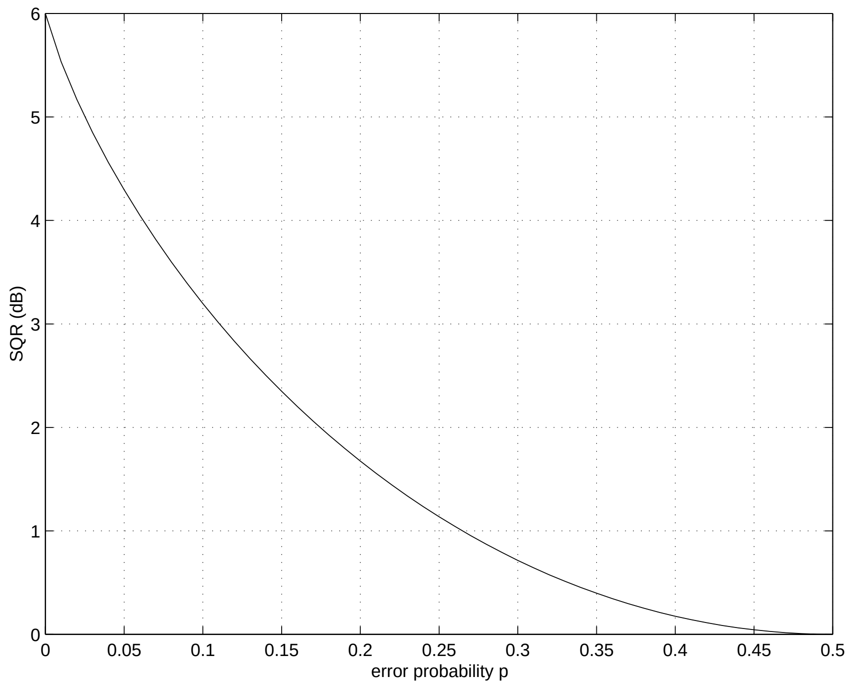


Figure 1.11: OPTA curve for gaussian source and BSC channel.

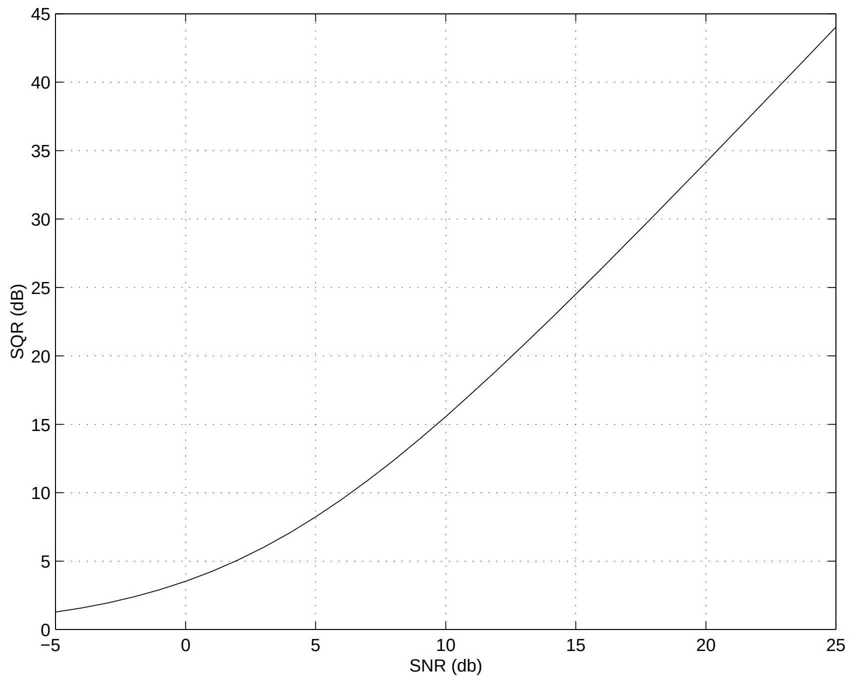


Figure 1.12: OPTA curve for gaussian source and AWGN channel.

1.5 State of the Art on JSCC

Since the last two decades, JSCC has attracted a lot of attention. Massey, in his seminal paper [73], is the first to analyse the advantages of combining source and channel coding in a single block. He claimed that it is in fact possible to design joint source and channel coders with **smaller complexity** and at least the same performance as tandem systems since a tendem system is a special cases of joint source and channel coding. Nowadays, several techniques are available which cover a wide range of applications and which treat the JSCC problem from different perspectives. In this section, a brief survey on these JSCC techniques is presented. We have followed the same approach of [95] by distinguishing four classes of JSCC systems. This survey is also based on the classification presented in [91] and [86]. Other excellent surveys on present time JSCC techniques can be found in [36] and [121].

JSCC techniques can be divided into four main classes:

- Concatenated joint source-channel coders
- Joint source-channel decoders
- Robust source coders
- Joint optimization of source and channel coders

In the following, each one of theses classes are described.

1.5.1 Concatenated joint source-channel coders

The main idea of the techniques belonging to this class is that source and channel coders are separately optimized, but a trade off between the source rate and the channel rate is made so that the the overall distortion D is minimized.

The work of Modestino and his co-authors for the transmission of still images over noisy channels belongs to this class [77, 78, 79]. In [77], the performance of a 2-D differential pulse code modulation combined with convolutional codes is analyzed over noisy channels. In [78], a 2-D DCT coder is concatenated to the same convolutional codes and the same trade offs source-rate-channel-rate is studied. Finally, in [79] a 2-D tree encoding system is compared to the results obtained in [78]. The conclusions of these series of papers are that for large channel SNR, better performances in terms of source distortion are obtained when more bits are used for source encoding. On the other hand, when the channel degradation is large, by reducing the source rate R_s and increasing the channel rate R_c in the same proportion, the performance of the whole system is greately improved. Thus, an intelligent trade off has to be done between source and channel coders so as to improve the performances when the transmission is done over noisy channels.

As an example, comparing a transmission system composed only of a DPCM system at $R_s = 2$ bits per pixel (bpp) with a concatenated systems composed of a source coder with $R_s = 1$ bpp and a $R_c = 1/2$ convolutional code, it was noticed that for a channel SNR=15 dB, the former system outperformed the concatenated system in 5 dB of SQR. However, when the channel SNR was reduced to 10 dB, the concatenated system outperformed the source coder in 12 dB.

In [122], instead of trading source and channel code rates, a new algorithm for index assignment of source codewords is proposed to reduce the overall distortion introduced by noisy channels. This algorithm is called pseudo-gray coding. In this method, the Hamming distance of the binary labels is commensurate with the euclidean distance of the channel codevectors. This way, small errors in the received word introduce little distortion in the decoded source. Pseudo-gray coding provides redundancy-free error protection for vector quantizers since the binary indexes associated to the source codevectors correspond to the channel symbols of the channel constellation. Another algorithm which performs a judicious assignment of the binary labels associated to the source codewords of a vector quantizer was proposed in [33]. The proposed algorithm is suboptimal in general; however, for examples involving a small number of codevectors, this algorithm achieved the optimum solution.

1.5.2 Joint source-channel decoders

These methods are motivated by the practical shortcomings of Shannon's separation theorem: the failure to remove all redundancy and the inadequacy of conventional channel coding methods, specially for very noisy and/or fading channels. These techniques capitalize on the first shortcoming to mitigate the second shortcoming. They can provide channel robustness without incurring any increase in rate, and without modifying the encoder for the noisy channel. Thus, these methods are potentially useful for broadcast transmission, where there is no feedback channel from decoder to encoder.

In some of these techniques, it is actually desired that the source coders do not remove all the natural redundancy of the source and hence, this *a priori* information can be used to provide better decisions at the decoder. This way, the decoder uses both the received information from the channel and the a priori information from the source by considering all possibilities and selecting the most probable.

The most important results on this subject were provided by Sayood *et al.* [95, 94]. In order to take into account the source redundancy during the decoding operation, instead of decoding one source symbol at a time, a set of received source symbols is treated. To do so, new metrics are used which are similar to the branch metrics of covolutional code decoding. As a result, the Viterbi algorithm can be used together with this new metrics to reconstruct the source. It was shown that significant gains could be obtained with this method.

In addition to the source a priori information for joint decoding, a priori channel information can also be used at the decoder, and by means of Maximum A Posteriori (MAP) decoding, the performance in terms of source distortion can be increased. Certainly, if the channel characteristics are known and can be modelled, they can be incorporated into the decoding metrics as more information becomes known about the received sequence and better decisions can be made [1].

More recently, Hagenauer proposed the modification of the Viterbi algorithm which uses a priori or a posteriori information about the source bit probability for better decoding in addition to soft inputs and channel information [55]. This algorithm is called APRI-SOVA. When applied to the full rate GSM speech codec, results showed that with this algorithm, the channel SNR in a bad mobile environment could be lowered by 2 or 3 dB resulting in the same voice quality.

Then, in [120], the APRI-SOVA was modified and applied to the transmission of still images over a AWGN channel. The source coders consisted of a DCT coder and a subband coder. Significant gains were obtained, specially in bad transmission environments. Furthermore, it was shown that it is better not to remove the statistical redundancy in source-coded bits by using a variable length coding like the Lempel-Ziv algorithm [93], where error propagation occurrs even with a single bit error. Instead, such redundancy can be more efficiently utilized at the receiver end by applying the APRI-SOVA.

1.5.3 Robust source coders

In this kind of coders, the quantizer is optimized in such a way that the channel statistical distribution is taken into account during both the source quantization and the codebook design. It must be noted that these coders serve as simultaneous source and channel coders. In addition, it must be noticed that these systems do not try to correct transmission errors but rather to decrease the impact of these errors in the reconstruction of the source.

The design of scalar quantizers optimized for a given channel probability model was first suggested by Kunterbach and Wintz [65]. They developed a generalized Lloyd-Max algorithm which take into account the channel noise so as to minimize the overall distortion. New expressions for the nearest neighbor condition and the centroid equation were found.

Later, Farvardin and Vaishampayan [37] extend the work of Kunterbach for the case where the number of reproduction codewords at the receiving end is different from the number of codewords at the encoder. In addition, they also analyzed the index assignment during the codebook design procedure. Kumazawa *et al.*, in [64] generalized the results of Farvardin to vector quantization. The LBG algorithm, adapated to the expressions for the nearest neighbor condition and the centroid equation was used. This new technique is called Channel Optimized Vector Quantization (COVQ).

Dunham and Gray [34], on the other hand, found the necessary conditions for the design of joint source-channel trellis coders and Ayanoğlu and Gray proposed algorithms for the design of such coders, based on the LBG algorithm [9]. In [119], trellis coded quantization (TCQ) [71] was designed for noisy channels and the codebook design algorithm presented in [9] was modified to account for the TCQ features. The performance obtained with this algorithm were close to Ayanoğlu's algorithm but with smaller computational complexity.

Another way of implementing joint source-channel trellis coders was proposed by Rodríguez in [91]. This algorithm employs a training sequence and an iterative procedure where transmitter and receiver are used to design the codebook. The main difference with respect to other trellis quantizers is that the codebook update is performed at the receiving end. To do so, a Viterbi algorithm is employed at the receiver. This way, two Viterbi algorithms are required during the optimization of the reproduction codebook. The main advantage of this approach is that the channel disturbances are inherently taken into account and absorbed by the codebook update. Results close to Ayanoğlu's approach were also obtained.

A different approach in the design of robust source coders was recently proposed by Skoglund in [107] and [108]. He proposes new iterative algorithms for "soft" decoding of vector quantizers over noisy channels. MAP decoding is utilized and the resulting algorithm is optimum according to mean-squared error criterion.

1.5.4 Joint optimization of source and channel coders

There are two ways of performing a joint optimization between source and channel coders. In [113], Vaishampayan *et al.* proposed an algorithm for the design of vector quantizers subject to the modulation signal sets. The design criterion was the minimization of the MSE at the receiver. Such scheme attempts to match the source sensitivities to the modulation signal set in a structured manner.

A joint TCQ/TCM scheme was developed by Fischer and Marcellin in [42] which combined both coders to match quantization MSE to euclidean distance in TCM [16]. However, this system does not perform as well as expected when the transmission channel is very noisy. The reason for this poor behavior is that both coders are separately designed and the only interaction between source and channel coders is done during the mapping from equal MSE to euclidean distance. To solve this problem, in [119] an algorithm based on the LBG algorithm which optimizes the TCQ and the TCM is proposed. This time, the performance of the system when the transmission channel is poor is largely improved.

In [106], another adaptation between the codewords of a vector quantizer and the signal constellation is proposed by means of neural networks. In this algorithm, the neural network finds the best assignment between the source codewords and the modulation signal set.

Chei and Ho extended the work of Wang and Fischer and developed a new algorithm for optimum soft decoding for combined TCQ/TCM schemes in rayleigh fading channels [28]. In addition to the iterative algorithm for the joint optimization of the TCQ and TCM schemes, the MAP algorithm [12] is used as a minimum mean-squared error decoder.

In Chapter 6 of this thesis, we present another technique which uses the joint source-channel trellis coding design algorithm of Ayanoğlu and the MAP algorithm for soft source decoding.

Chapter 2

Joint Source-Channel Trellis Coding

Last Chapter presented the theory behind joint source-channel coding. Different techniques which have been proposed in the last years were outlined and we saw the potential advantage of using JSCC techniques over tandem systems, specially when complexity is the main parameter design and when the transmission channel is very noisy.

In this chapter, we present the concepts and design algorithms for the JSCC technique that we have chosen for hardware implementation, the Joint Source-Channel Trellis Coding (JSCTC) technique proposed by Ayanoğlu [9, 34]. We chose this JSCC technique because it treats the JSCC problem in a "pure" fashion, where source and channel coding is embedded into a single entity. That is, JSCTC is considered as a single source and channel coder since the quantization of the source and the protection against channel errors is done in the same block. Other algorithms such as those described in sections 1.5.1, 1.5.2 and 1.5.4 employ separate source and channel coders which are already well-known and which have been modified to incorporate the JSCC charecteristics. Hence, these systems do not treat the JSCC as a single operation. Other techniques such as Rodríguez algorithm and COVQ are also pure JSCC techniques. However, the computational load of these techniques is higher.

We begin this chapter with a general description of trellis source coding with noiseless assumptions, and then, a detailed description of the joint source-channel trellis coding technique is given. In fact, we will see that the main differences between these two techniques reside in the distortion measure utilized to perform both the quantization process and the codebook design algorithm. The distortion measure results in an optimum quantization of the source when transmitted through a noisy channel and when no channel coding is considered. The derivation of the distortion measure allowing this "protection" against channel errors and the codebook design algorithms are presented.

2.1 Trellis Source Coding

Trellis source coding is a quantization technique proposed from the discovery that encoding-decoding structures similar in nature to convolutional channel coding-decoding could also be used for source coding applications yielding performances arbitrarily close to the theoretical limits given by the rate-distortion theory [58, 83, 115]. In addition, it has been shown that trellis source coding systems are more robust to channel errors than vector quantizers and need less complexity for the same performance requirements [50]. Trellis source coding is particularly well suited for speech coding applications, specially in model-based source coding where instead of transmitting the samples of the source waveform, the parameters of a synthetic model representing the source are quantized and transmitted [36, 51, 87, 97, 110].

A trellis quantizer can be thought of as a quantizer with memory, that is, a quantizer whose encoder output or channel symbol, u_k, depends on the current and prior source symbols $\{x_k, x_{k-1}, x_{k-2}, \cdots, x_{k-L}\}$, and its decoder output $\hat{x}_k$ depends on the current and prior channel symbols $\{u_k, u_{k-1}, u_{k-2}, \cdots, u_{k-K}\}$, where parameter K is known as the trellis coder's *constraint lenght.* Let us first describe the decoding process.

Consider a sequence of length L_{TS} of q-ary channel symbols $\mathbf{u} = \{u_k\}_{k=0}^{L_{TS}-1}$ entering from left to rigth a $\log_2 q(K-1)$-bit shift register, as shown in figure 2.1 for $K = 3$ and $q = 2$ ($u_k \in \{0, 1\}$). This shift register represents a finite state machine (FSM) with q^{K-1} states whose transition diagram is also shown in figure 2.1. Each time a channel symbol u_k enters the shift register, the FSM changes to the next state S_{k+1} according to its transition diagram. In a general way, the next state is given by the mapping

$$S_{k+1} = f(u_k, S_k) \qquad k = 0, 1, \cdots \tag{2.1}$$

where f is called the next-state function. In our particular case where the decoder is a shift register, the current state of the FSM is given by the channel symbols $u_{k-1}, u_{k-2}, \cdots, u_{k-K+1}$ and the next state is given by

$$S_{k+1} = f(u_k, S_k) = u_k, u_{k-1}, u_{k-2}, \cdots, u_{k-K+2} \tag{2.2}$$

that is, as described above, the next state of the FSM is given by simply shifting the current input symbol into the leftmost position of the shift register.

In order to visualize the evolution in time of the different states visited by the FSM, a trellis diagram is commonly employed, as depicted in figure 2.1(c). A trellis diagram is a bidimensional representation of the FSM where the horizontal axis denotes time and the vertical axis represents the states of the FSM. The transitions or branches connecting two states in successive time instants represent the channel symbol u_k entering the shift register. In the case where $q = 2$, only two branches leave and reach each state, indicating whether a 0 or a 1 channel symbol enters the FSM. Notice that there are 2^K trellis branches per trellis stage. Each path in the trellis diagram represents a unique channel sequence $\mathbf{u}$; conversely, different channel sequences will never trace the same path through the trellis.

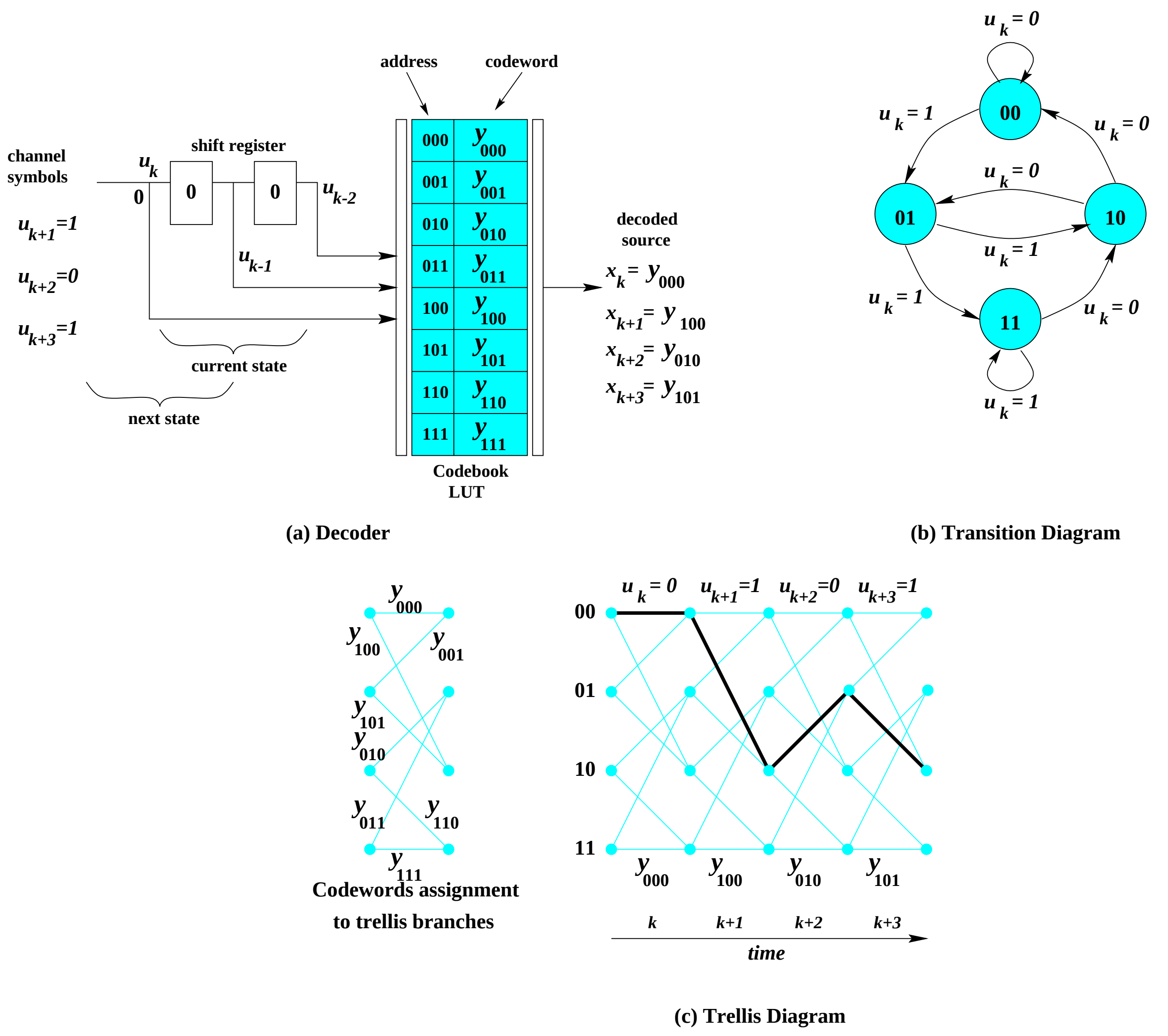


Figure 2.1: Trellis quantizer decoder.

If we associate a reproduction codebook $\mathcal{C} = \{y_i\}_{i=0}^{2^K-1}$ to the branches of the trellis diagram, the channel sequence $\mathbf{u}$ generates in turn a sequence of reproduction codewords $\mathbf{y} = \{y_{i(k)}\}_{k=0}^{L_{TS}-1}$, where $y_{i(k)}$ denotes the codeword associated to branch i visited by the channel symbol u_k at time k. This sequence of reproduction codewords is the representation $\hat{\mathbf{x}}$ of the source sequence at the receiver end.

In practice, the decoder of the trellis quantizer may be implemented with the shift register described above, addressing a Look-Up Table (LUT) which contains the reproduction codebook $\mathcal{C}$. The address of the LUT is given by the current channel symbol u_k and the current state of the FSM; that is, the address is simply given by the most recent K bits.

At the encoder end, the channel sequence $\mathbf{u}$ is generated by a trellis search algorithm. This search algorithm employs the trellis structure of the decoder and a copy of the reproduction codebook in order to search for the path in the trellis that minimizes the distortion between the source sequence $\mathbf{x} = \{x_k\}_{k=0}^{L_{TS}-1}$ and the reproduction codeword sequence $\mathbf{y} = \{y_{i(k)}\}$ associated to that trellis path. The fidelity criterion used to find this minimum distortion path is the MSE defined in the previous Chapter. Thus, the trellis search algorithm finds the trellis path yielding the minimum value of the MSE given by

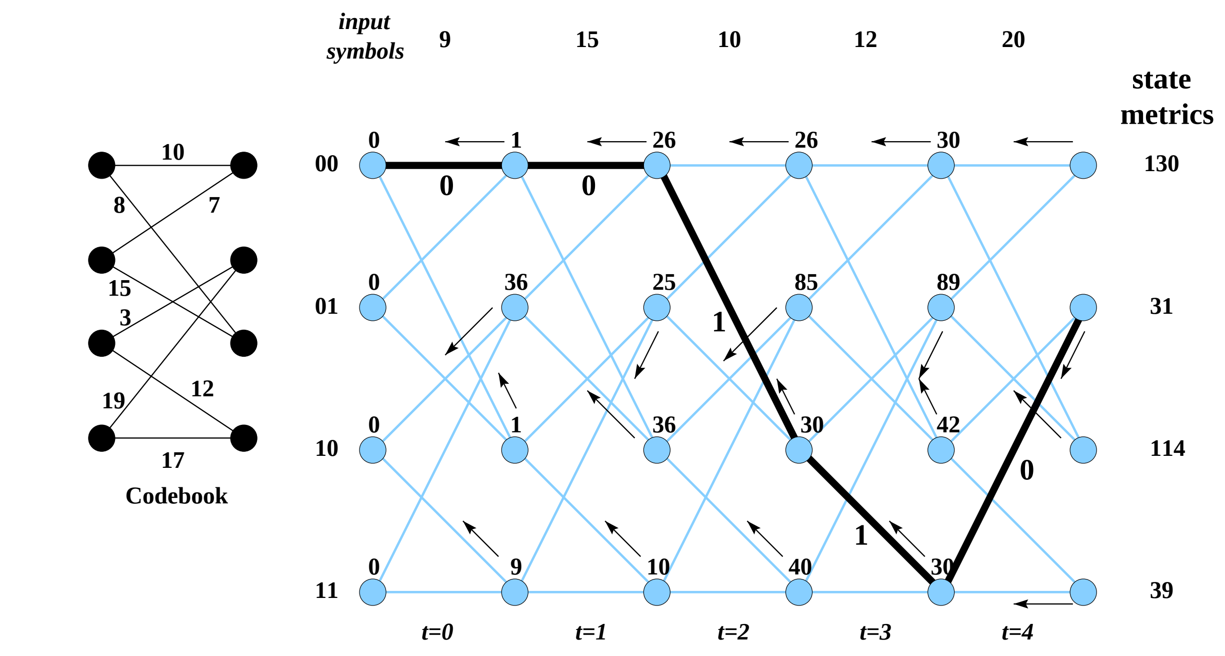
$$MSE = \frac{1}{L_{TS}} \sum_{k=0}^{L_{TS}-1} d(x_k, y_{i(k)}) \tag{2.3}$$

where $d(x_k, y_{i(k)}) = (x_k - y_{i(k)})^2$ is the per-letter distortion measure.

An example and overall structure of a trellis quantizer is illustrated in figure 2.2 for a 4-state trellis. The trellis search algorithm in the example is performed with the Viterbi algorithm [43, 114] which will be explained in a more detailed way in Chapter 4. For the moment, we will give a rough description of its operation.

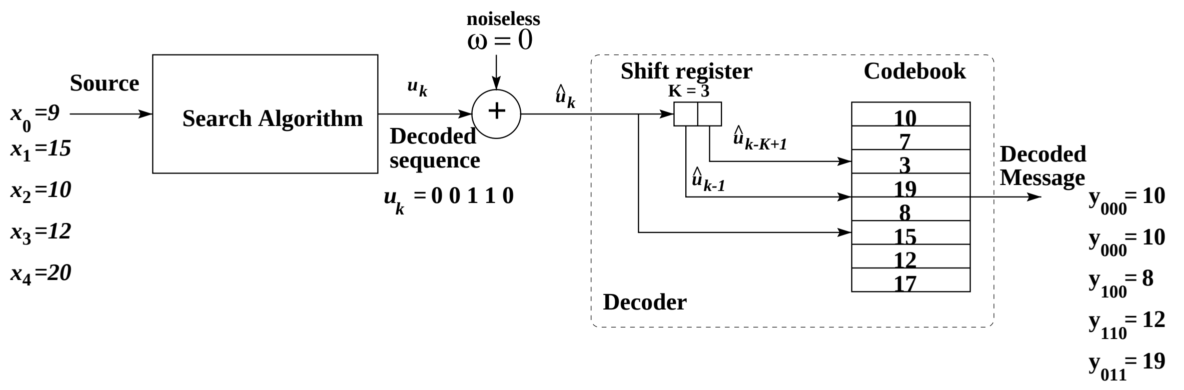
At every time instant k, when two paths enter the same state, the one having the smallest distortion is retained. To do so, each state keeps the value of the accumulated per-letter distortions between the current and previous source symbols, and the reproduction codewords associated to the trellis branches that form the path entering the state. This is shown in the figure by the distortion values at the top of each state. To compute the new distortion associated to a given state at time $k+1$, the per-letter distortions of the two branches entering a given state are computed and accumulated to the distortion associated to the trellis states from which these branches are coming at time k. The smallest accumulated distortion is chosen and the state keeps track of the retained branch, as shown by the arrows in the figure. This way, at every time instant, each state in the trellis has a unique path. The process of retaining a single path per trellis state is repeated until all the source sequence is treated. Then, the trellis state having the least accumulated distortion is chosen and the binary sequence $\mathbf{u}$ associated to its path is the compressed representation of the source sequence $\mathbf{x}$ which is transmitted to the receiver. This least distortion path is represented in the figure by the bold line.

It must be pointed out that the trellis search does not necessarily map the source symbol x_k into the codeword $y_{i(k)}$ yielding the minimum distortion. What the trellis search algorithm does it to map the entire source sequence into the minimum average distortion sequence of reproduction codewords.



Decoded sequence	0	0	1	1	0
Reproduction codewords sequence	10	10	8	12	19

a) operation example



b) complet system

Figure 2.2: Trellis quantizer structure and operation example.

Finally, after transmission of the binary sequence $\mathbf{u}$ and assuming that no channel errors were produced, this sequence enters the decoder in order to generate the reproduction codewords $\hat{\mathbf{x}} = \mathbf{y}$ which will represent the source sequence $\mathbf{x}$ at the receiver end.

2.1.1 Codebook Desing

In the same manner as the Lloyd-Max algorithm was adapted to vector quantization, resulting in the LBG algorithm [67], a codebook design algorithm for trellis quantizers, based on the Lloy-Max method was developed by Stewart [110]. It consists in the same training sequence-based iterative procedure but the encoding operation is performed by the trellis search algorithm. Refer to page 14 for a description of the codebook design algorithm.

2.1.2 Codebook Initialization

The stewart codebook design algorithm also generates locally optimum solutions and if the initial codebook $\mathcal{C}^0$ is not well guessed, it is likely that the algorithm produces low-performance codebooks. To make a good guess for $\mathcal{C}^0$, several methods have been proposed within the context of trellis quantization [51]. Nevertheless, an algorithm that has been shown to provide excellent results in practical situations is the extension algorithm proposed also by Stewart [110].

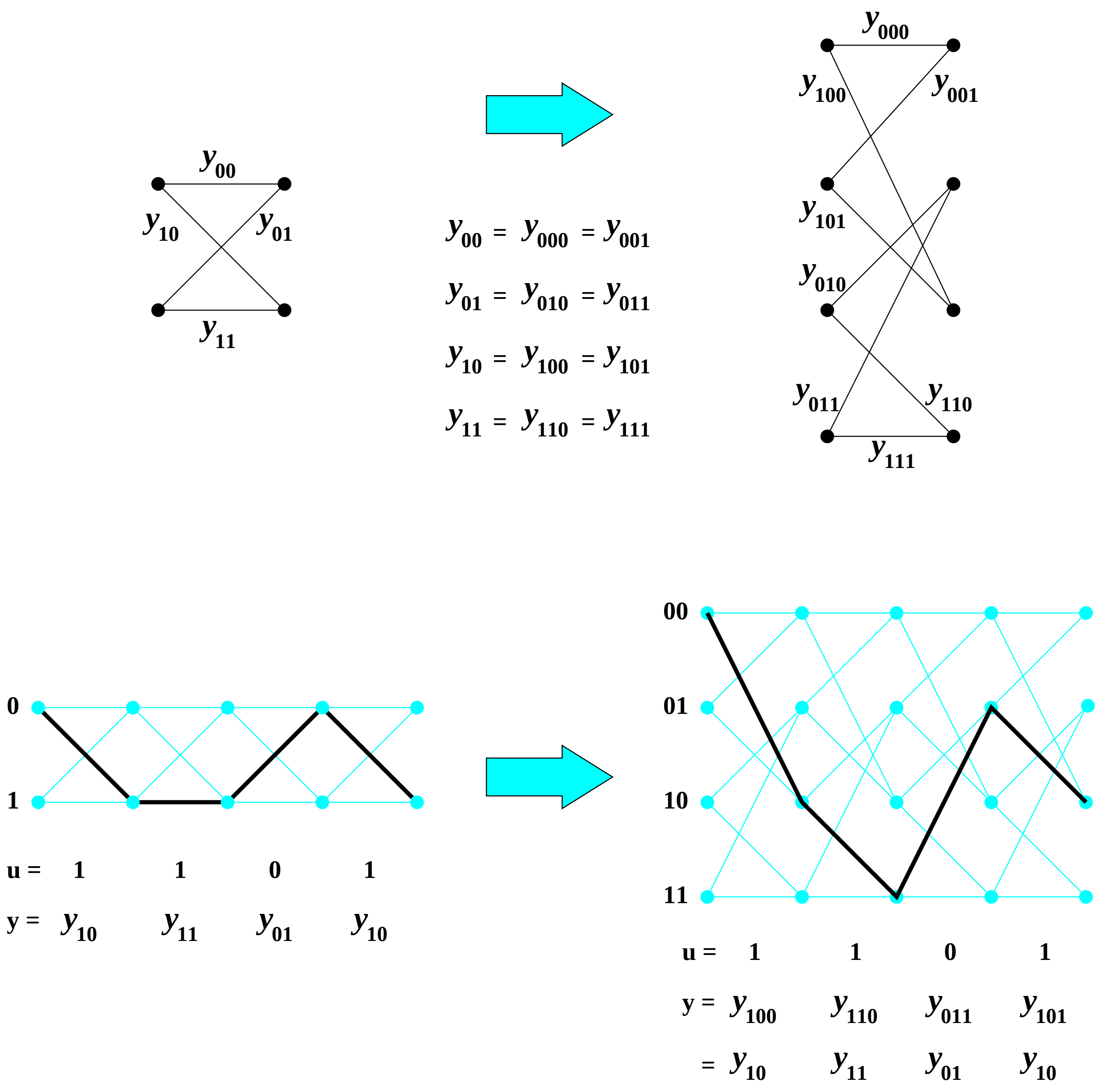
The main idea of this algorithm is to design a reproduction codebook for a K-constraint length trellis, from a $K-1$-constraint length. Thus, the decoder can begin with a single stage register and iteratively design longer codebooks until the desired constraint length is reached. The main advantage of this method is the design of reproduction codebooks from scratch for sources with unknown characteristics.

The extension algorithm can be thought of as the trellis counterpart of the splitting algorithm of vector quantizers since increasing by one the constraint length of the trellis implies duplicating the number of reproduction codewords.

It must be pointed out that attention must be paid during the codeword assignment from a $K-1$-constraint length quantizer to a K-constraint length one. The aim is to start the codebook design operation of the K-constraint length quantizer with a distortion identical to the distortion of the previous $K-1$-constraint length quantizer. In the way we have defined the labels of the reproduction codewords in the trellis (see figure 2.1), the rule for assigning the codewords of the $K-1$-length quantizer to the new K-constraint length codebook is

$$y_i \;=\; y'_{2i} \;=\; y'_{2i+1} \qquad i = 0 \cdots 2^{K-1} - 1 \tag{2.4}$$

where y_i' denotes the codeword associated to branch i of the new K-length codebook. This way, during the first iteration of the codebook design procedure of the new constraint-length quantizer, the channel sequence $\mathbf{u}$ and the distortion D^0 will be the same as the ones obtained during the last iteration of the $K-1$-constraint length codebook design operation. This is shown if figure 2.3.



y_{00} y_{10} y_{01} y_{11}
$y_{00} = y_{000} = y_{001}$
$y_{01} = y_{010} = y_{011}$
$y_{10} = y_{100} = y_{101}$
$y_{11} = y_{110} = y_{111}$
y_{000} y_{100} y_{001} y_{101} y_{010} y_{011} y_{110} y_{111}
0
1
u = 1 1 0 1
y = y_{10} y_{11} y_{01} y_{10}
00
01
10
11
u = 1 1 0 1
y = y_{100} y_{110} y_{011} y_{101}
= y_{10} y_{11} y_{01} y_{10}

Figure 2.3: Codeword assignment in the Extension algorithm.

2.2 Joint Source-Channel Trellis Coding

The trellis source coding technique explained above can be adapted to the JSCC case with performances arbitrarily close to the theoretical limits given by the OPTA function. The major goals of this approach are [9]:

- the design of compression systems that are robust to channel errors,
- the reduction of system complexity due to "tandemizing" the trellis quantizer with a channel protection technique and,
- for certain applications, the complete elimination of channel coding. This results in higher transmission rates dedicated only for encoding the source which in turn results in a better performance.

It must be pointed out that this technique acts like a simultaneous source coding and error protection technique. That is to say, the aim is not to correct transmission errors but to reduce the distortion introduced by these errors during the reconstruction of the source sequence.

The theorem showing that JSCTC yields performances arbitrarily close to the OPTA function was demonstrated by Dunhan and Gray in [34]. They showed that such good performances could be achieved by replacing the conventional noiseless distortion measures by distortion measures equal to the expected value of the distortion between the source symbol and the corresponding reproduction codewords; where the expectation is taken over the channel distribution, that is, over the channel error probability.

Consider the source sequence $\mathbf{x}$ which has been encoded at the encoder end by the sequence of reproduction codewords $\mathbf{y} = \{y_{i(k)}\}_{k=0}^{L_{TS}-1}$. The overall distortion that has to be minimized is given by

$$D = \sum_{k=0}^{L_{TS}-1} \sum_{j=0}^{2^K-1} d(x_k, y_j) \cdot Pr(y_j|y_{i(k)}) \tag{2.5}$$

where $Pr(y_j|y_{i(k)})$ is the probability of receiving codeword y_j given that codeword $y_{i(k)}$ was actually sent and $d(x_k, y_j)$ is the distortion between the source symbol x_k and the reproduction codeword y_j at the receiver end.

When a BSC channel is cosidered, the error probabilities $Pr(y_j|y_i)$ are given by the expression [89]

$$Pr(y_j|y_i) = p^{d(i,j)} \cdot (1-p)^{K-d(i,j)} \tag{2.6}$$

where p is the transition error probability of the BSC channel. For AWGN channels and BPSK modulation, this transition error probability is given by [89]

$$p = Q\left(\sqrt{2\frac{E_b}{N_o}}\right) \tag{2.7}$$

We can see that $Pr(y_j|y_{i(k)})$ depends on both the modulation technique used and the channel conditions or channel SNR [1]. In addition, notice that this system assumes that the transmitter has some information about this channel SNR. Fortunately, such information is commonly available in practical communication systems.

If $d(x_k, y_j)$ is the euclidean distance, the per-letter distortion measure in the JSCTC context is given by

$$d(x_k, y_i) = \sum_{j=0}^{j=2^K-1} (x_k - y_j)^2 \cdot Pr(y_j|y_i) \tag{2.8}$$

where y_i is the reproduction codeword associated to branch i.

Example

Consider the source sequence $\mathbf{x} = \{5, 5, 0, 3, ...\}$ wich is trellis quantized with the noiseless and JSCTC quantizers using the reproduction codebook $\mathcal{C} = \{4, 6, 1, 25\}$ associated to a 2-state trellis, as shown in figure 2.4, and that will be transmitted through a BSC channel with $p = 0.1$.

In the noiseless case, the per-letter distortion measure is the euclidean distance and the best path in the trellis is the one having the least MSE defined in equation 2.3. The bold line represents the minimum distortion path. The channel sequence corresponding to this path is $\mathbf{u} = \{0, 1, 0, 0, ...\}$ and the distortion introduced by the quantization process is 4.

In the JSCTC case, the per-letter distortion measure is given by equation 2.8 and the error probabilities $Pr(y_j|y_{i(k)})$ are computed with equation 2.6. The channel sequence corresponding the path having the best **expectation** of the distortion is $\mathbf{u} = \{0, 0, 0, 0, ...\}$, with a total MSE of 19.

Suppose now that during transmission, an error is produced on the third bit of each channel sequence such that the received channel sequence for the noiseless case becomes $\mathbf{u} = \{0, 1, 1, 0, ...\}$ whereas the received channel sequence in the JSCTC case is $\mathbf{u} = \{0, 0, 1, 0, ...\}$. At the decoder, the reconstructed source with the noiseless quantization has a total distortion of 631 while the distortion introduced by the JSCTC is only of 42.

[1]Notice that this technique is not restricted to BSC channels or BPSK modulation.

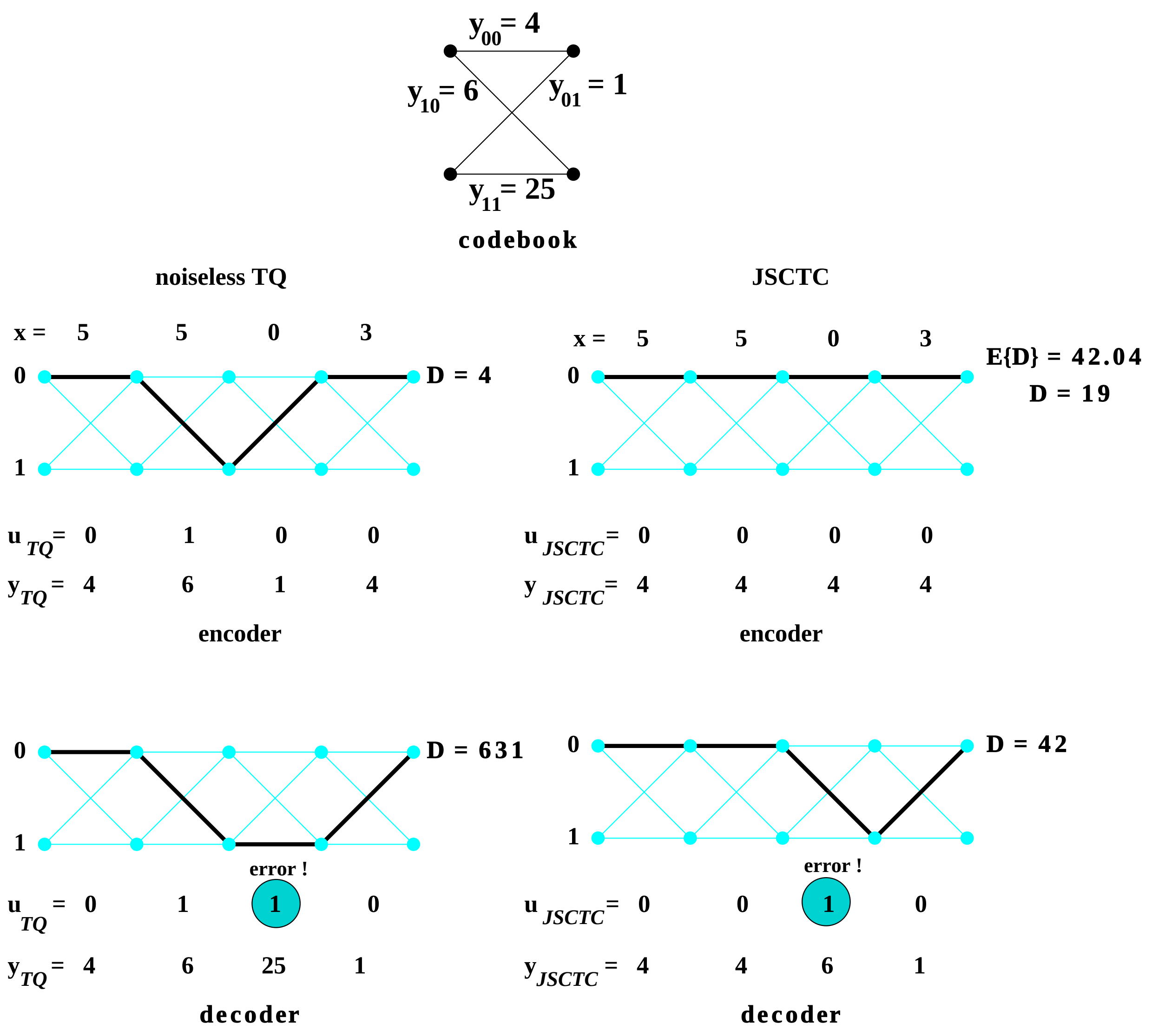


Figure 2.4: JSCTC vs Noiseless Trellis Quantization.

We can see that, eventhough at the encoder end the reproduction codeword sequence found by the JSCTC coder introduces higher distortion, at the receiver, the expectation of the distortion introduced by transmission errors is decreased.

2.2.1 Codebook Design and Initialization

Ayanoğlu in [9], proposes the design of JSCTC systems with the same approach given by Stewart. In fact, it is a straightforward adaptation of Stewart's work to the context of JSCTC. He demonstrates that the fact of using the modified distortion measure described above does not affect the convergence and monotonically decreasing distortion properties of the Stewart algorithm. The only difference between Stewart's and Ayanoğlu's algorithm is the per-letter distortion measure employed for encoding the source sequence, and the codebook update which is governed by the centroid equation. This centroid can be obtained by setting partial derivatives of equation 2.5 with respect to y_j^m to zero. We thus have

$$y_j^{m+1} = \frac{1}{\sum_{i=0}^{2^K-1} Pr(y_j|y_i)||Q_i||} \sum_{i=0}^{2^K-1} Pr(y_j|y_i) \sum_{k:x_k \in Q_i} x_k \tag{2.9}$$

This equation can be seen as the centroid of equation 1.12 (page 12) "weighted" by the channel error probability. For instance, if the transmission channel is noiseless, $Pr(y_j|y_i) = \delta_{ij}$[2]and equations 1.12 and 2.9 become identical.

Concerning codebook initialization, Ayanoğlu also showed that the extension algorithm outperformed other codebook initialization methods. In addition, he proposed a second extension algorithm which consists in the generation of a noisy channel codebook from a noiseless channel codebook of the same constraint length. This is also an iterative algorithm. It starts by obtaining an optimized codebook $\mathcal{C}$ for noiseless channel conditions. Then, by means of a perturbation constant p which is added repeatedly to the error probability $Pr(y_j|y_i)$, the ayanoğlu codebook design algorithm is run until obtaining an optimized codebook for that particular value of $Pr(y_j|y_i)$. This operation is iterated until the optimized codebook for the desired error probability is designed.

[2]$\delta_{i,j}$ is the Kronecker delta function defined as

$$\delta_{ij} = \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases}$$

2.2.2 Performances

In this section, the performance of the JSCTC technique is evaluated for different sources. Simulations are presented for the BSC channel which is governed by the transition error probability p, and for the AWGN channel whose error probability is given by equation 2.7. Comparisons with noiseless trellis quantization is highlighted for the BSC channel. In addition, the JSCTC technique is compared to a tandem system comprising a noiseless trellis quantizer and a convolutional code for error control coding. All the simulations were performed for 1 bit per sample quantization.

Figures 2.5 and 2.6 present the source distortion performance of the JSCTC system and the noiseless trellis quantizer over a BSC channel with transition error probabilities in the range $[0.0, 0.5]$. Figure 2.5 corresponds to a zero-mean, unit variance gaussian source and figure 2.6 to a first order Gauss-Markov source with autoregression coefficient 0.9. The constraint length of the trellis quantizers is $K = 5$.

We can see from the figures, that, as it was claimed before, when the transmission channel is noiseless, the performance of both quantizers is exactly the same. Nevertheless, as the transmission channel degrades, the performance of the noiseless trellis quantization is rapidly decreased whereas that of the JSCTC system decreases very smoothly, approaching the upper bound given by the OPTA curve (see figure 2.5).

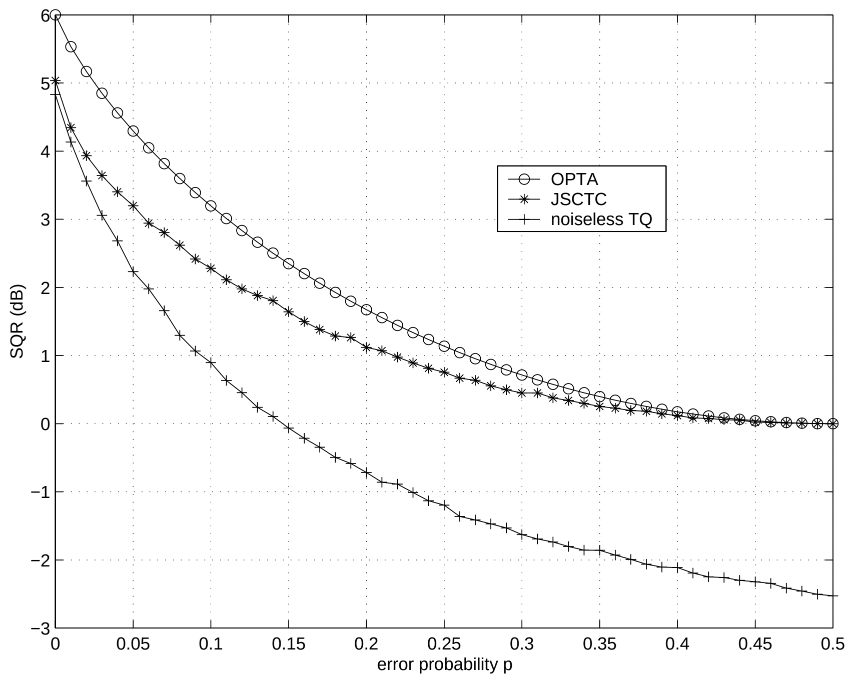


Figure 2.5: JSCTC vs TQ Performance for BSC and gaussian source.

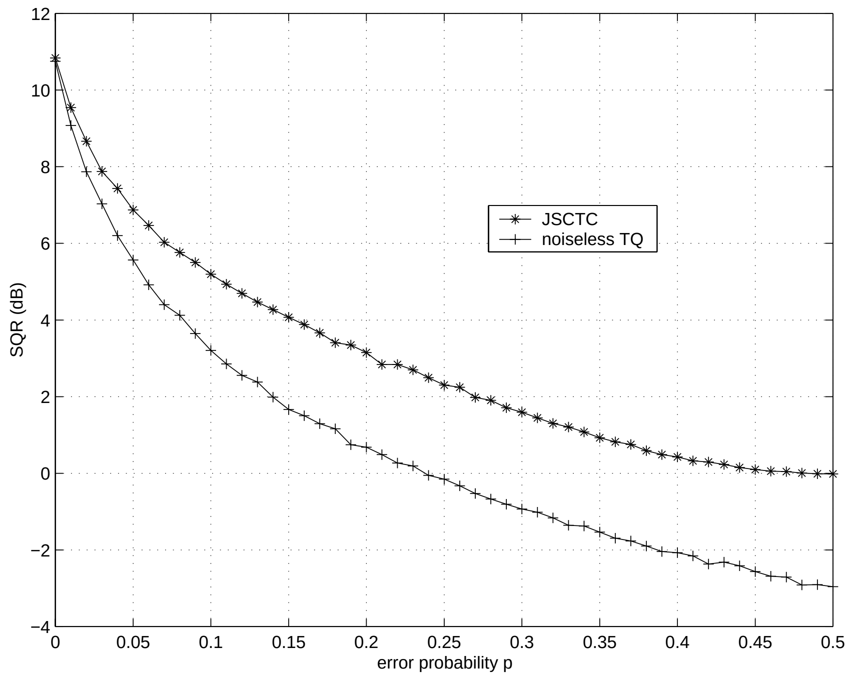


Figure 2.6: JSCTC vs TQ Performance for BSC and markov source.

Figure 2.7 compares now the source distortion performance between the JSCTC technique and a tandem system composed of a 1 bit per sample noiseless trellis quantizer and a R= 1/2-rate convolutional code. Both systems have a $K = 5$ constraint length and operate over an AWGN channel. The information to be transmitted is a gaussian source.

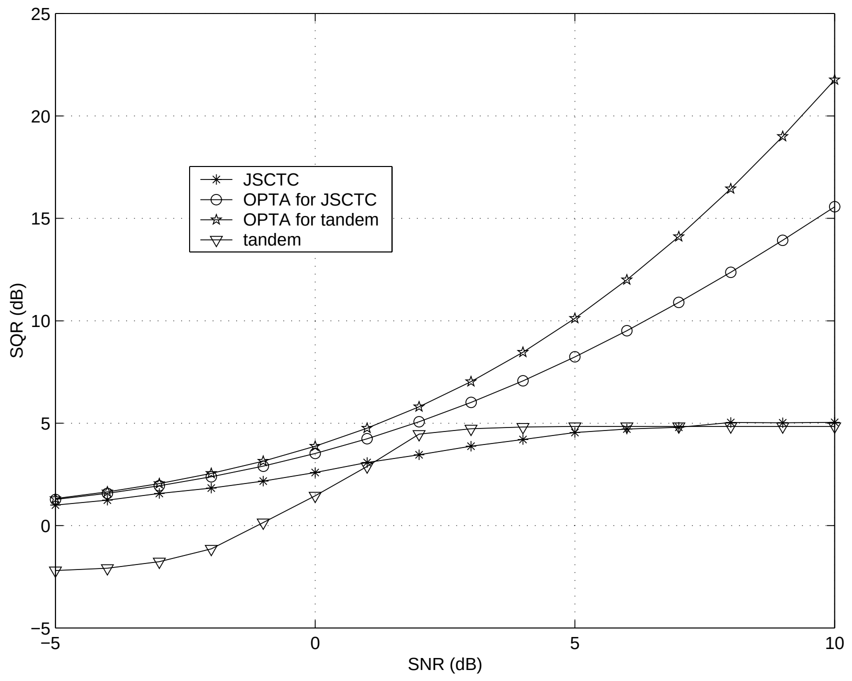


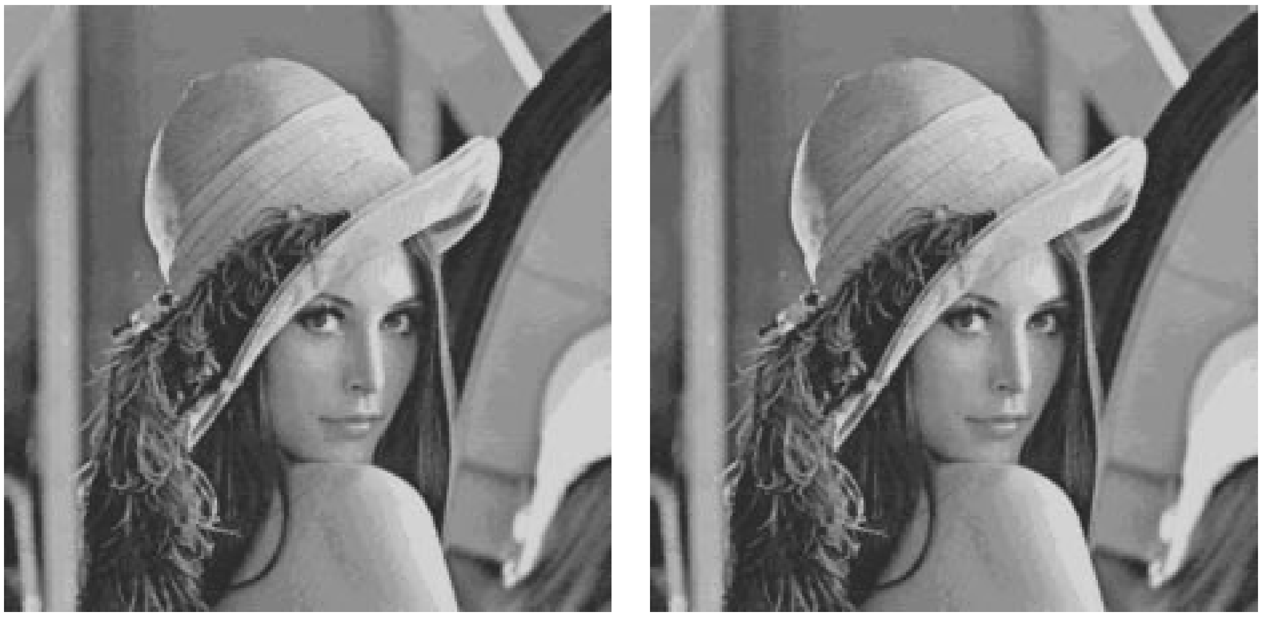
Figure 2.7: JSCTC vs tandem system performance for AWGN channel and gaussian source.

Two aspects are worth pointing out from this figure. First, the same behaviour as in figure 2.5 can be observed; that is, when the channel quality is optimum, the performance of both systems is equivalent. However, as the transmission channel worsens, the error correction capability of the convolutional code is exceeded and uncorrected transmission errors introduce large distortion to the decoded source. Notice, however, that in the range $0 - 5$ dB of channel SNR, the tandem system is closer to its theoretical limit than the JSCTC system.

The other aspect is complexity. Two trellis search algorithms are required by the tandem system, one for the trellis quantization and one for decoding the convolutional code. As we will see later, the complexity of trellis search algorithms increases exponentially with the constraint length of the trellis search. This is a good reason for prefering JSCTC over the tandem system.

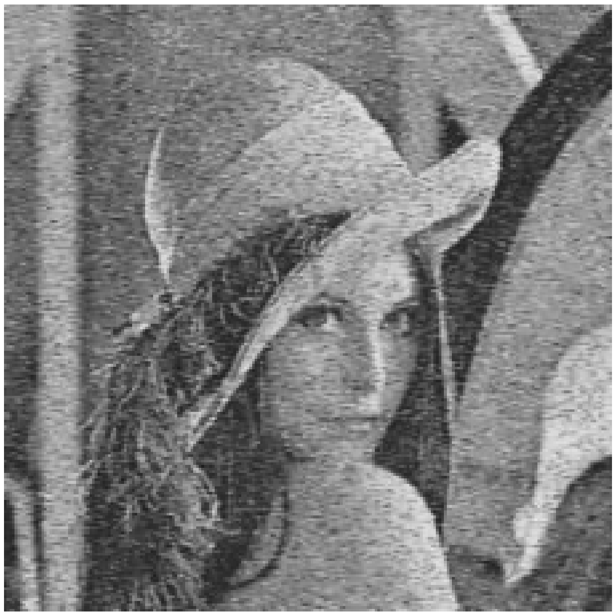
Finally, figures 2.9 through 2.11 present Lenna images originally coded with 8 bits per pixel (bpp) (255 grey levels) encoded with the JSCTC and noiseless quantizers, and transmitted through an AWGN channel for SNR=-5, 0 and 10 dB. Notice the improved image quality obtained with the JSCTC system when the channel is very noisy.

Figure 2.8: Original Lenna image 8 bpp.



(a) noiseless TQ; $K = 10$ *(b)* JSCTC; $K = 10$

Figure 2.9: Lenna image encoded at 1 bpp. SNR=10dB.


(a) noiseless TQ; $K = 10$

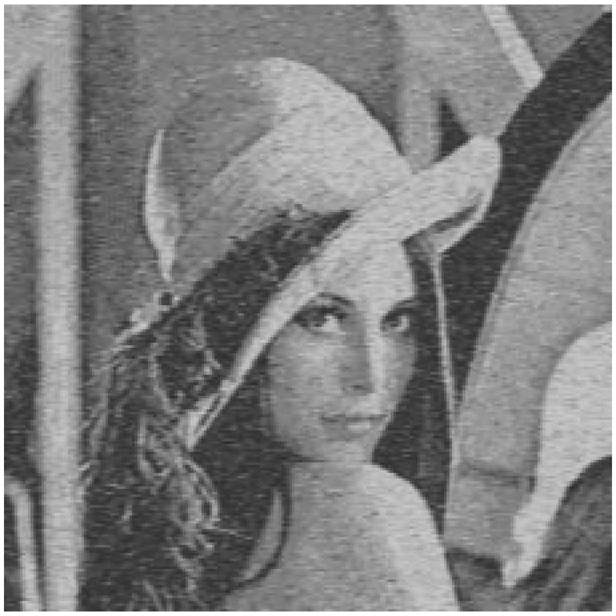
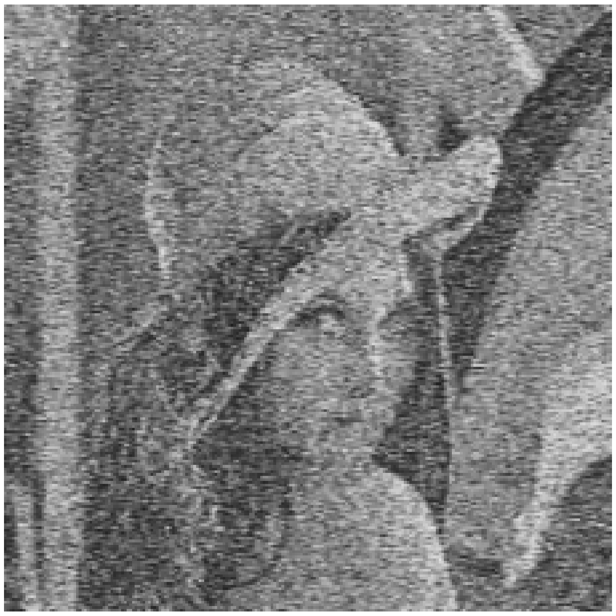

(b) JSCTC; $K = 10$

Figure 2.10: Lenna image encoded at 1 bpp. SNR=0dB.


(a) noiseless TQ; $K = 10$

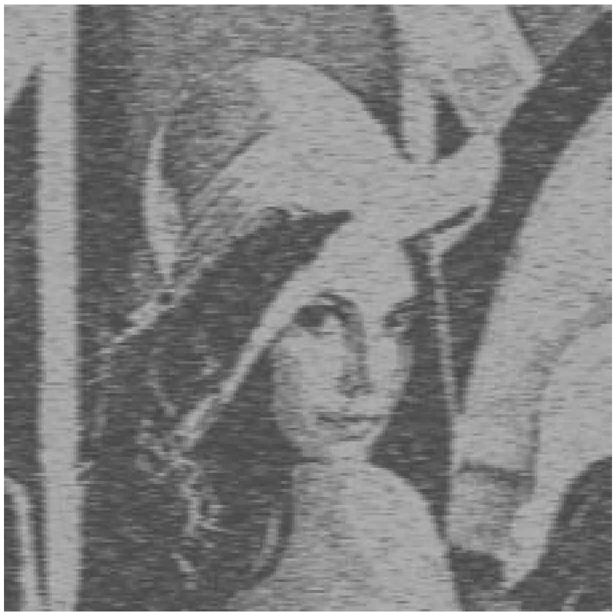

(b) JSCTC; $K = 10$

Figure 2.11: Lenna image encoded at 1 bpp. SNR=-5dB.

A final comment regarding the improved performance of the JSCTC technique over the noiseless trellis quantization is in order. Figures 2.10 and 2.11 highlight the reason why JSCTC introduces less distortion than the noiseless trellis quantizer during source decoding. Notice that the image decoded with the noiseless quantizer has a larger contrast, that is, the image has darker and brighter pixels than the images decoded with the JSCTC system. This is because the codebook of the noiseless quantizer has a larger range than the JSCTC codebook, as shown in figure 2.12. For instance, the higher and lower codewords of the noiseless codebook are 35 and 225, respectively; as a result, if an error occurs on a bit which had to decode a dark pixel and the decoder decodes instead a bright pixel, a large distortion is introduced. On the other hand, the higher and lower codewords of the JSCTC codebooks are 55 and 190 for the 0 dB codebook and 75 and 170 for the -5 dB codebook; since the codewords of the JSCTC codebook are closer to each other, a transmission error does not introduce distortions as large as the ones introduced by the noiseless decoder. Of course, as described in the example at the beginning of this section, at the encoder the image encoded with the JSCTC codebook is not so good.

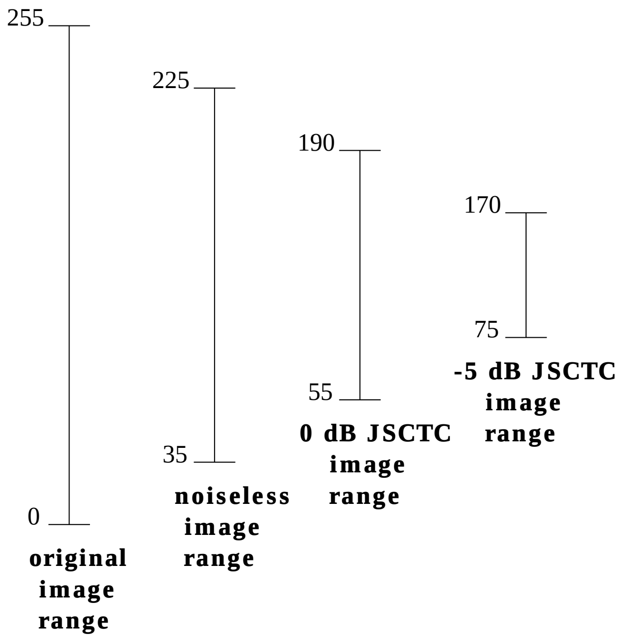


Figure 2.12: Pixel range of the image decoded with the noiseless TQ and JSCTC.

2.3 Conclusion

In this chapter we have seen the potential advantages of joint source-channel trellis coding over noiseless trellis quantization. Simulations were presented pointing out the superiority of this technique, specially when the quality of the transmission channel is very poor. Nevertheless, there is a price to pay since the computational load of the JSCTC distortion measure and the codebook design and codebook intialization algorithms are quite large.

This large complexity results in a limited size of the reproduction codebook which causes in turn a degradation in performance since the codebook codewords cannot represent the input source in the best possible way. As a consequence, means to reduce this computational load are required without being prejudicial to the system's performance. In the next chapters, ways to achieve this complexity reduction without penalizing the overall performance will be described. This reduction in complexity is related to the two main parts of the JSCTC algorithm, namely the computation of the distortion measure and the trellis search algorithm employed for source quantization.

Chapter 3

Joint Source-Channel Trellis Coding Architecture

In the previous chapter, we presented the underlying principles and theory of joint source-channel trellis coding. We saw that in this tehcnique the entire communication system is a trellis source coder-decoder pair which is able to outperform conventional tandem systems when the conditions of the transmission channel are very poor.

This chapter deals with hardware architectures for the implementation of the JSCTC technique presented in the previous chapter. These architectures comprises both the codebook design algorithm and the distortion measure required during the trellis quantization. The goal of this chapter is to study the transition from algorithm to architecture, that is, to accomplish an accurate adequation of the algorithm to be implemented, to an architecture that fulfils the performance requirements with an acceptable hardware complexity. When this cannot be achieved, a trade off performance-hardware feasibility is done, by modifying the architecture so that its hardware costs and/or processing time are reduced.

The outline of the Chapter is as follows. In section 3.1, two architectures for the codebook design algorithm are presented. Then, in section 3.2, we will show that the same architecture for the codebook design can be utilized to implement the distortion measure of the quantization algorithm because both tasks can be modified so as to meet the same type of arithmetic operations. Section 3.3 gives some measures to increase the performance of the architecture in terms of latency. Then, simple hardware modifications of the global architecture are highlighted in section 3.4, in order for the codebook initialization algorithm to be implemented in the hardware architecture.

3.1 Architecture for Codebook Design

At a first glance, it may appear that the hardware implementation of the codebook design algorithm is of no importance since this kind of tasks are generally done off-line, in software, prior to the quantization of the source. Nevertheless, as indicated in [27], in some applications the high complexity of this task and the need for repeated access to large files of training data generally places severe limitations on the size of the training sequence used for the codebook design procedure. In such cases, a hardware architecture for the codebook design is needed. Moreover, since the trellis quantization process is embedded into the codebook design algorithm, the hardware ressources are not largely increased if the optimization algorithm is implemented in hardware. In addition, we will see that the distortion measure of the quantization process can employ the same hardware ressources.

As we saw in the previous chapter, the codebook design algorithm alternately performs two operations:

1. Trellis quantization of the training sequence employing a trellis search algorithm,
2. Codebook update which calculates the centroid equation given by

$$y_j^{m+1} = \frac{1}{\sum_{i=0}^{2^K-1} Pr(y_j|y_i) \cdot ||Q_i||} \sum_{i=0}^{2^K-1} Pr(y_j|y_i) \sum_{k:x_k \in Q_i} x_k \tag{3.1}$$

Recall that a training sequence $\mathbf{x}$ of length L_{TS} is partitioned by the quantization operation in 2^K (or less) sets Q. Each set Q_i ($i = 0 \cdots 2^K - 1$) contains the training samples x_k that were represented by codeword y_i during the trellis quantization, and $||Q_i||$, the cardinality of Q_i, denotes the total number of training samples that were coded with this codeword. The quantization process produces in turn a binary sequence or channel sequence $\mathbf{u}$ which is the compressed version of the training sequence.

Equation (3.1) suggests two ways of implementing the codebook update. These two ways are described by the pseudo-program of figure 3.1.

In words, the first pseudo-program states that a first step would consist in accumulating the training samples that were coded by the same codeword y_i and computing the cardinality of its corresponding partition set Q_i, and then, a second step consists in performing the centroid computation weighted by the transition error probabilities $Pr(y_j|y_i)$. The other possibility given by the second pseudo-program is to compute the numerator and denominator terms of equation (3.1) at the same time as the training sample x_k is being decoded, and then perform the corresponding divisions so that the final codebook is updated.

```
1. Codebook Update with respect to codeword label $i$
    /*Trellis encoding*/
    for k=0 to $L_{TS}$
    {
      $u_k$ = trellis_encoding($x_k$)
      codeword_label = decode_codeword_label($u_k$)
      $Acc_{codeword\_label}$ += $x_k$
      $||Q_{codeword\_label}||$ = $||Q_{codeword\_label}|| + 1$
    }

    /*Centroid computation*/
    for j=0 to $2^{K-1}$
      for i=0 to $2^{K-1}$
      {
        $Pr(y_j|y_i)$ = channel_transition_probability(i,j)
        $NUM_j$ += $Pr(y_j|y_i)$ $\cdot$ $Acc_i$
        $DEN_j$ += $Pr(y_j|y_i)$ $\cdot$ $||Q_i||$
        $y_j^{m+1}$ = $\frac{NUM_j}{DEN_j}$
      }

2. Codebook Update with respect to codeword label $j$
    /*Simultaneous trellis encoding and*/
    /*generation of numerator and denominator terms*/
    for k=0 to $L_{TS}$
    {
      $u_k$ = trellis_encoding($x_k$)
      codeword_label = decode_codeword_label($u_k$)
      for j=0 to $2^{K-1}$
      {
        $Pr(y_j|y_i)$ = channel_transition_probability(j,codeword_label)
        $NUM_j$ += $Pr(y_j|y_{codeword\_label})$ $\cdot$ $x_k$    $(x_k \in Q_{codeword\_label})$
        $DEN_j$ += $Pr(y_j|y_{codeword\_label})$
      }
    }

    /*Centroid computation*/
    for j=0 to $2^{K-1}$
      $y_j^{m+1}$ = $\frac{NUM_j}{DEN_j}$
```

Figure 3.1: Pseudo-programs describing to ways of implementing the codebook design algorithm.

3.1.1 Architecture for Method 1

The hardware architecture for the codebook design algorithm, implemented with the first method is shown in figure 3.2. For simplicity, the trellis search algorithm will be seen as a black box and we will assume that it delivers the binary sequence $\mathbf{u}$ corresponding to the sequence of reproduction codewords $\mathbf{y_{i(k)}}$ that matches the input sequence $\mathbf{x}$ in the best way. Discussions about the trellis search algorithm and its hardware architectures will be presented in the next chapter.

The three steps of the codebook desing algorithm are highlighted in the figure:

1. **Trellis Encoding** (ligth grey in the figure): the output bit u_k delivered by the trellis quantizer enters the *codeword label decoder* block in order to know the label l_i of the reproduction codeword y_i that was used to represent the k^{th} training symbol. The training sample x_k is added to the previous training samples that were encoded with codeword y_i and the cardinality of this set is incremented. The accumulated values are then stored again in the *accumulation memory* at the address given by the decoded label l_i. This operation is repeated until all the training sequence is processed.

2. **Halt Test** (darker grey in the figure): the distortion D^m introduced by the trellis quantization is compared to the distortion D^{m-1} of the previous iteration according to the halt condition (see equation 1.14 in page 14). If the halt condition holds, the circuit stops and the current codebook C^m is used for the quantization of the source. Otherwise, step 3 is performed.

3. **Codebook Update** (darkest grey in the figure): In this step, *counter1* and *counter 2* start working. *Counter1* serves to generate the codeword labels l_j whereas *counter 2* generates the codeword labels l_i of the centroid equation (3.1). *Counter 2* is now used as the address of the *accumulation memory* to retrieve the $\sum_{Q_i} x_k$ and $||Q_i||$ terms. At the same time, the ouputs of the two counters enter the *channel transition probability* block to compute the error probabilities $Pr(y_j|y_i)$. This value is then multiplied by the outputs of the *accumulation memory.* The results of this operation are stored in flip-flops *acc*. This way, the NUM_j and DEN_j terms of pseudo-program 1 are being computed. The process is repeated until *counter 2* finishes counting. Next, the division is done with the contents of the *acc* flip-flops, and the new codeword y_j^{m+1} is stored in the *codebook memory.* Afterwards, *counter 1* is increased by one, *counter 2* restarts working and the process is repeated for the new value of label l_j. The whole procedure is repeated until *counter 1* reaches its final value, indicating that the entire codebook has been updated.

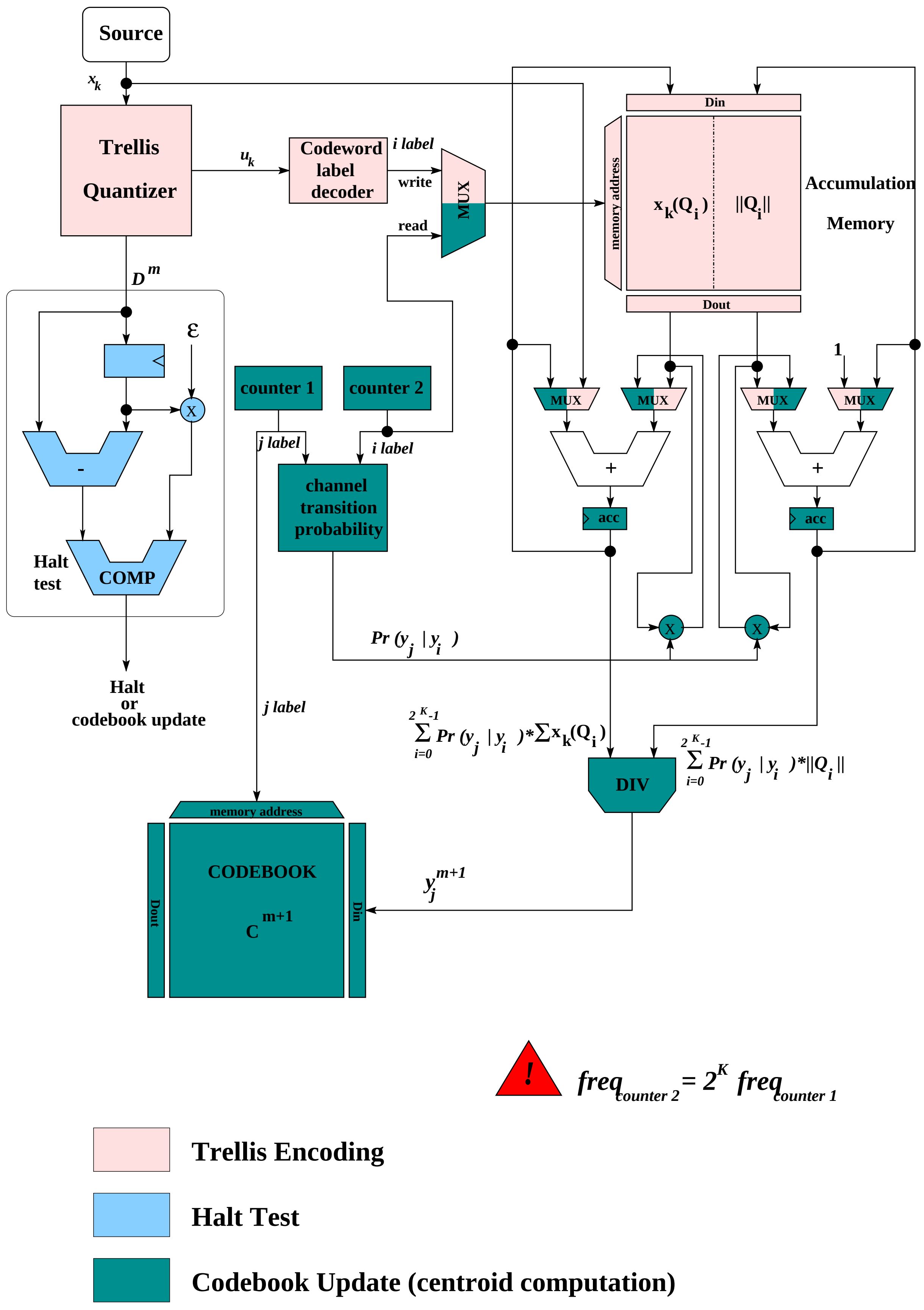


Figure 3.2: Hardware architecture for the first method of codebook design.

Notice that the frequency of *counter 2* is 2^K times faster than the frequency of *counter 1*

$$\frac{freq_{counter2}}{freq_{counter1}} = 2^K \tag{3.2}$$

3.1.2 Architecture for Method 2

Figure 3.3 shows the hardware architecture for the second pseudo-program. The steps towards the codebook update are:

1. **Simultaneous Trellis Quantization and NUM and DEN accumulation** (light grey): For every training sample x_k, its corresponding bit u_k is decoded as label l_i. *Counter 1* acts as the label generator of codewords y_j. The channel transition probability with respect to the decoded label and counter 1's label is computed. This probability is multiplied by the training sample x_k and the result as well as the error probability are accumulated to the contents of the accumulation memory whose address is given by *counter 1*. This way, the NUM_j and DEN_j terms of the second pseudo-progam are being computed and stored in their corresponding address. This operation is repeated until *counter 1* finishes its count and until the last training sample is quantized.
2. **Halt Test** (darker grey): After the training sequence has been quantized, the quantizer has the m^{th} iteration distortion. The halt condition is checked.
3. **Codebook Update** (darkest grey): If the codebook update is needed, *counter 1*, which generates the l_j lables, reads the contents of the *accumulation memory* where the NUM_j and DEN_j terms were stored. The division is computed and the result is stored in the *codebook memory*. The write address is given by *counter 1*.

In this case, the operation frequency of *counter 1* is 2^K times the frequency of the system

$$\frac{freq_{counter1}}{freq_{system}} = 2^K \tag{3.3}$$

A final remark is in order. As indicated earlier, the way the *accumulation memory* is used changes in each approach. In the first architecture, each memory location contains the accumulated training symbols belonging to a given set Q_i together with its cardinality $||Q_i||$. On the other hand, in the second architecture each memory location contains the contribution of each training symbol x_k belonging to set Q_i, to the new value of codeword y_j. Notice that each contribution is weighted by the channel transition probability.

The following sections describe the inner structures of the *codeword label decoder block* and the *channel transition probability* block.

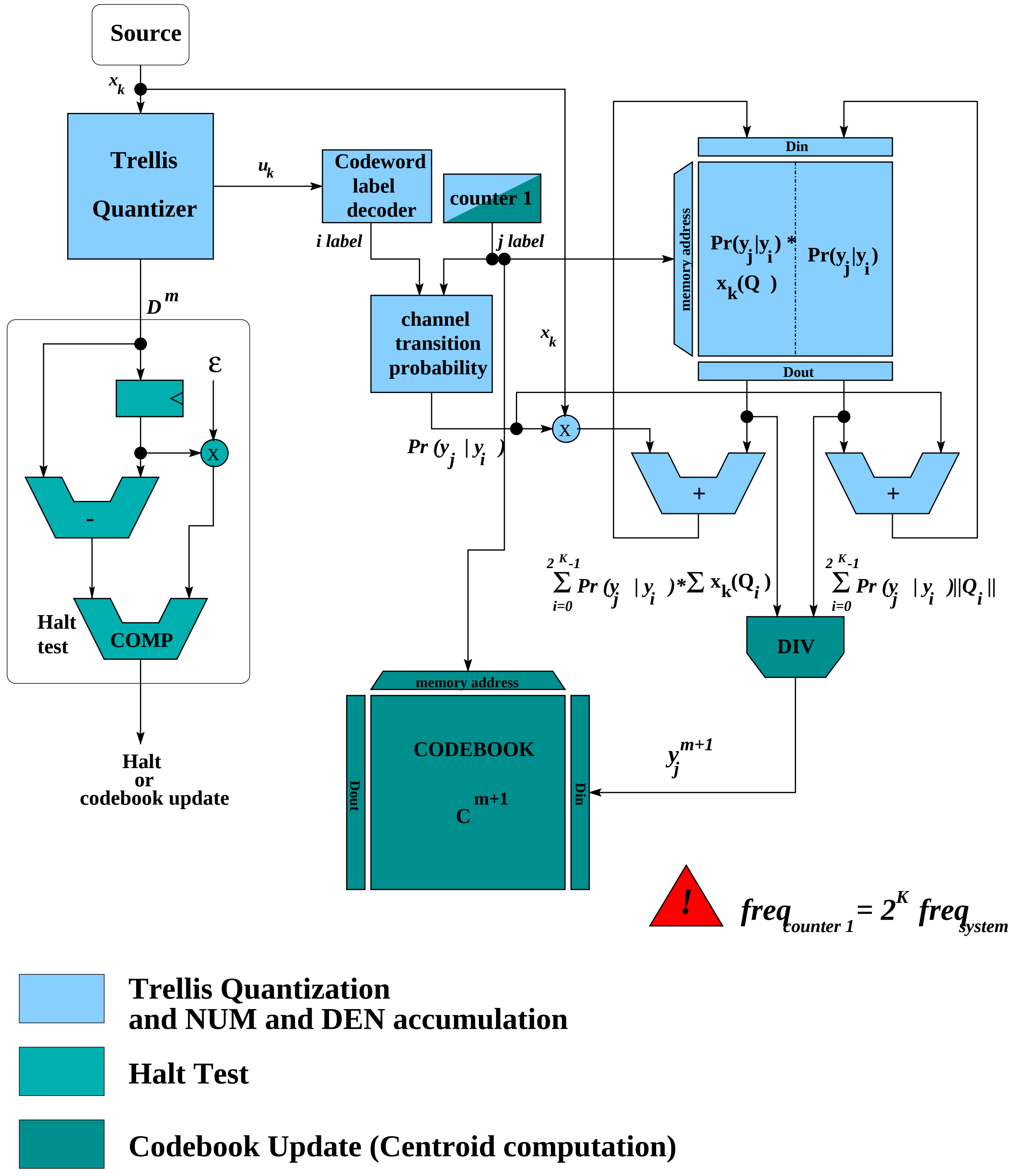


Figure 3.3: Hardware architecture for the second method of codebook design.

3.1.3 Codeword Label Decoder

The *codeword label decoder* can be implemented with a K-bit shift register. Figure 3.4 depicts the label decoder. The bitstream $\mathbf{u}$ enters the shift register; the output of each register re-creates the trellis states visited by the sequence $\mathbf{u}$ during the training sequence quantization, as indicated by the trellis diagram in the figure.

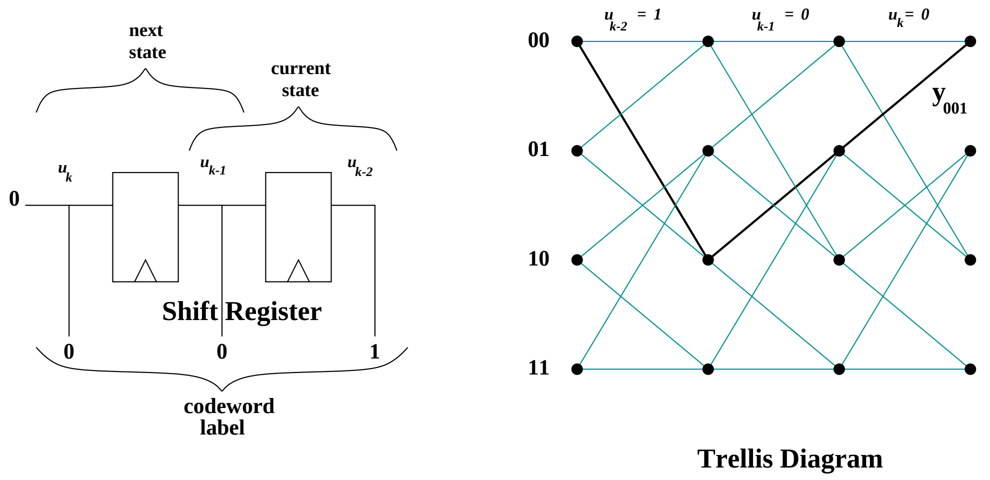


Figure 3.4: Codeword Label Decoder.

The codeword label is formed by the current input bit u_k and the outputs of the shift register. In this K-bit word, the $K-1$ LSBs denote the current state of the trellis and the $K-1$ MSB bits are the next state. The codeword label is thus $y_{u_k,u_{k-1},u_{k-2},\cdots,u_{k-K+1}}$.

3.1.4 Channel Transition Probability

The *channel transition probability* generator is described in figure 3.5. This block is implemented with an array of K XOR gates, an encoder circuit and a memory containing all the channel transition probabilities $Pr(y_j|y_i)$. Remember that for a binary symmetric channel the channel transition probability is given by

$$Pr(l_j|l_i) = p^{d(l_i,l_j)} \cdot (1-p)^{K-d(l_i,l_j)} \tag{3.4}$$

where p is the error probability and $d(l_i, l_j)$ is the hamming distance between labels l_i and l_j.

The range of the hamming distance is

$$0 \leq d(l_i, l_j) \leq K \tag{3.5}$$

Hence, a $K+1$-word memory is needed to store all the possible channel transition probabilities. The memory address is given by the encoder block. This block translates the output of the XOR array into a $\log_2(K+1)$-bit word that indicates the number of differing bits between the two input labels l_i and l_j.

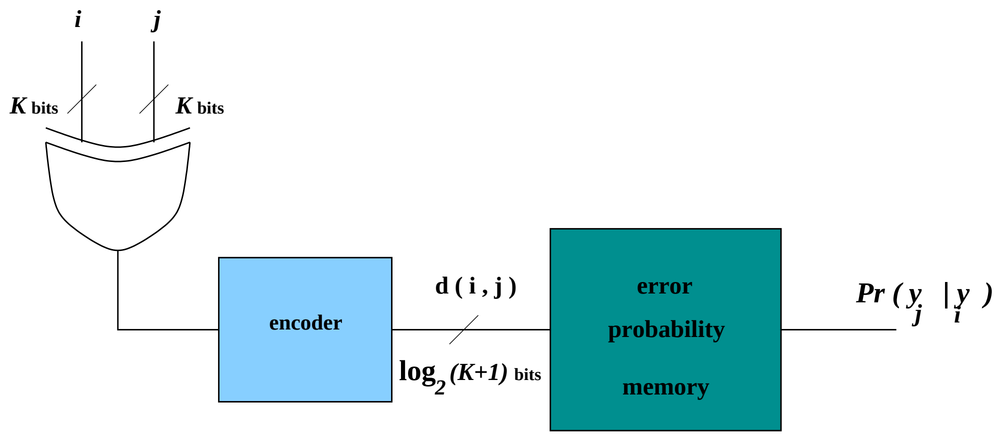


Figure 3.5: Channel Transition Probability Block.

3.2 Hardware Implementation of the Distortion Measure

A second issue in the robust trellis quantization algorithm is the implementation of the distortion measure of the trellis search algorithm. This distortion measure was defined as

$$d(x, y_i) = \sum_{j=0}^{2^K-1} (x - y_j)^2 \cdot Pr(y_j|y_i) \tag{3.6}$$

Table 3.1 shows the computational requirements per source symbol of this operation in terms of its arithmetic operations. As we can see, a direct hardware implementation of this expression demands a large amount of circuitry. Nevertheless, as in the case of vector quantization architectures [32, 62, 118], since the reproduction codebook remains unchanged between each codebook design iteration and during the quantization process, we can reduce the complexity of the distortion measure by precalculating the terms that do not depend on the input symbols x_k. Indeed, if equation (3.6) is expanded, the branch metric can be expressed as

Table 3.1: Computational requirements of the distortion measure in the JSCTC algorithm.

Operation	Type	Number of operations
$x - y_j$	substraction	$2^K \cdot 2^K$
$(x - y_j)^2$	square	$2^K \cdot 2^K$
$(x - y_j)^2 \cdot Pr(y_j \vert y_i)$	multiplication	$2^K \cdot 2^K$
$\sum_{j=0}^{2^K-1}(x - y_j)^2 \cdot Pr(y_j \vert y_i)$	addition	$2^K \cdot 2^K - 1$

$$d(x) = A_i x^2 - B_i x + C_i \tag{3.7}$$

where

$$\begin{aligned} A_i &= \sum_{j=0}^{2^K-1} Pr(y_j|y_i) = 1 & (3.8) \\ B_i &= 2 \cdot \sum_{j=0}^{2^K-1} y_j \cdot Pr(y_j|y_i) & (3.9) \\ C_i &= \sum_{j=0}^{2^K-1} y_j^2 \cdot Pr(y_j|y_i) & (3.10) \end{aligned}$$

Since $A_i = 1$, x^2 is common to all distortion measures and can thus be discarded. The B_i and C_i coefficients are now independent of the input samples x_k; thus, they can be computed off-line, prior to the quantization process. The distortion measure can now be expressed in the form

$$d(x) = -B_i x + C_i \tag{3.11}$$

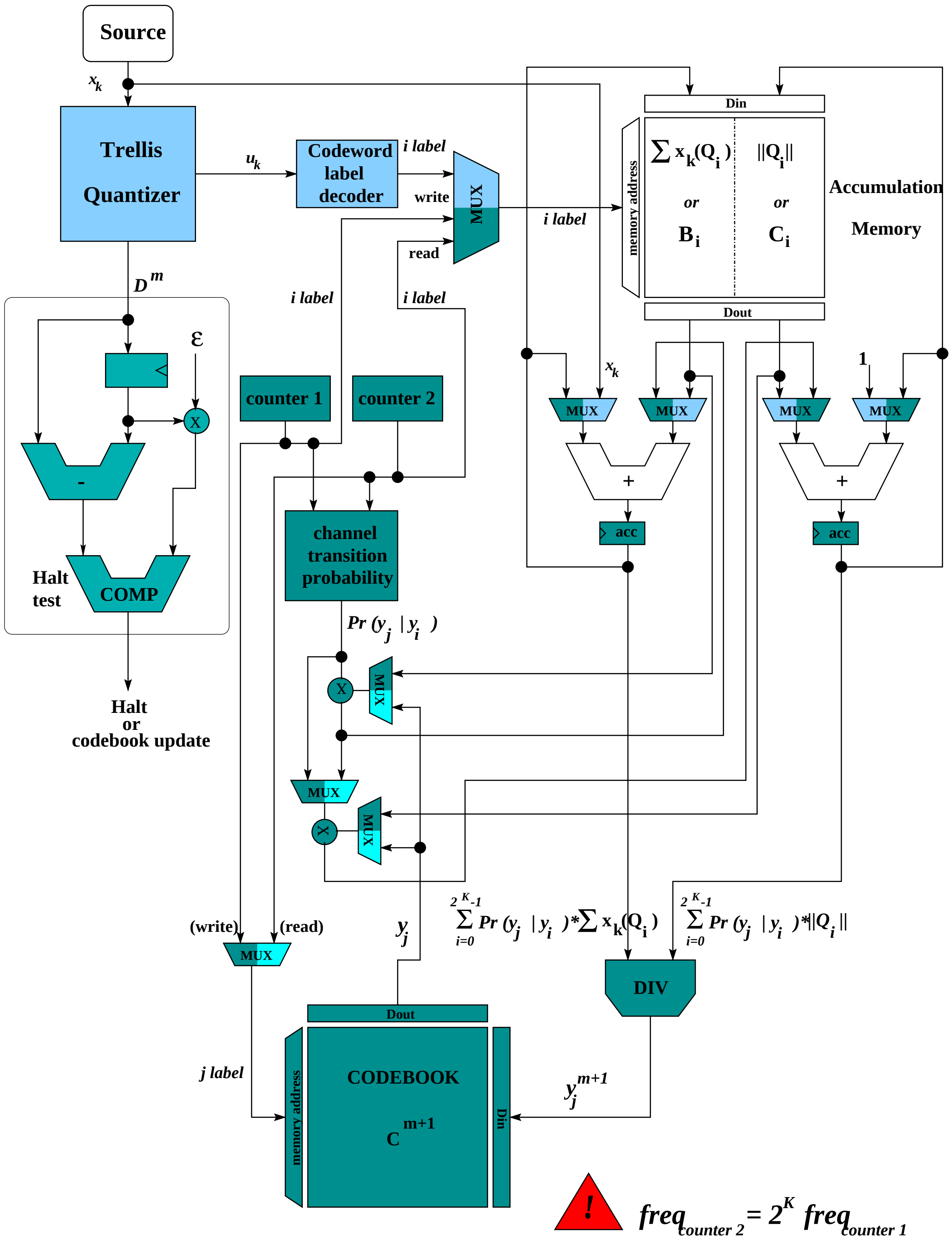
which consists in one multiplication and one addition per source symbol and per reproduction codeword. The complexity has been reduced from $O(2^{K^2})$ to $O(2^K)$.

If we take a closer look at the B_i and C_i coefficients, and the expression of the numerator and denominator terms of the centroid equation (3.1), we see that they consist in the same kind of operations. As a result, the same architecture proposed for the codebook update can be used for the computation of the B_i and C_i coefficients. All what is left to do is to manage the sharing of the different hardware ressources.

3.2.1 Global Architecture for Method 1

The complete hardware architecture for method 1 is presented in figure 3.6. The functioning of this architecture is the following:

1. **Trellis Quantization**: The operation of this step is the same as the one described in figure 3.2.
2. **Halt Test**: *idem*
3. **Codebook Update and B_i, C_i Calculation**: In this step, *counter 2*, which generates the l_j labels, reads the *accumulation memory* and the *accumulator circuit* (adder and flip-flop acc) produces the NUM_j and DEN_j terms. When *counter 2* finishes counting, the divider takes the outputs of the two accumulators and calculates the new codeword y_j^{m+1} which is in turn stored in the *codebook memory.* The address bus of the *codebook memory* is controlled by *counter 1.* When all the codewords are updated, the control of the address bus corresponding to the *codebook memory* is now passed to *counter 2*; conversely, *counter 1* commands the address bus of the *accumulation memory.* The new codewords y_j^{m+1} are read, multiplied by the channel transition probabilities and accumulated to the previous processed codewords in the *accumulator circuit.* This process is iterated until *counter 2* arrives to its maximum value. Then, the accumulator ouputs are stored in the *accumulation memory* at the address given by *counter 1.* This way, the first B_i and C_i coefficients are calculated. At the end, when *counter 1* finishes counting, all the B_i and C_i coefficients have been computed.



🟦 **Trellis Encoding**

🟩 **Halt Test**

◩ **Codebook Update (centroid computation) and B_i, C_i computation**

Figure 3.6: Global architecture for method 1.

3.2.2 Global Architecture for Method 2

The complete hardware architecture for method 2 is described in figure 3.7. Its functioning consists in the following steps:

1. **Trellis Quantization**: the same as in figure 3.3.
2. **Halt Condition**: *idem*
3. **Codebook Update**: *idem*
4. B_i **and** C_i **Calculation** (lightest gray): *Counter 1* commands the address bus of the *codebook memory* and *counter 2* commands the*accumulation memory*. Codewords $y^{m+1}_{counter1}$ are read from the *codebook memory*, they are multiplied by the channel transition probabilities $(Pr(y_{counter1} \mid y_{counter2})$ and accumulated to the previous processed codewords in the *accumulator circuit*. When *counter 1* finishes counting, the *accumulator* ouputs are stored in the *accumulation memory* whose address is given by *counter 2's* value. This way, when *counter 1* finishes counting, all the B_i and C_i coefficients are computed.

A final comment on the mutual independence of the codebook design and the quantization process is in order. In fact, both operations are not totally independent since the trellis quantizer starts decoding the bits u_k before the entire training sequence has been encoded. As a consequence, the B_i and C_i coefficients would be lost if the training samples of a given set Q_i and its cardinality $||Q_i||$ are accumulated and stored in the accumulation memory. A solution to this problem is to use a buffer that stores the bitstream $\mathbf{u}$ until all the training sequence is coded, or to employ two accumulation memories, one dedicated specially to the centroid computation and the other to the computation of the B_i and C_i coefficients.

3.3 Measures for Increasing Performance

In the last section two hardware architectures for the codebook update and the distortion measure were presented. Each approach has its advantages and disadvantages. For instance, if we assume that the clocks of counter 2 and the trellis quantizer in the first architecture are the same, the codebook is updated after $L_{TS} + 2(2^K \cdot 2^K)$ cylces. The first L_{TS} cycles are needed to decode the codeword labels and $2^K \cdot 2^K$ cycles are needed to update the codebook. Afterwards, other $2^K \cdot 2^K$ cycles are needed to compute the B_i and C_i coefficients.

Concerning architecture 2, if counter 1 has the same frequency as counter 2 of architecture 1, the circuit takes $2^K \cdot L_{TS} + 2^K$ cycles to update the codebook. The first $2^K \cdot L_{TS}$ cycles are due to the fact that for each bit u_k, the terms $Pr(y_j|y_i) \cdot x_k$ have to be computed

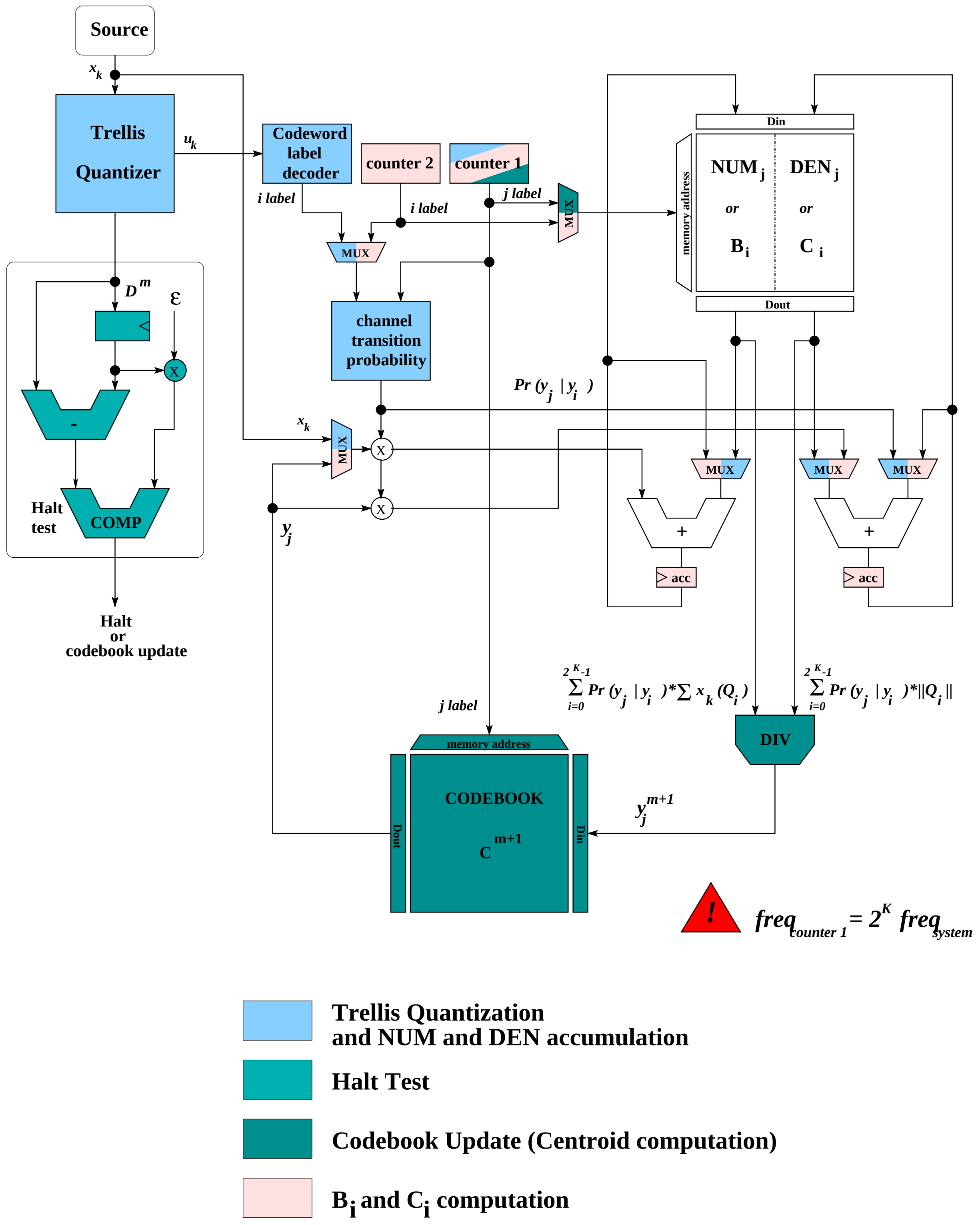


Figure 3.7: Global architecture for method 2.

for each j ($j = 0 \cdots 2^K$). The last 2^K cycles are required to divide the NUM_j and DEN_j terms.

Table 3.2 summarizes the latency of each approach. As an example, consider a training sequence of 100 samples and a $K = 3$ constraint length trellis quantizer. The first architecture requires 164 cycles to complete one iteration of the codebook desing whereas the second architecture needs 808 cylces. Eventhough the latency of the second architecture is far larger than that of the first architecture, its optimization is easier.

Table 3.2: Codebook Design Latency of Architectures 1 and 2.

Operation	Architecture 1	Architecture 2
Codebook update	$L_{TS} + 2^K \cdot 2^K$	$2^K \cdot L_{TS}$
B_i and C_i computation	$2^K \cdot 2^K$	2^K

It is well-known that sequential processing of signals leads to the throughput being the critical measure. In addition to the throughput, storage requirements, regularity and modularity, and parallelism are further characteristics that are relevant to the implementation. Because of the iterative and sequential nature of the robust trellis quantization algorithm, pipelining for circuit optimization is of little help. We can however appeal to parallelism to optimize and increase the throughput.

At first thought, it can be argued that circuit optimization through the use of parallelism is not important since the codebook design operation is performed only once, prior to the source quantization. Nevertheless, since the B_i and C_i coefficients, needed during the trellis quantization, are computed in the same circuit as for the codebook design, parallelization of this architecture may be required if the trellis search algorithm is to be parallelized as well. Indeed, for high speed applications, it is common use to parallelize the trellis search algorithm (usually a Viterbi decoder) so as to increase the system throughput [39]. For instance, in applications where a total parallelism is required, each trellis state is provided with its own processor to update its metric. As a result, 2^K distortion measures must be simultaneously available in order for the $2^K - 1$ processors to update the state metrics at the same time. In our case, if the computation of the B_i and C_i coefficients is parallelized, the trellis quantizer architecture can also be parallelized; this way, the system throughput is increased. Moreover, the latency of the codebook design procedure is also reduced. Thus, we kill two birds with a single shot. In the following sections, parallel architectures for the two methods are proposed.

3.3.1 Parallelization of Method 1

The parrallel structure of the architecture for method 1 is shown in figure 3.8. In this case, the circuit is duplicated n_1 times so that the accumulation memory is divided into n_1 memory banks. Each block has its own accumulator and multiplier circuits. Since both counters control the address bus of the accumulation memory, they have to be modified in such a way that counter 1 ranges from 0 to $2^K - 1$ in the codebook updating mode and from 0 to $\left(\frac{2^K}{n_1} - 1\right)$ in the B_i and C_i computation mode. Conversely, counter 2 ranges from 0 to $\left(\frac{2^K}{n_1} - 1\right)$ in the codebook updating mode and from 0 to $2^K - 1$ in B_i and C_i computation mode. This is due to the fact that the accumulation memories are reduced to $\left(\frac{2^K}{n_1} - 1\right)$ memory addresses whereas the codebook memory remains with 2^K memory addresses. The circuit works as follows:

1. **Trellis Quantization** (lightest grey): The codeword label l_i decoded with input bit u_k enters the *control* block which serves to generate the memory address of the memory bank corresponding to that codeword label. In other words, from the K bits that form the codeword label l_i, the $\log_2 n_1$ MSB bits are used to select the *accumulation memory* bank, and the rest are used to select the memory address of the selected memory bank. The process continues until all the training samples have been coded.

2. **Codebook Update**: In this mode, the $\sum x_k$ and $||Q_i||$ terms of each memory bank are read with *counter 2*. These terms are multiplied by the channel error probability and accumulated in their corresponding *accumulator circuits*. *Counter 1's* value is used as the codeword label l_i needed for the channel error probability computation. Notice that when *counter 2* finishes, each *accumulator* in each block contains pieces of accumulated terms NUM_j and DEN_j. That is, block 1 contains NUM_0 and DEN_0, NUM_1 and DEN_1, $\cdots$, $NUM_{\left(\frac{2K}{n_1}-1\right)}$ and $DEN_{\left(\frac{2K}{n_1}-1\right)}$; block 2 contains $NUM_{\left(\frac{2K}{n_1}\right)}$ and $DEN_{\left(\frac{2K}{n_1}\right)}$, $NUM_{\left(\frac{2K}{n_1}+1\right)}$ and $DEN_{\left(\frac{2K}{n_1}+1\right)}$, $\cdots$, $NUM_{\left(2\frac{2K}{n_1}-1\right)}$ and $DEN_{\left(2\frac{2K}{n_1}-1\right)}$, and so on. Thus,two n_1-operand adders are needed to compute the final NUM_j and DEN_j terms for the centroid computation. Finally, the respective divisions are performed to compute the new codeword y_j^{m+1} which is stored in the codebook memory.

3. B_i **and** C_i **Computation**: Once the reproduction codebook is updated, *counter 2* switches to the 2^K-range mode and *counter 1* to the $\frac{2^K}{n_1}$-range mode respectively. *Counter 2* reads in sequential order the new codewords for the computation of the new B_i and C_i coefficients as described previously. They are stored in the accumulation memory whose address bus is controlled by *counter 1*. At the end of this operation, each memory bank contains $\frac{2^K}{n_1}$ new B_i and C_i coefficients.

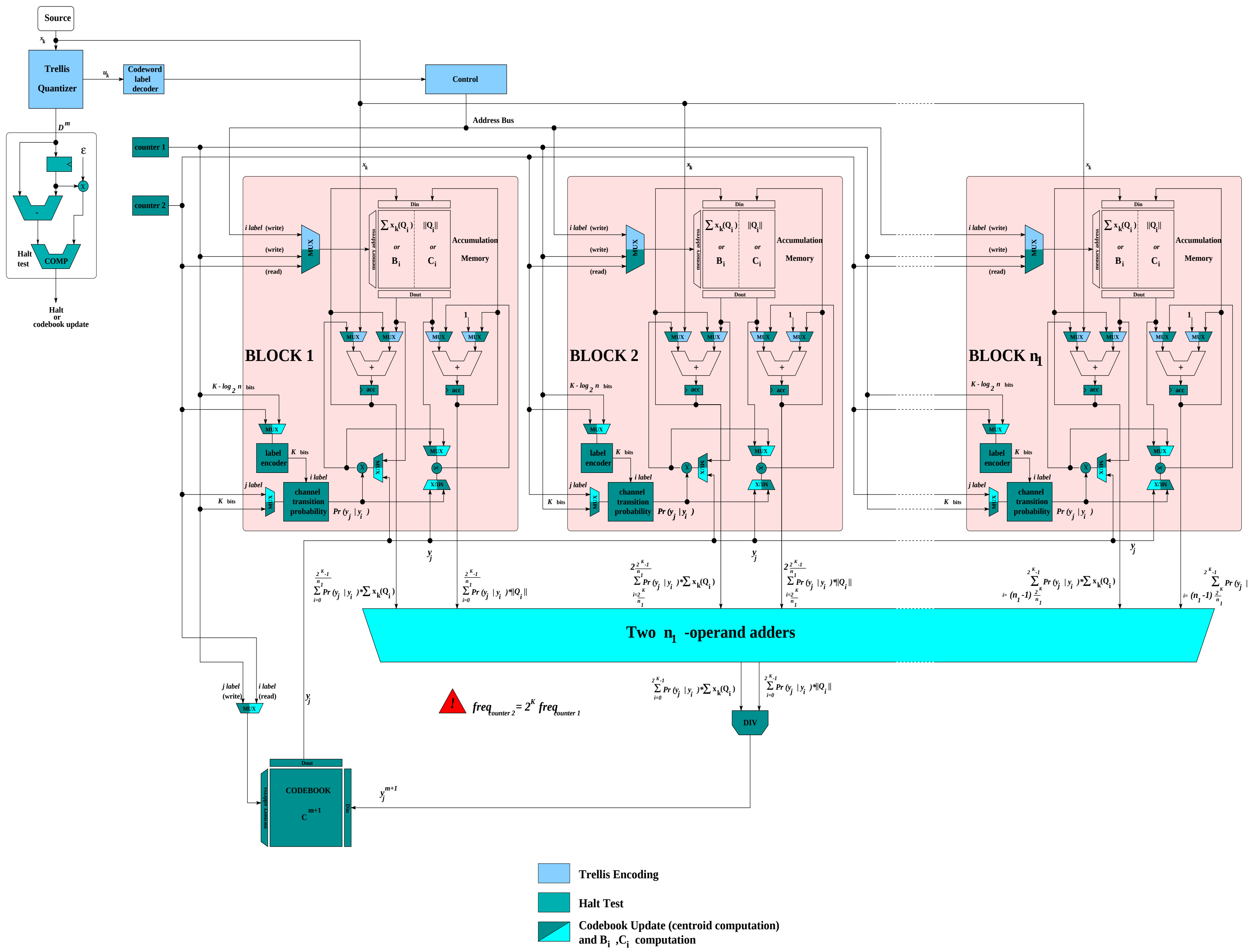


Figure 3.8: Parallel architecture of method 1.

Notice that when one of the two counters is working in the $\left(\frac{2^K}{n_1}\right)$-range mode, its output enters the *label encoder* block shown next to the *channel transition probability* block. This block converts the counters' current value into a 2^K-range value representing the codeword label l_i. This way, the channel transition probabilities can be calculated. One way to perform this conversion is to simply append $\log_2 n_1$ bits to the counter bits in each block so that a K-bit word is created. The value of this $\log_2 n_1$ bits is permanent in each block.

As an example, assume that $K = 6$ (32 trellis states) and $n_1 = 4$. We have 4 memory banks with 16 words each, addressed by the 4-bit counter. Hence, two bits need to be appended to the counter bits in order to create a 6-bit word that generates the 6-bit codeword label l_i. Now, if block 1 is reserved to the appending bits 00, then it contains the codeword labels from $00|0000$ up to $00|1111$. In the same manner, if block 2 uses the 01 bit pattern, it contains the labels $01|0000$ up to $01|1111$; codeword labels from $10|0000$ to $10|1111$ correspond to block 3 and the codeword labels ranging from $11|0000$ to $11|1111$ are reserved to block 4. This way, all the reproduction codewords are used in one iteration of counter 2.

It must be pointed out that parallelism is also possible in counter 1's "dimension". In other words, counter 1 can be constrained to the range $[0 \cdots \frac{2^K}{n_2} - 1]$ so that the computation of the new codebook is reduced. In this case, the codebook memory is divided into n_2 memory banks with $\frac{2^K}{n_2}$ words each. In addition, the computation time of the B_i and C_i coeffecients is also reduced. Nonetheless, there is a penalty in hardware complexity because in this new parallelization, n_2 n_1-operand adders and one final n_2-operand adder are needed to complete the codebook update and the coefficient computation. Moreover, since the computation of the B_i and C_i coefficients is performed only once, their parallelization is of no interest.

A final comment is in order. For parallel implementations of the trellis search algorithm, the B_i and C_i coefficients needed to update a single state metric must be stored in different memory banks so that they can be simultaneously read from memory. In this case, the label encoder described previously only has to change the position of its corresponding bit pattern. For instance, assuming that we have the same conditions as before, the processor that updates the state metric of the 0 state needs the B_0, C_0 and B_1, C_1 coefficients to compute the distortion measure. Hence, if the bit pattern for block 1 in figure 3.8 is $0xxxx0$, where $xxxx$ is the counter value, and the bit pattern for block 2 is $0xxxx1$, then the coeffcients B_0, C_0 and B_1, C_1 are available for the computation of the two distortion measures. In the same way, if the bit pattern for block 3 is $1xxxx0$, and that of block 4 is $1xxxx1$, then four distortion measures can be computed for the trellis branches steming from the same states.

3.3.2 Parallelization of Method 2

Figure 3.9 presents the parallel architecture for method 2. Like method 1, counter 1 ranges from 0 to $\left(\frac{2^K}{n_1} - 1\right)$ in codebook update mode and from 0 to $2^K - 1$ in B_i and C_i computation mode. Counter 2, on the other hand, ranges from 0 to $\left(\frac{2^K}{n_1} - 1\right)$ and is only used in the B_i and C_i computation mode. The two main modes of operation are:

1. **Trellis Quantization and Codebook Update**: It works in the same way as in figure 3.7 except for the computations duration. Now, only $\frac{2^K}{n_1}$ cycles are needed between successive bits u_k to compute the pieces of the NUM_j and DEN_j terms. The training sample x_k is distributed to all the n_1 blocks for the computation of each NUM_j and DEN_j term. At the end of the trellis quantization, *counter 1* restarts so that n_1 new codewords y_j^{m+1} are computed in parallel and stored in their corresponding codebook memory banks.

2. **B_i and C_i computation**: In this mode, *counter 1*, which now ranges from 0 to $2^K - 1$, reads the codewords y_j sequentially. These codewords are distributed to all the blocks for the computation, accumulation and storage of each B_i and C_i coefficient. *Counter 2* commands the memory address of the *accumulation memory.* Notice that the outputs of *counter 2* enter the *label encoder* block in order to convert its value into a K-bit word representing codeword label l_i. The *CE (chip enable)* block next to the codebook memories serves to enable that codebook memory storing the codeword corresponding to *counter 1's* value. This way, only one codeword is read at a time without having problems with the data bus.

In the same way as in architecture 1, parallelism for the computation of the B_i and C_i coefficients is also possible by giving the control of the codebook memory address bus to counter 1, instead of counter 2. In such a case, each accumulation memory contains partial accumulations of the B_i and C_i coefficients in the same way as for the NUM_j and DEN_j terms in architecture 1. As a consequence, a n_1-operand adder is needed to compute the total value of these coefficients. Nonetheless, as explained earlier, the optimization of the computation of these coefficients is not relevant.

3.3.3 Architecture Comparison

From a hardware ressources point of view, both architectures are practically the same except for the n_1-operand adder required by architecture 1. Table 3.3 shows a comparison of both architectures in terms of latency. For simplicity, we have assumed that one multiplication and one division can be done in one clock cycle.

Although the latency of architecture 1 is smaller, architecture 2 is easier to parallelize. Moreover, for full parallelism, the hardware ressources and the latency of architecture 2

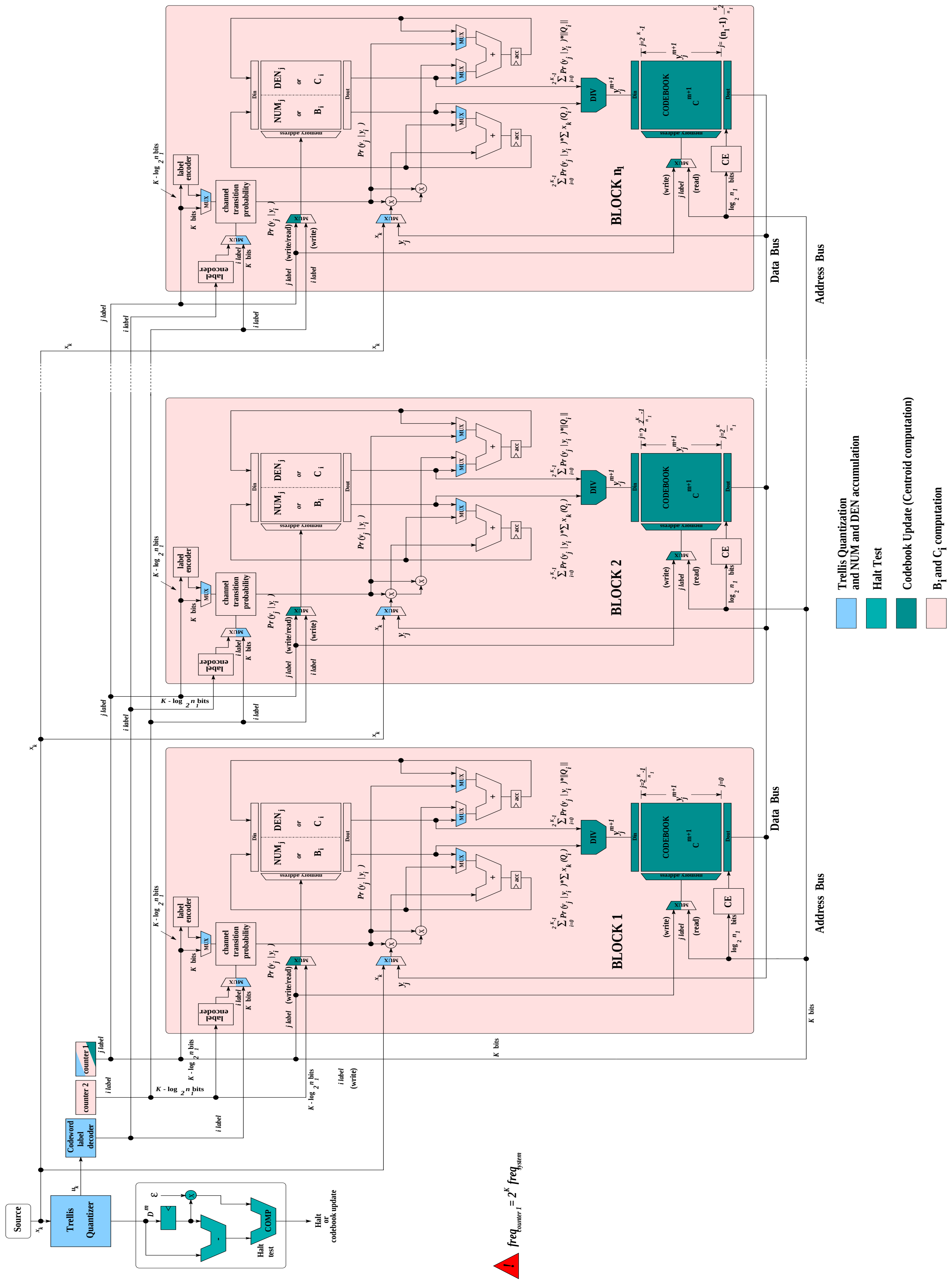


Figure 3.9: Parallel architecture of method 2.

Table 3.3: Latency comparison between architectures 1 and 2.

Operation	Architecture 1	Architecture 2
Trellis Quantization	L_{TS}	$\frac{2^K}{n_1} \cdot L_{TS}$
Codebook Update	$2^K \cdot \left(\frac{2^K}{n_1} + t_{adder}\right)$	$\frac{2^K}{n_1}$
B_i and C_i computation	$2^K \cdot \frac{2^K}{n_1}$	$2^K \cdot \frac{2^K}{n_1}$

($L_{TS} + 1 + 2^K$ cycles) is smaller than the latency of architecture 1. At the end, the target application dictates the choice of the architecture. For example, for a full parallel implementation of the robust trellis quantizer, architecture 2 is preferable over architecture 1; however, for sequential implementations, architecture 1 is more attractive.

3.4 Adaptability to the Extension Algorithm

In section 2.1.2 of chapter 2, we pointed out the great sensibility of the codebook design algorithm to the initial or seed codebook. Consequently, another algorithm, called the extension algorithm was introduced in order to solve the impasse on how the codebook design is initialized. Remember that the extension algorithm consists in iteratively using the codebook design algorithm, from a 1-bit scalar quantization ($K = 0$) to the final desired constraint length $K_{ex} = K_f$, where the final codebook of a given constraint length K_{ex} is used as the initial codebook for the $K_{ex} + 1$ constraint length trellis.

The architectures presented in this chapter can be easily modified to account for the extension algorithm. In the following, the different parts of the architecture, taking into account the extension algorithm are presented.

3.4.1 Shift Registers and Counters

Shift registers and counters can easily manage the extension algorithm by enabling only the first K_{ex} registers from the total length of K_f. This way, the codeword label decoder with total length K_f is able to decode K_{ex}-bit codewords during iteration *ex* of the extension algorithm. In the same manner, the counters addressing the accumulation and codebook memories are able to address $2^{K_{ex}}$ locations out of a total of 2^{K_f}.

3.4.2 Channel Transition Probability

The adaptation of this part of the system is a little bit more complicated. The channel transition probability of a binary symmetric channel was defined in equation (3.4). Figure 3.10 shows an example of the different channel transition probabilities as a function of the Hamming distance $d(i,j)$ for a $K = 3$ and $K = 4$-bit codeword labels. We can see that the channel transition probabilities of the $K = 4$-bit codeword labels can be updated from those of the $K = 3$-bit case. In general, the expression for updating the channel transition probabilities of a K-bit codeword labels from a $K - 1$-bit labels is

$$Pr^{K_{ex}}_{d(i,j)}(y_j|y_i) = Pr^{K_{ex}-1}_{d(i,j)}(y_j|y_i) \cdot (1-p) = Pr^{K_{ex}-1}_{d(i,j)-1}(y_j|y_i) \cdot p \tag{3.12}$$

that is, the channel transition probability of K_{ex}-bit codeword labels with Hamming distance $d(i,j)$ can be obtained from the channel transition probability of $Kex - 1$-bit labels with the same Hamming distance $d(i,j)$, multiplied by the term $(1 - p)$. Other option is to multiply the channel transition probability corresponding to the $Kex - 1$-bit labels with hamming distance $d(i,j) - 1$ by the error probability p. Notice that when $d(j,i) = K_{ex}$, the expression for updating the channel transition probabilities of the new constraint length coder becomes

$$Pr^{K_{ex}}_{d(i,j)=K_{ex}}(y_j|y_i) = Pr^{K_{ex}-1}_{d(i,j)=K_{ex}-1}(y_j|y_i) \cdot p \tag{3.13}$$

The new architecture to update the channel transition probabilities is shown in figure 3.11. In this case, a multiplexer and a multiplier were added to the circuit of figure 3.5. The multiplexer is controlled by the encoder output so that, when $d(i,j) = K_{ex}$, the memory output is multiplied by p; otherwise, the output is multiplied by $(1 - p)$.

In addition, a control circuit is needed in order to detect those channel transition probabilities that have already been updated. This can be done with a circuit based on flags, which are set to one when its corresponding hamming distance has been computed. This way, the updating of the channel transition probabilities is performed only once per hamming distance.

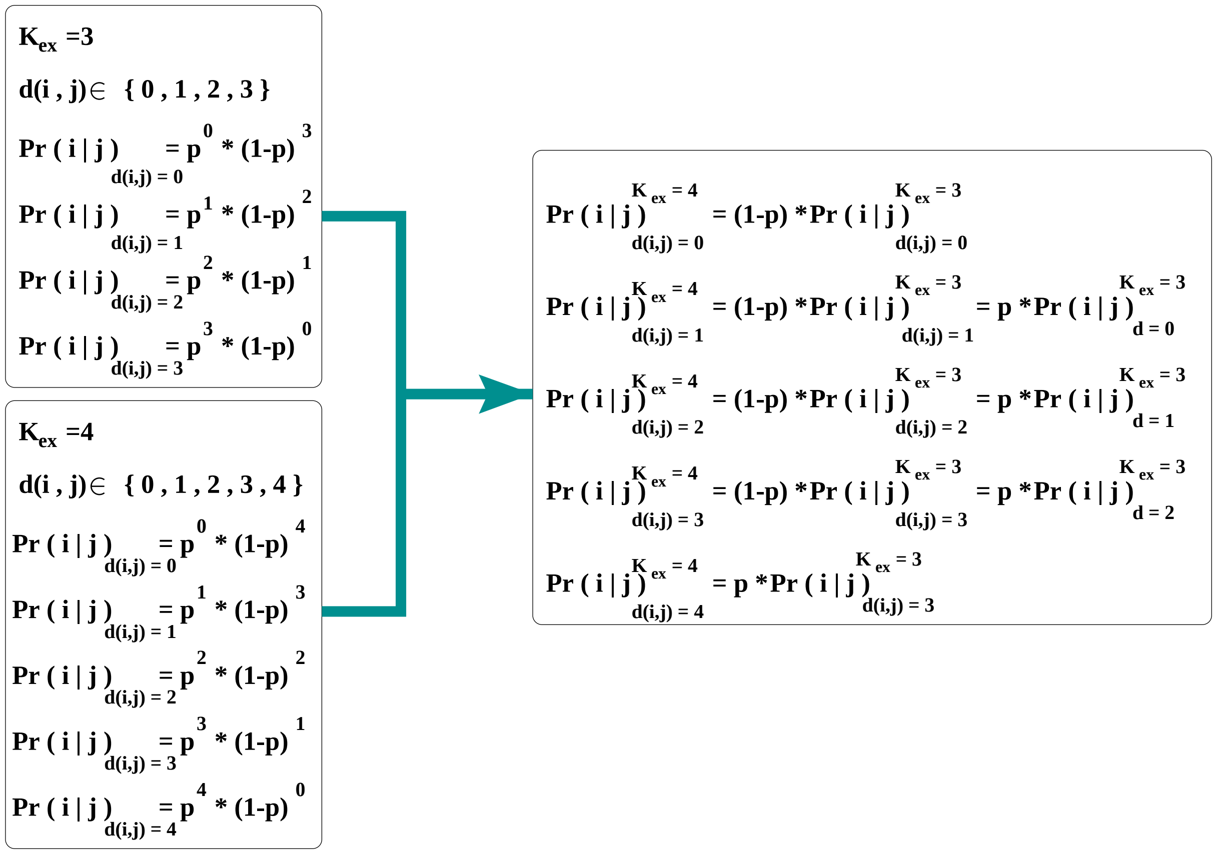

$K_{ex} = 3$
$d(i,j) \in \{0,1,2,3\}$
$Pr(i|j)_{d(i,j)=0} = p^0 * (1-p)^3$
$Pr(i|j)_{d(i,j)=1} = p^1 * (1-p)^2$
$Pr(i|j)_{d(i,j)=2} = p^2 * (1-p)^1$
$Pr(i|j)_{d(i,j)=3} = p^3 * (1-p)^0$
$K_{ex} = 4$
$d(i,j) \in \{0,1,2,3,4\}$
$Pr(i|j)_{d(i,j)=0} = p^0 * (1-p)^4$
$Pr(i|j)_{d(i,j)=1} = p^1 * (1-p)^3$
$Pr(i|j)_{d(i,j)=2} = p^2 * (1-p)^2$
$Pr(i|j)_{d(i,j)=3} = p^3 * (1-p)^1$
$Pr(i|j)_{d(i,j)=4} = p^4 * (1-p)^0$
$Pr(i|j)^{K_{ex}=4}_{d(i,j)=0} = (1-p) * Pr(i|j)^{K_{ex}=3}_{d(i,j)=0}$
$Pr(i|j)^{K_{ex}=4}_{d(i,j)=1} = (1-p) * Pr(i|j)^{K_{ex}=3}_{d(i,j)=1} = p * Pr(i|j)^{K_{ex}=3}_{d=0}$
$Pr(i|j)^{K_{ex}=4}_{d(i,j)=2} = (1-p) * Pr(i|j)^{K_{ex}=3}_{d(i,j)=2} = p * Pr(i|j)^{K_{ex}=3}_{d=1}$
$Pr(i|j)^{K_{ex}=4}_{d(i,j)=3} = (1-p) * Pr(i|j)^{K_{ex}=3}_{d(i,j)=3} = p * Pr(i|j)^{K_{ex}=3}_{d=2}$
$Pr(i|j)^{K_{ex}=4}_{d(i,j)=4} = p * Pr(i|j)^{K_{ex}=3}_{d(i,j)=3}$

Figure 3.10: Channel transition probability dependency between successive constraint lengths in the extension algorithm.

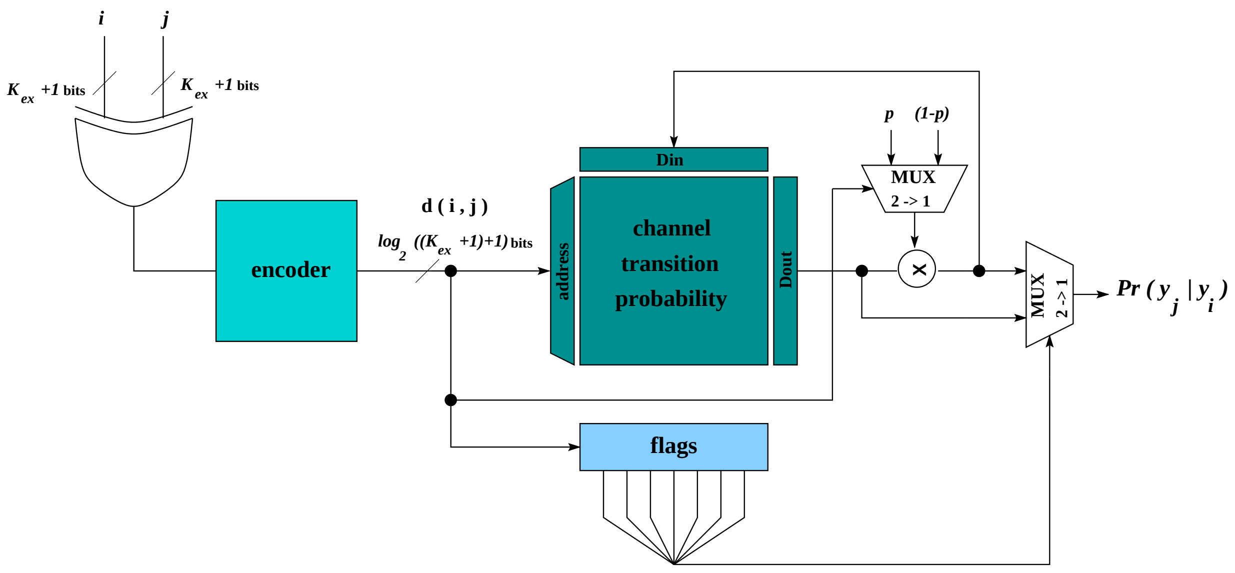


Figure 3.11: Architecture for updating the channel transition probability memory.

3.4.3 Codebook Memory

When the halt condition of the codebook design algorithm holds, the current codebook has to be duplicated from $2^{K_{ex}-1}$ codewords to $2^{K_{ex}}$. Instead of duplicating the codebook in the same codebook memory, we can proceed directly to the computation of the B_i and C_i coefficients of the new constraint lenght coder. To do so, each codeword y_j of the $2^{K_{ex}-1}$-codeword codebook is read twice and multiplied by the new updated channel error probabilities $P(y_{j0}|y_i)$ and $P(y_{j1}|y_i)$, where the j label ranges from 0 to $2^{K_{ex}-1}$. Each term is accumulated to the other codewords y_j $(j = 0 \cdots 2^{K_{ex-1}})$ that were also read twice. This way, in the first iteration of the codebook design optimization of the new K_{ex}-constraint length coder, $2^{K_{ex}}$ new B_i and C_i coefficients are ready to be used.

The codebook duplication procedure is shown in figure 3.12. When the counter used to read the codebook memory passes from $K_{ex} - 1$ to K_{ex} bits, only the $K_{ex} - 1$ MSBs are taken into account in the duplication procedure. This allows a double reading of each codeword y_j in successive time instants. Then, during a normal operation of the codebook optimization, all the counter bits are used to address the codebook memory.

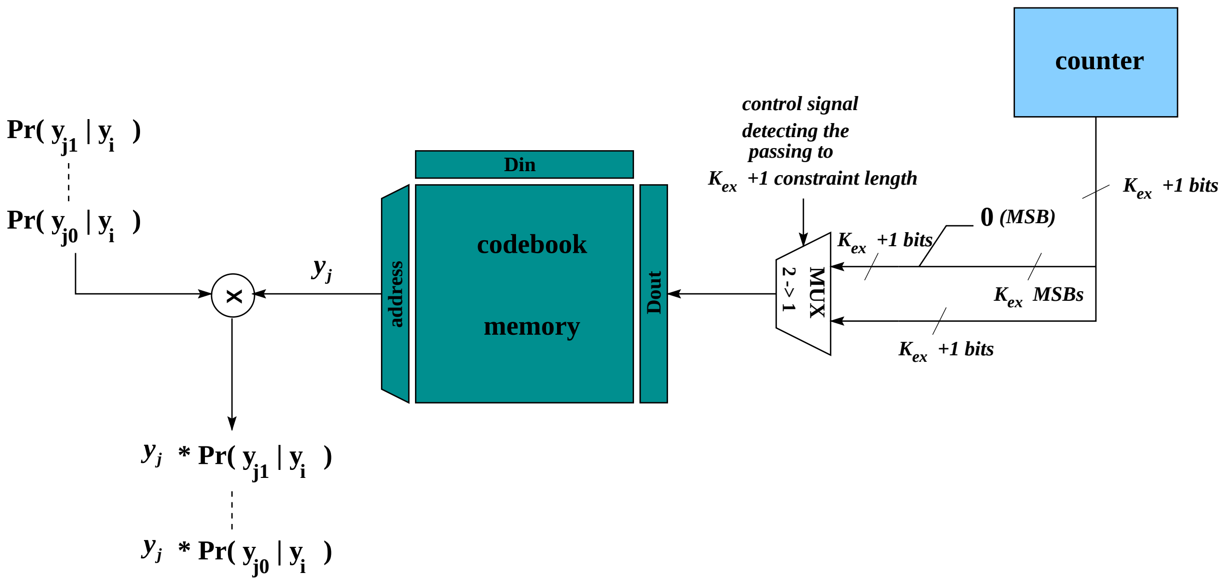


Figure 3.12: Architecture for codebook duplication in the extension algorithm.

3.5 Conclusion

In this Chapter we have presented two hardware architectures for the codebook design algorithm and the distortion measure of Ayanoğlu's JSCTC technique. One of the main drawbacks of this technique was the large computational load of the distortion measure $(O(N^2))$. This distortion measure takes into account all the reproduction codewords to quantize each symbol of the source sequence. Hence, when the reproduction codebook is large, the computational load becomes prohibitively complex. Nevertheless, following the same approach as in vector quantization, we showed that this computational load can be reduced to $O(N)$ by modifying the distortion measure and precomputing those terms in the modified distortion measure that do not depend on the source symbol. In addition, thanks to this modification in the distortion measure, the same architecture can be used for the codebook design procedure.

Measures to increase the performance of the proposed architectures were presented. These architectures allow both a parallelization of the trellis quantization and a reduction in the latency of the codebook design operation. Finally, these architectures were made reconfigurable in order to account for the extension algorithm. This way, the entire codebook design procedure can be implemented in hardware.

Chapter 4

Study of a Suboptimum VLSI Architecture for Joint Source-Channel Trellis Coding

This chapter introduces the basic component of the robust trellis quantizer: the trellis search algorithm. Roughly speaking, trellis search algorithms are methods whose goal is to find the path in a weighted graph wich connects two given nodes S_i and S_o, with the property that the sum of the weights of all the branches producing the path is minimized over all such paths. The sum of the partial weights of each branch is called a *metric* and indicates how well the path is being travelled over. In our case, the weight of all the branches in the trellis is the distortion measure discussed in the last Chapter and which is known as *branch metric*. So far we have considered the trellis quantizer as a black box delivering the best path in the trellis. In this chapter, architectural aspects for the trellis quantization are presented.

Among all the trellis search algorithms that exist, the Viterbi algorithm is perhaps the most famous and the one for which the most hadware architectures have been proposed. This is because the Viterbi decoder is the optimum trellis search algorithm for almost all trellis coding applications. Nevetheless, since our will is to continue with the algorithm-architecture adequation strategy, the goal of this chapter is to study the effects of using a suboptimum trellis search algorithm for the implementation of the JSCTC scheme. The idea is to reduce the harwdare complexity of the trellis quantizer while maintaining as much as possible the performace that would have been obtained with the use of the optimum Viterbi algorithm.

The chapter is organized as follows. Section 4.1 presents the basic principles of the Viterbi algorithm. A description of the conventional hardware architectures is presented. Then, in section 4.2 we give some arguments that make us reconsider the use of the Viterbi algorithm for the implementation of the robust trellis quantizer. More precisely, some aspects of the global communication system that is considered (only a trellis source coder-decoder pair) make us realize that the Viterbi algorithm may not result in the most efficient trellis search for our application. After a brief survey on suboptimum trellis search algorithms (section 4.3), we finally show in section 4.4 that the use of the M algorithm allows an important reduction in the system's hardware complexity while the overall system performance is maintained very close to that of the optimum system.

4.1 Architectural Issues of the Viterbi Algorithm

The aim of this section is to present the Viterbi algorithm in an architectural context. Different aspects of its hardware architectures such as operation, storage requirements, computational load, latency and measures for increasing its performance are outlined. This way, an assessement of the weak points of the algorithm can be done with respect to the system's overall performance so that alternative solutions can be proposed. In addition, by understanding this algorithm, the operation of suboptimum trellis searches can be elucidated more easily and the algorithm which is best suited to our needs can be selected.

4.1.1 The Viterbi Algorithm

The Viterbi Algorithm (VA) finds the best path in a trellis by calculating a measure of similarity, or distance, between a received sequence (pixels of an image or channel symbols) and all the trellis paths entering each state [43, 114]. This measure of similarity is called the *path metric*. The algorithm reduces the computational load by taking advantage of the trellis structure. When two paths enter the same state, the one having the best path metric is chosen; this path is called the *surviving path* of the state. There is a surviving path associated to each trellis state. The early rejection of the unlikely paths reduces the decoding complexity.

The algorithm can be divided into two main operations:

1. **Path metric updating**
2. **Survivor memory management**

During the **path metric updating** operation, the selection of the branches that create the surviving path arriving to each state is accomplished. This is done in the way described in figure 4.1 for a 4-state trellis. At every time step k, a path metric PM_i^k is assigned to each trellis state i ($i = 0 \cdots 2^{K-1}$). In order to calculate the new path metric of state j at time $k+1$, the branch metric $BM_{i \to j}^{k+1}$ connecting state i at time k to state j at time $k+1$ is computed and added to the path metric PM_i^k. In the figure there are two branches arriving to each state, thus, the new path metric of state j is found by the recursion expression

$$PM_j^{k+1} = min\left(PM_i^k + BM_{i\to j}^{k+1}\ ;\ PM_l^k + BM_{l\to j}^{k+1}\right) \qquad j = 0\cdots 2^{K-1} \tag{4.1}$$

that is, the path metric of state j at time $k+1$ is given by the minimum value between the addition of the branch metric connecting states i and j with the path metric of state j, and the addition of the branch metric connecting states l and j with the path metric of state l. This is done for all the trellis states. Equation 4.1 is referred to as the *Add-Compare-Select* (ACS) iteration.

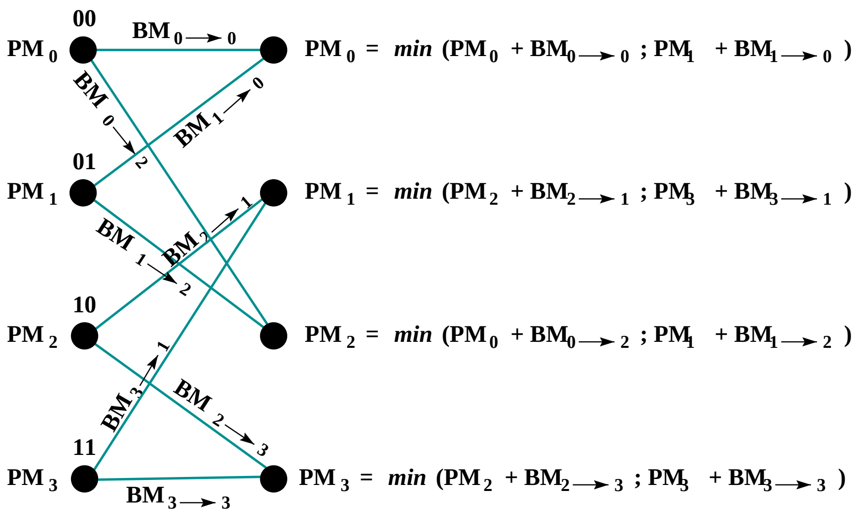

00
01
10
11
PM 0
PM 1
PM 2
PM 3
BM 0→0
BM 0→2
BM 1→0
BM 1→2
BM 2→1
BM 2→3
BM 3→1
BM 3→3
PM 0 = min (PM 0 + BM 0→0 ; PM 1 + BM 1→0)
PM 1 = min (PM 2 + BM 2→1 ; PM 3 + BM 3→1)
PM 2 = min (PM 0 + BM 0→2 ; PM 1 + BM 1→2)
PM 3 = min (PM 2 + BM 2→3 ; PM 3 + BM 3→3)

Figure 4.1: Path metric updating in the Viterbi algorithm.

In order to find the best path in the trellis, the algorithm must know which are the branches that constitute the surviving path arriving to each state. To do so, a *decision bit* is generated at every ACS iteration, one decision bit per trellis state. The decision bits generated at a given time instant k form a *decision vector* V_k. Conventionally, the decision bit is labeled as

0 if the surviving path arriving to a given state comes by the upper branch

1 if the surviving path comes by the lower branch

This way the algorithm can trace, at every time instant, the evolution of the surviving path arriving to each state.

Upon the processing of all the input symbols, the **path metric updating** operation is terminated. Each trellis state S_i has its own final path metric $PM_i^{L_{TS}}$ and its own sequence of decision bits u_i. Then, the **survivor memory management** operation takes place.

Starting from the trellis state having the best path metric, its decision bits are read in backward direction from the last generated decision bit to the first one. The decision bits allow the re-creation of the trellis states visited by the best surviving path in the forward direction during the **path metric updating** operation and hence the output binary sequence can be decoded.

Figure 4.2 shows an example of operation of the Viterbi algorithm for a source coding application, where the branch metric is the squared error. At the beginning, the path metrics are intialized as

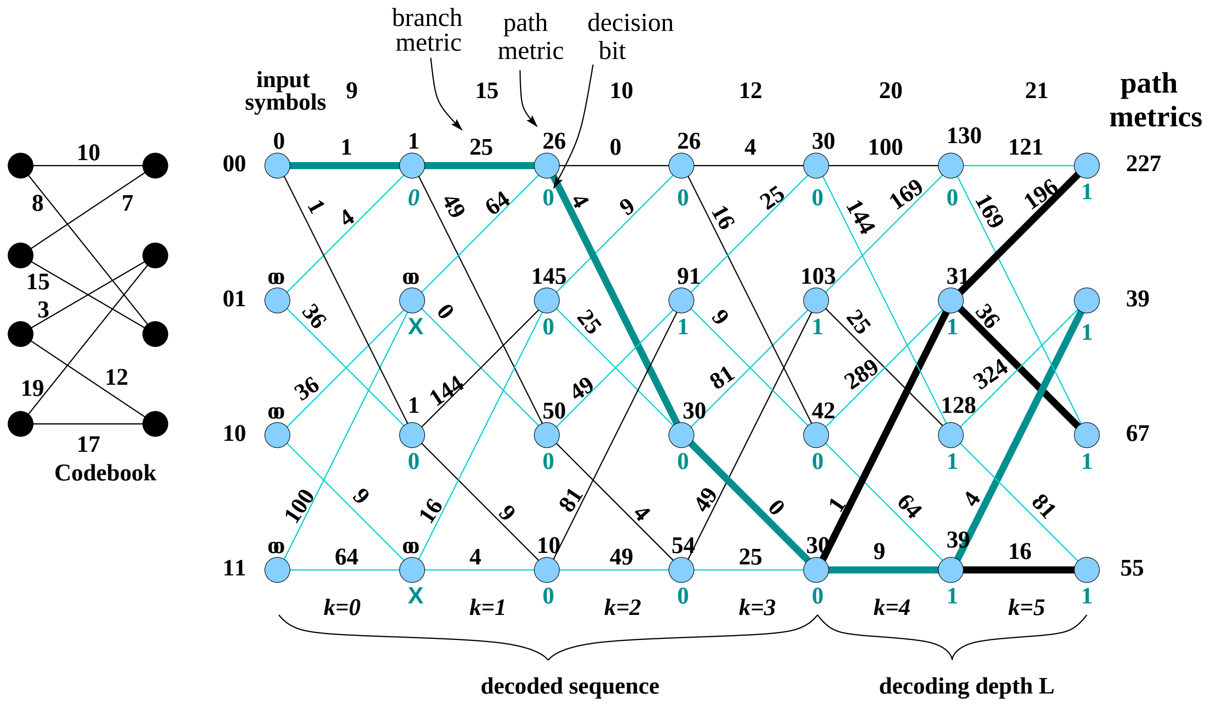
$$\begin{aligned} PM_0 &= 0 \\ PM_i &= \infty \quad i = 1 \cdots 2^{K-1} \end{aligned}$$

so that the surviving paths are forced to come from a unique state. Each branch has its associated branch metric and each trellis state has its own path metric and decision bit as shown in the figure. During the forward direction of the algorithm, that is, during the **path metric updating**, at each time instant every path metric is computed and the decision vectors are generated. The thin grey lines in figure 4.2 indicate the trellis paths that have been discarded during **path metric updating**.

When the entire source sequence has been processed, the best surviving path is traced back through the trellis, as indicated by the bold grey line in the figure, so that the binary sequence associated to this path can be found. The binary sequence can be decoded by taking the MSB of each state visited by the surviving path.

A very important feature of trellis search algorithms is the *decoding depth* of the trellis. This feature states that all the surviving paths associated to each state are likely to merge into a single path after a certain number L of trellis states have been visited during the **survivor memory management** operation. In the figure, the decoding depth is $L = 2$ and is indicated by the bold lines steming from each state at $k = 5$ in backward direction. In other words, each time the *survivor memory management* operation has processed L decision vectors, it is very likely that the surviving paths at time $k = k - L$ come from the same state, and for $k < k - L$, the segment of the 2^{K-1} surviving paths are identical. As a result, the **survivor memory management** operation can start from any trellis state and be sure that the decoded bit at $k = k - L$ is part of the final decoded sequence. Moreover, this feature allows the survivor decoding operation to be performed before the complete **path metric updating** operation is achieved, and hence, the latency of the algorithm can be reduced. Typically, the length of the decoding depth is [85]

$$L \approx 6 \cdot K \tag{4.2}$$



Decision vectors						
	0	0	0	0	0	1
	X	0	1	1	1	1
	0	0	0	0	1	1
	X	0	0	0	1	1

							Departure state
visited states	00	00	00	10	11	11	01
Decoded sequence	0	0	1	1	1	0	

Figure 4.2: Operation example of the Viterbi algorithm.

In order to design hardware architectures for the Viterbi algorithm, the usual approach is to divide its implementation into three main blocks, the **branch metric** unit (BMU), the **Add-Compare-Select** unit (ACSU) and the **survivor memory unit** (SMU), as indicated in figure 4.3. As the name suggests, the ACS unit performs the ACS recursion (equation (4.1)) and the SMU unit performs the surivivor memory management operation. Since the branch metric unit was already explained in the previous Chapter, only the ACS unit and SMU unit will be discussed in the reminder of this section.

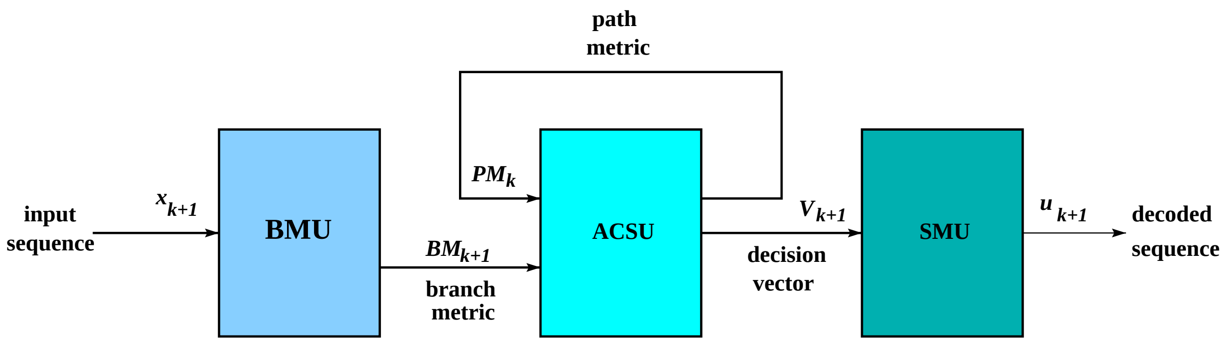


Figure 4.3: Block diagram of the Viterbi decoder.

4.1.2 The Add-Compare-Select Unit

The basic processing element of the Add-Compare-Select unit is presented in figure 4.4. This simple structure performs exactly the operation of equation 4.1. In spite of the simplicity of the basic processor, high-speed implementations of the Viterbi algorithm are difficult because the ACS operation contains a feedback loop which is non-linear and inherently serial in nature due to the data dependency of the recursion [84]. Since path metrics of time k depend on previous path metrics, a limit on the processing rate of the Viterbi algorithm is imposed since the use of parallellism and pipelining is of little help. Consequently, the ACS recursion is the critical path and bottleneck of the Viterbi algorithm.

The logical solution to this data dependency problem is the use of full parallel architectures where a single processing element is reserved to the path metric updating of a single state. Nevertheless, the problem that arises with this new approach is the arrangement and layout design of the ACS unit since the data dependency affects the required interprocessor communication. As a result, a trade off throughput-complexity has to be done.

A lot of work has been dedicated to the optimization of the interprocessor communication; however, the resulting hardware architectures are so complex that they are limited to constraint lengths in the order of $K = 7$. Different approaches have been proposed to design efficient architectures for the ACS unit. For a detailed treatement on this subject, the reader is referred to [19, 39, 40, 53, 75, 84, 90, 100, 109].

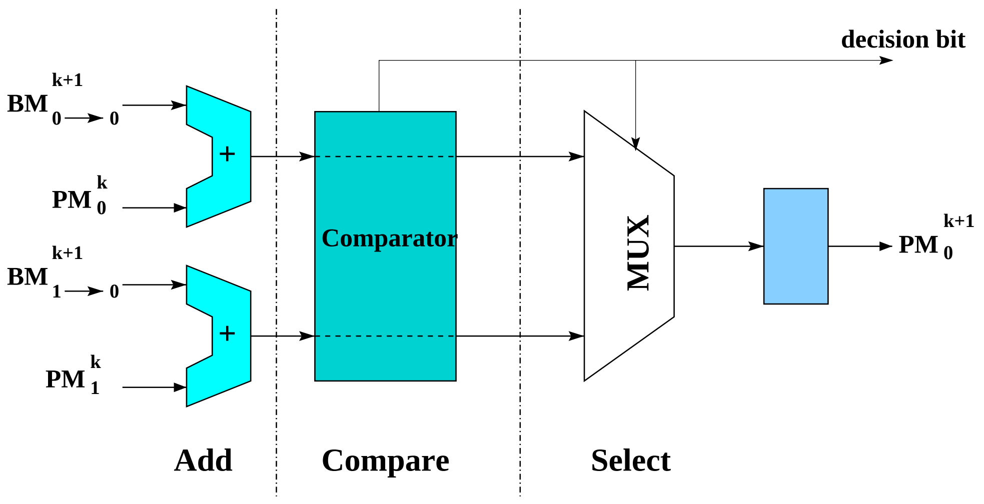


Figure 4.4: Basic processing element of the ACS operation.

4.1.3 Survivor Memory Management

The two most important methods to perform the survivor memory management in Viterbi decoders are:

- **The Register Exchange algorithm** and
- **The Trace-Back algorithm**

Each approach has its advantages and disadvantages; thus, before deciding which approach will be used, a careful study must be done so as to know which parameters are of primary importance: high throughput, latency, storage density, storage requirements, circuit area and wiring, power consumption, etc.

4.1.3.1 Register Exchange Algorithm

The Register Exchange algorithm (RE) is the most straightforward method to manage the decision vectors produced during the path metric updating operation [29]. This method consists in storing in a bank of registers the L most recent decision bits of the surviving path corresponding to a given state. The bank of registers is interconnected in tha same manner as the structure of the trellis diagram, as shown in figure 4.5 for a 4-state trellis. Each row in the bank of registers contains the surviving path of its corresponding state, that is to say, the first row is assigned to state 0, the second one to state 1 and so on. For each new decision vector generated by the ACS unit, the register contents are interchanged according to the decision bit controlling the multiplexers of each row in the register array; then, the new decision vector is inserted at the rightmost column of the register bank and the oldest bit coming out of the array is the decoded bit. Since the decoding depth of the trellis is respected, the surviving paths merge with high probability and hence the oldest decision bit of any state can be chosen as the decoded bit.

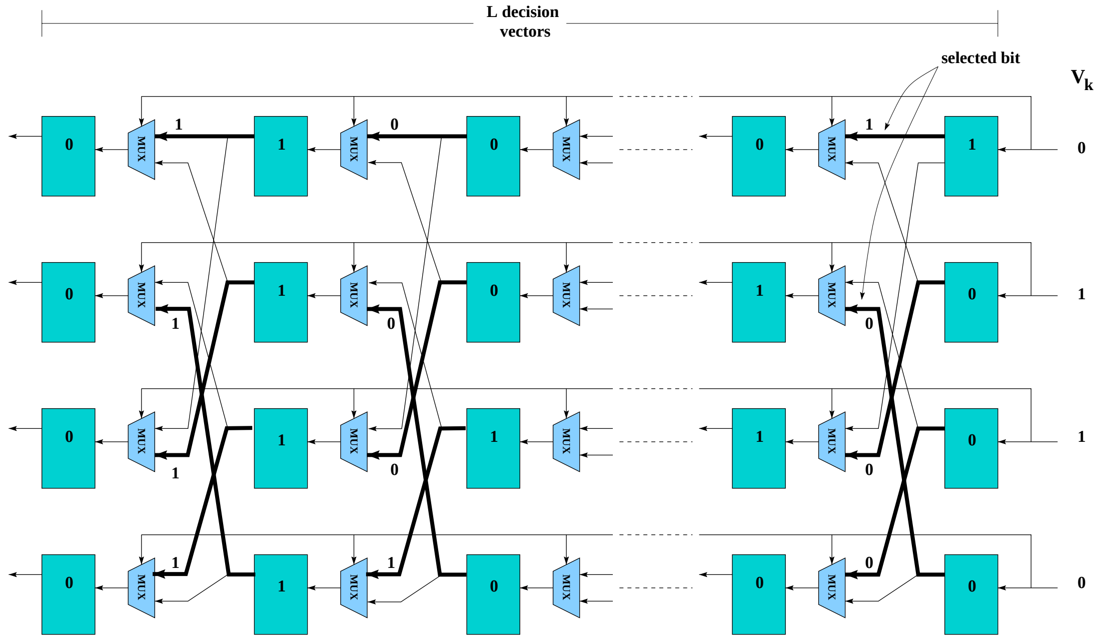


Figure 4.5: Register Exchange architecture.

The main advantage of this architecture is its small latency. After L cycles of the survivor memory management operation, the architecture is able to deliver the decoded sequence at a throughput rate that is matched to the rate at which the ACS unit delivers new decision vectors. Another advantage of this architecture is the regularity and simplicity of the processing elements (a register and a multiplexer). However, it is clear that when the number of trellis states is large, this architecture becomes impractical due to the poor density of storage and the large area required to interconnect the processing elements. In addition, since the registers in the array are updated at every cycle, the power dissipation of this architecture is quite large. Consequently, register exchange architectures are attractive only when the size of the trellis is small ($K \sim 4$).

4.1.3.2 Trace-Back Algorithm

In the Trace-Back algorithm (TB), the decision bits generated by the ACS unit at each processing interval act as pointers to the visited states of the surviving paths [90]. The principle of operation is very simple and is the "natural" way of decoding the output sequence from the decision vectors. The idea underlying the trace-back operation consists in the utilization of the decision bits to find the ancestor states of the best surviving path. More precisely, the trellis state visited by the surviving path at time k, together with its decision bit, serve to find the ancestor state visited by the surviving path at time $k-1$.

This way, all the trellis states visited by the surviving path can be *traced back* in time and thus, the decoded sequence can be found.

Figure 4.6 presents the hardware structure that implements the trace-back algorithm for the example of figure 4.2. During the **path metric updating** operation, the decision vectors generated by the ACS unit are stored in the *trace back* memory at the address given by time index k. Once the trace back memory has been filled in (in our example, at $k = 5$), the state having the best path metric (01 state) is stored in the shift register and the trace back memory is read in backward direction, starting from the last decision vector (at $k = 5$). Then, the multiplexer selects the decision bit corresponding to the contents of the shift register (1), the shift register is shifted one position to the left and the decision bit is appended at the rightmost position of the register. This way, the visited state at time $k = 4$ is found (11 state). At the next processing cycle, the decision vector of time $k = 4$ is retrieved from memory, the multiplexer selects the decision bit according to the shift register value and this decision bit is inserted into the shift register such that the visited state of time $k = 3$ is found (11 state). This procedure is continued until the entire trace back memory is read. During the trace back operation, the output bit of the shift register (the MSB of the visited states) is stored into a LIFO so that the decoded sequence can be arranged in reverse order as they were generated by the ACS unit. The grey boxes of the trace-back memory in the figure indicate the decision bits selected during the trace-back operation.

The main advantages of the TB algorithm are the simplicity of operation, low power consumption and the large storage density since we can use RAMs to store the decision vectors. As a consequence, this algorithm is specially attractive for trellises with large number of states.

On the other hand, the main drawback of this approach is the decoding delay since it has to wait for all the received sequence to be processed by the ACS unit before starting the trace-back operation. Nevertheless, thanks to the decoding depth property of the VA, we can use a trace back memory of length L to perform the trace back procedure. This way, the storage requirements and the decoding delay may be reduced. However, the problem that arises with such a hardware implementation is that only one bit is decoded each time a trace back operation of length L is accomplished. In addition, we have to take into account that during the trace back procedure more decision vectors are created by the ACS unit, which need to be stored. Clearly, the architecture has problems of storage management, read/write conflicts and inter-symbol decoding because there is a gap of L cycles between successive decoded bits.

As in the case of the hardware implementation of the ACS unit, a lot of hardware structures based on the TB algorithm have been proposed. These architectures attempt to solve the problems described above. In appendix A, the most important methods to improve the hardware implentation of the trace-back algorithm are highlighted. The unfamiliarized reader is encouraged to take a look at these architectures since in Chapter 5 we will present some architectures that are based on the same approaches.

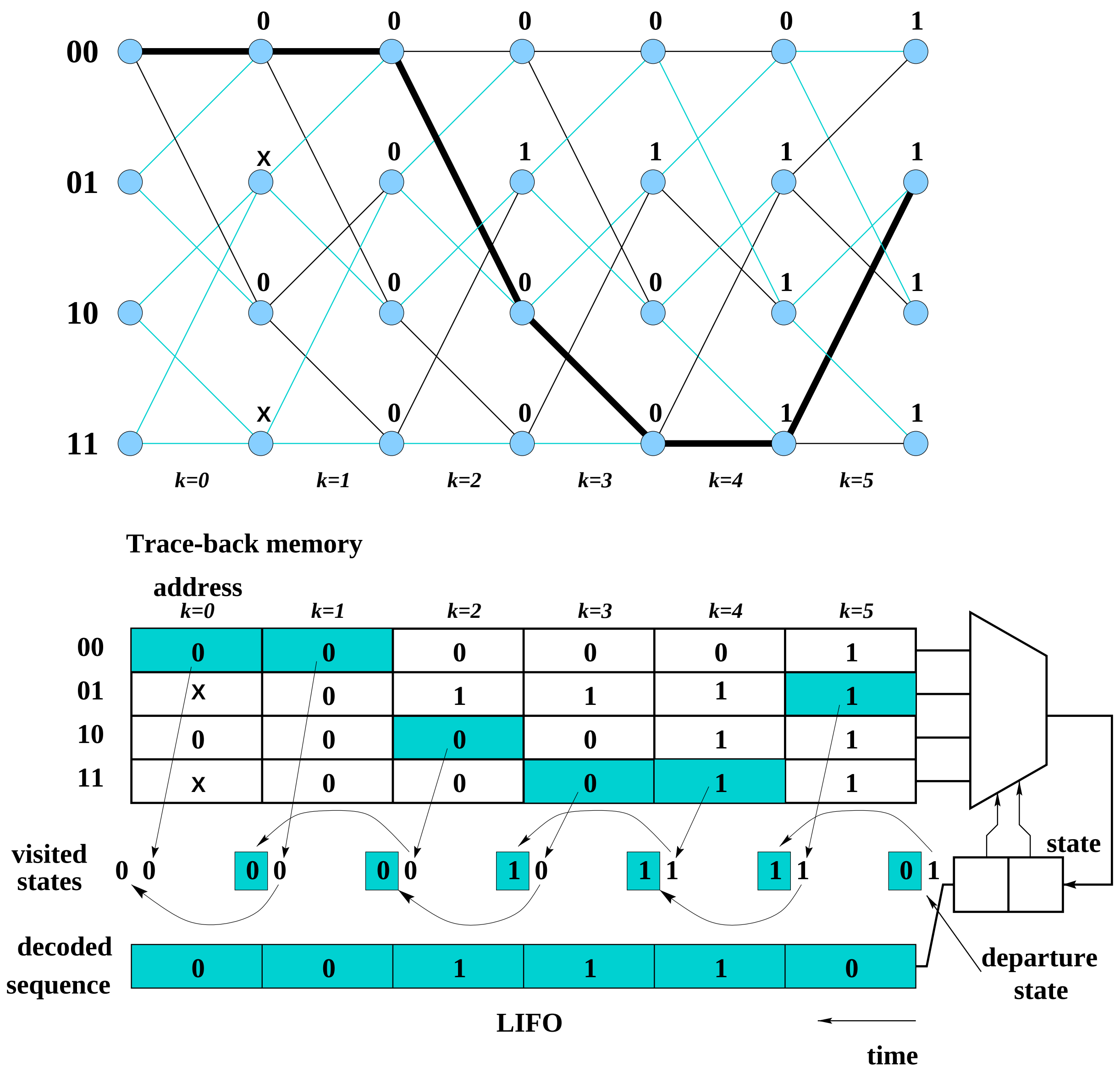


Figure 4.6: Operation Example of the Trace-Back Algorithm.

4.2 Quest for a Suboptimum Trellis Search Algorithm

In Chapter 2 it was shown that the use of the Viterbi algorithm in JSCTC allowed us to attain an overall performance that is superior to that of a tandem system composed of a trellis quantizer and a convolutional code. Nevertheless, notice that this superiority was achieved for noisy channel conditions and for large values of the trellis constraint length. In general, the larger the constraint length of the trellis, the better the overall performance. This is due to the fact that when the constraint length is large, the reproduction codebook is richer, with the result that the reproduction codewords are better adapted to the statistical distribution of the source.

In the previous section, the hardware implementation of the Viterbi algorithm was discussed. Clearly, the complexity of this algorithm grows exponentially with the constraint length K. For instance, at each processing interval, the VA must compute

- 2^K branch metrics
- 2^{K-1} ACS operations which consist in 2^K additions and 2^{K-1} comparisons

In addition, the minimum storage requirements are $L \times 2^{K-1}$ decision bits. As a result, the primary difficulty with Viterbi JSCTC is that the proximity to the OPTA curve is not achievable in practice because only small constraint length trellis quantizers can be used. In addition, it is well known that in source coding applications, eventhough many states are required, only a few paths need to be pursued [2, 4, 8, 51, 79]; that is, it is possible to obtain source codes with an excellent performance even if the search through the trellis is not the optimum one [80, 81, 87, 110]. Consequently, it is clear that the amount of computations of the Viterbi algorithm is not always needed and it may lead to low-performance JSCTC systems because of this complexity constraint.

On the other hand, in the context of JSCTC, finding the least distortion path at the encoder end is not very helpfull since that path will not be entirely reconstructed at the decoder end since the transmission system lacks of a channel correction scheme. In other words, the performance obtained by the system does not deserve the effort employed to find the optimum path. This means that the Viterbi algorithm still performs 2^K multiplications and additions per input symbol x_k, some of which are wasted effort. Consequently, it is desirable to have a search procedure whose encoding effort allows a certain increase in the number of trellis states so that the overall performance can be increased because the reproduction codebook is richer. This way, for low channel SNR, the abscence of a channel protection scheme is substituted by a richer reproduction codebook which, in addition to the joint source-channel codebook optimization, permits the decoder to resist the distortion introduced by the transmission channel.

Sequential coding has come to be an attractive alternative to the Viterbi algorithm [7]. The decoding effort of sequential coding is essentially independent of the constraint length and thus larger trellises can be used. This reduction in hardware complexity can lead to the design of more efficient JSCTC systems.

It is convenient to define the features that are general to all the search algorithms. As indicated earlier, the aim of every search algorithm is to find a path with the best metric. In a general way, the search process begins at a root state and continues until some (or all) path(s) reach(es) the decoding depth L. At this point, the algorithm selects the best path and releases the oldest branch of this path as output. The search process resumes until some path again reaches L branches; then, the best path is selected and its oldest branch is released as output. The procedure continues indefinitely until all the input sequence have been coded. As in the case of the VA, a *path metric* is associated to every path in the trellis.

All the algorithms peform the following steps to find the minimum metric path:

1. **Path Extension**: The algorithm extends one branch at each time instant k in a process which includes:
   (a) computing or retrieving the codeword symbols associated to the branch,
   (b) fetching the input symbol corresponding to the time instant k (source symbols to be encoded or channel symbols to be decoded)
   (c) calculating the metric increment or *branch metric*
   (d) updating the new *path metric* for that path together with its length and terminal state.
2. **Ambiguity Check**: The algorithm checks if two paths diverge and then remerge to a same state. In this case, the path having the best path metric is retained because the other path will never become the best path. As a result, it is useless to keep retaining this path. This will be explained in more detail in the next chapter.
3. **Path Deletion**: The algorithm deletes an entire path. The deletion occurs whenever the numbre of paths stored exceeds a given value or when a path fails an ambiguity check.
4. **Release Output Symbol**: The algorithm releases as output the oldest symbol of the best path.

In the case of the Viterbi algorithm, the first three steps are performed by the ACS and branch metric units described above. Specifically, the first three items of the path extension are done at the branch metric unit whereas the last step as well as the ambiguity check and path deletion are inherently performed in the ACS unit. The fact that the ACS unit selects one path arriving to each state out of two, makes the path deletion to be performed automatically. In addition, thanks to the decoding depth L, the ambiguity check is saved, it does not need to be performed by the Viterbi decoder. Finally, the release output symbol step is performed by the survivor memory unit. Notice that it is possible to release a block of output symbols at once if the decoding depth is respected.

4.3 Classification of the Trellis Search Algorithms

There are several ways of classifying trellis search algorithms. One way of classifying the algorithms is by knowing whether they need sorting operations or not. Yet, a more accurate classification of the algorithms is by the way they perform the extension and search through the trellis. In this way, trellis search algorithms fall into three categories [7]:

1. **Metric-first**: In this class, the contender paths are ranked according to its metric and only the path having the best metric is extended and explored.
2. **Depth-first**: These algorithms search along a single promising path until its metric falls below a discard criterion. In that case, the trellis is backtracked to explore alternative branches. Like in the metric-first class, in depth-first algorithms only one paths is explored at a time.
3. **Breadth-first**: In this case, all contending paths are extended during a processing interval and pruned according to a discard criterion based on the metric [8].

There are several trellis search algorithms existing in the literature. The reader is referred to [6, 7, 26, 54, 96] for a deeper insight into those algorithms. Only one example of each search class will be given here.

4.3.1 Metric-first example: The Stack Algorithm

The Stack algorithm is a member of the metric-first trellis search algorithms [3, 59]. In this algorithm, an ordered list or stack of previously examined paths of different lengths is kept in storage. Each stack entry contains a path together with its metric; the path with the smallest (or largest) metric is placed at the top of the list and the others are placed in order of increasing (or decreasing) metric. At each decoding step, the top path in the stack is extended by computing the branch metrics of its 2 succeeding branches and adding these to the metric of the top path to form 2 new paths. The top path is deleted from the stack, its 2 succesors are inserted, and the stack is rearranged in order of increasing (decreasing) metric values. When the top path in the stack is at the end of the trellis, the algorithm terminates.

The steps of the stack algorithm are:

1. Load the stack with the origin state in the tree, whose metric is taken to be zero
2. Compute the metrics of the successors of the top path in the stack
3. Delete the top path from the stack
4. Insert the new paths in the stack, rearrange the stack in order of increasing (decreasing) metric values and retain only the best S paths
5. If the top path in the stack ends at a teminal state in the tree, stop. Otherwise, return to step 2

When the algorithm terminates, the top path in the stack is taken as the decoded sequence.

4.3.2 Depth-first example: The Fano Algorithm

When the stack algorithm is working on a particular partial path that it considers is not likely to be part of the correct path (i.e. when the metric of the partial path is inferior to the metric of one or other partial paths in the stack) the algorithm jumps to the partial path that appears to be the one most likely to grow into the correct path.

In the Fano algorithm [24, 35], when a particular path is decided to be abandoned, a different approach is taken. Rather than jump to a completely different path, the algorithm "backtracks" in the trellis by removing the most recently appended branch from the path. The algorithm then examines the alternative branches that could take the place of the one just removed.

The Fano algorithm stores only the path from the initial state S_0 to the current state S_k, so it cannot use comparisons with other paths to help decide what to do with the current working path. Rather than jumping from state to state looking for a better path, as in the stack algorithm, the Fano algorithm must decide if the current path is good enough to continue building the path. This decision is accomplished by comparing the new path metric to a threshold. If this metric is better than the threshold, the algorithm moves forward to the corresponding new branch to reach the new working state S_{k+1}. If the new path metric is not better than the threshold, the algorithm moves backwards to the state S_k leading to the present state S_{k+1}. This eliminates the need for storing the path metrics of previously examined states. Nevertheless, some states are visited more than once and in this case their path metrics must be recomputed.

If no path can be found whose path metric is better than the threshold, the threshold is lowered and the decoder moves forward again and tries to keep moving forward with this lower threshold. Each time a given state is revisited in the forward direction, the threshold is lower than on the previous visit to that same state. This prevents infinite looping in the algorithm and the decoder eventually reaches the end of the tree, at which point the algorithm terminates. We can see that this algorithm is very cumbersome for hardware implementations.

4.3.3 Breadth-first example: The M algorithm

As stated above, in the M algorithm [60], paths are extended at the same time and the number of retained paths is fixed: only the best M paths are retained at every processing interval. In the M algorithm the search is synchronous, that is, all paths have the same length. Notice that the Viterbi algorithm falls into this class of trellis search algorithms. At every time instant, the Viterbi algorithm extends 2^K new paths by the branch metric unit and the ACS unit performs the selection and deletion of 2^{K-1} paths; when the decoding depth is reached, the oldest branch of the best path is released as output.

In its specific operation, the M algorithm moves forward by extending the M paths it has retained to form $2M$ new paths. All the terminal branches are compared to the input data corresponding to this depth, the branch metrics are computed and the M poorest paths are deleted. The heart of the algorithm is a sorting algorithm which serves to delete these paths.

The steps of the M algorithm are:

1. Obtain departure states
2. Perform ambiguity check and release output symbol
3. Extend 2 paths from each retained path and save them in a list
4. Order the list to find the best M paths
5. Delete the remaining paths. Go to step 2

4.3.4 Algorithm Selection

In VLSI implementations of trellis coding techniques, algorithms such as Fano and stack are seldom used because the computational load is not largely reduced as compared to the Viterbi algoritm [7]. In addition, as we have seen, they present a random computing time which make very difficult their hardware implementation. As a result, these algorithms are rather interesting for software applications. To our knowledge, there has never been reported hardware implementations of the Fano algorithm. For the stack algorithm, however, recently, a VLSI architecture for fast stack decoders was proposed where the operation of sorting the stack is alleviated by using a systolic priority queue that delivers the best path in fixed periods of time [66].

Breadth-first algorithms, on the other hand, perform a constant number of computations per cycle. This regularity of operations and inherent parallelism renders the Viterbi and M algorithm very suitable for VLSI implementations and are therefore of greater interest here. It must be pointed out that in [101], another breadth-first trellis decoding algorithm, called the T algorithm, was introduced for application to sequence estimation. The decoding algorithm maintains a variable number of paths which depends on the channel noise encountered. This algorithm exhibits an error-rate versus average-computational-complexity behavior that is much superior to the Viterbi algorithm and which also improves the M algorithm. However, because of the adaptive computational load, a buffer for received samples is required and the computing time is also variable. Consequently, the algorithm that we have selected for JSCTC is the M algorithm.

4.4 JSCTC with the M Algorithm

The JSCTC scheme implemented with the M algorithm was used for the transmission of different sources through a BSC channel. In the following, simulation results are presented and compared to the same JSCTC scheme but performed with the Viterbi algorithm. The comparison is made in terms of overall performance and computational complexity.

4.4.1 Computational Load Comparison with the VA

The Computational Load (CL) of the VA and M algorithm (MA) is located mostly in the branch metric computation and in the path metric updating. This computational load can be expressed as

$$\begin{aligned} CL(VA) &= 2^K \text{ multiplications } + 2^K \text{ additions } \textbf{(branch metrics)} \\ &\quad +2^K - 1 \text{ ACS operations} \qquad (4.3) \\ CL(MA) &= 2M \text{ multiplications } + 2M \text{ additions } \textbf{(branch metrics)} \\ &\quad +2M \text{additions } \textbf{(path metric updating)} \\ &\quad +[(\log_2 M)^2 + \log_2 M]M \text{ sorting operations}^1 \qquad (4.4) \end{aligned}$$

If the ACS operation, multiplications and comparison elements are transformed into addition operations, we can have a more accurate estimation of the reduction in the computational load achieved by the M algorithm. Table 4.1 shows an estimation in terms

[1]This expression for the sorting operation was taken from [82] where a hardware implementation of the M algorithm is proposed. It denotes the number of comparison elements required by a Batcher's odd-even sorting network [11]. In the next Chapter we will show that this hardware complexity can be reduced.

of addition operations of the computational load per input symbol of the Viterbi and M algorithms. We have assumed that one multiplication, one ACS operation and one comparison element correspond to 8, 3 and 2 addition operations respectively.

Table 4.1: Estimation of the computational load per input symbol of the Viterbi and M algorithms.

Viterbi		M=8	M=16	M=32
K	additionss	additions	additions	additions
5	336	352	960	2560
6	672	352	960	2560
7	1344	352	960	2560
8	2688	352	960	2560
9	5378	352	960	2560
10	10752	352	960	2560

We can see that while the M algorithm is independent of the trellis constraint length, the Viterbi algorithm has an exponential dependency. Moreover, as we will see in the next section, this significant reduction of the computational load (see for instance, the last row of table 4.1) does not introduce considerable performance degradations.

A thorougher hardware complexity comparison, this time considering new ways of implementing the M-path selection and the survivor memory management will be made in the next chapter.

4.4.2 Peformance Comparison with the VA

Figure 4.7 and 4.8 show the performance of the suboptimum JSCTC scheme for a zero-mean, unit variance gaussian source and a first-order gauss markov source with autoregression coefficient 0.9, respectively. Both sources were transmitted through a BSC channel with the error probability p evenly distributed from $p = 0$ to $p = 0.5$.

In the case of the gaussian source, the JSCTC was performed with a $K = 5$ constraint length trellis (16 states) and only two surviving paths were retained at each trellis stage. We can see from the figure that there is practically no performance degradation between the optimum and suboptimum trellis searches, and however, the computational load was reduced by a factor of 7.

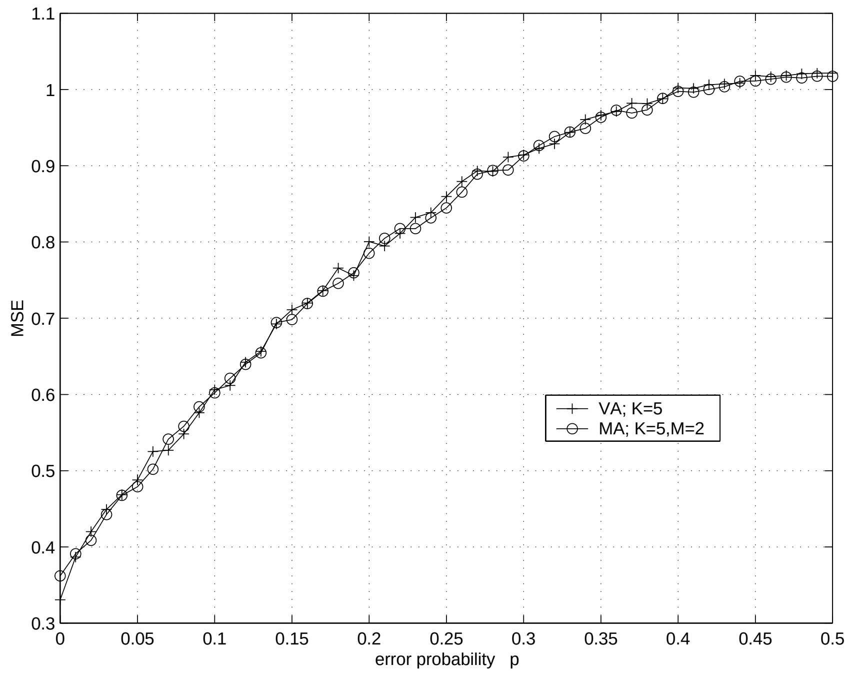


Figure 4.7: Suboptimum robust trellis quantizer performance on gaussian sources.

Regarding the first order gauss-markov source, the suboptimum JSCTC scheme was performed with a $K = 10$ constraint length coder and only 8 surviving paths were retained. Two important aspects can be pointed out from this figure. First, with the same computational load (VA, $K = 5$ curve), the M algorithm outperforms the Viterbi quantizer because the reproduction codebook is larger. Second, with a reduction of the computational load by a factor of 31, a performance degradation of 0.3 dB is introduced by the M algorithm. Hence, the M algorithm offers an excellent trade off performance-hardware complexity.

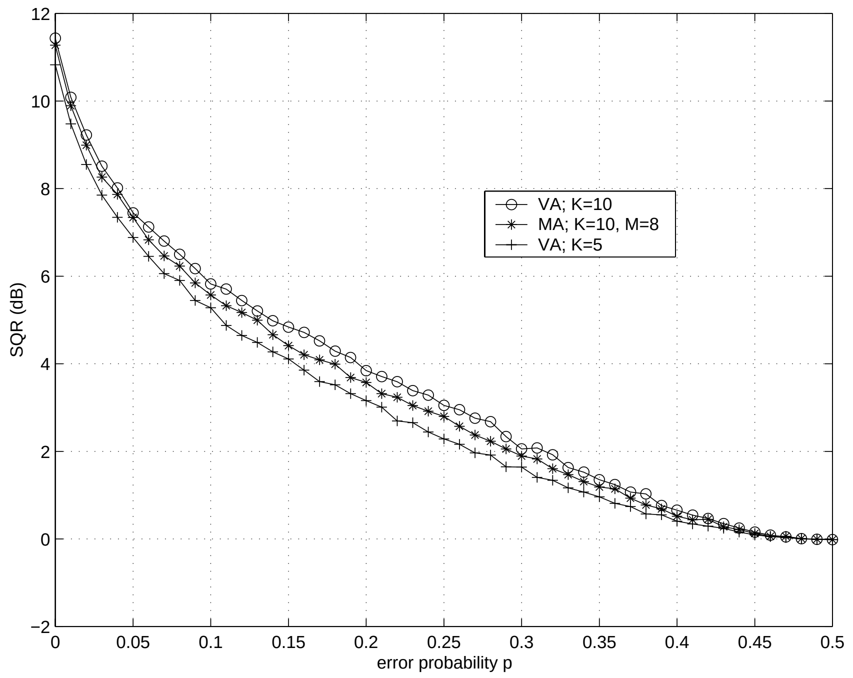


Figure 4.8: Suboptimum robust trellis quantizer performance on first-order gauss-markov sources.

In the following, the Viterbi and M algorithms are compared for the transmission of the Lenna image through an AWGN channel. Figure 4.9 presents the performance of JSCTC with the VA and MA, in terms of the SQR as a function of the channel SNR. We can see that for the same trellis constraint length $K = 10$, the performance degradation introduced by the M algorithm is not very important, specially for our region of interest, that is, when the transmission channel is noisy.

Table 4.2 shows the degradation in dB introduced by this suboptimum trellis search with respect to the Viterbi JSCTC. We can see from this table that if the computational load of the robust trellis quantizer is reduced in a factor of 11 (M=16), the maximum performance degradation at low channel SNR is of 0.2 dB whereas for large SNR the difference attains 0.33 dB. On the other hand, when the computational load is reduced 31 times (M=8), the maximum performance degradation is 0.4dB at low channel SNR and 0.7 dB for large SNR. Finally, by reducing the computational load 90 times (M=4), the maximum distortion obtained is 0.83 dB and for large SNR the degradation reaches 1.3 dB. Notice that even the suboptimum JSCTC scheme with M=4 surviving paths outperforms a Viterbi quantizer with 64 states; as indicated earlier, this is due to the fact that the reproduction codebook of a $K = 10$ trellis coder has a larger number of codewords which are better adapted to the statistical distribution of the source.

To better elucidate this last remark, figure 4.10 presents the performance of the optimum and suboptimum trellis searches when the trellis quantizers have the same computational load. Figure 4.10(a) shows the performance of a $K = 5$ Viterbi quantizer and a

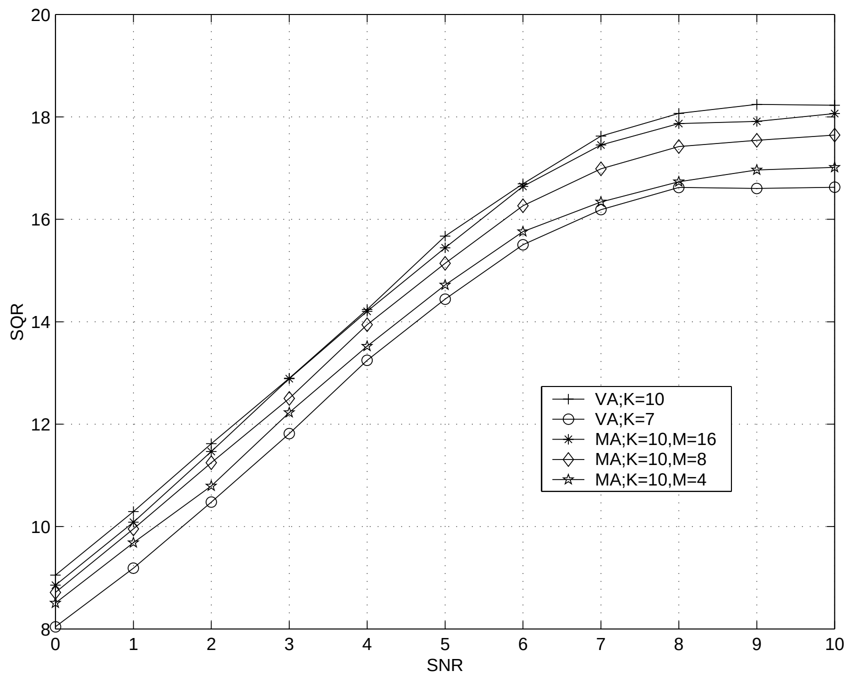
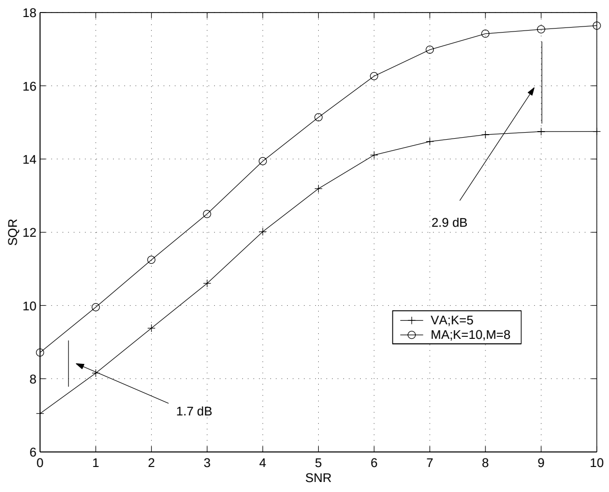


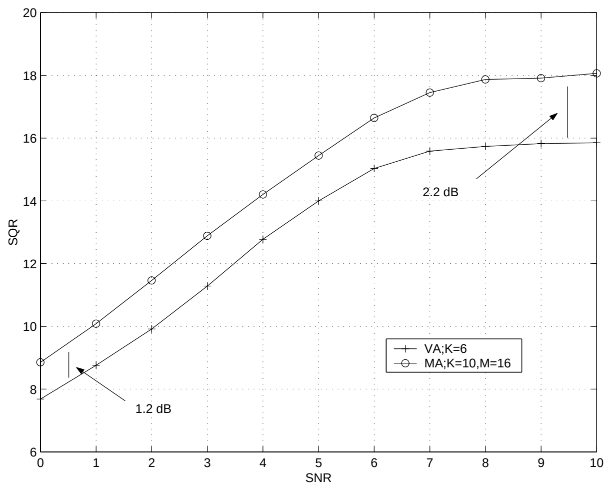
Figure 4.9: SQR vs SNR performance of the Viterbi and M algorithm for the transmission of the Lenna image.

Table 4.2: Performance degradation introduced by the suboptimum trellis search.

SNR	Viterbi	M=16		M=8		M=4	
	SQR (dB)	SQR (dB)	Δ SQR	SQR (dB)	Δ SQR	SQR (dB)	Δ SQR
0	9.05	8.85	0.2	8.71	0.34	8.5	0.55
1	10.29	10.08	0.21	9.95	0.34	9.68	0.61
2	11.61	11.46	0.15	11.24	0.37	10.79	0.83
3	12.90	12.89	0.01	12.50	0.40	12.23	0.67
4	14.24	14.20	0.04	13.94	0.30	13.52	0.72
5	15.67	15.44	0.23	15.13	0.50	14.72	0.95
6	16.69	16.64	0.05	16.26	0.43	15.76	0.93
7	17.63	17.45	0.18	16.99	0.64	16.33	1.28
8	18.06	17.86	0.20	17.42	0.64	16.73	1.33
9	18.24	17.91	0.33	17.54	0.70	16.97	1.27
10	18.23	18.06	0.17	17.65	0.58	17.02	1.21



(a) VA; K=5 and MA; K=10,M=8



(b) VA; K=6 and MA; K=10,M=16

Figure 4.10: SQR vs SNr performance of the Viterbi and M algorithm with the same hardware complexity.

$K = 10$, $M = 8$ M quantizer. Notice that for the same computational load, the performance obtained with the M quantizer is much better than that of the Viterbi quantizer (gains from 1.7 to 2.9 dB). On the other hand, with a $K = 6$ Viterbi quantizer and $K = 10$, $M = 16$ M quantizer, gains from 1.2 to 2.2 dB are obtained (see figure 4.10(b)).

It can be argued that channel mismatch can have a prejudicial influence on the suboptimum trellis search. Actually, this can be certainly the case since the reproduction codebook may not be well designed. Nonetheless, simulation results strongly suggest that channel mismatch between the codebook design and the quantization operation do not introduce large distortions, at least when the transmission channel is noisy, as shown in figure 4.11 for trellis quantizers whose codebooks were designed with channel estimations of 0, 3 and 10 dB. A source was quantized with these three codebooks and transmitted through the AWGN channel at different SNR values. We can see that when the transmission channel is noisy, the degradation in performance introduced by this channel mismatch is not important (0.2 dB) except, of course, for the extreme case where the codebook has been designed for a channel SNR=10 dB, meaning that the transmission is considered as noiseless and obviously, this is not the case!! As a result, we can conclude that the robustness of this suboptimum trellis quantizer is quite good.

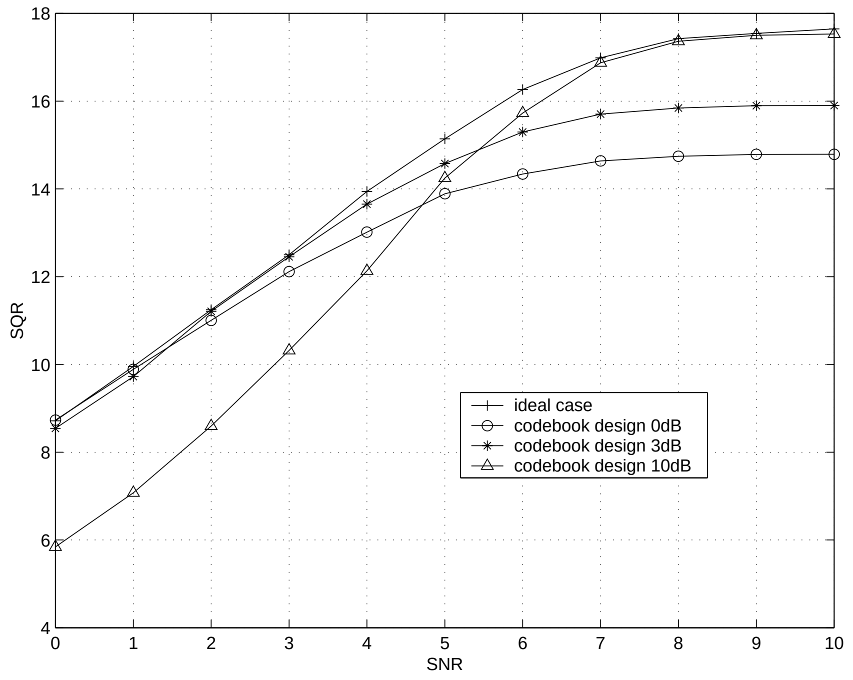
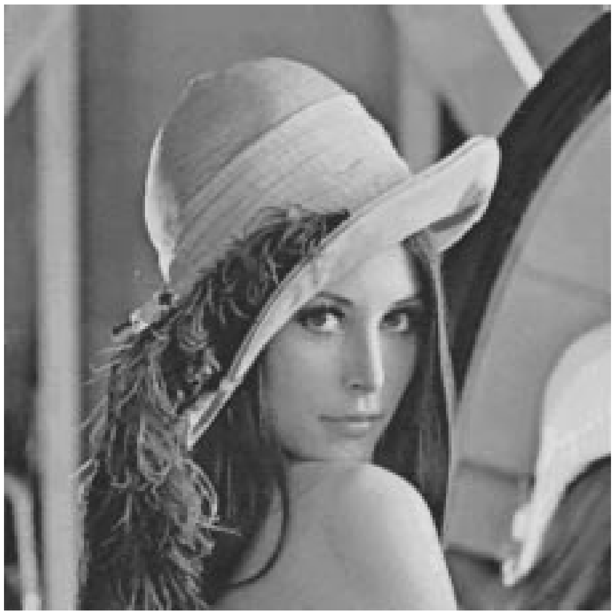


Figure 4.11: Effects of channel mismatch in suboptimum trellis quantization.

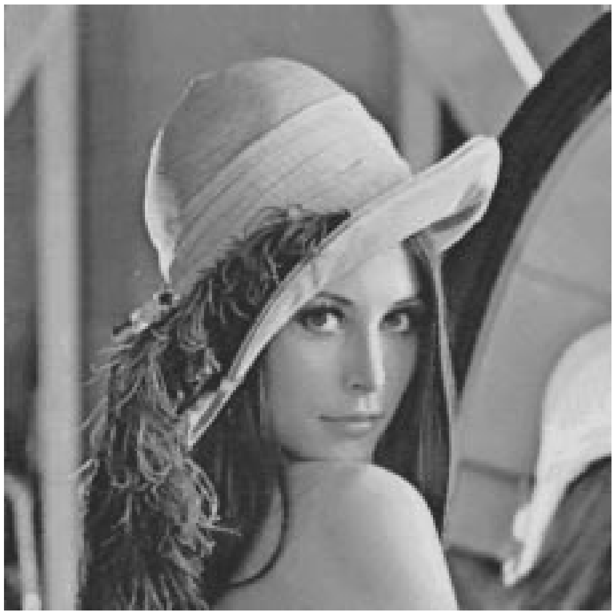
Finally, different Lenna images are shown from figure 4.12 to 4.15 which were quantized with the Viterbi and M algorithms. The images were transmitted through an AWGN channel at SNR =0 and 10dB. In the first two figures (4.12 and 4.13), a comparison is made between JSCTC coders of the same constriant length ($K = 10$) but with different hardware complexity. Notice that the image quality of the reconstructed images that were quantized with the M algorithm are very close to that of the Viterbi algorithm, eventhough the hardware complexity was largely reduced. Finally, the last two figures compare the Viterbi and M algorithm for quantizers with the same hardware complexity. It must be noted the significant improvement of the reconstructed images when coded with the M algorithm.

(a) original image

(b) VA; $K = 10$ constraint length

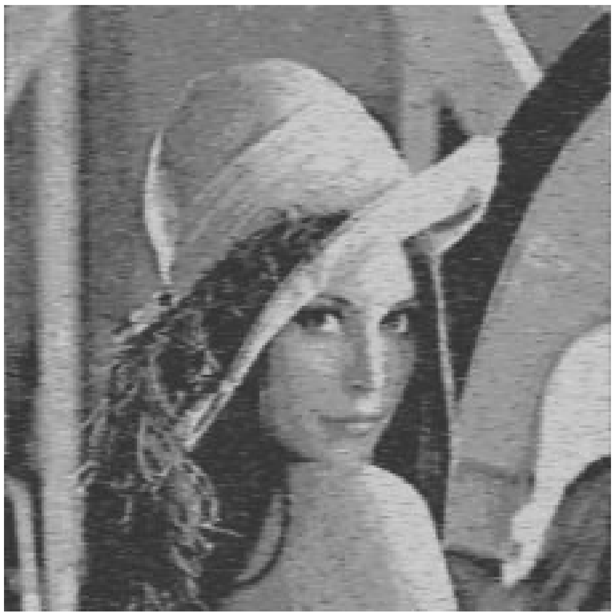


(c) MA; $K = 10$,$M = 16$

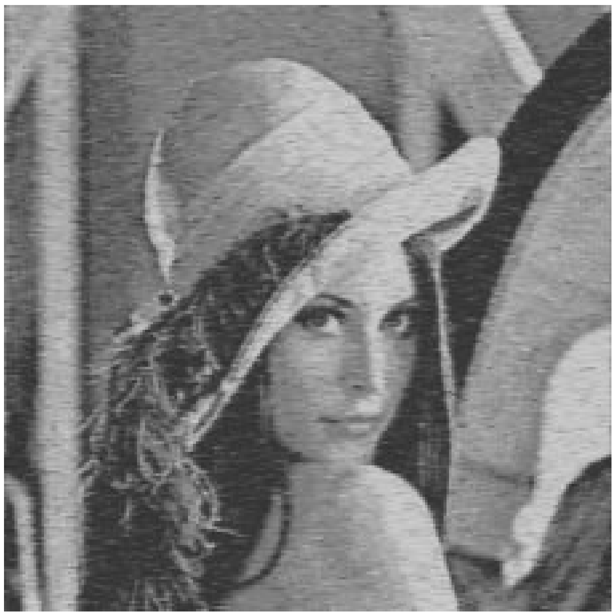


(d) MA; $K = 10$,$M = 8$

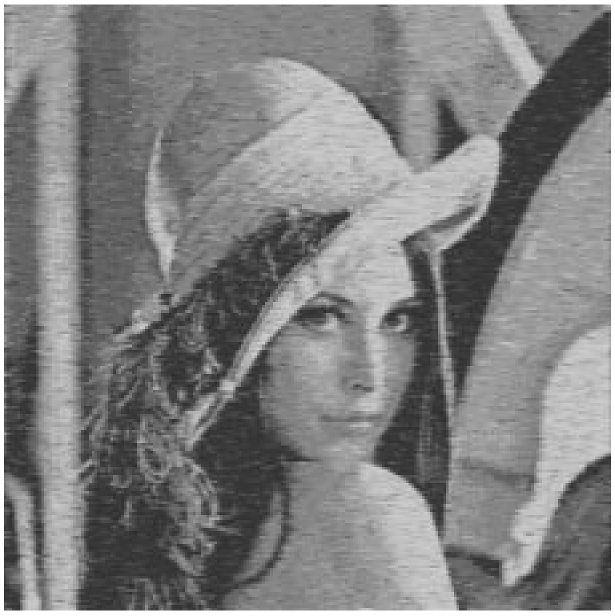
Figure 4.12: Overall performance with reduced computational load at SNR=10 dB



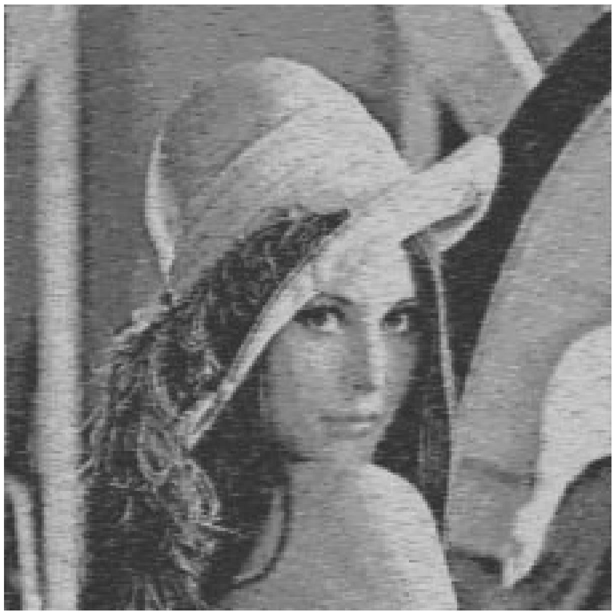
(a) VA; $K = 10$ constraint length



(b) MA; $K = 10$,$M = 16$

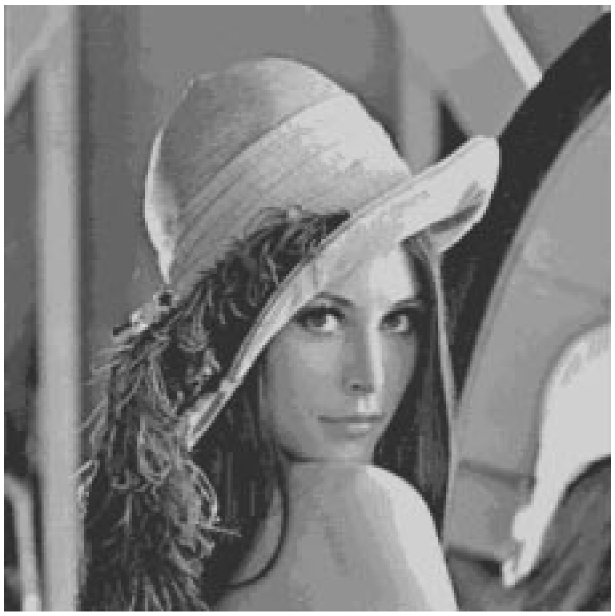


(c) MA; $K = 10$,$M = 8$

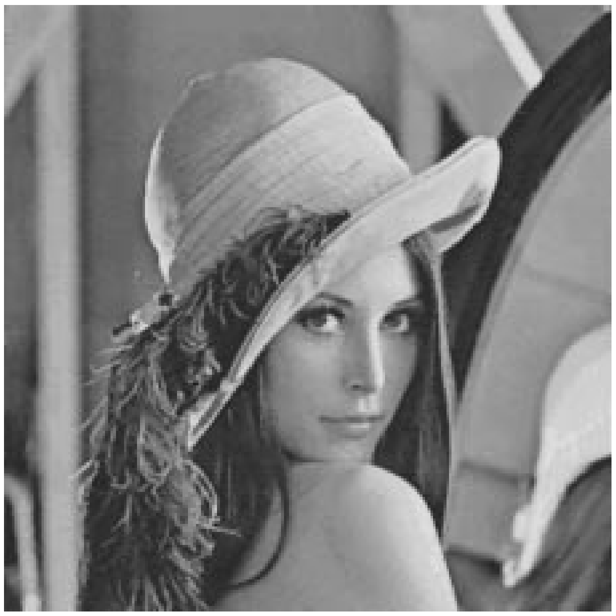


(d) MA; $K = 10$,$M = 4$

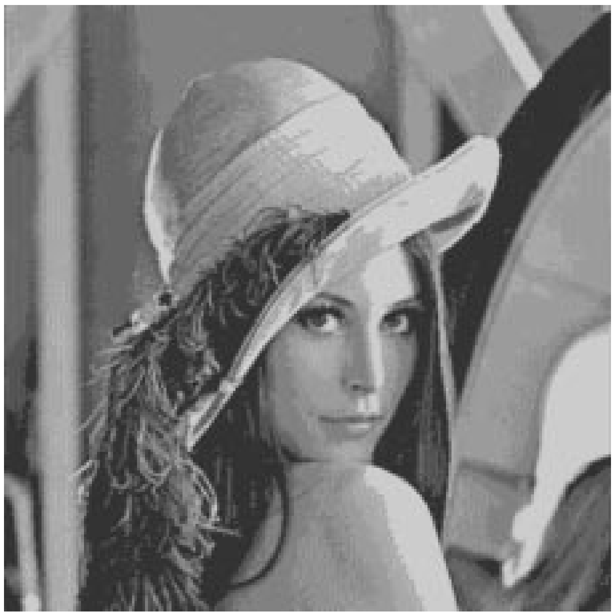
Figure 4.13: Overall performance with reduced computational load at SNR=0 dB



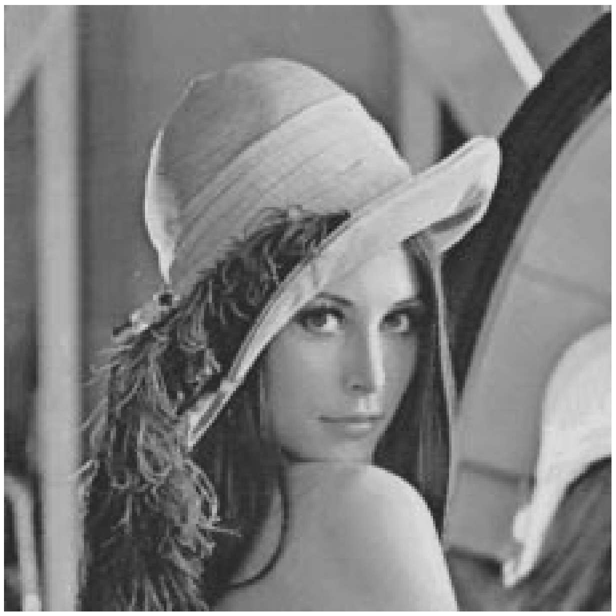
(a) VA; $K = 6$ constraint length



(b) MA; $K = 10$,$M = 16$

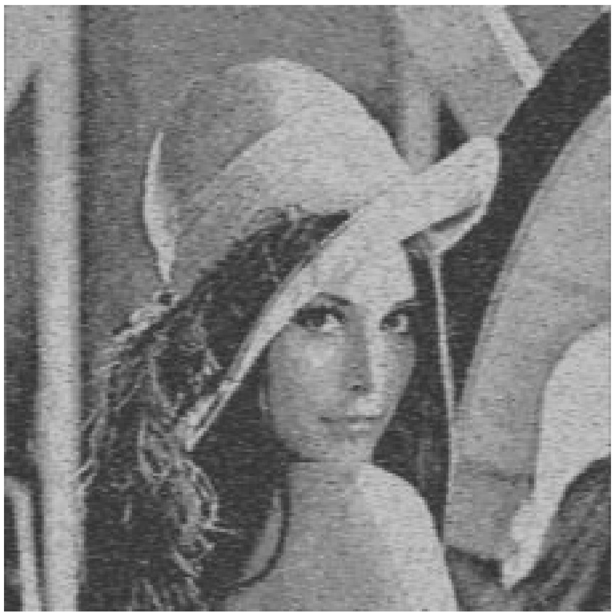


(c) VA; $K = 5$ constraint length

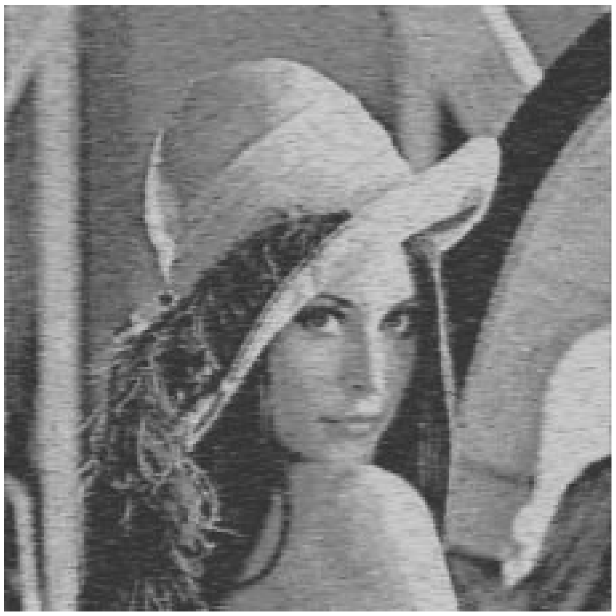


(d) MA; $K = 10$,$M = 8$

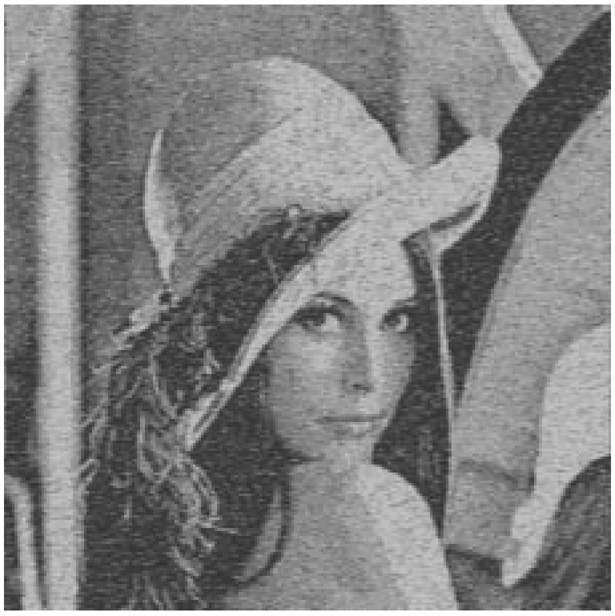
Figure 4.14: JSCTC overall performance with similar computational load at SNR=10 dB



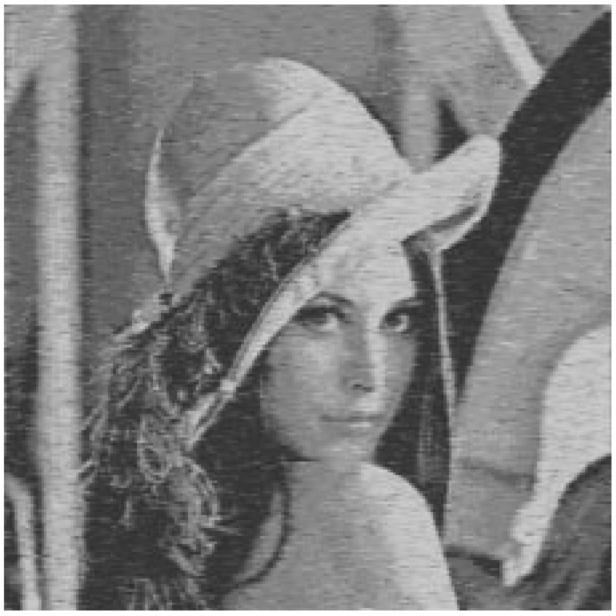
(a) VA; $K = 6$ constraint length



(b) MA; $K = 10$,$M = 16$



(c) VA; $K = 5$ constraint length



(d) MA; $K = 10$,$M = 8$

Figure 4.15: JSCTC overall performance with similar computational load at SNR=0 dB

4.5 Conclusions

We have presented a study of the performance of the JSCTC technique when a suboptimum trellis search algorithm is utilized. It was shown that the use of this suboptimum search introduces very little distortion as compared to the optimum search, while the computational complexity was largely reduced. This is due to the fact that the small computational requirements of the M algorithm allows the use or larger trellises. As a result, the reproduction codebook is richer and better adapted to the source statistics. This way, the channel optimized codebook, in addition to the use of larger trellises substitute the utilization of a channel protection scheme. Simulation results were presented were all these arguments were corroborated. The next step will consist in the hardware implementation of the M algorithm. This will be treated in the next Chapter.

Chapter 5

VLSI Architectures for the M Algorithm

In the last chapter, we showed how, in source coding applications, the M algorithm may yield more efficient schemes than the Viterbi algorithm. In addition, the M algorithm was also selected because of its regularity and inherent parallelism.

In addition to source coding, another application where the M algorithm has probed its efficiency is sequence estimation for intersymbol interference [116]. When the transmission channel is bandlimited (as it is almost always the case), the transmitted sequence is dispersed or spread, causing the signal pulses to overlap. As a result, the estimation of a given symbol does not depend only on the symbol itself but also on the last transmitted symbols. This phenomena can be seen as an unwanted coding where the channel acts as a convolutional coder that receives K transmitted symbols and generates a waveform that is a mixture of these symbols. Thus, a trellis search algorithm can be used to estimate the transmitted sequence. In cases where the channel bandwidth is very limited, symbol overlapping attains several trasmitted symbols; hence, an effective Viterbi algorithm can be very complex. For this reason, a suboptimum search like the M algorithm can lead to more efficient sequence estimations than the Viterbi algorithm [10, 68, 74, 92, 105].

In this Chapter, VLSI architectures are presented for the M algorithm. A thorough study of this algorithm is presented in order to elucidate its working operation and be able to propose efficient hardware architectures that take advantage of the algorithm's features. The outline of the chapter is the following. In section 5.1 the M algorithm is presented in a more detailed fashion. The different characteristics of each part of the algorithm are highlighted. Specifically, an important feature of the path extension stage is pointed out which allows a better treatment of the ambiguity check stage described in section 4.2 of the previous chapter. Then, section 5.2 presents a hardware study of the sorting operation, the main component for selecting the best M paths. This section comprises a state of the art in sorting architectures employed to implement the M algorithm and three new approaches to select the M best paths are highlighted. In section 5.3, a new method for the survivor memory management is proposed. This method is based on the Trace-Back technique employed in the survivor memory management of Viterbi decoders, but modified to account for the M algorithm's characeristics. Section 5.4 gives a more refined hardware comparison between the Viterbi and M algorithms. This comparison is made in terms of hardware ressources, storage requirements, power consumption and latency. Finally, a hardware implementation of the M algorithm, implemented at the COMELEC department is outlined in section 5.5.

It must be pointed out that to our knowledge, no hardware implementations of the M algorithm employing our ideas concerning the sorting operation and the trace back-based survivor memory management have ever been reported in the litterature.

5.1 The M Algorithm revisited

Section 4.3 on page 83 described the functioning of the M algorithm. A more detailed description of the algorithm is given for a better understanding of its operation. In this manner, we can define its four main steps as follows:

1. **Extension:** the M surviving paths at time k are extended to their two successors at time $k+1$. For these new $2M$ paths, their corresponding path metric and decision words are also computed.
2. **Rejection of merging paths:** It is likely that two distinct paths merge at time $k+1$. This means that for the remaining processing intervals these two paths will share the same branch metrics, paths metrics and decision words. If these merging paths are not eliminated, the effective number of explored paths in the trellis is reduced and the performance of the algorithm may decrease. Therefore, when two paths merge into a single state, the one with the largest metric must be discarded. This is equivalent to the ACS operation in the Viterbi algorithm.
3. **Selection of the best M paths:** the $2M$ paths are sorted and the M paths having the smallest path metrics are retained for the next extension step.
4. **Decoding of output sequence:** when the decoding depth L is reached, the oldest bit of the best metric path is released as ouput.

Step 2 and 3 imply the use of sorting circuits and, as we will see, the hardware complexity of sorting circuits can be very large. As a result, in practice the M algorithm is advantageous only when

$$M << 2^{K-1} \tag{5.1}$$

that is, in the cases where the number of surviving paths is much smaller than the number of trellis states.

An example of operation of the M algorithm is presented in figure 5.1 for an 8-state trellis and $M = 4$ surviving paths. The algorithm starts with an initilization process where a root node (state 000) is extended to generate its two successors (states 000 and 100). Then, the branch metrics (not shown in the figure) are computed and the path metrics are updated. This initialization continues until $M = 4$ surviving paths are generated (time $k = 1$). Upon the generation of the four surviving paths, at every time instant k, each survivor is extended to its two successors, the branch metrics per succesor are computed and the path metric updating is done as well as the selection of the 4 best paths. Each state in the figure has its associated path metrics at each time instant. The trellis branches in grey color denote the discarded paths and the branches in black color are the surviving paths. Notice that at time $k = 6$ there are two paths merging to state 000 and two paths merging to the state 100. In this case, the **rejection of merging paths** step is executed. The paths having the largest metrics are eliminated (path coming from state 000 and arriving to state 000 with a 38 path metric, and path coming from state 001 and arriving to state 100 with a 35 path metric). The algorithm continues in the usual way until all the input sequence has been coded. Then, like in Viterbi decoding, starting from the state which has the smallest metric, the surviving path corresponding to this state is traced back and the decoded sequence is found. The surviving path is shown by the bold line in the trellis diagram and its associated binary sequence is shown at the bottom of the figure.

In the same manner as in the Viterbi algorithm, the M algorithm can be divided into two main operations: **path metrics updating and selection**, and **survivor memory management** [47, 48]. The former operation consists in the computation of the branch metrics and the selection of the M best paths, and the latter consists in the decoding of the binary sequence $\mathbf{u}$ associated to the best path in the trellis. The advantage of this division is that we can focus on each part of the algorithm separately and propose different approaches for each component.

The block diagram of the M algorithm is shown in figure 5.2. In the *path extension and path metric updating* block, the new $2M$ paths are generated together with their associated states and decision bit. In the *sorting and selection block*, the M paths having the smallest distortions are selected and the merging paths are detected and discarded. Once the best M paths are selected, their decision bits feed the *survivor memory management* block for sequence decoding.

Figure 5.3 shows the internal structure of the path extender. Let $Z(k)$ be the set of state labels associated to the M survivors of time k and $S^i(k)$ be the state label associated to the i^{th} survivor of time k. We can write this in the form

$$Z(k) \quad = \quad \{S^i(k)\}_{0 \leq i < M-1} \tag{5.2}$$

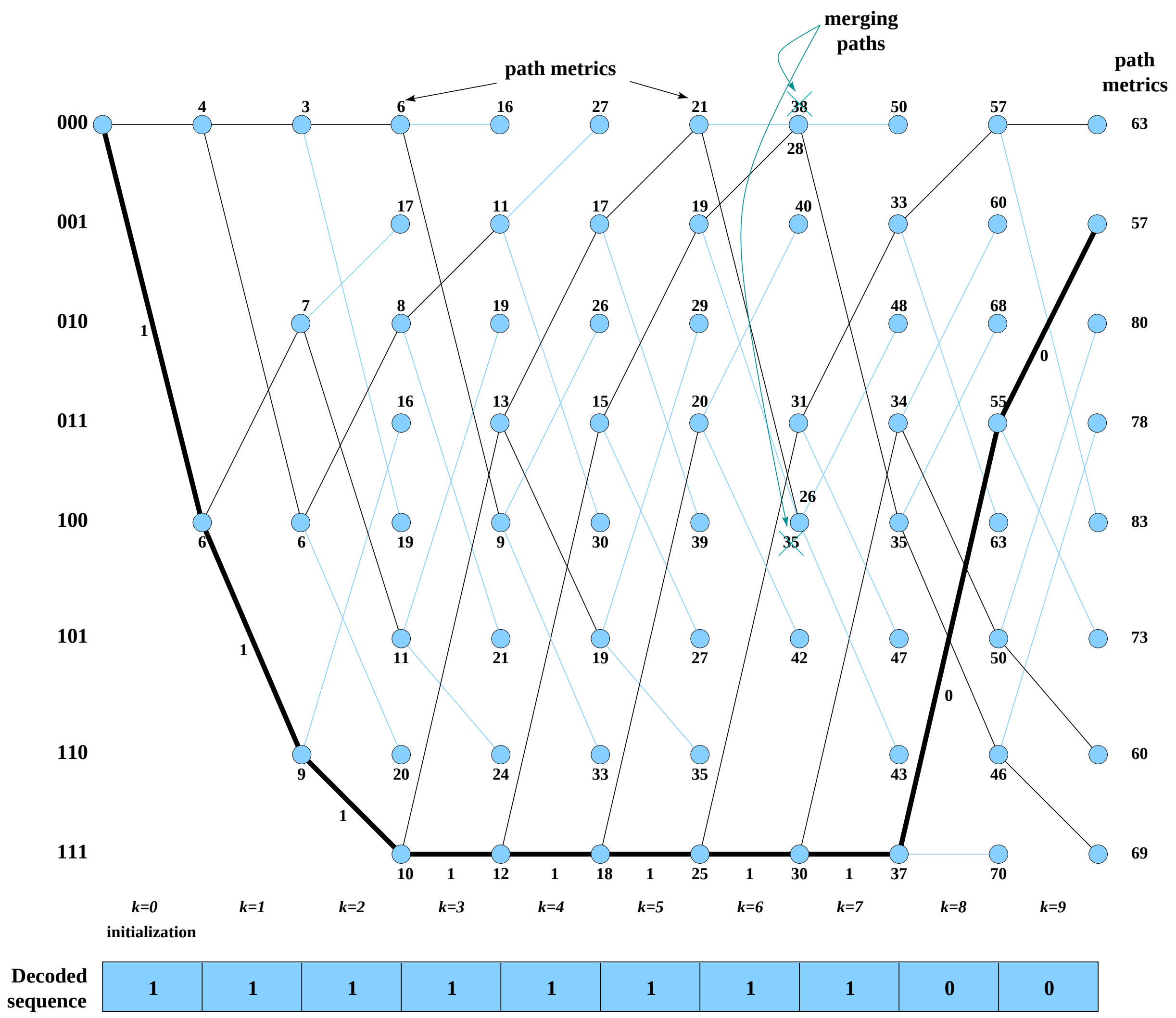


Figure 5.1: Operation example of the M algorithm.

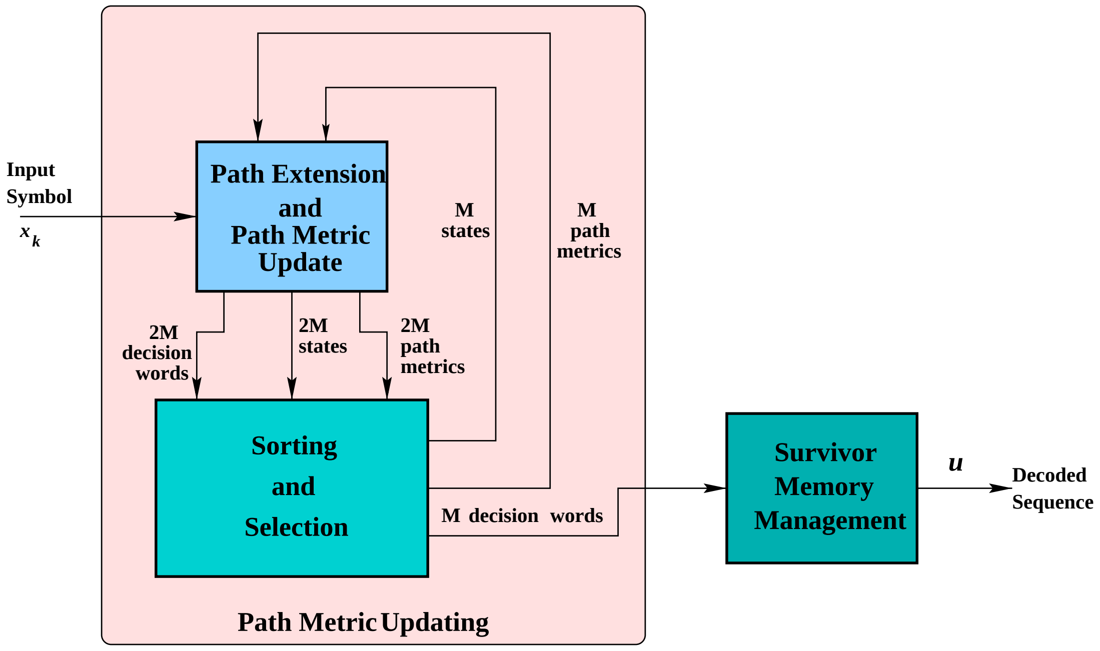


Figure 5.2: Block Diagram of the M Algorithm Architecture.

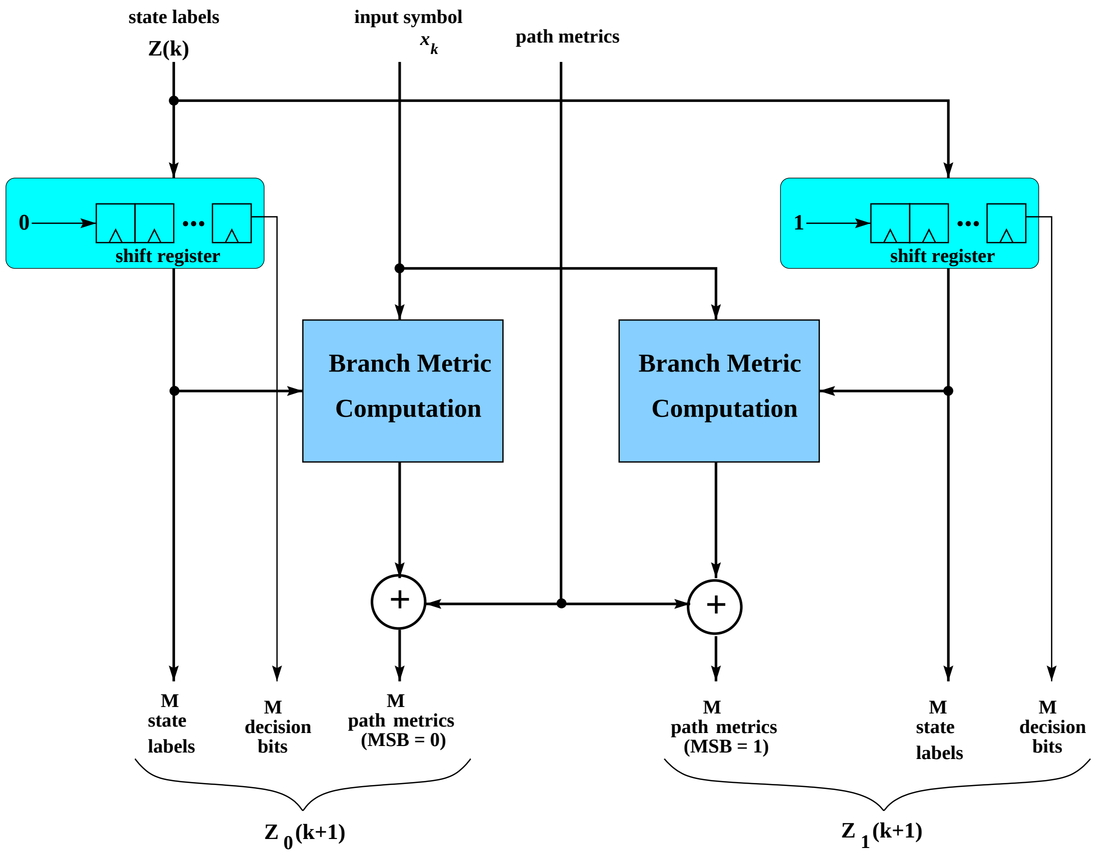


Figure 5.3: Architecture for Path Extension.

The state labels can be represented by the binary contents of the shift registers associated to each survivor

$$S^i(k) \quad = \quad (u^i_{k-1}, u^i_{k-2}, \cdots, u^i_{k-K+1})_2 \tag{5.3}$$

where the index 2 denotes binary representation.

During the extension operation, each state label $S^i(k)$ is extended to its two successors of time $k+1$. These successors are $S^i_0(k+1) = (0, u^i_{k-1}, \cdots, u^i_{k-K+2})_2$ for the arrival of a 0 and $S^i_1(k+1) = (1, u^i_{k-1}, \cdots, u^i_{k-K+2})_2$ for the arrival of a 1. The $2M$ extended paths can be clustered into two sets, $Z_0(k+1)$ and $Z_1(k+1)$, which contain the successors generated with the arrival of a 0 and a 1 respectively, that is,

$$Z_0(k+1) = \{S^i_0(k+1)\}_{0\leq i<M-1} \quad = \quad \{(0, u^i_{k-1}, \cdots, u^i_{k-K+2})_2\}_{0\leq i<M-1} \tag{5.4}$$
$$Z_1(k+1) = \{S^i_1(k+1)\}_{0\leq i<M-1} \quad = \quad \{(1, u^i_{k-1}, \cdots, u^i_{k-K+2})_2\}_{0\leq i<M-1} \tag{5.5}$$

We can notice that

$$Z_0(k) \bigcap Z_1(k) = \emptyset \tag{5.6}$$

since the MSB of the state labels is different in the two sets. This property will be defined as **Mutual independence of extended sets**.

On the other hand, if $S^i(k) > S^{i+1}(k)$ then

$$S^i_0(k+1) \geq S^{i+1}_0(k+1) \tag{5.7}$$

and

$$S^i_1(k+1) \geq S^{i+1}_1(k+1) \tag{5.8}$$

In words, if the state labels $Z(k)$ associated to the M survivors of time k are in sorted order, then the two sets, $Z_0(k+1)$ and $Z_1(k+1)$ will be in sorted order too. This property will be denoted as **State Label Sorting Conservation of Extended Paths**. We will see that these two properties allow a significant reduction in the hardware complexity of the sorting architecture.

From a hardware architecture point of view, figure 5.4 presents the different operations that the algorithm must perform at every processing interval. To describe these operations, consider the time instant from $t = 6$ to $t = 7$ of figure 5.1. Each survivor has its associated path metric and surviving path of lenght L. Notice that the three righmost bits of each surviving path denote its associated state (read from right to left). The algorithm begins by extending the four surviving paths to their two corresponding successors. This is accomplished by appending a 1 and a 0 bit to the states associated to each surviving path at the rightmost position (MSB). Then, the branch metrics of the new extended paths are retrieved or computed and added to the path metric of each new extended path. This way the path metrics of the eight new paths are updated. Next, the eight path metrics are sorted so as to detect the best four paths. Notice that in this block the ambiguity check or path merging deletion is done. Once all the paths are sorted in increasing order, the first four paths are retained for the next processing interval. Finally, when the decoding depth is attained, the oldest bit of the best surviving path is released as output (darkest bit at the leftmost position of the best path in the figure). This process is repeated until all the input sequence has been coded.

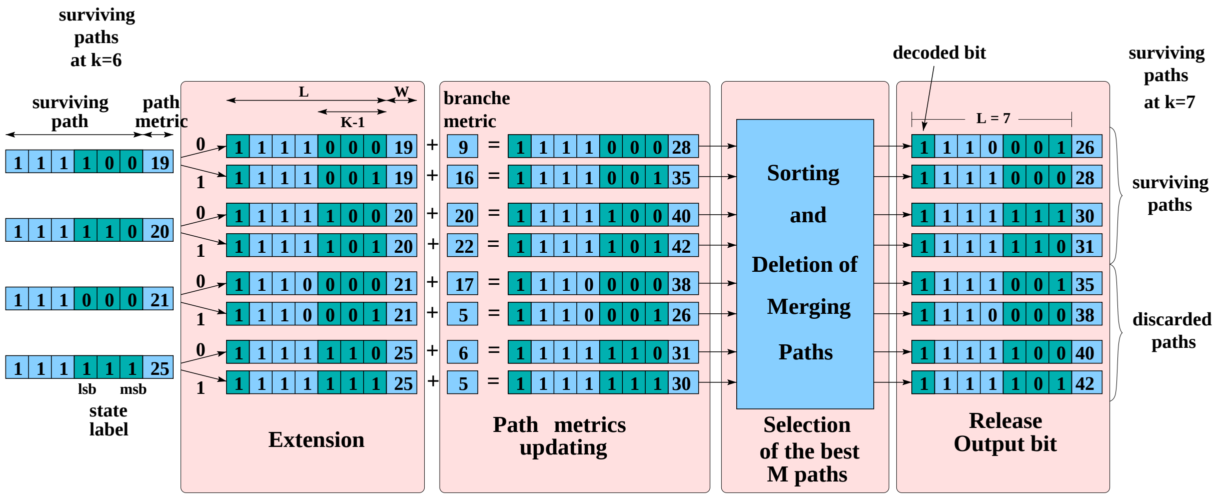


Figure 5.4: Hardware operations performed by the M algorithm at every trellis stage.

As we can see from this first discussion, the bulky part of the M algorithm resides in the selection of the best M paths and in the survivor memory management. The complexity of the path extension block is not important, except for the branch metric computation which depends on the application and which was explained in chapter 4. In the following, the main parts of the M algorithm are treated, namely, the sorting architecture and the survivor memory management unit.

5.2 The Sorting Architecture

The sorting operation is the heart of the path metric updating. Therefore, fast and regular sorting algorithms are required in order for the M algorihtm to be implemented in VLSI circuits. Sorting algorithms can be divided into two main classes: parallel sorting and serial sorting [20, 46]. In the former, all the values to be sorted are processed together in an interconnection network fashion; conversely, in the latter it is assumed that the values arrive serially to the sorting circuit in such a way that the new value to sort is inserted in an already ordered list. Obviously, the choice of the sorting algorithm depends on the target application. If, for example, a parallel implementation of the M algorithm is required, sorting networks result in the best solution. On the other hand, if the architecture were to be implemented in programmable architectures such as FPGAs or DSP processors, the logical solution is to employ serial sorting algorithms. In this section, only parallel algorithms are considered.

We based our study of the sorting architecture in the Batcher's odd-even and bitonic sorting networks [11]. The basic processing element of these networks is a comparison-exchange element, as shown in figure 5.5. It compares its two inputs and the largest input is released according to the arrow head (H output) whereas the smallest input is released by the upper output line (L output). At the end of this chapter, an FPGA implementation is presented which employs a serial sorting algorithm [25]. In appendix B, a description of the operation of the sorting networks employed in our work is presented. It is assumed that the reader is familiarized with Batcher's sorting networks. Otherwise, it is recommended to consult the appendix for a thorough explanation of these sorting structures.

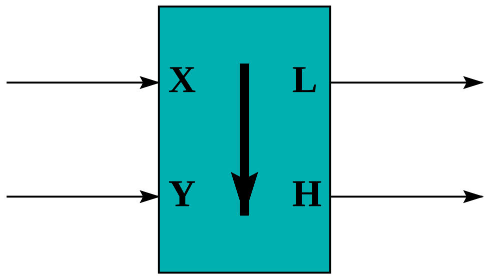


Figure 5.5: Comparison-exchange element for sorting networks.

Before starting the study of the sorting architectures, a description of the different ways the rejection of merging paths can be dealt with is presented.

5.2.1 Introduction of Path Merge Rejection within the Sorting Circuit

As we saw in section 4.2, ambiguity check is a very important step towards the design of high performance trellis search algorithms. In the M algorithm, the ambiguity check is crucial so that the full search capability of the algorithm can be profited. The most common ways of implementing the ambiguity check in the M algorithm are:

- **State comparison:** The states associated to the surviving paths are compared at each stage of the trellis and whenever two paths have the same state (we say that a *path merging* is produced), the one having the largest metric is discarded.
- **Comparison of the oldest symbols:** The paths whose branches differ from that of the best path at depth L are discarded. This way, initial divergence of the merging paths is detected and rejected. This is illustrated in figure 5.6. The survivors whose surviving path passes by the dashed line must be discarded because if one of these paths become the best path in the following trellis stages, its oldest bit will be released as output causing conflicts in the generation of the decoded sequence.

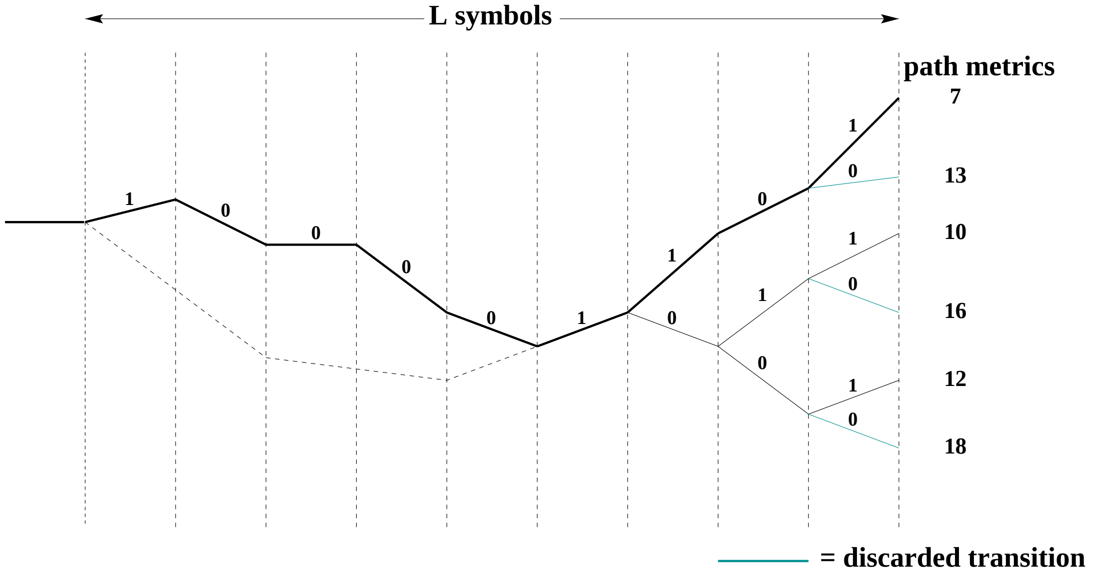


Figure 5.6: Merging paths during the construction of the surviving paths (M=3).

The second solution presents two main drawbacks. First, it must be pointed out that this method introduces a lag of L symbols in rejecting the merging paths. The problem that comes out is that during this interval, path merging is not tested. As a consequence, merging paths can slip into the best M paths, discarding other paths that might have become the overall best path. The full search capability is thus reduced. Moreover, this problem can be important if the decoding depth is large.

The second disadvantage is that this approach can only be adopted for *register-exchange-like* methods of the survivor memory management. Indeed, in order to compare the last decision bits of all the surviving paths, these bits must be available always, and this can only be accomplished by storing them in banks of registers (see figure 5.4). As a result, this method cannot be implemented in trace-back-based approaches for the survivor memory management since in this method only the path corresponding to the best survivor is re-created. Therefore, in spite of its higher complexity, the state comparison method will be utilized in this thesis to perform the path merging rejection.

Regarding the state comparison method, the first problem that arises is that the sorting network is unable to handle the sorting and rejection operations simultaneously. Figure 5.7 shows an example of a $M = 4$ odd-even sorting network which has been modified to account for the path merging rejection. There are two kinds of information feeding the sorting network, the state metrics and the state labels associated to each contender path. Each comparison-exchage element compares the state labels of their corresponding inputs and if these labels are the same, an ∞-metric is assigned to the contender path having the largest metric so that it goes down all along the sorting network. This way, the output metrics are in sorted order and the best paths all have different state labels. Notice that an additional compare-exchange element is required to ensure the complete arrangement of the output list, that is, to force the ∞-metric contenders to be placed at the botton of the sorting network. An additional sorting layer is thus required.

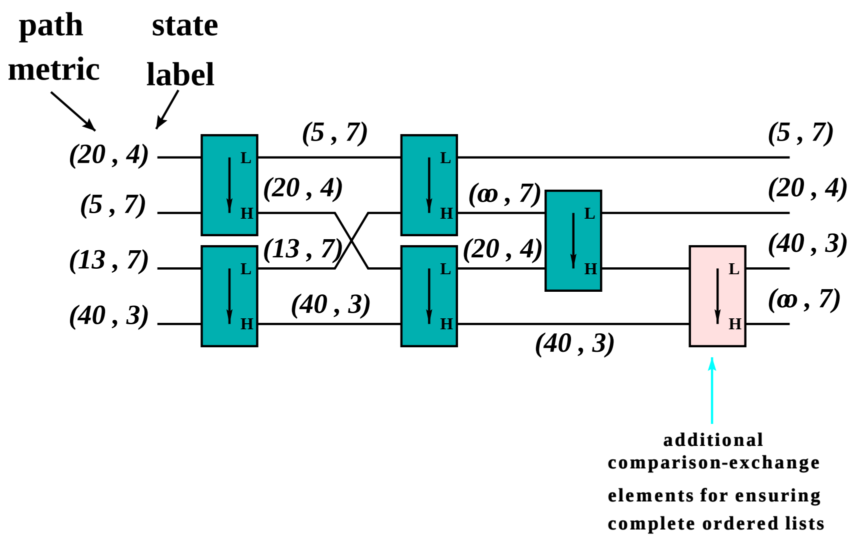


Figure 5.7: Path merging rejection in odd-even sorting networks. 4-item structure.

The problem with this approach is that generalizations to higher number of surviving paths is not straightforward. To illustrate this, figure 5.8 shows an example of a sorting network for $M = 8$ surviving paths. Notice that eventhough the output metrics are in sorted order, the structure failed in the rejection of merging paths since there are two survivors among the M best paths with the same state label. In addition, notice the modification that the sorting network suffers in its last layer. Once again, this modification is to guarantee that the ∞-metric paths descend to the bottom of the network.

In the following section, different architectures that have been proposed to select the best M paths, with a simultaneous rejection of merging paths are presented.

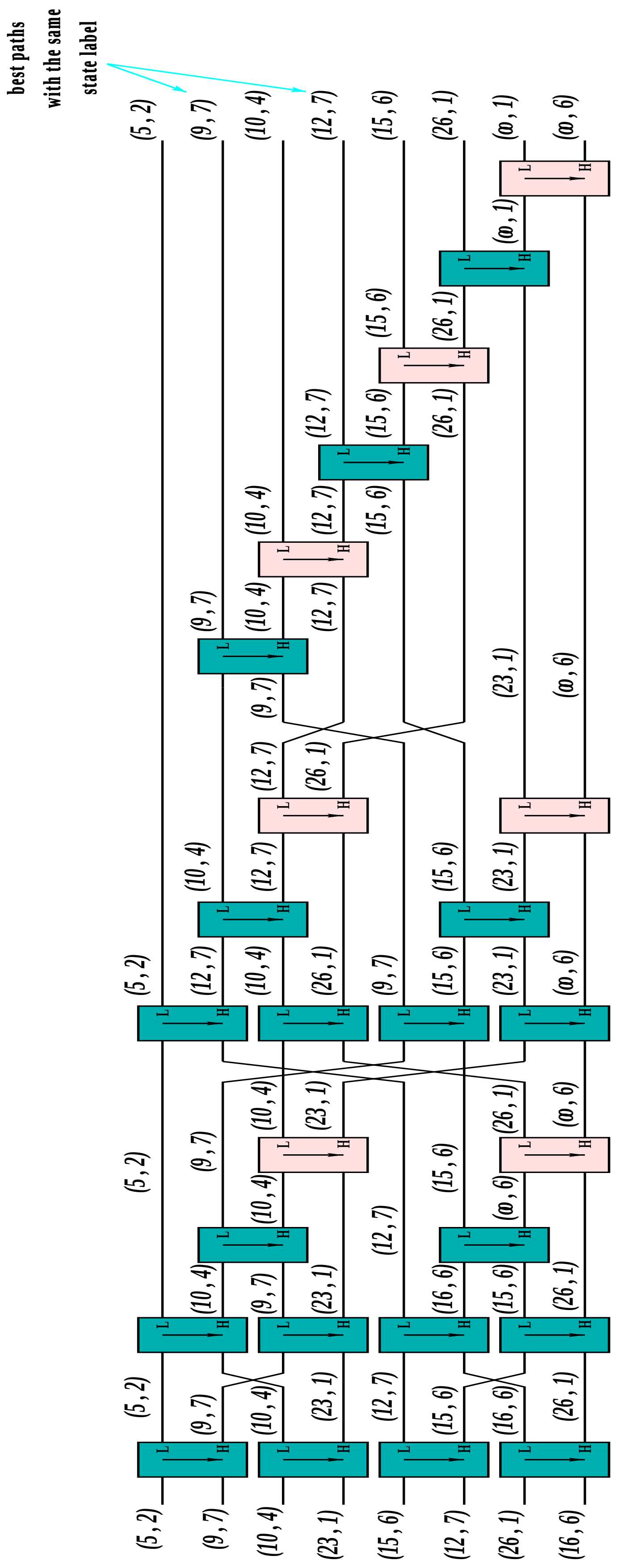


Figure 5.8: Path merging rejection in odd-even sorting networks. 8-item structure.

5.2.2 State of the Art in Sorting Architectures for the M Algorithm.

Eventhough the M algorithm has proven its effectiveness in different applications, very few hardware implementations have been reported. The first hardware implementation of a sequential encoder of speech utilizing the M algorithm is due to Anderson and Ho [5]. The sorting algorithm used was the serial insertion technique (see appendix B). The circuit was implemented with TTL (Transistor Transistor Logic) components (late 70's).

More recent reported work dates from the late 80's. Mohan and Sood [82] present a multiprocessor architecture for VLSI implementation of the M algorithm. The sorting algorithm used is Batcher's odd-even sorting network. In a very creative way, the sorting network is substituted by a column of M processors and an interconnection newtwork. Each processor performs the extension step, the computation of the branch metrics and the updating of the state metrics. Then, to select the best paths, the processors and the interconnection network are repeteadely used to simulate the stages of the sorting network. The processors perform the corresponding comparisons and the interconnection nerwork performs the corresponding switching operations to direct the different metrics to their corresponding processor for comparison. This is done in circular form, and by changing the states of the switches in the interconnection network, a processor can read from different places. In this way, the interstage data transfer required within the sorting network can be done using only M processors, as opposed to $[(log_2 M)^2 \log_2 M] \cdot M$ for a direct implementation. Nevertheless, it must be noticed that the interconnection networks are also composed of comparators, mutliplexers, demultiplexers etc. which do not allow a significant reduction in the hardware complexity.

Simmons presents a series of papers where different sorting algorithms are utilized [102, 103, 104]. These architectures are based on a common underlying structure; only the sorting methods differ. The sorting algorithms employed are the bitonic sorting network [11], odd-even transposition, insertion and weavesorting [96]. In order to alleviate the hardware complexity and power consumption, the processing is done bit-serially. Yet, there is a penalty in increased latency. In [102], a nonsorting VLSI structure is proposed. Simmons noticed that the sorting operation and its associated complexity can be avoided by using selection algorithms instead of sorting. This way, although the best M paths are not delivered in sorted order, they are still selected. As a result, the circuit area is largely reduced, meaning that larger trellises and larger values of M can be implemented on a single chip.

Concerning the rejection of merging paths, Simmons proposes a solution to the problem encountered in the comparison of the oldest symbols method by comparing the entire surviving paths to that of the best survivor during the first L trellis stages. If an initial divergence at any point in the trellis is followed by a remerging to a common state, a rejection flag is set and the remerging path is deleted. As we can see, this solution can become more complex that the state comparison approach.

On the other hand, Mohan and Sood describe a way of implementing an optimum path merge rejection by using the state comparison method without modifying the structure of the sorting network [82]. Figure 5.9 depicts their approach. The selection of the best M paths is carried out according to two criteria. First, a sorting operation is performed with respect to the state labels; then, a second sorting is done, this time with respect to the path metrics. After the first sorting, it is guaranteed that the merging paths, if any, are adjacent; then, the state labels of adjacent survivors are compared and an ∞-metric is assigned to the path with the largest metric. This way, after the second sorting, the survivors with infinite metric are placed in the lowest positions of the ordered lists and the best M paths will all have different state labels.

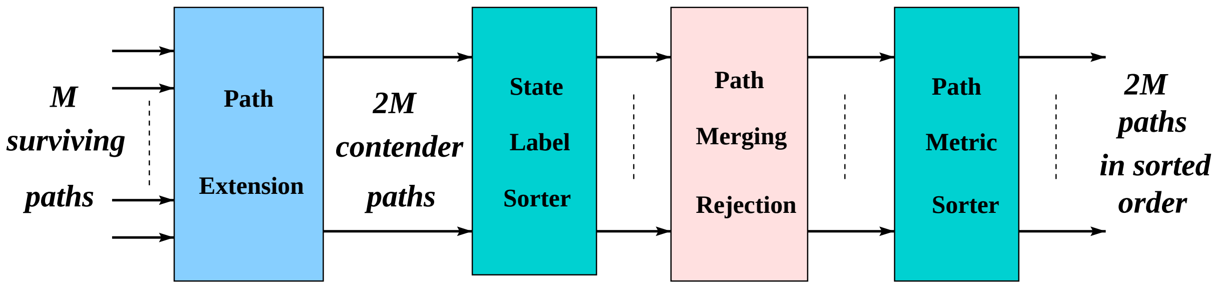


Figure 5.9: Mohan and Sood's approach for path merging rejection.

Notice that the hardware complexity of this approach suffers no augmentation since the same M-processor layer with the interconnection network are used for both sorting operations. We must take into account, however, that eventhough this approach is optimum and no hardware augmentation is required, the latency of the sorting operation is doubled since two sorting operations are now performed. In the next sections new ways of implementing the selection of the M best paths which improve both the hardware complexity and the latency are presented.

5.2.3 First Method for Selecting the Best Paths: Path Merge Detectors

In section 5.2.1, we saw that the sorting network cannot manage path merging rejection by itself. In this section we are going to study a way of modifying the sorting network so that the selection of the best M paths and the rejection of merging paths can be carried out on a single sorting operation.

We begin this discussion by remembering the **mutual independence of extended sets** property described in section 5.1. It was shown that the extended paths could be divided into two sets and only those extended paths that belong to the same set are likely to become merging paths. As a result, it is useless to perform the test for path merging rejection on the 2M-item list. Instead, this test can only be performed on each set of the new extended paths as shown in figure 5.10. This way, two M-item sorting networks performing path merging rejection are needed, together with a final $2M$ merging network to sort all the 2M items[1]. We will see that this new approach can alleviate the hardware requirements and the latency of the architecture.

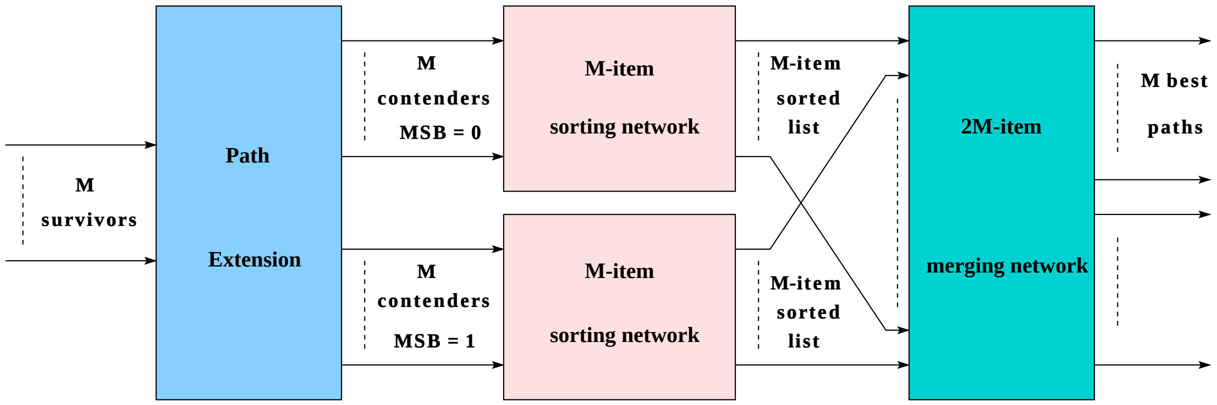


Figure 5.10: Separation of the extended paths into two sets for path merging detection.

In this new approach, the comparison-exchange elements now become *comparison-exchange-rejection* elements (CER) whose internal structure is shown in figure 5.11(a). They are composed of two comparators, one for metric comparison and the other for merging detection. When the state labels associated to the input metrics are the same, a saturation value $((111\cdots111)_2)$ is assigned to the survivor with the largest metric. Otherwise, the CER element behaves as a normal comparison-exchange element. It must be pointed out that the complexity of the new CER element is not significant (5 gates per bit) and its latency can be neglected.

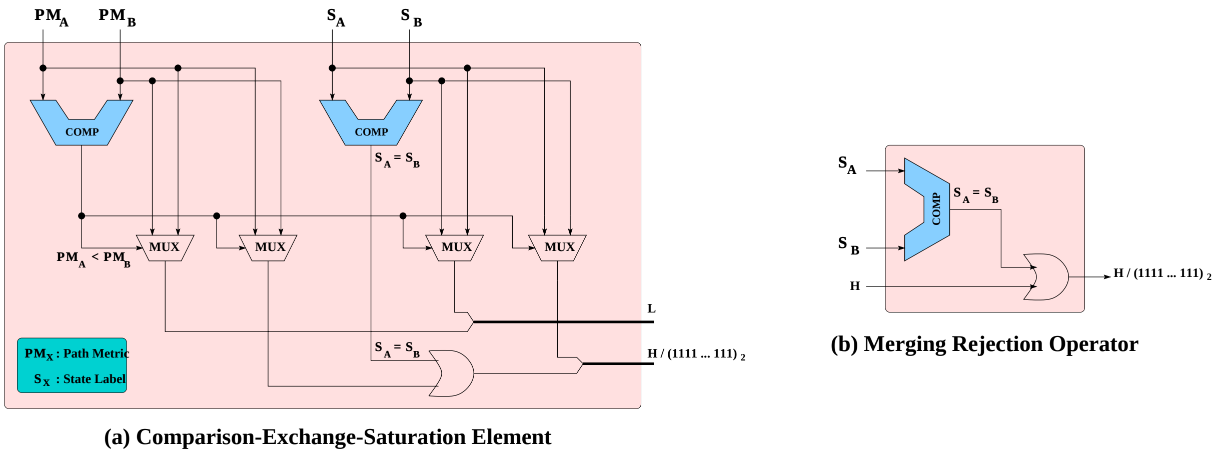


Figure 5.11: Internal Structure of the new operators utilized in the sorting network. a) comparison-exchange-saturation; b) merging rejection operator.

We saw in the previous paragraphs that by using comparison-exchange-rejection elements in the odd-even sorting network did not guarantee the simultaneous rejection of

[1]Refer to appendix B for definitions and differences between sorting and merging networks

merging paths and the selection of the best ones. Therefore, in addition to the CER elements, other operator, called *merging rejection operator* (MRO), is also needed to compare those surviving paths who might never be compared during the sorting operation. The merging rejection operator consists only of a state label comparator and a saturation circuit as shown in figure 5.11.b. This way, when the two state labels are the same, the largest metric path is saturated to the $(111\cdots111)_2$ metric. Notice that it must be already known which survivor has the smallest metric before entering the MRO.

The new sorting network performing simultaneous sorting and path merge rejection is presented in figure 5.12 for a 4-item list. Notice that the merging rejection operators must be carefully placed in the sorting network since not all the contender paths are compared with each other within the network. For instance, it is possible that input In_1 in the figure might never be compared to input In_4 (in the case, for example, where In_1 is the smallest metric and In_4 the largest one). A MRO is thus required between the L output of the CER element 3 and the H output of the CER element 4. The arrow in the MRO indicates the metric that is saturated if necessary. Accordingly, it is likely that the L output of CER 4 and 3 will never be compared (for example, when the L output of CER 3 is the input In_1 and the L output of CER 4 is the input In_3 or In_4). Hence, another MRO is needed between these two outputs. In the same manner, a last MRO is required between the H output of CER 3 and the H output of CER 4. Notice that the final comparison-exchange element is still needed for the complet sorting of the list.

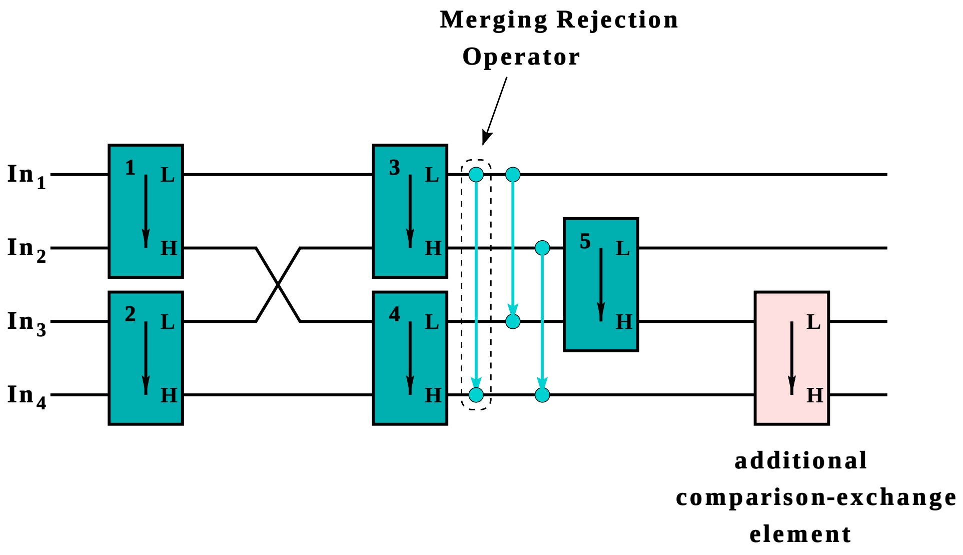

Merging Rejection Operator
In1
In2
In3
In4
1
2
3
4
5
L
H
additional comparison-exchange element

Figure 5.12: Odd-even sorting network for $M = 4$ performing simultaneous sorting and path merging rejection.

In general, with this new configuration of the sorting network, the merging networks which constitute the sorting network need

$$\frac{M}{2} - 1 \tag{5.9}$$

additional comparison-exchange elements.

On the other hand, we can see that the placing of MROs is a combinatorial problem that of how many combinations of two elements are possible in a list of M items. Hence, the total number of MROs in a M-survivor merging network is

$$\binom{M}{2} = \frac{M}{2} \cdot (M-1) \tag{5.10}$$

For example, for the 4-metric list of figure 5.12, six MROs are needed. However, because the CER elements act also as MRO, only 3 additional operators are required. In general, a M-item merging network needs

$$\log_2 \frac{M}{2} \cdot \frac{M}{2} + 1 \tag{5.11}$$

comparison-exchange elements. Thus, the number of additional MROs per merging network is

$$\frac{M}{2} \cdot (M-1) - \left(\log_2 \frac{M}{2} \cdot \frac{M}{2} + 1\right) \tag{5.12}$$

The reader is invited to corroborate that in this new structure of the odd-even sorting network the final list is in sorted order and merging paths are indeed rejected.

Figure 5.13 illustrates an $M = 8$-survivor sorting network with merging path rejection. As stated above, the 4-item merging networks need 1 additional comparison-exchange element and 3 MROs. On the other hand, the 8-item merging network needs 3 additional comparison-exchange elemensts and 28 MROs, from which 9 are already used within the CER elements. We need thus to place 19 additional operators to ensure the complete rejection of merging paths. The position of the merging rejection operators is shown in the figure.

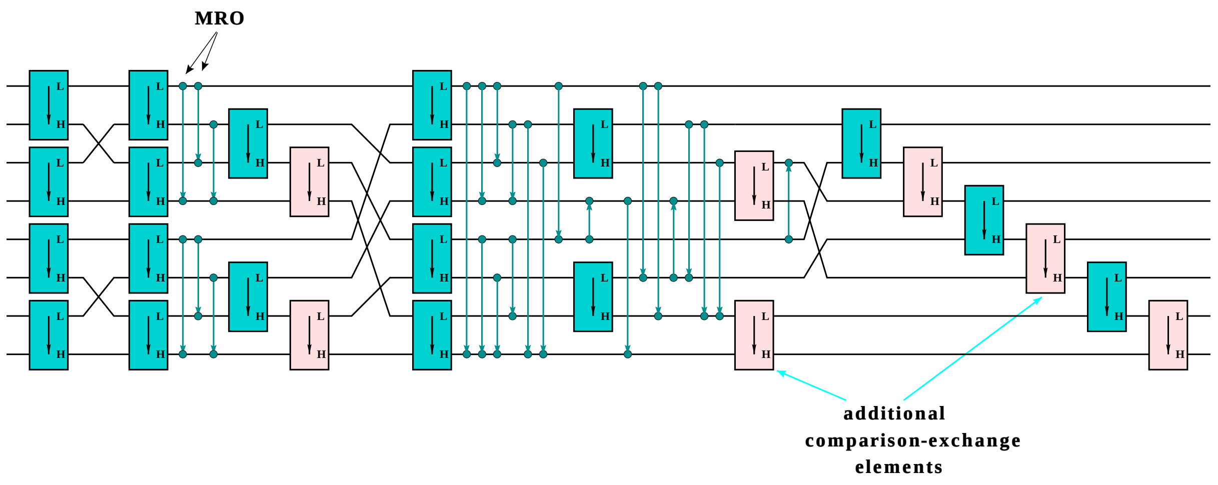


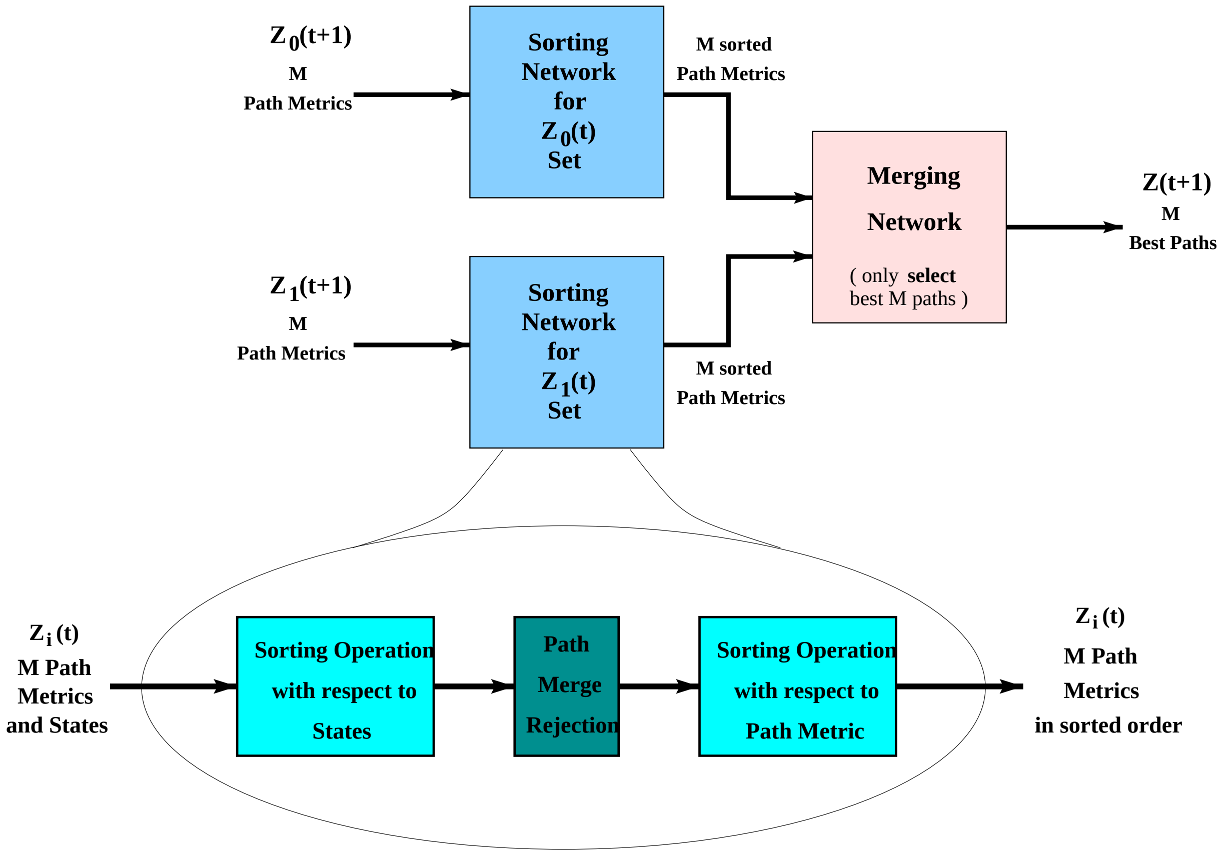
Figure 5.13: Odd-even sorting network for $M = 8$ with simultaneous sorting and path merging rejection.

So far, the placing of the path merging detectors was done in a trial-and-error approach. This means that an extremely complex problem arises as the number of surviving paths increases. For example, for an $M = 16$ trellis search, 95 merging operators must be placed in the 16-item merging network. Analogically, for a $M = 32$ trellis search, 431 merging operators are requried. Clearly, for large M it is practically impossible to know the exact location of the merging operators in the merging network. By means of a computer program we would be able to know the different paths that a given input can follow through the sorting network and hence, the locations of the merging rejection operators can be known. Nevertheless, we have to take into account that this path depends on the other inputs to the sorting network. That is to say, eventhough the inputs to the odd-even merging network are arranged in sorted order (the input is divided into two ordered lists of $\frac{M}{2}$ items each), the shuffle that they experiment through the network makes very hard to forecast or trace the paths followed by the different contender paths. Therefore, the problem of finding the exact locations of the merging rejection operators is left open and we opt to perform this placing in an empirical way or decide to search for alternative solutions.

Notice however that this first method for path selection can be used in suboptimum implementations of the M algorithm. That is, eventhough this method is not optimum since merging paths are not detected in a precise way, at least, thanks to the merging rejection operators, the probability of selecting two surviving paths with the same state label is largely reduced.

5.2.4 Second Method for Selecting the Best Paths: Combined Sorting and Selection

Another way of achieving optimum path merge rejection with a reduction in hardware complexity consists in the use of Mohan and Sood's approach but with some algorithmic modifications. A first modification consists in the division of the extended paths into two sets; we saw in the previous section that this division can be advantageous from a hardware complexity point of view. A second modification can be done by observing that if the decoding depth L of the trellis search is respected, then, decoding the source sequence from any surviving path will not introduce large performance degradations as long as this surving path belongs to the best M surviving paths. Therefore, instead of sorting the entire $2M$ list to obtain the best M paths in sorted order, we can sort independently the two sets of the extended paths with Mohan and Sood's approach and once we have the two lists in sorted order, the best M paths are selected (**not sorted**) from the two lists. This is illustrated in figure 5.14.



Z_0(t+1) M Path Metrics
Sorting Network for Z_0(t) Set
M sorted Path Metrics
Merging Network (only select best M paths)
Z(t+1) M Best Paths
Z_1(t+1) M Path Metrics
Sorting Network for Z_1(t) Set
M sorted Path Metrics
Z_i (t) M Path Metrics and States
Sorting Operation with respect to States
Path Merge Rejection
Sorting Operation with respect to Path Metric
Z_i (t) M Path Metrics in sorted order

Figure 5.14: Block Diagram of the Combined Sorting and Selection Method.

The path merging rejection block in the inner structure of each sorting network is composed of a column of M path merging detectors comparing adjacent states. The circuit for implementing path merging rejection is shown in figure 5.15. When the state labels in the inputs to the path merging detector are the same and path metric 1 (PM_1) is smaller than path metric 2 (PM_2), that is $PM_1 \ < \ PM_2 \ = 0$, the PM_1 output line copies the PM_1 input and the PM_2 output line is set to the saturation value $(111 \cdots 11)_2$. If the two state labels are the same but $PM_1 \ < \ PM_2 \ = 0$, then the input PM_2 is transferred to the output line PM_2 and the PM_1 output is saturated. Finally, if the state labels are different, the two inputs are transferred to their corresponding outputs.

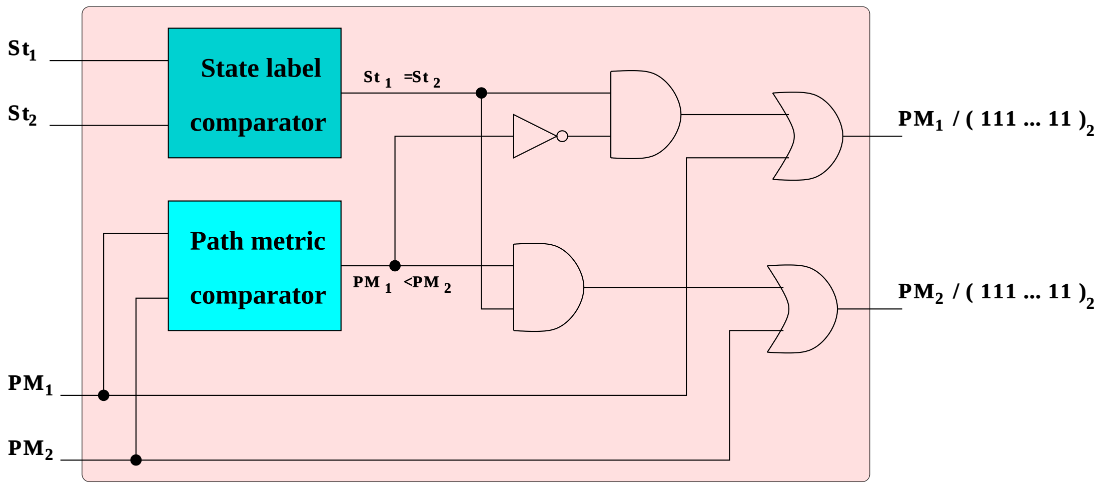


Figure 5.15: Circuit diagram of a path merge rejection cell.

The selection of the best M paths can be very easily accomplished by noticing that in the bitonic merging network explained in appendix B, after the first vertical layer of comparison-exchange elements, the upper half of the resulting list contains already the best M paths among the total list of $2M$ items, provided that the input list is bitonic. This is illustrated in figure 5.16. Notice that the two lists entering this one-layer merging circuit must be in sorted order.

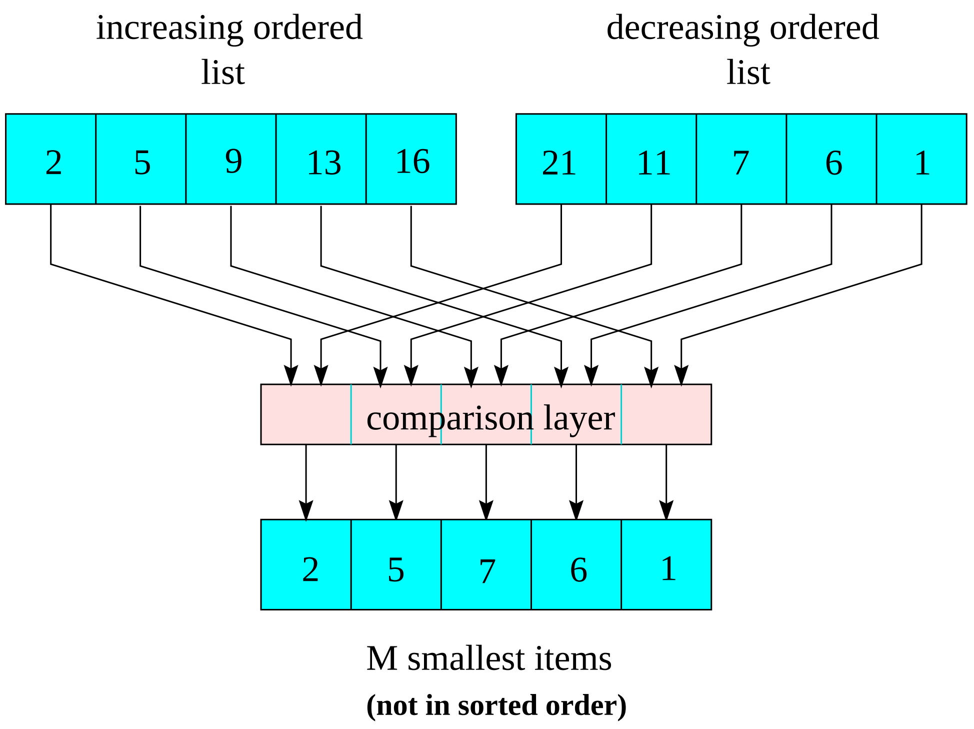


Figure 5.16: Selection of the best M paths with a one-layer bitonic merging.

An example of the entire sorting network is shown in figure 5.17. As indicated, the L outputs of the bitonic merge have the best M paths. Moreover, these paths have distinct state labels since the path merge rejection was performed during the sorting of each set $Z_i(k+1)$.

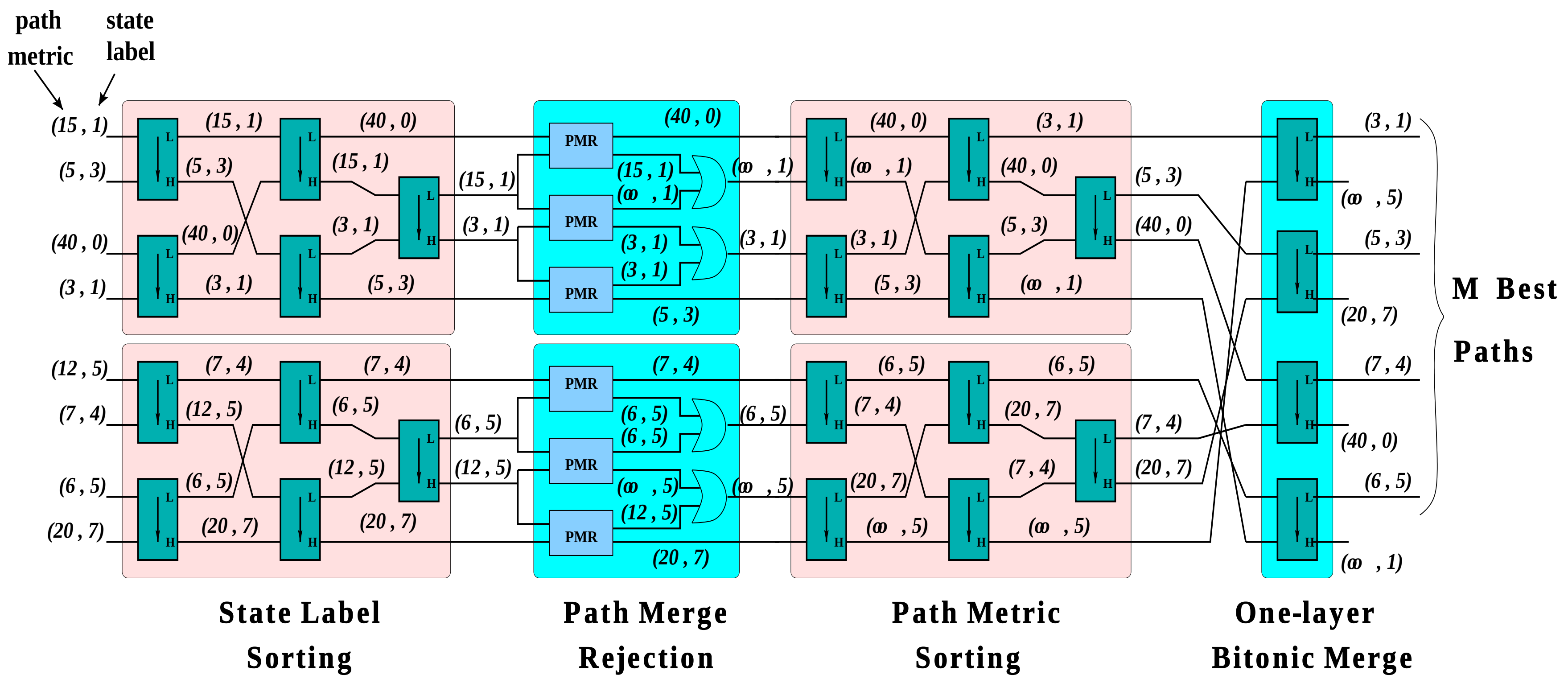


Figure 5.17: Sorting Example of the Combined Sorting and Selection Method.

Regarding the hardware complexity, table 5.1 presents the number of comparison-exchange elements used in the proposed method and in Mohan and Sood's approach. The expressions indicating the number of comparison-exchange elements required in each approach are

Mohan and Sood

$$2\left\{\frac{2M}{4}\left[(\log_2 2M)^2 - \log_2 2M + 4\right] - 1\right\} \tag{5.13}$$

Combined Sorting and Selection

$$4\left\{\frac{M}{4}\left[(\log_2 M)^2 - \log_2 M + 4\right] - 1\right\} + M \tag{5.14}$$

Coefficient 2 in Mohan and Sood's expression indicates that the sorting operations of a $2M$-item list is performed twice, one for sorting the state labels and the other for sorting the path metrics. Coefficient 4 in our proposed method indicates that each set $Z_i(k+1)$ composed of M items each is sorted twice. Then, these sortings are followed by a final layer of M comparison-exchange elements to select the best paths. We can see that a reduction in hardware complexity is achieved with practically no performance degradation.

Table 5.1: Comparison-exchange elements used in the combined sorting and selection method and in Mohan and Sood's approach.

M	Mohan and Sood	Proposed Method	Hardware Reduction (%)
2	10	6	40
4	38	24	37
8	126	84	33.3
16	382	268	29.8
32	1086	796	26.7
64	2942	2236	24
128	7678	6012	21.6
256	19454	15612	19.7

It can be argued that the previous comparison is not fair since Mohan and Sood's approach delivers the best M paths in sorted order whereas the combined sorting and selection method not. Table 5.2 gives the number of comparison-exchange elements used in both methods, but Mohan and Sood's approach is now modified so that it only selects the best M paths, though not necessarilly in sorted order. The new expression to find the number of comparison-exchange elements in the Mohan and Sood's combined sorting and selection approach is

$$\frac{2M}{4}\left[(\log_2 2M)^2 - \log_2 2M + 4\right] - 1 + 2\left\{\frac{M}{4}\left[(\log_2 M)^2 - \log_2 M + 4\right] - 1\right\} + M \tag{5.15}$$

that is, one sorting operation of $2M$ items is done for the sorting of the state labels and then two sortings of two M-item lists are performed for the selection of the best M paths. We can see that even in this case, our proposed method presents a lower hardware complexity.

Table 5.2: Comparison-exchange elements used in the proposed method and in Mohan and Sood's combined sorting and selection.

M	Mohan and Sood's combined sorting and selection	Proposed Method	Hardware Reduction (%)
2	9	6	33.3
4	33	24	27.2
8	109	84	22.9
16	333	268	19.5
32	957	796	16.8
64	2621	2236	14.7
128	6909	6012	13
256	17661	15612	11.6

Concerning the latency of both methods, table 5.3 shows the number of vertical layers needed by our proposed method and Mohan and Sood's approach for both complet sorted list and combined sorting and selection of the best M paths. The critical path expressions for these three cases are

Mohan and Sood's complete sorting

$$2\left[\frac{(1+\log_2 2M)\cdot\log_2 2M}{2}\right]\cdot t_p \tag{5.16}$$

Mohan and Sood's combined sorting and selection

$$\left[\frac{(1+\log_2 2M)\cdot\log_2 2M}{2}+\frac{(1+\log_2 M)\cdot\log_2 M}{2}+1\right]\cdot t_p \tag{5.17}$$

Combined Sorting and Selection

$$\left[\frac{(1+\log_2 M)\cdot\log_2 M}{2}+1\right]\cdot t_p \tag{5.18}$$

The first expression comes from the fact that 2 sorting networks of $2M$ items are used. In the second expression one $2M$-item lists is sorted followed by two M-item lists sorted in parallel and one final bitonic layer. Finally, in the last expression two M-item lists are sorted in parallel followed by an additional bitonic layer. Notice that in both cases the latency of the proposed method is lower than Mohan and Sood's approach.

Table 5.3: Architecture latency of the combined sorting and selction method and Mohan and Sood's approach.

M	Mohan and Sood's complete sorting	Mohan and Sood's combined sorting and selection	Proposed Method	Latency Reduction (%)	
2	6	5	2	67	60
4	12	10	4	67	60
8	20	17	7	65	58
16	30	26	11	63	57
32	42	37	16	62	56
64	56	50	22	61	56
128	72	65	29	58	55
256	90	82	37	59	55

5.2.5 Third Method for Selecting the Best Paths: Delayed State Label Sorting

One last method for selecting the best paths which further reduces the hardware complexity of the sorting structure consists in the following observation. As we saw in section 5.1, the *State Label Sorting Conservation of Extended Paths* property states that if the M surviving paths are sorted with respect to the state labels before being extended, then after the extension step of the next trellis stage the two sets $Z_0(k+1)$ and $Z_1(k+1)$ will have their corresponding state labels arranged in sorted order too. This means that the sorting network which sorts the state labels of the extended paths can be delayed and performed after the path metrics sorting and the merging of the ordered lists $Z_0(k+1)$ and $Z_1(k+1)$. Figure 5.18 presents the block diagram of such method.

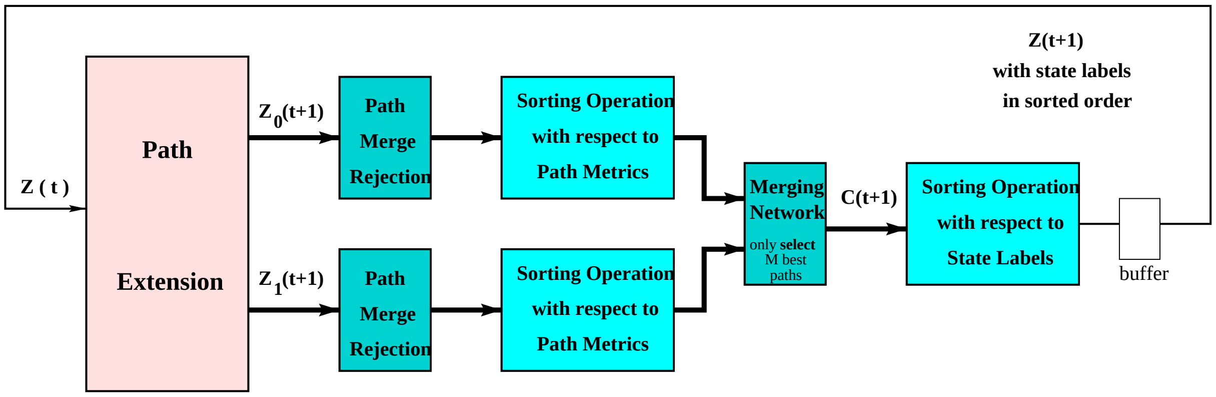


Figure 5.18: Block Diagram of the Delayed State Label Sorting Method.

After the division of the extended paths into the sets $Z_0(k+1)$ and $Z_1(k+1)$, the state labels of each set are in sorted order; hence, an initial path merging rejection between adjacent contender paths is done. Then, each set is sorted, this time with respect to the path metrics, and a one-layer bitonic merging is done so as to find the M surviving paths from the two sets. Finally, once the best paths are selected, a final M-item sorting network is employed to sort the selected paths with respect to the state labels. This way, after the extension stage of the next processing interval the contender paths will be arranged according to the state label.

An example of the operation of this method is given in figure 5.19. Notice that this new approach allow us to further improve the path merging rejection cell of figure 5.15. Indeed, the path merging rejection can be performed with a single state label comparator, the first layer of comparator-exchange elements of the path metric sorting network and a merging rejection operator (MRO), as shown in figure 5.19. We only need to compare the state labels of the middle values on each set. If these are the same, the first and third input to the path metric sorting network are switched so that the metrics of these merging paths are compared; then, the MRO saturates the contender with the largest metric.

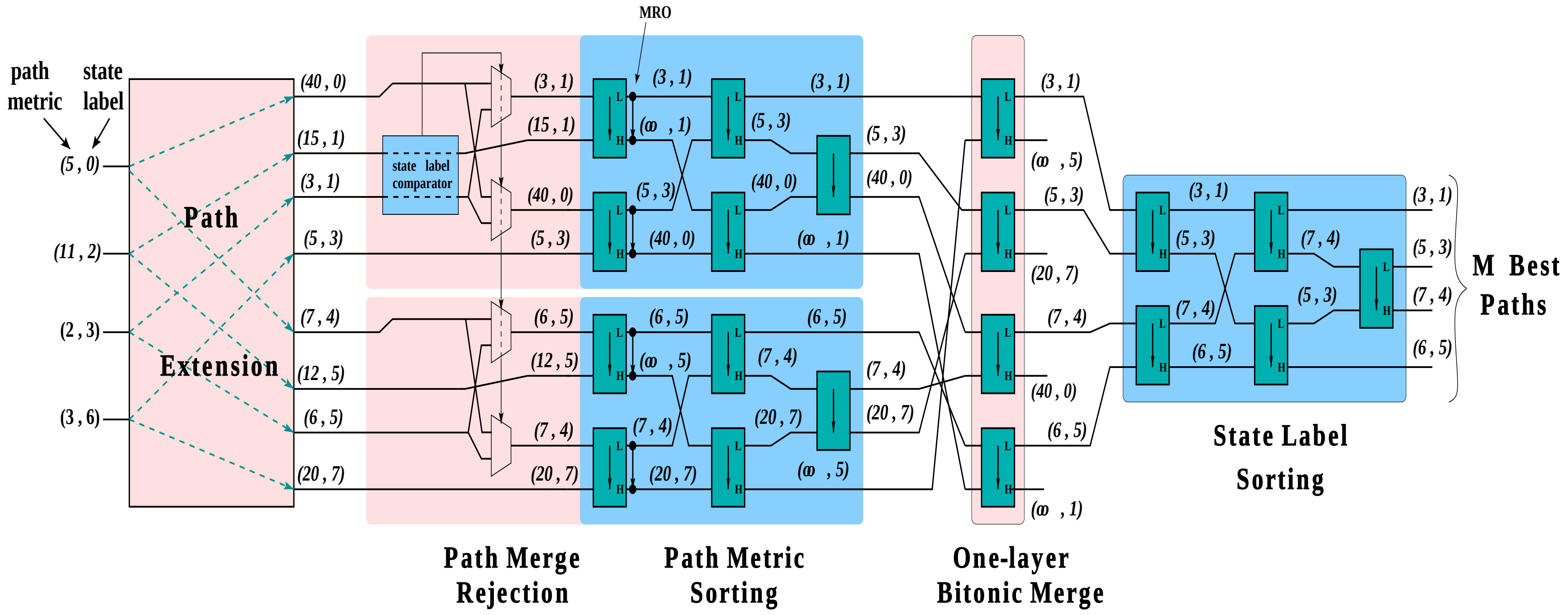


Figure 5.19: Example of the Delayed State Label Sorting.

If the state labels are not the same, the circuit works as a normal sorting circuit. Notice also that only one state label comparator is needed since the path extension is symmetrical, that is, if a path remerging occurs in set $Z_0(k+1)$, the same remerging occurs in $Z_1(k+1)$. As a result, the same state label comparator can control the path merging rejection of both sets. Moreover, this modification permits to save 7 comparators.

The main advantage of this new structure with respect to the previous one is the reduction in hardware. The complexity model of this method is given by

$$3\left\{\frac{M}{4}\left[(\log_2 M)^2 - \log_2 M + 4\right] - 1\right\} + M \tag{5.19}$$

This means that only three M-item odd-even sorting networks and one layer of bitonic merge are needed. Table 5.4 shows the hardware complexity of the delayed state label sorting method and its hardware reduction achieved with regard to Mohan and Sood's approach and the combined sorting and selection method of the previous section. Notice that the latency of the architecture is maintained since only a change in the arrangement of the sorting networks was made.

Table 5.4: Hardware complexity comparison between the delayed state label sorting network, the combined sorting and selection method and Mohan and Sood's approach.

M	Mohan and Sood's complete sorting	Combined Sorting and selection	Delayed State Label Sorting	Hardware Reduction (%)	
2	9	6	5	50	17
4	33	24	19	50	21
8	109	84	65	48	23
16	333	268	205	46	24
32	957	796	605	44	24
64	2621	2236	1693	42	24
128	6909	6012	4541	41	24
256	17661	15612	11773	39	25

5.2.6 Conclusion

In this section we have presented three new methods to perform the selection of the M best paths in the M algorithm. One of these methods, however, does not perform an optimum rejection of merging paths. Yet, it can be considered for suboptimum implementations or for implementations where the number of surviving paths is small. These three methods offer a significant reduction in both hardware requirements and latency. One reason for choosing the state comparison as the path merging rejection method is because this method is well suited for trace-back-based structures of the survivor memory management. In the next section, we complete our study of the M algorithm by proposing a new method to implement the survivor memory management based on the trace-back algorithm.

5.3 Survivor Memory Management

So far, we have focused only on the path metric updating block of figure 5.2. In this section an insight into the different techniques available for survivor memory management are presented and methods for increasing the hardware performance are highlighted.

In Chapter 4 we presented the two most important survivor memory management techniques for the Viterbi algorithm, namely the *Register Exchange* (RE) approach and the *Trace-Back* (TB) algorithm. Since the M algorithm belongs to the same class of trellis search as the Viterbi algorithm, the way the decision bits are produced during the path metric updating is the same; that is, at each processing interval, a decision bit per trellis state is generated indicating whether the upper or the lower branch arriving to a given state is the survivor. As a result, the same survivor memory management techniques that are used in Viterbi decoders can be used in the M algorithm as well. Yet, we will see that some modifications must be done for this to be true (See appendix A for more details).

5.3.1 Conventional Approach of Survivor Memory Management in the M Algorithm.

Conventionally, all the reported architectures of the M algorithm employ a register-exchange-like approach. The difference with the Viterbi algorithm resides in the fact that each surviving path holds all the information correspoding to it, namely the path metric, the associated state label and the last L decision bits, and this information is moved as a single entity all along the sorting network until the surviving paths are selected.

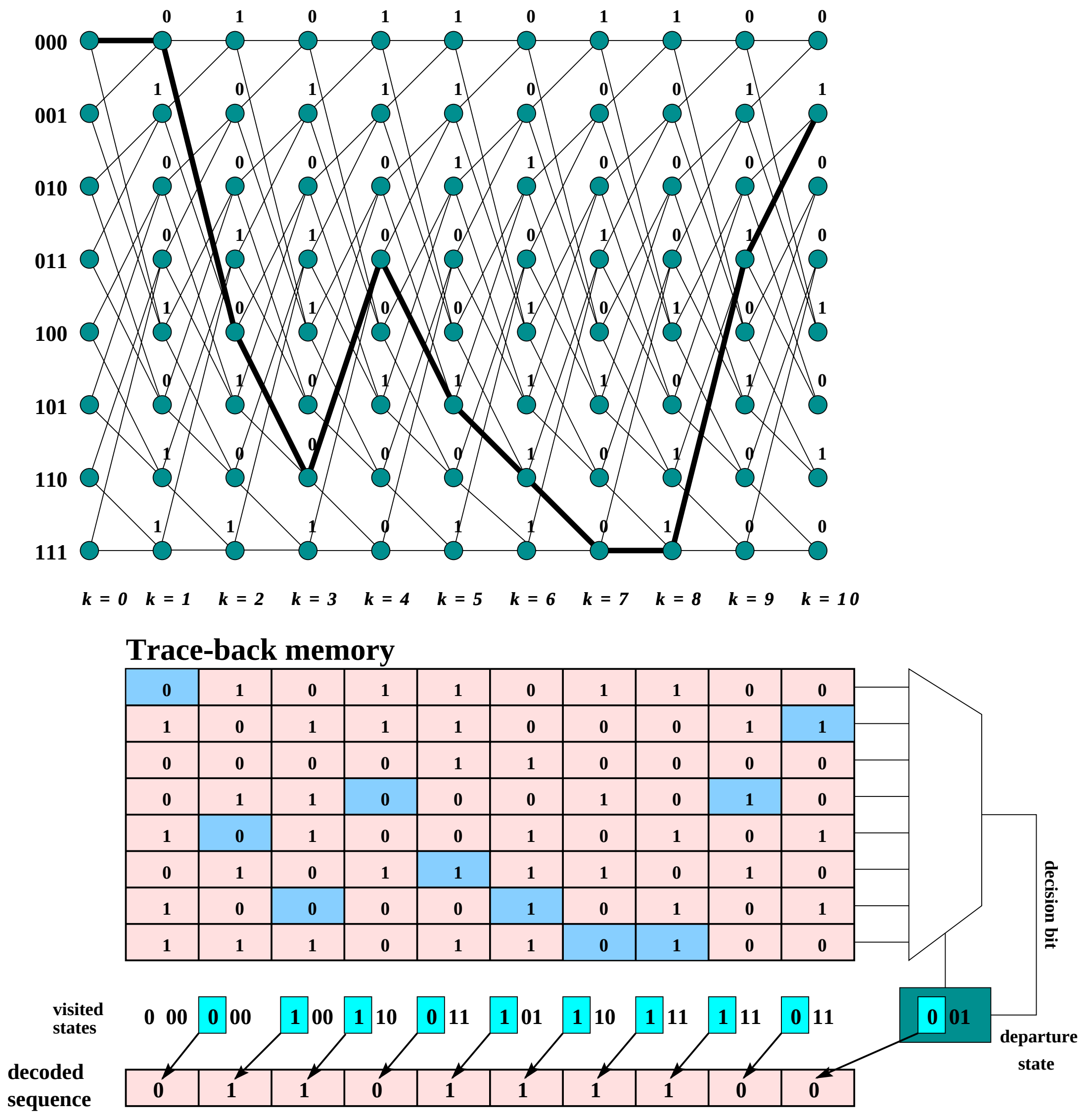
The general approach adopted by all the reported architectures was presented in figure 5.4 (page 105). Each surviving path is composed of a $W+L$-bit word, W bits correspond to the path metric of the surviving path and the last L bits correspond to the decision bits; from these L bits, the $K-1$ most recent denote the current state label of the surviving path. All this information is stored in an array of register and multiplexers like the RE approach of the Viterbi algorithm; nonetheless, when the extended paths enter the sorting network, the comparison-exchange elements that compose the sorting network must be able to support the data flow of all this information. When the number of surviving paths and the decoding depth of the algorithm is not very large, this kind of structures are suitable for VLSI implementations because of its simplicity (only registers and multiplexers are required) and regularity. However, for applications where large trellises and surviving paths are needed, this structure becomes impractical because the length of the binary word associated to each path must be increased. Thus, in the same manner as in Viterbi implementations, eventhough the latency of this structure is very small (L clock cycles), the architecture becomes bulky for large M and L. To alleviate this massive flow of information, interconnection networks are used [82] or bits are processed serially so as to obtain a more compact structure but with its corresponding penalty in decoding latency [102, 103, 104]. In any case, this movement of information is still power consumming since all the $M \times (W+L)$ bits move through the sorting network simultaneously [23]. Therefore, other solutions have to be proposed, and this solution may be the trace-back algorithm.

5.3.2 Trace-Back Techniques Adapted to the M Algorithm

The Trace-Back procedure can lead to very efficient implementations of Viterbi decoders [31, 41, 85]. Its large density of storage allows the design of small area and low power circuits. However, the main disadvantage of this approach is its latency; nevertheless, different methods to overcome this drawback are known.

In the same manner as the register exchange approach, the trace-back procedure needs some modifications so that it can be applied to the M algorithm. The problem that arises when we try to use the trace-back procedure in the M algorithm is that the decision bits generated by the path extension block of figure 5.3 only correspond to the best M paths, that is, to M trellis states. Clearly, at each time step, different trellis states survive and hence, we cannot predict the position of the decision bit corresponding to a given state as in the Viterbi algorithm since we are not sure that this state will survive at the next time step. In addition, these decision bits are shuffled by the sorting network in such a way that the exact location of the decision bit corresponding to a survivor state cannot be known in advance.

Let us explain this in a more detailed way with an illustrative example. Figure 5.20 shows the operation of the trace-back procedure in the Viterbi algorithm. In this case, at every processing interval, 2^{K-1} decision bits, one for **each** trellis state, are generated and stored in the trace-back memory. Thus, we already know which decision bit corresponds to a given state. For example, the first bit in the decision vector corresponds **always** to



0	1	0	1	1	0	1	1	0	0
1	0	1	1	1	0	0	0	1	1
0	0	0	0	1	1	0	0	0	0
0	1	1	0	0	0	1	0	1	0
1	0	1	0	0	1	0	1	0	1
0	1	0	1	1	1	1	0	1	0
1	0	0	0	0	1	0	1	0	1
1	1	1	0	1	1	0	1	0	0

visited states	0 00	0 00	1 00	1 10	0 11	1 01	1 10	1 11	1 11	0 11
decoded sequence	0	1	1	0	1	1	1	1	0	0

departure state: 0 01

Figure 5.20: Example of the Trace-Back procedure in Viterbi decoders.

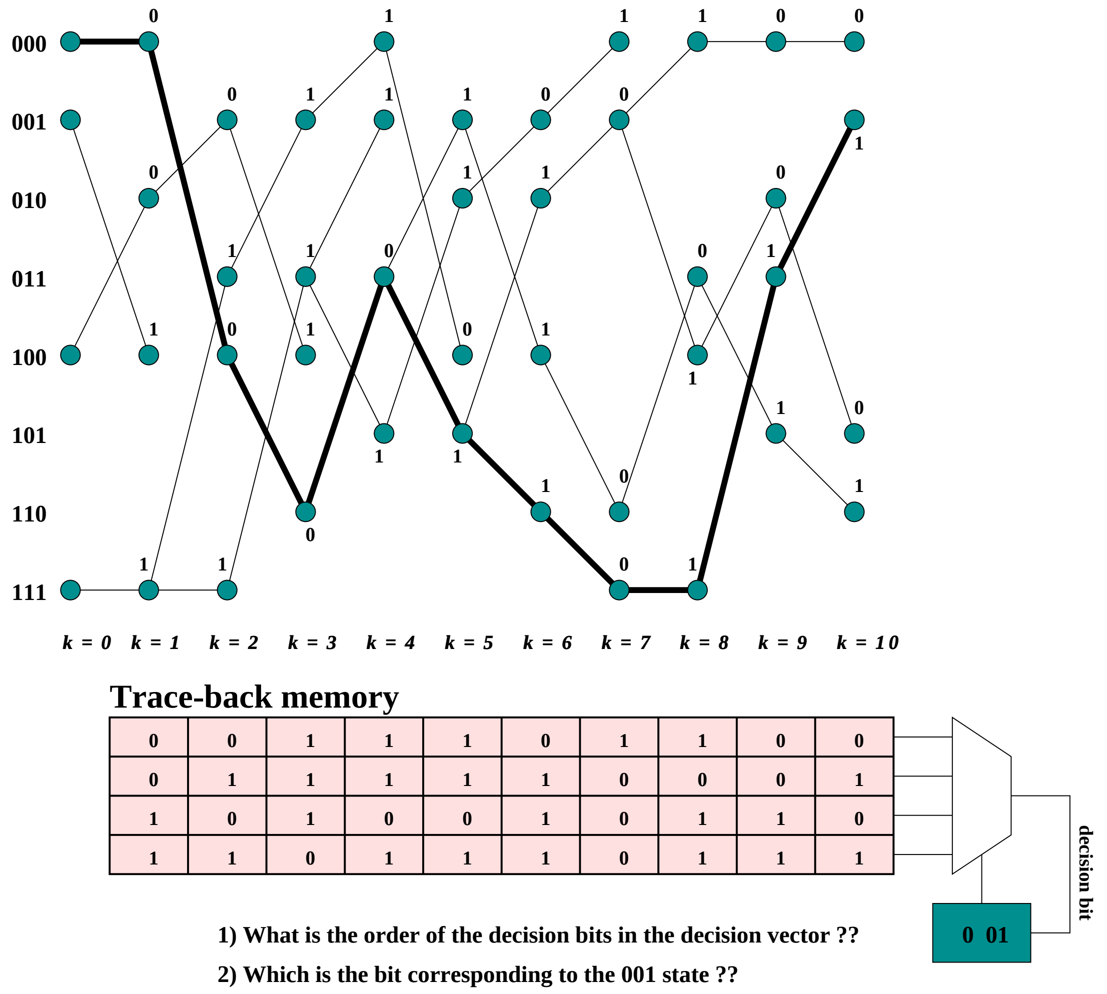
state $00\cdots00$, the second bit corresponds **always** to the $00\cdots01$ state and so on. This way, the steps that the trace-back procedure performs at every processing interval are:

1. Starting from $k = L$, select the trellis state i of time k which has the best metric and load its value into the contents of a shift register,
2. Read the decision vector corresponding to address k,
3. From this decision vector, select the i^{th} decision bit,
4. Shift the contents of the shift register one position to the left and insert the decision bit in the LSB position of the shift register. This step creates the state visited by the surviving path in the previous time step,
5. Set $k = k - 1$ and go to step 2.

Notice that there is a direct correspondance between the trellis state visited by the surviving path and the position of the decision bit into the decision vector. The trellis state acts thus as a pointer to the decision vector. Notice that this correspondance, *trellis state-decision bit position*, is maintained fixed throughout the entire decoding process because all the 2^{K-1} states are available at each processing interval.

On the other hand, figure 5.21 shows the problem encountered in the M algorithm when the same Trace-Back procedure is applied. In this case, the memory words are only M bits wide. As a result, when the Trace-Back procedure begins from the best trellis state, we cannot know which decision bit must be selected in the decision vector since the correspondance *trellis state-decision bit position* is not observed; that is, the trellis states visited by the surviving path cannot be pointed to by the state labels anymore.

To solve this problem, a naive solution could be the use of the same trace-back structure employed in the Viterbi algorithm. In other words, we can utilize the same $L \times 2^{K-1}$ trace back memory and update only those decision bits in the decision vector corresponding to the M surviving paths. This way, the Trace-Back decoding of the overall surviving path can be performed in the same manner as it is done in Viterbi decoders. Clearly this is not the wisest solution for two reasons. First, the waste in hardware ressources is extremely large, specially in the cases where the number of surviving paths is far smaller than the number of trellis states. Consider, for example, the case where the constraint length is $K = 10$ and the number of surviving paths is $M = 8$; if $L = 70$ we would need a 70×512-bit memory, from which only 8 bits per memory address are used. This means that from the 35840 bits that can be stored in the memory, only 4096 have meaningfull information. We are thus wasting almost 9 times the hardware ressources with its corresponding circuit area. Second, if we use this structure for the decoding of the surviving paths, the storage requirements needed for the M algorithm are still $L \times 2^{K-1}$ decision bits, the same requirements as for the Viterbi algorithm.



Trace-back memory

0	0	1	1	1	0	1	1	0	0
0	1	1	1	1	1	0	0	0	1
1	0	1	0	0	1	0	1	1	0
1	1	0	1	1	1	0	1	1	1

1) What is the order of the decision bits in the decision vector ??

2) Which is the bit corresponding to the 001 state ??

Figure 5.21: Wrong Trace-Back Procedure in the M algorithm.

5.3.2.1 Trace-Back Technique by Comparison of Visited States.

Another solution that we called *Trace-Back by comparison of visited states* is shown in figure 5.22. As we can see, the architecture is composed of the usual trace-back memory and shift register controlling a multiplexer, and an additional set of M comparators. Now, not only the decision bits are stored in the trace-back memory but the state labels of the surviving paths as well. Consequently, the size of the trace-back memory is $L \times M \cdot K$; that is, the decision vector is divided into M packets of K bits, $K-1$ correspond to the state label and 1 bit to the decision bit. The operation of this structure is:

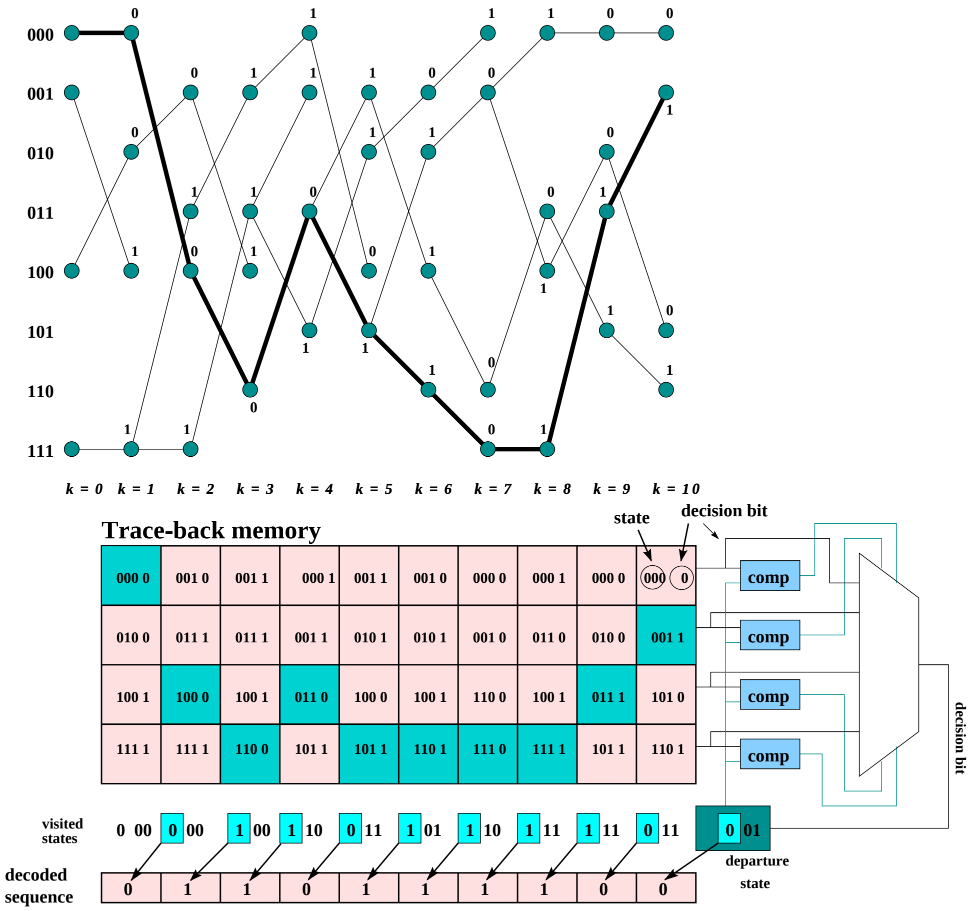


Figure 5.22: Trace-Back by comparison of visited states.

1. Select the trellis state of time $k = L$ which has the best metric and load its value into the contents of the shift register,

2. Read the decision vector corresponding to address k,

3. Compare the contents of the shift register with the M state labels of this decision vector. The decision bit of the K-bit packet that contains the same state as the one stored in the shift register is selected by the multiplexer,

4. The contents of the shift register is shifted one position to the left and the decision bit is inserted in the LSB position,
5. Set $k = k - 1$ and go to step. 2

The hardware requirements of this approach are:

- 1 $L \times (M \cdot K)$-size memory
- M $K - 1$-bit comparators
- 1 multiplexer
- 1 $K - 1$-bit shift register

This time, the reduction of the storage requirements is important. For a $K = 10$ trellis coder/decoder with $M = 8$ surviving paths, the amount of information to store is $L \times 80$ bits, as opposed to the $L \times 512$ bits required by the Viterbi algorithm. The reduction in storage requirements achieved by this architecture is in the order of 7. Nevertheless, the size of the decision vector is still large.

5.3.2.2 Trace-Back Technique with Path-Number Pointer.

A first observation towards the improvement of the architecture is that as a matter of fact, even if the state label can act as a pointer to the decision bits, we still can produce another type of pointer to this purpose. Figure 5.23 illustrates this approach. After the selection of the best paths, a path number $(0 \cdots M-1)$ can be associated to these paths depending on its position in the decision vector, as shown in figure 5.23; this way, after the sorting operation, we are able to know the ancestors of the new surviving paths. The decision bit of the surviving path and the path number of its ancestor are then stored in the trace-back memory. Then, the path numbers are reinitialized and the process continues in this way until the last decision vector. The position in the decision vector k of the current decision bit and the path number of its ancestor at time $k-1$ depends on the current path number. In this way, the path number can serve as a new pointer to the decision bit of the previous processing interval which corresponds to the overall surviving path since now there is a direct correspondance between the path number and the position of the decision bit in the decision vector. As a result, each time a decision vector is read from the trace-back memory, we are able to know the position of the decision bit in the next decision vector and thus the overall surviving path can be decoded. This method is referred to as *Trace-Back technique with path-number pointer.*

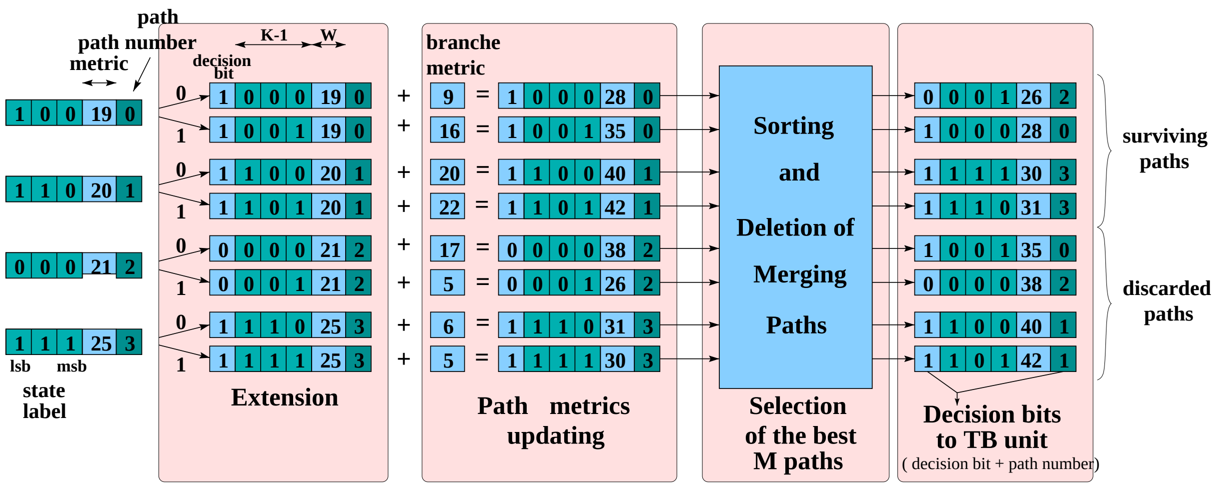


Figure 5.23: Pointer generation for TB decoding.

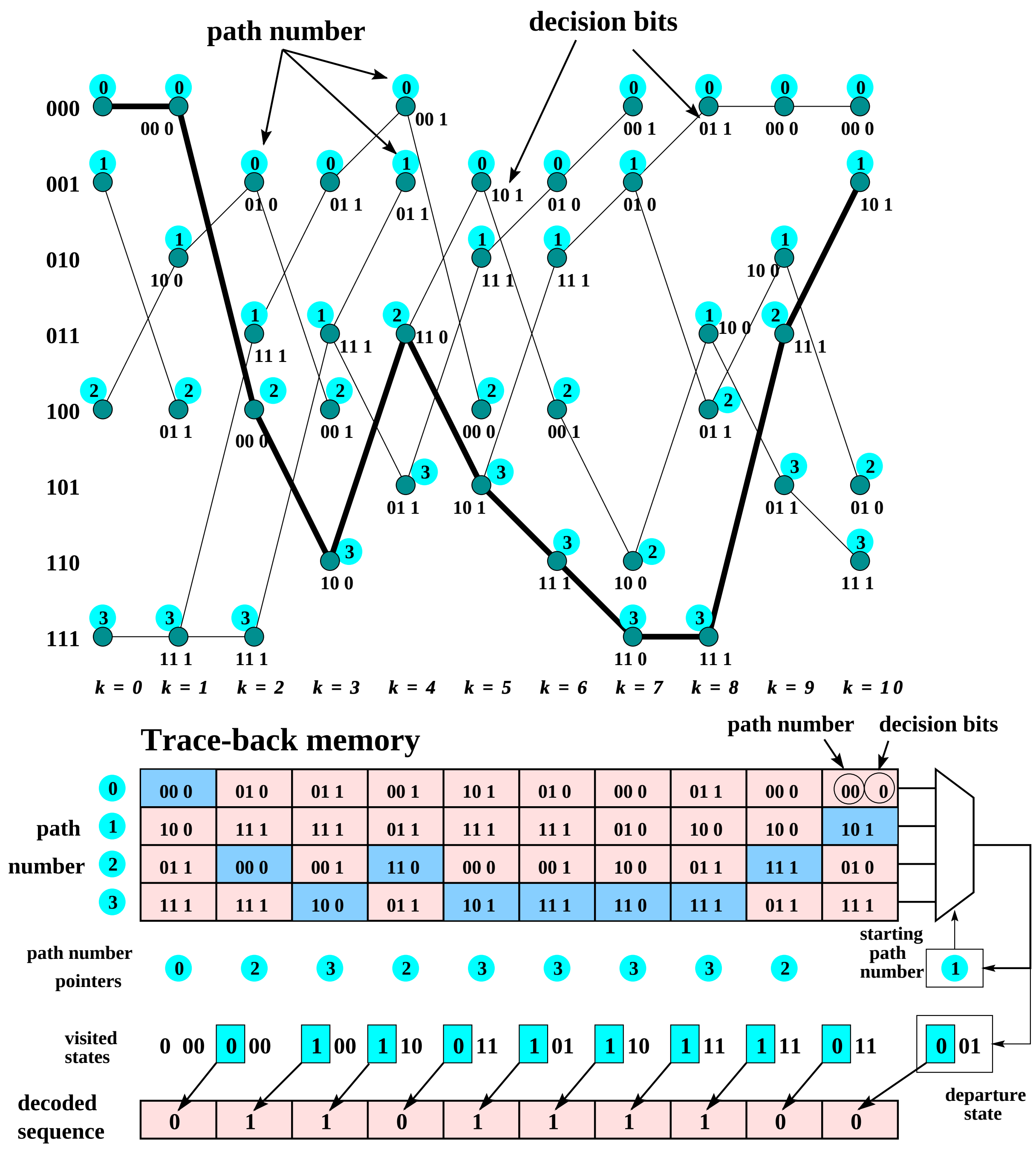


Figure 5.24: Trace-Back with Path-Number Pointer.

Figure 5.24 presents an example of operation of this new approach for a $K = 4$ trellis and $M = 4$ surviving paths. During the path metric updating stage (forward direction), once the M surviving paths are selected, they are enumerated from 0 to 3 (cercled numbers in the figure) according to the sorting operation; the decision vector at every processing interval is created by concatenating the decision bits and the path numbers of the ancestors of each surviving path. The concatenation depends on the current path number of the surviving paths; for example, in the figure, the 3 LSBs of the 12-bit decision vector at time $k = 5$ are 101 since the ancestor at time $k = 4$ of this first surviving path was labeled with path number 2 and the current decision bit is 1. The next three bits are 111 since the ancestor of this second path was labeled with number 3 and the current decision bit is 1; the other three bits correspond to 000, that is, the path's ancestor had the number 0 and the decision bit is 0. Finally, the last 3 bits are 101 which correspond to the surviving path whose ancestor had the path number 2 and the decision bit is 1. This process is repeated until the trace-back memory has been filled in.

During the trace-back operation, the register driving the multiplexer is set to the path number of the surviving path with the best path metric and the decision vector of address $k = 10$ is retrieved from memory. This first path number is 01, which means that the second 3 bits of the decision vector are selected (101). From these bits, the LSB is the decision bit which serves to decode the surviving path and the two MSBs indicate the next 3-bit packet (10) that will be selected in the next cycle. These bits are stored in the register and, at the next cycle, the decision vector of address $k = 9$ is read from memory, the third 3-bit packet is selected from the decision vector, the LSB decodes the surviving path and the MSBs indicate that the fourth 3-bit packet of the next decision vector at $k = 8$ will be chosen. This process is repeated until the last decision bit is read. This is the way the surviving path is decoded.

In general, because of the decoding depth property, we can start the trace-back procedure from the first three bits (path number $= 0$). The Path-Number-Trace-Back procedure consists in the following steps:

1. Set $k = L$ and the contents of the register to path number $= 0$
2. Read the decision vector of address k and select the bit packet given by the current path number. This packet contains $\log_2 M + 1$ bits.
3. Take the LSB to decode the surviving path and store the $\log_2 M$ MSBs in the register.
4. Set $k = k - 1$; if $k = 0$ halt, otherwise go to step 2

The main advantage of this architecture over the previous one is that the storage requirements become independent of the trellis constraint length (except, of course, for the decoding depth which still depends on the constraint length by the relation $L = 6 \cdot K$).

The hardware requirements of this structure are:

- 1 $L \times M(\log_2 M + 1)$-size memory
- 1 $\log_2 M$-bit register
- 1 $M(\log_2 M + 1)$ to $\log_2 M + 1$ multiplexer.

Comparing the storage requirements with the Viterbi algorithm, we find that a reduction in the order of $\frac{2^{K-1}}{M(\log_2 M+1)}$ is achieved. For instance, for $K = 10$ and $M = 8$, a reduction in a factor of 16 is obtained.

5.3.3 Improving Hardware Efficiency: Hybrid Architectures

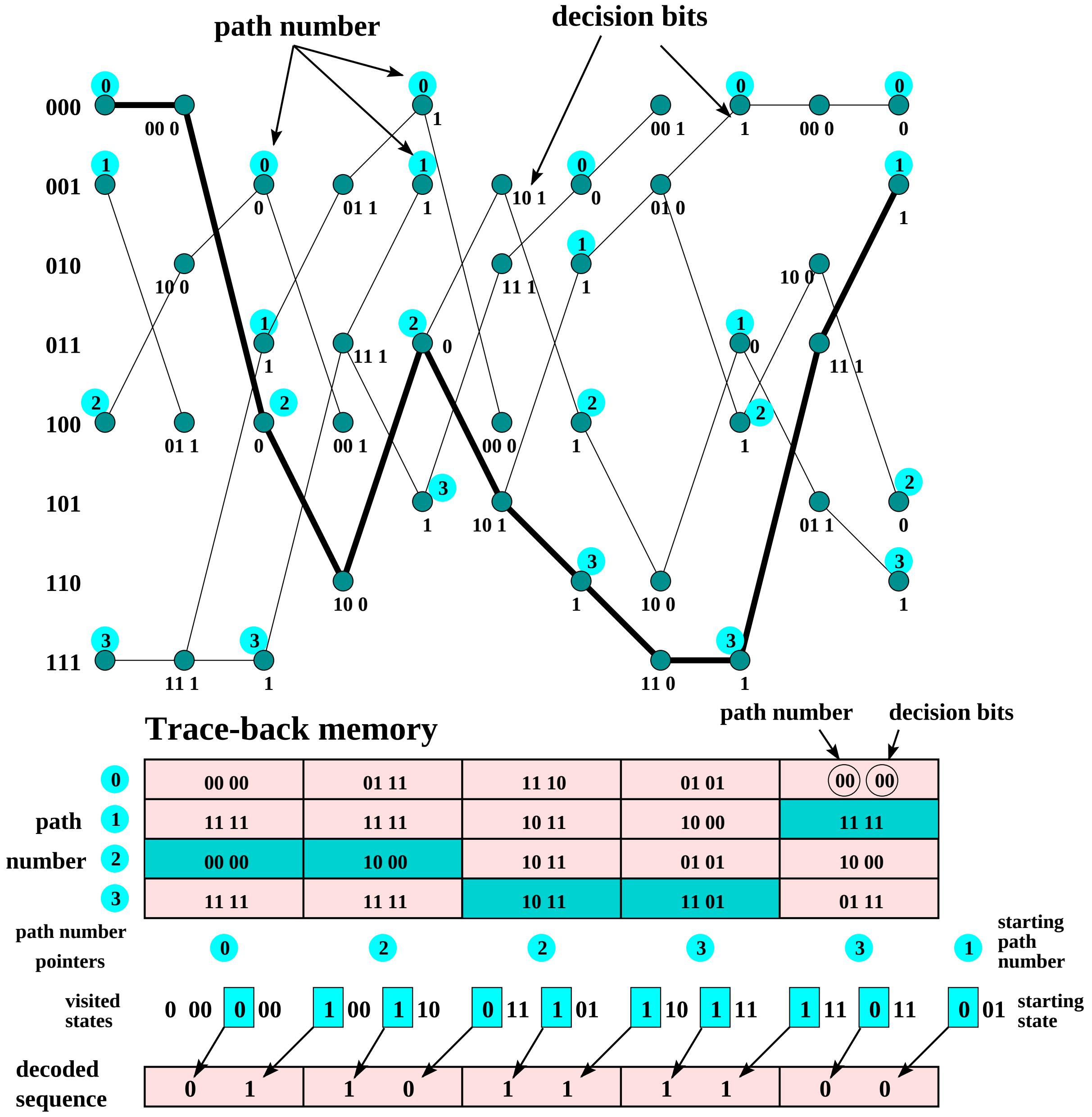
In the previous section we showed how the trace-back algorithm could be adapted to account for the features of the M algorithm so that sequence decoding can be performed in a more efficient way than with the register-exchange-like approach. The main advantage of this adaptation is that all the hardware architectures that have been proposed in the literature to improve trace-back realizations of the Viterbi algorithm can be utilized for the M algorithm as well, provided that the information about the path number is considered (see appendix A or [31, 41, 85]). Other solutions such as hybrid architectures employing both the register exchange approach and the trace-back algorithm can also be used although these must suffer little modifications to work; yet, like in the Viterbi algorithm, these structures allow significant reductions in storage requirements and latency.

5.3.3.1 Block Trace-Back for the M Algorithm.

The first observation to point out is that the trace-back method presented in the previous section is already working in an hybrid fashion because each surviving path has a $K-1$-bit shift register indicating the state label associated to this path (see figure 5.23); that is, the trace-back technique can be though of as a special case of the hybrid architecture where the register-exchange-like structure has a $K-1$-bit depth.

To improve this hybrid structure, we can simply increase the length of the registers containing the state labels of each surviving path to a length l ($l \geq K-1$). The decision bits delivered by the sorting network are stored therein prior to being stored in the trace-back memory. Then, each l cycles, the contents of this register are transferred into the trace-back memory together with the path number corresponding to time instant $k-l$ since this path number serves to find the ancestor state l cycles before. Figure 5.25 shows an example of operation of this new method for an 8-state trellis with $M = 4$ surviving paths and a register exchange depth of $l = 2$. As indicated above, each 2 cycles, the path numbers are initialized and are stored in the trace-back memory together with the

2 decision bits generated in these 2 cycles. Then, the trace-back operation takes place in the same way as before, except that this time, 2 bits are decoded per trace-back step.



path number					
0	00 00	01 11	11 10	01 01	00 00
1	11 11	11 11	10 11	10 00	11 11
2	00 00	10 00	10 11	01 01	10 00
3	11 11	11 11	10 11	11 01	01 11
path number pointers	0	2	2	3	3

path number pointers	0	2	2	3	3	1 starting path number
visited states	0 00 0 00	1 00 1 10	0 11 1 01	1 10 1 11	1 11 0 11	0 01 starting state
decoded sequence	0 1	1 0	1 1	1 1	0 0	

Figure 5.25: Operation example of a block trace-back adapted to the M algorithm.

The advantages of this method with respect to the trace-back approach presented in the previous section is the number of bits decoded per trace-back step. In this new approach, l bits are decoded at each time instant. In addition, the storage requirement are reduced since the trace-back pointer depends only on the path number; then, the number of decision bits that have to be stored every l cycles is

$$M \cdot (\log_2 M + l) \tag{5.20}$$

as opposed to

$$l \cdot M \cdot (\log_2 M + 1) \tag{5.21}$$

for the trace-back algorithm. A saving of $l \cdot M$ bits is thus achieved.

5.3.3.2 Forward Trace-Back for the M algorithm.

Other adaptations of hybrid structures applied to the Viterbi algorithm are possible. For instance, we can adapt the Forward trace-back proposed in [21] and [22] in the following manner.

Since the path number of each surving path commands the TB operation, we can initialize every L cycles an array of $M \times \log_2 M$ registers with the path numbers coming from the sorting network. Then, in the intermediate cycles between k and $k + L$, the path numbers coming out of the sorting network are used only to drive the shuffling of the register array. Thus, at the end of the L-cycle period, all the rows in the array contain the path number of the surviving path corresponding to the time instant k. This path number is used as the initial pointer of the trace-back operation, avoiding the initial trace-back procedure performed in the trace-back methods presented above which is required to attain the convergence length L.

An example of this method is presented in figure 5.26 for the same 8-state trellis and $M = 4$ with a decoding depth of $L = 6$. At $k = 3$, the register is initialized with the values $0 \cdot M - 1$. Then, during the path metric updating, the path numbers generated in each cycle control the shuffle of the different path numbers stored in the register array. For instance, at time $k = 4$, the first path indicates that its ancestor is path number 10; hence, the contents of the third register in the register array is tranferred to the contents of the first register (10). The second path indicates that its ancestor is path 11; thus, the contents of the fourth register in the register array is transferred to the second register (11). The same occurs to the other paths and during L cycles. At the end of these L cycles, all the registers in the register array contain the path number visited by the surviving path L cycles before (path 10 in the example).

Notice that in order to decode the output sequence, the trellis states of the surviving paths at the moment of initialization of the register array must be known (seef figure 5.26). To this purpose, a register containing the state labels of the suviving paths is also initialized every L cycles. When the decoding depth is attained, the path number stored

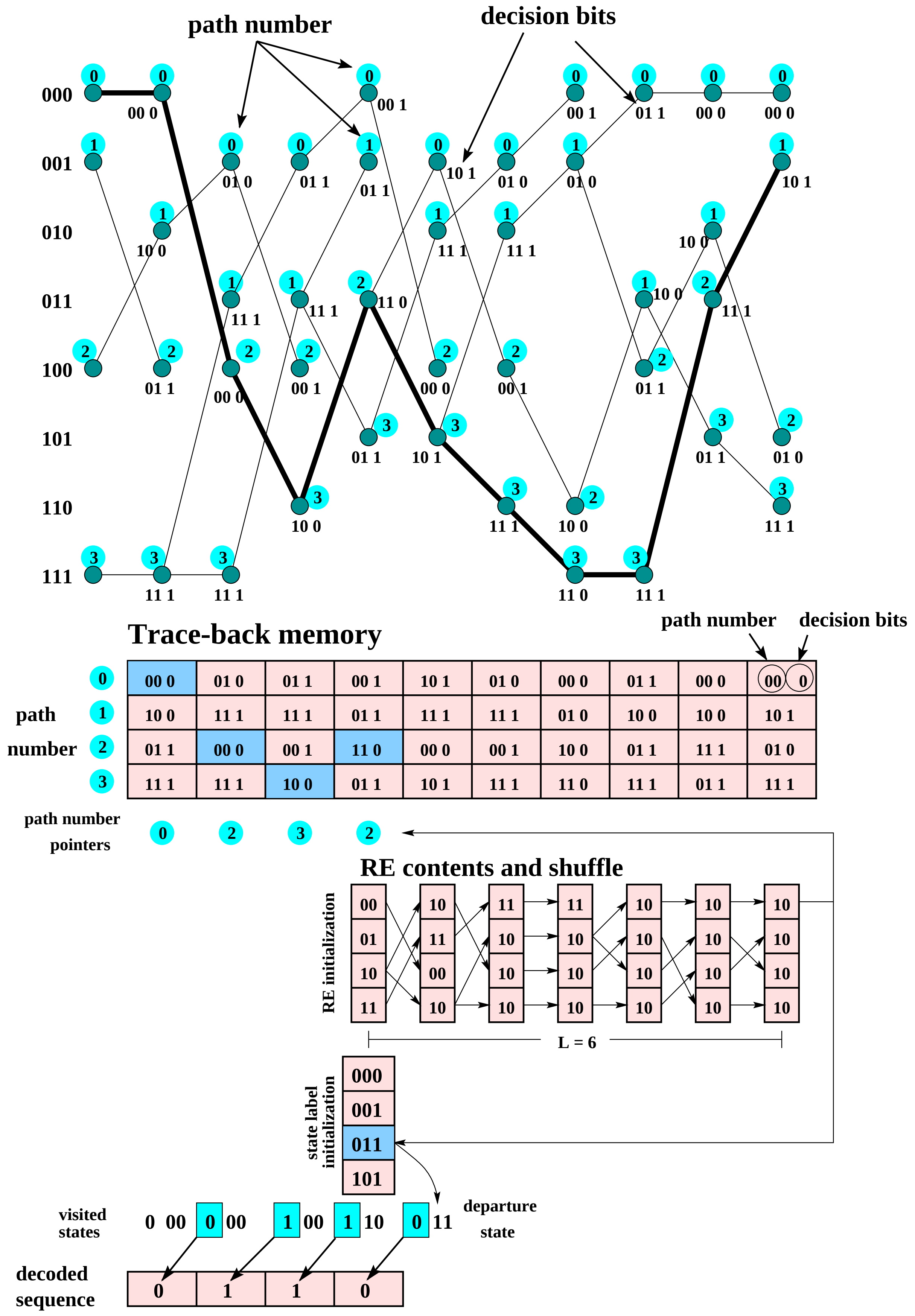


Figure 5.26: Forward Trace-back architecture for the M algorithm.

in the register exchange array selects the state label corresponding to that path number. This way, the decoded sequence can be found.

As we can see from all these TB examples, the trace-back procedure is an algorithm that can be very easily adapted to the M algorithm. Consequently, all the hardware architectures proposed to implement Viterbi decoders can be employed for the M algorithm. In addition, the main advantage with respect to the RE approach is that the complexity of the sorting network is significantly reduced because the surviving paths are reduced from $L+W$-bit words to $M \cdot (log_2 M + 1) + W$-bit words. This is very important because the flow of information in the sorting circuit is reduced, which reduces in turn the power consumption and the hardware complexity of the sorting network.

5.4 Hardware Comparison with the Viterbi Algorithm

We have seen that the survivor memory management of the Viterbi and M algorithms is very similar. Basically, this procedure consists in the storage and retrievement of the decision bits into and from a memory. Therefore, the hardware complexity of the survivor memory management is not very significant in terms of computations but rather in terms of storage. Table 5.5 shows the storage requirements of both algorithms for the RE procedure and the TB approach.

Table 5.5: Storage Requirements of the Viterbi and M algorithms.

	Viterbi algorithm	M algorithm
Register Exchange	$L \times 2^{K-1}$	$L \times M$
Trace Back	$L \times 2^{K-1}$	$L \times M(\log_M +1)$

Table 5.6 compares the computational load of the branch metric computation and path metric updating of both algorithms. As an illustrative example, consider the simulation results presented in the previous Chapter. We saw that for a trellis quantizer with a $K = 10$ constaint length, the performance of the M algorithm with $M = 8$ surviving paths was very close to that of the Viterbi algorithm. In this case, the overall computational requirements of the Viterbi algorithm are:

- 1024 branch metric computations per trellis stage
- 512 ACS operations (1024 additions + 512 comparisons + 512-address memory)
- Storage for L decision vectors of 512 bits each

On the other hand, the hardware requirements of the M algorithm are:

- 16 branch metric computations per trellis stage
- 65 comparisons-exchange operations
- Storage for L decision vectors of 32 bits each

Table 5.6: Viterbi and M Algorithm Computational Load for Branch Metric Computation and Path Metric Updating.

	Viterbi algorithm	M algorithm
Branch Metric	2^K	$2M$
Path Metric Updating	2^{K-1}	$3\left\{\frac{M}{4}\left[(\log_2 M)^2 - \log_2 M + 4\right] - 1\right\} + M$

Using the same complexity model of the previous chapter, namely

1 ACS operation = 3 additions

1 comparison-exchange = 2 additions

the Viterbi algorithm needs 1536 additions per processing cycle to update the path metrics while the M algorithm only needs 130.

A final comment is in order. It must be noted that in spite of the larger compuational load and storage requirements, the Viterbi algorithm is still potentially more efficient than the M algorithm. This is because the critical path in the Viterbi algorithm is given by the ACS unit only, whereas the M algorithm needs to perform sorting procedures to find the best paths. Hence, as indicated above, the M algorithm could be more attractive in applications where the number of surviving paths to retain is much smaller than the number of trellis states

5.5 Hardware Realization of the M Algorithm

A hardware architecture for the M algorithm was implemented on a FPGA circuit as part of a 4-month-last year project in electronics engineering [25]. The aim of this implementation was to estimate the hardware complexity and operation speed of the path metric updating and the survivor memory management blocks. The circuit was designed for a $K = 10$ trellis constraint length and $M = 8$ surviving paths. No specific application was intended for this architecture design; as a result, the branch metric unit was not taken into account. It was only assumed that this unit delivered two branch metrics per clock cycle. The computation precision was assumed at 10 bits for the branch metrics and 14 bits for the path metrics.

5.5.1 Sorting Architecture

The serial nature of the branch metric computation suggested a serial implementation of the sorting circuit. In this implemetation of the M algorithm, a parallel insertion circuit was selected. The features of this sorting circuit were described in section B.2. Remember that in this circuit M comparison elements are required and the sorting operation is performed in M clock cycles. Eventhough this method is not the most efficient serial sorting algorithm, its main advantage is that the path merging rejection and the sorting operation can be done at the same time, making this algorithm a very attractive solution for path metric updating.

The sorting architecture is the same as the one presented in figure B.11. Only an additional state comparator was required to account for the path merging rejection. Addtional control signals were also needed to manage the cases where path merging occurs. These control signals consisted in enabling or disabling the shifts of the inserted elements when two path metrics had the same state label.

The *mutually independence of extended sets* property was used to divide the path metrics selection into two sorting architectures of M items each. This way, the latency of the parallel insertion is reduced from 16 clock cycles to only 8. In addition, since this architecture is serial and not all the path metrics of the surviving paths are required at the same time to carry out the sorting operation, an optimization in the hardware complexity of the bitonic merging layer was done; only one comparison element was used to implement the merging operation of the two M-item lists. In order for the merging circuit to do this, the parallel insertion architectures were designed so as to deliver the sorting lists serailly, as described in figure 5.27. Finally, during M clock cycles the M surviving paths are selected. Notice that in this new structure, the bitonic merge and the sorting of new path metrics is performed simultaneously. As a result, an addtional control scheme was needed to distinguish between the current sorted lists and the one that is being created.

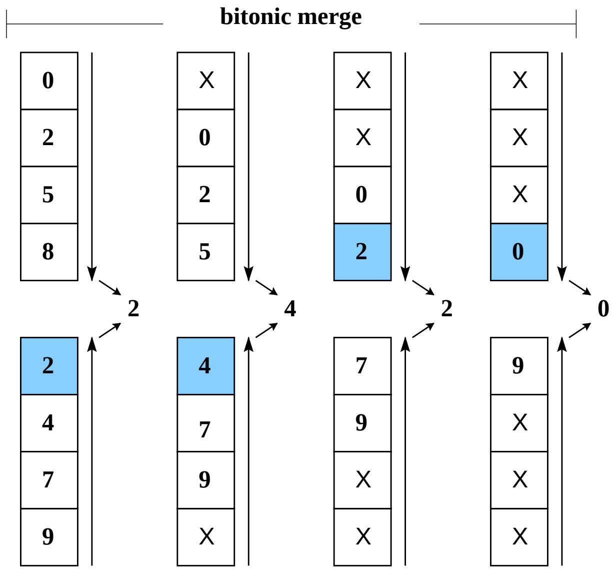


Figure 5.27: Simultaneous parallel insertion and bitonic merging of path metrics.

5.5.2 Survivor Memory Managements Architecture

The survivor memory management block was implemented with the trace-back algorithm. Due to the serial sorting of the path metrics, only one pointer was needed to perform the trace-back operation.

The trace-back memory was implemented with M FIFOs. For this reason, the surviving paths stored in each FIFO needed to be re-circulated during $L-1$ cycles before starting the trace-back operation. In this way, the first decision vectors are stored as if they were the last inputs to the FIFO and then, the trace-back operation can be done as usual.

5.5.3 Logic Synthesis Results

The architecture was synthesized on Altera's Flex10kE FPGA (EPF10K100EFC484-1). The number of logic cells and registers employed was 2485 and 1083 respectively, and the clock rate achieved by the circuit was 43 Mhz (3 Mbits/s).

5.6 Conclusion

In this chapter we have presented new architectures for the M algorithm. As in the case of Viterbi decoders, the hardware design was divided into two main blocks, as opposed to the conventional approach where only register-exchange-like architectures are possible.

We proposed new methods for parallel sorting networks which exploit the algorithmic properties of the M algorithm to reduce its hardware requirements and latency. Reductions of up to 50%, as compared to previous reported work were achieved.

Regarding the survivor memory management, new trace-back techniques adapted to the M algorithm were proposed. These ideas were borrowed from hardware structures for the Viterbi algorithm. In addition, to further improve the efficiency of the architecture in both hardware requirements and latency, hybrid approaches which employ the register exchange approach and the trace-back algorithm were developed. These structures are also based on hybrid implementations of the Viterbi algorithm. It must be pointed out that with the adaptation proposed in this thesis, namely the use of a pointer generated during the path extension and which indicates the position of the surviving path in the trace-back memory, all the trace-back and hybrid structures that have been proposed for the implementation of Viterbi decoders can be straightforwardly applied for the implementation of the M algorithm.

Chapter 6

Joint Optimization of a Trellis Source and Convolutional Channel Coders with Soft Source Decoding

The work presented so far has consisted in the study of a joint source and channel trellis coding technique constituted by a robust trellis coder-decoder pair. This study has been done from both a theoretical and architectural points of view. We showed that the tradeoff performance-hardware complexity is quite good, as compared to the so called “tandem” system, specially for degraded conditions of the transmission channel. In addition, we showed that further reduction in hardware complexity was achievable by using suboptimum trellis search algorithms, namely the M algorithm, with negligible performance degradation. Nevertheless, as indicated in [91], the use of a binary symmetric channel models to measure the effects of the transmission channel on the communication system is not a realistic approach since channel parameters such as bandwidth expansion, modulation, etc. are not considered.

This Chapter is devoted to the study of a joint source and channel coding technique consisting in the joint optimizaton of a trellis source coder and a convolutional code for transmissions over the AWGN channel. In this way, the transmission channel can be treated in a more realistic way. In addition to the joint optimization, at the receiver end, the same idea that was used for the robust quantization process consisting in finding a “weighted centroid” of all the possible reproduction codewords is employed. To this purpose, the BCJR algorithm [12] is used to provide the channel estimation necessary to perform this “soft source decoding”. In this manner, the distortion introduced by channel errors during the reconstruction of the source can be further reduced.

Other works with similar underlying ideas have also been reported in the litterature where a Trellis Coded Quantizer and a Trellis Coded Modulator are jointly optmized [42, 119, 15]. It must be pointed out that a very similar work was independently developed in [28] for the case of a TCQ encoder jointly optimized with a TCM coder employing the

BCJR algorithm to deliver channel soft information that is used during source decoding.

The outline of the Chapter is the following. In section 6.1, the complete system is described. Then, previous work on the joint optimization of trellis source codes and channel coding systems is presented in section 6.2. Section 6.3 presents the analytical computation of the pairwise error probability needed for the codebook design and the encoding processes. This error probability is estimated with a modified transfer function of the convolutional code. This way, no extensive monte-carlo simulations are required as is the case of previous reported work. Discussions about the codebook design procedure performed with the extension algorithm which must take into account the convolutional code's error probability are given in section 6.4. Section 6.5 elucidates the use of the MAP algorithm for "soft source decoding". Finally simulation results with benchmarking sources are given in section 6.6.

6.1 System Description

The joint source-channel coder that we are going to study throughout this Chapter is shown in figure 6.1. It consists of a robust trellis quantizer and a convolutional coder at the transmitter end and the BCJR decoder performing joint source-channel decoding at the receiver end. As before, the goal of this system is not to correct transmission errors but to minimize their effects on the decoded source.

The *robust trellis quantization* block in the figure is the same quantizer that we have been studying and whose purpose is to minimize the expectation of the distortion between the original source $\mathbf{x}$ and the one decoded at the receiver end $\hat{\mathbf{x}}$. The same branch metric as the one described in Chapter 2 is employed. Remember that this distortion measure is defined as

$$\begin{aligned} d(x_k, y_j) &= E\{(x_k - y_i)^2 | y_j\} \qquad i = 0...2^K - 1 \\ &= \sum_{i=0}^{2^K-1} (x_k - y_i)^2 \; Pr(y_i|y_j) \end{aligned} \tag{6.1}$$

The output binary sequence $\mathbf{u}$ delivered by the trellis quantizer is now encoded by the *convolutional coder* so as to produce the channel sequence $\mathbf{v}$ that will be transmitted through the AWGN channel. It must be pointed out that both coders employ the same trellis, as indicated in the figure, so that each trellis branch is labeled with a unique reproduction codeword-channel symbol pair. This way of labeling the trellis branches is very usefull to simplify the codebook design operation and the soft source decoding.

During the transmission of the channel symbols v_k through the AWGN channel, these are corrupted such that the received signal corresponding to channel symbol v_k is

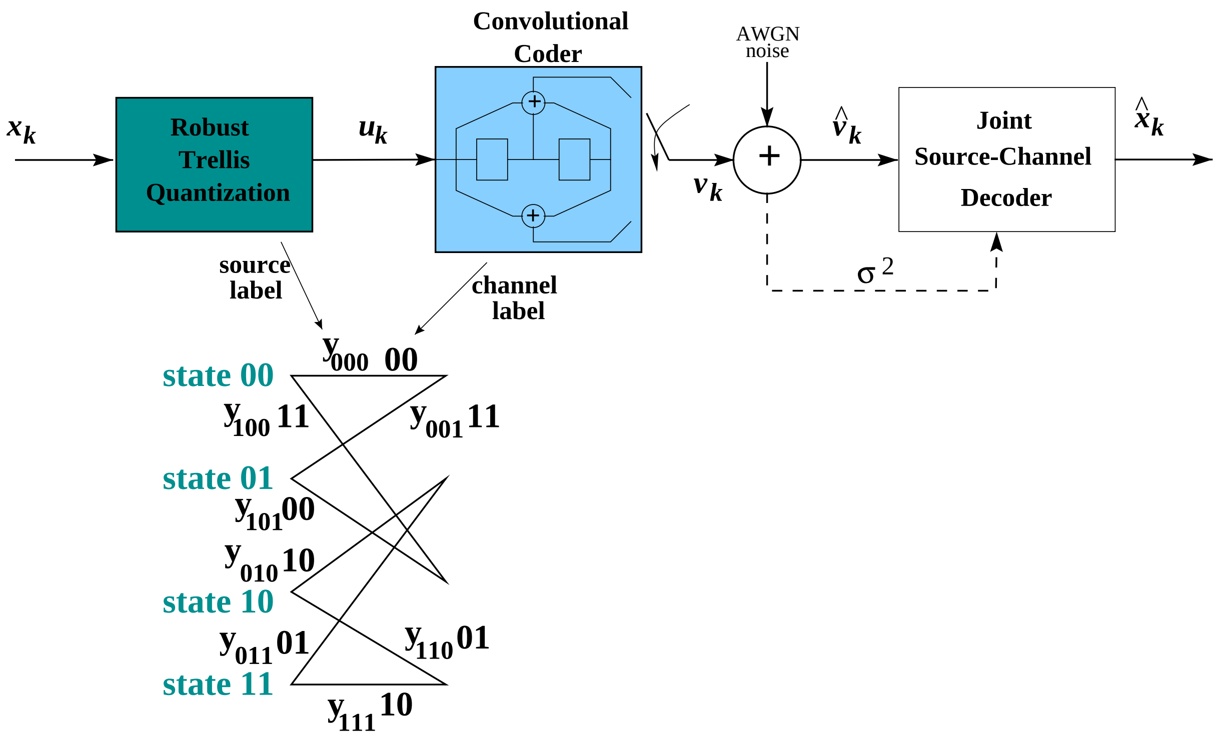


Figure 6.1: Joint Source-Channel Trellis Coding Scheme.

$$\hat{v}_k = v_k + w_k \tag{6.2}$$

where w_k is a zero-mean gaussian random variable. At the receiver end, the received sequence $\hat{\mathbf{v}}$ is decoded by the *joint source-channel decoder* block which consists of the MAP decoder performing soft source decoding, that is, the soft information computed by the MAP algorithm is used to compute a reproduction sample $\hat{x}_k$ that attempts to minimize the mean square error according to the channel distribution. The idea is to use the BCJR algorithm for Maximum A Posteriori decoding, that is, to obtain the a posteriori probabilities (APP) of the trellis branches, and in this way, the weighted centroid of all the reproduction codewords can be estimated and used as the decoded symbol $\hat{x}_k$. Notice that the BCJR algorithm is not employed to estimate the bit error probability since, as indicated earlier, we are not interested in correcting transmission errors but rather to decrease the distortion that they might introduced to the source decoding operation.

6.2 Previous Work

As indicated at the beginning of this Chapter, previous work employing similar ideas to the joint optimization of trellis source codes and channel coding techniques have already been reported. In [42], Fischer and Marcellin present a joint source-channel technique consisting in the combination of trellis-coded quantization with trellis-coded modulation. They use the same trellises for both coders and a consistent labeling between quantization levels and modulation symbols. This way, the euclidean distance in the channel is made commensurate with the quantization noise in the source encoder; as a result, transmission errors of small euclidean distance cause small additional distortion to the decoded source. Nevertheless, the results presented in [42] are for separately designed TCQ and TCM coders, resulting in a large performance degradation as the channel SNR decreases. To avoid this dramatic drop of performance, Wang and Fischer proposed the joint optimization of a TCQ-TCM system with successfull results [119]. The codebook design algorithm is the same as the one presented in Chapter 2, but modified to the TCQ case. In addition, since the computation of the error probability needed to compute the expectation of the distortion between the input and decoded symbols is TCM dependent, they conclude that an analytical computation of this error probability is quite complex so that a numerical method based on quasi-newton optimization is used to design the TCM system. As a consequence, the final codebook design algorithm of this system is quite cumbersome since two training sequences are required, one for the TCQ coder and the other for the TCM system, and a montecarlo simulation consisting in the iterative coding and decoding of source and channel sequences is performed.

In [15], the same approach as in [119] is used but modified to the joint optimization of a TCQ and a convolutional coder. The design algorithm also implies montecarlo simulations which include TCQ encoding, channel encoding, insertion of channel errors, soft decision channel decoding with the Viterbi algorithm and TCQ decoding. Recently, Chei and Ho presented the design of an optimal soft decoding for combined TCQ/TCM over rayleigh fading channels [28]. The main difference with respect to previous reported work is the utilization of the BCJR algorithm as a minimum mean-square error decoder. The results presented in this work outperform those of the previous reported systems because of this new idea of using the BCJR algorithm as source decoder. Nevertheless, this system also employs a codebook design algorithm which iterates source coding, channel decoding with estimation of the channel statistical distribution and source decoding. This is due to the fact that analytical computations of the channel error probability for TCM systems are a quite difficult task.

In the next section we will show that it is indeed possible to compute the pairwise error probability $Pr(y_i|y_j)$ in an analytical way by considering the reproduction codewords y_i in the derivation of the transfer function of the convolutional code.

6.3 The Pairwise Error Probability

The codebook design procedure and the source quantization require the knowledge of the pairwise error probability $Pr(y_i|y_j)$. We have seen that when the communication system consists only of the robust trellis source coder and the binary symmetric channel, these error probabilities are expressed as

$$P(y_i|y_j) = p^{d_H(i,j)} \cdot (1-p)^{K-d_H(i,j)} \tag{6.3}$$

However, in the system of figure 6.1, the error probability depends now on the channel code and its exact derivation is quite difficult, specially for low channel SNR. This is the reason why the usual approaches to deal with this problem are based on monte-carlo simulations. Nevertheless, in the same manner as the computation of upper bounds on the bit error probability of convolutional codes [69, 89, 116], a tight upper bound on the pairwise error probability $P(y_i|y_j)$ can be obtained. Certainly, since the trellises for the source and channel coders are the same and there is a one-to-one mapping between a reproduction codeword and channel symbol, the probability $P(y_i|y_j)$ is equal to the probability that the Viterbi algorithm decodes the channel symbol corresponding to branch i instead of decoding the channel symbol corresponding to branch j. This probability depends on the code generator polynomial, i.e. the weight distribution of all the paths in the trellis.

6.3.1 Analytical Derivation of the Pairwise Error Probability

Assuming without loss of generality that the all-zero channel sequence is transmitted, it can be shown that the first-event error probability, which is defined as the probability that another path in the trellis merging for the first time with the all-zero path at a given node has a better metric than the all-zero metric, is upper bounded in the form (see [89])

$$P_e \leq \sum_{d=d_{free}}^{\infty} a(d)\ P_b(d) \tag{6.4}$$

where d_{free} is the free distance of the convolutional code, coefficient $a(d)$ denotes the number of trellis paths of distance d from the all-zero path that merge with the all-zero path for the first time and $P_b(d)$ is the probability of error in a pairwise comparison of two paths that differ in d bits and defined as

$$P_b(d) = Q\left(\sqrt{2\frac{E_b}{N_o}R_c\ d}\right) \tag{6.5}$$

where R_c is the channel code rate and $\frac{E_b}{N_o}$ is the received SNR per bit. Carrying out the same reasoning, it can be shown that the pairwise error probability $Pr(y_i|y_0)$ can be upper bounded in the form

$$P(y_i|y_0) \leq \sum_{d=d_{free}}^{\infty} a_i(d)\ P_b(d) \tag{6.6}$$

where coefficient $a_i(d)$ indicates the number of trellis paths of distance d from the all-zero path that diverge from the all-zero path, pass by the trellis branch whose reproduction codeword is y_i and remerge into the all-zero path for the first time. Since the channel code is linear, equation (6.6) which was obtained for the all-zero path can be generalized to all the trellis branches and pairwise error probabilities in the form

$$P(y_i|y_j) = P(y_{i\oplus j}|y_0) \tag{6.7}$$

6.3.2 Derivation of Coefficient $a_i(d)$ from the Transfer Function a the Convolutional Code

The coefficients $a_i(d)$ can be obtained from the state transition diagram of the convolutional code in the same manner as the distance properties and the error rate performance of the code are derived. To do so, the transfer function $T(D)$ of the convolutional code is obtained [89]. Remember that variable D serves to have an indication of the Hamming distance of the sequence of output bits corresponding to a given path in the trellis with respect to the sequence of output bits corresponding to the all-zero path. In our case, in order to obtain coefficient $a_i(d)$, in addition to variable D, a label y_j $(j = 0 \cdots 2^K - 1))$ corresponding to the reproduction codeword associated to branch j is added to the state transition diagram of the code. This way, a modified transfer function $T(D, y_j)$ is derived which depends on D and y_j. Coefficient $a_i(d)$ can then be obtained by the coefficients of the polynomial $P_i(D)$ which is defined as

$$P_i(D) = \left.\frac{\partial T(D, y_j)}{\partial y_i}\right|_{y_j=1 \quad j\neq i} \tag{6.8}$$

Let us explain the derivation of coefficient $a_i(d)$ in a more detailed way with an illustrative example. Consider the transition diagram shown in figure 6.2(a) corresponding to the joint source-channel trellis diagram presented in figure 6.1. Each branch in the transition diagram is labeled with its corresponding reproduction codeword y_j and with variable D

whose exponent denotes the Hamming distance between the output bits of a given branch and the all-zero branch.

For ease of explanation, two additional dummy labels J and N are assigned to each branch in the following manner:

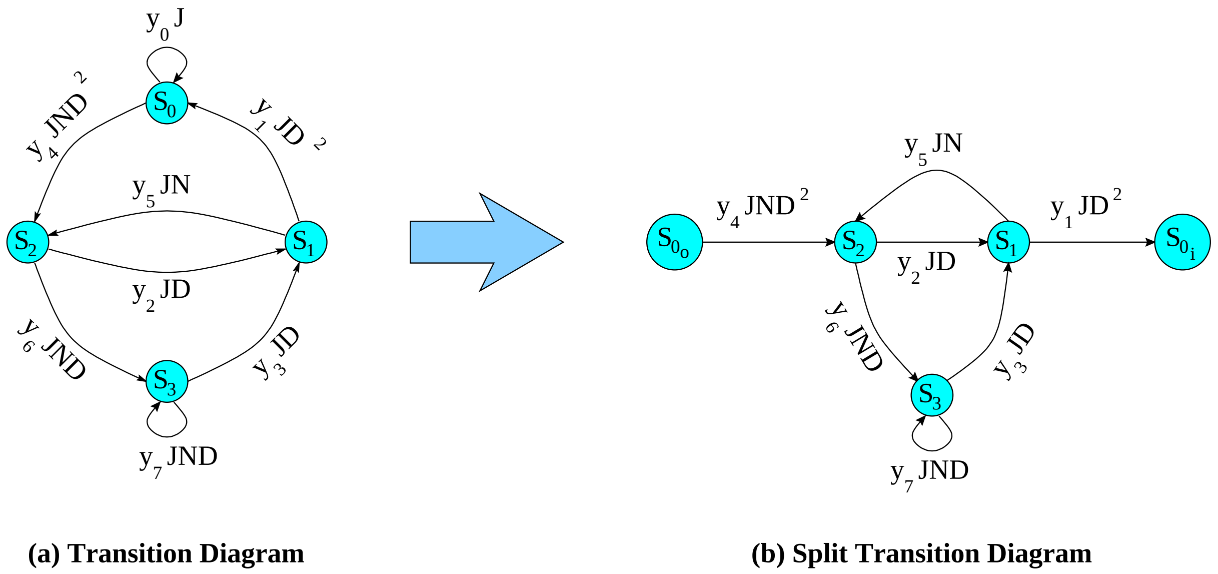


Figure 6.2: State transition diagram for a joint trellis source code and rate 1/2, K=3 convolutional code.

J is a counting variable which indicates the number of branches in a given path from the moment it diverges from the all-zero state to the moment it merges with the all-zero state

N is a variable assigned to those branches generated by the input bit 1, that is, this variable keeps the score of the number of ones that enter the shift register of the convolutional code

In order to obtain the transfer function of the convolutional code, the zero state in the state transition diagram is split into an input and output node, as illustrated in figure 6.2(b). From this split diagram we can write the four state equations as

$$\begin{aligned} S_2 &= JND^2y_4\, S_{0_o} \;+\; JNy_5\, S_1 \\ S_3 &= JNDy_7\, S_3 \;+\; JNDy_6\, S_2 \\ S_1 &= JDy_2\, S_2 \;+\; JDy_3\, S_3 \\ S_{0_i} &= JD^2y_1\, S_1 \end{aligned} \tag{6.9}$$

The transfer function of the code, defined as $T(D,N,J,y_j) = \frac{S_{0_i}}{S_{0_o}}$, is obtained by solving these state equations. We thus have

$$\begin{aligned}
T(D,N,J,y_j) &= \frac{J^3ND^5y_1y_2y_4 \;+\; J^4N^2D^6y_1y_3y_4y_6 \;-\; J^4N^2D^6y_1y_2y_4y_7}{1 \;-\; JNDy_7 \;-\; J^2NDy_2y_5 \;-\; J^3N^2D^2y_3y_5y_6 \;+\; J^3N^2D^2y_2y_5y_6} \\
&= \overbrace{J^3ND^5y_4y_2y_1}^{\text{path 1}} + \overbrace{J^4N^2D^6y_4y_6y_3y_1}^{\text{path 2}} + \overbrace{J^5N^2D^6y_4y_2^2y_5y_1}^{\text{path 3}} \\
&\quad + \overbrace{J^5N^3D^7y_4y_6y_7y_3y_1}^{\text{path 4}} + \overbrace{2J^6N^3D^7y_4y_6y_5y_3y_2y_1}^{\text{paths 5 and 6}} + \overbrace{J^6N^4D^8y_4y_6y_7^2y_3y_1}^{\text{path 7}} \\
&\quad + \overbrace{J^7N^3D^7y_4y_2^3y_5^2y_1}^{\text{path 8}} + \cdots
\end{aligned} \tag{6.10}$$

This polynomial expression indicates all the paths in the trellis that diverge from and merge with the all-zero state for the first time, and provides information concerning the Hamming weight of the input sequence (exponent of N), the Hamming weigth of the ouput sequence (exponent of D), the path length (exponent of J) and the reproduction codewords y_j visited by a given path. Figure 6.3 shows the first eight paths obtained by expression 6.10. We can see that the first path of equation 6.10 has a length of 3 input bits, the hamming weight of this 3-bit input sequence is 1, the hamming weight of the convolutional coded sequence is 5 and the reproduction codewords passed by this path are y_4, y_2 and y_1; the second path has a length of 4 input bits, the hamming weigth of the input and coded sequences is 2 and 6 repectively, and the reproduction codewords visited by this path are y_4, y_6, y_3 and y_1. Thus, this modified transfer function gives us, in addition to the distance properties of the convolutional code, the reproduction codewords visited by a given path.

Finally, the number of paths of hamming distance D passing by codeword y_i may be obtained by differentiating $T(D,N,J,y_j)$ with respect to y_i, according to equation 6.8. As an example, in order to know the number of paths passing by codeword y_2 in the trellis diagram of figure 6.3, equation 6.10 is differentiated with respect to y_2 and all the dummy variables J and N and the codewords y_j, where $j \neq 2$, are set to one. We thus obtain

$$\left.\frac{\partial T(D,N,J,y_j)}{\partial y_2}\right|_{N=J=y_j=1 \quad j\neq 2} = D^5 \;+\; 2D^6 \;+\; 5D^7 \;+\; \cdots \tag{6.11}$$

This expression indicates that there is one path passing by codeword y_2 with a Hamming distance of 5, there are 2 paths with Hamming distance 6, five paths with Hamming

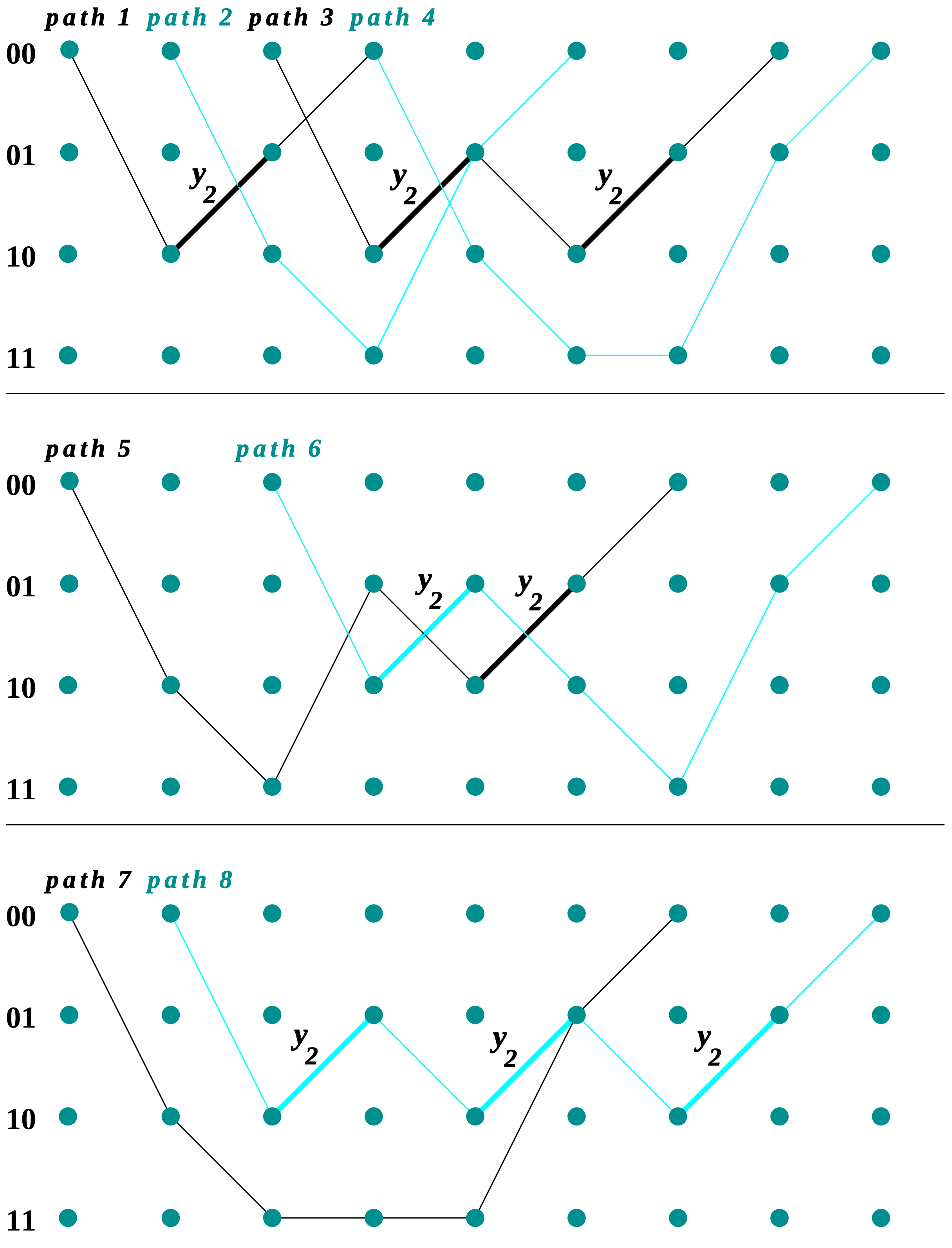


Figure 6.3: First paths of expresion 6.10

distance 7 and so on. These first eight paths correspond to paths 1, 3 (this path passes twice by codeword y_2), 5, 6 and 8 (this path passes 3 times by codeword y_2) as indicated in figure 6.3.

Since the convolutional code does not necessarily have a fixed length, it is not possible to determine the exact number of paths passing by a given branch in the trellis. In practice, we have obtained the polynomial $P_i(D)$ of each trellis branch i by restricting the trellis to a L-size frame of trellis stages.

Tables 6.1 and 6.2 show the computation results of the pairwise error probabilities corresponding to a R=1/2 rate, $K = 7$ convolutional code with polynomial generators $(133, 171)_{octal}$ and for channel SNR = 0, 1, 2 and 3 dB. The pairwise error probability ($P_{sim}(y_i|y_0)$) was obtained with a Monte-carlo simulation where an all-zero sequence of length 10^6 was encoded, transmitted and decoded with the Viterbi algorithm. The error probability ($P_{ub}(y_i|y_0)$) represents the upper bound obtained with equation (6.6). These error probabilities were normilized to 1 so as to obtain the pairwise error probabilities $P_{norm}(y_i|y_0)$. To this purpose, the term $P_{sim}(y_0|y_0)$ obtained by simulation is used according to

$$P_{norm}(y_j|y_0) = P_{ub}(y_j|y_0) \cdot \frac{1 - P_s(y_0|y_0)}{\sum_{m>0} P_{ub}(y_m|y_0)} \tag{6.12}$$

Table 6.1: Simulated and estimated pairwise error probabilites for SNR=0 and 1 dB

	SNR = 0 dB (10^{-4})					SNR = 1 dB (10^{-4})				
branch i	1	3	4	7	9	12	15	18	20	31
$P_{sim}(y_i|y_0)$	73.5	36.6	64.7	20.3	59.7	5.3	8.5	11.5	11.1	3.3
$P_{ub}(y_i|y_0)$	2861.3	2342.2	3918.6	1637.9	3805.1	32.1	56.8	76.2	74.9	29.2
$P_{norm}(y_i|y_0)$	38.1	31.2	52.2	21.7	50.6	4.9	8.7	11.7	11.5	4.4

Table 6.2: Simulated and estimated pairwise error probabilites for SNR=2 and 3 dB

	SNR = 2 dB (10^{-4})					SNR = 3 dB (10^{-4})				
branch i	36	48	63	76	88	96	100	112	120	127
$P_{sim}(y_i|y_0)$	0.97	0.05	0.06	0.3	0.43	0.04	0.0	0.05	0.04	0.1
$P_{ub}(y_i|y_0)$	1.6	0.05	0.2	0.4	0.5	0.7	0.07	1.3	1.3	1.6
$P_{norm}(y_i|y_0)$	1.01	0.03	0.11	0.2	0.3	0.0	0.0	0.0	0.0	0.04

As we can see from the tables, the matching between the simulated and estimated pairwise error probabilites is quite close; the differences may come from the insufficient number of trellis stages considered for the computation of polynomial $P_i(D)$. As a conclusion, from these estimated values of the pairwise error probabilities, quasi-optimum robust source quantization can be performed.

6.4 Codebook Design

In section 2.2.1, page 37, two codebook design algorithms were described, based on the extension algorithm. So far, the first method of the extension algorithm has been employed. Recall that in this method, the idea is to start from a $K-1$-constraint length quantizer with initial codebook C^0_{K-1} and run the codebook design algorithm to obtain an optimized codebook C^f_{K-1}; then, codebook C^f_{K-1} is extended to a K-constraint length quantizer by appending a register element at the LSB position. In order to obtain an initial codebook C^0_K for this new constraint length quantizer, codeword y_i, which is associated to branch i from codebook C^f_{K-1} is now associated to branches $2i$ and $2i+1$ ($i = 0 \cdots 2^{K-1}-1$) of the new codebook C^0_K. Then, the codebook design algorithm is used to obtain the final codebook C^f_K. This method is alternately applied from C^0_0 (a unique codeword) to C^f_K.

In the system proposed here where a convolutional code is added to the trellis quantizer, the extension algorithm has to be adapted because the pairwise error probabilities obtained in the previous section correspond to a fixed constraint length quantizer. Consequently, the pairwise error probabilities used in the optimization process of the 2^K-size codebook have to be reduced for the optimization of the 2^{K-1}-size codebook. To do so, a reduction of the trellis is done by using the expression

$$P^{K-1}(y_i|y_0) = P^K(y_{2i}|y_0) + P^K(y_{2i+1}|y_0) \tag{6.13}$$

that is, the probabilities corresponding to those branches having the same $K-1$ LSBs are added to form the pairwise error probability of the new $K-1$-bit label codewords. As a result, in order to optimize the reproduction codebook with the first method of the extension algorithm, a trellis reduction is first performed from 2^{K-1} states to 2^0 and then the trellis extension algorithm is used from 2^0 states to 2^{K-1}.

Another possibility consists in the use of the second method of the extension algorithm. This algorithm seems to be best suited since the constraint length remains constant. However, it was observed that the performance obtained with both extension algorithms were quite close.

6.5 The BCJR Algorithm as Soft Source Decoder

Conventionally, decoding a communcation system employing a trellis source coder and a convolutional code implies the concatenation of the Viterbi decoder for channel decoding and the K-length shift register addressing a 2^K-word Look-Up Table for source decoding. The Viterbi algorithm decodes the binary sequence $\hat{\mathbf{u}}$ from the channel sequence $\hat{\mathbf{v}}$ that was corrupted by the transmission channel (see figure 6.1). Then, the source decoder decodes the source sequence $\hat{\mathbf{x}} = \hat{\mathbf{y}}_\mathbf{i}$ from the decoded binary sequence $\hat{\mathbf{u}}$. As we can see, these two operations are performed separately and the source decoder has no knowledge of the distortion caused by the transmission channel. In order for the source decoder to take advantage of the channel characteristics during the source reconstruction, the source and channel decoding operations can be merged into a single decoding operation so as to obtain a soft source decoder which minimizes the expectation of the distortion introduced by the transmission channel. This is justified by observing that the mean square error between the original source $\mathbf{x}$ and the one reconstructed at the receiver end $\hat{\mathbf{x}}$ can be bounded with

$$D \leq \sum_{k=0}^{L_{TS}-1} (x_k - y_{i(k)})^2 + \sum_{k=0}^{L_{TS}-1} (y_{i(k)} - \hat{x}_k)^2 \tag{6.14}$$

where the first term corresponds to the distortion introduced by the quantization process at the encoder end and the second term indicates the distortion introduced by the transmission errors. Decreasing the overall distortion at the decoder end can only be accomplished with the second term since the distortion introduced by the quantization process is due to a lossy compression operation, and minimizing its distortion is not possible.

In the same manner as the reproduction codebook is updated during the codebook design algorithm described in Chapter 2, the expectation of the second term of equation 6.14 can be minimized by using the weighted centroid (see also [121])

$$\hat{x}_k = \frac{1}{\sum_{i=0}^{2^K-1} P_k(y_i)} \sum_{i=0}^{2^K-1} y_i P_k(y_i) \tag{6.15}$$

where $P_k(y_i)$ is the probability of decoding source codeword y_i, given all the received channel sequence $\hat{\mathbf{v}}$.

The channel information, i.e. $P_k(y_i)$, required for the estimation of the reproduction sample $\hat{x}$ can be obtained in an aposteriori way by the BCJR algorithm. According to [12], the objective of the decoder is to examine the received channel sequence $\hat{\mathbf{v}}$ and estimate the a posteriori probabilities of the trellis states and branches. Putting the probability $P_k(y_i)$ in terms of the notation employed in [12], we have

$$P_k(y_i) = \sigma_k(m', m) = P(S_{k-1} = m'; S_k = m; \hat{v}_1^{\tau}) \tag{6.16}$$

where m' and m are the previous and current states, respectively, of the branch labeled with the reproduction codeword y_i and $\hat{v}_1^{\tau}$ is a frame of the received channel symbols.

It was shown that $\sigma_k(m', m)$ could be expressed in the form

$$\sigma_k(m', m) = \alpha_{k-1}(m') \cdot \gamma_k(m', m) \cdot \beta_k(m) \tag{6.17}$$

where $\gamma_k(m', m)$, $\alpha_k(m)$ and $\beta_k(m)$ are, respectively, the branch probability and the forward and backward recursions. These three parameters are defined as

$$\alpha_k(m) = \sum_{m'=0}^{2^{K-1}-1} P(S_{k-1} = m'; S_k = m; \hat{v}_1^{k}) \tag{6.18}$$

$$\alpha_k(m) = \sum_{m'} \alpha_{k-1}(m') \cdot \gamma_k(m', m) \tag{6.19}$$

$$\beta_k(m) = \sum_{m'=0}^{2^{K-1}-1} P(S_{k+1} = m'; \hat{v}_{k+1}^{\tau} | S_k = m) \tag{6.20}$$

$$\beta_k(m) = \sum_{m'} \beta_{k+1}(m') \cdot \gamma_{k+1}(m, m') \tag{6.21}$$

$$\gamma_k(m', m) = \exp\left\{ \frac{-1}{2\sigma^2} \parallel \hat{v}_k - c_k(m', m) \parallel^2 \right\} \tag{6.22}$$

where σ^2 is the variance of the channel noise, $\hat{v}_k$ is the received channel symbol and $c_k(m', m)$ is the expected channel symbol associated to the trellis branch whose previous and current state are m' and m, respectively[1]. Consequently, equation 6.15 can be rewritten in the form

$$\hat{x}_k = \frac{1}{\sum_{m'} \sum_m \sigma_k(m', m)} \sum_{m'} \sum_m y_{(m', m)} \sigma_k(m', m) \tag{6.23}$$

where $y_{(m', m)}$ is the reproduction codeword associated to the branch that links states m' and m. Using this expression for decoding the source, the distortion introduced by the transmission channel can be minimized.

[1]For more details on the BCJR algorithm, the reader is referred to [12] and [44]

6.6 Simulation Results

The proposed system was simulated with a gaussian and first-order gauss-markov sources transmitted through an AWGN channel. In addition, the system was compared to a tandem system consisting of a trellis quantizer concatenated to a convolutional code.

The two systems used different decoding strategies. One strategy consisted in joint MAP decoding, as explained in the previous section; the other one consisted in a Viterbi decoder concatenated to the trellis decoder discussed in Chapter 2. We thus have four different system combinations:

1. Tandem system with joint MAP decoding. This will be referred to as Tandem-MAP
2. Tandem system with concatenated Viterbi and shift register source decoder. This will be called Tandem-vit
3. The proposed system, that is, JSCTC with a convolutional code and MAP decoding (JSCTC-MAP)
4. The proposed system but instead of the MAP decoder, a concatenation of a Viterbi decoder and the shift register source decoder is used (JSCTC-vit)

The source rate R_s is one bit per sample and the channel coder consisted of a K=7, rate $R_c = 1/2$ convolutional code with generator polynomials $(133, 171)_{octal}$.

Figure 6.4 shows the performance of the four systems in the case of a gaussian source transmitted through a AWGN channel. We can point out from this figure that MAP decoding results in a very efficient way to reduce the distortion introduced by transmission errors (curve “Tandem-MAP”). Notice how, by using the a posteriori channel information, the decoder is able to improve the tandem system’s performance (curve “Tandem vit) for about 2.5 dB (at channel SNR=0 dB). On the other hand, by using the JSCTC algorithm for codebook design and source quantization, we can further improve the overall performance (curves “JSCTC-vit” and “JSCTC-MAP”) between 0.8 and 1 dB of SQR.

Figure 6.5 presents results obtained with a first order gauss-markov source. The same behaviour as in the above example can be observed: MAP decoding improves the tandem system’s performance, and with the addition of the JSCTC technique, the performance is further improved.

Finally, figures 6.6, 6.7 and 6.8 present Lenna images coded with the Tandem-map and JSCTC-MAP systems for transmissions at SNR=-3, -1 and 0 dB. Notice the improved performance of the proposed approach over the tandem system, specially for very noisy channels.

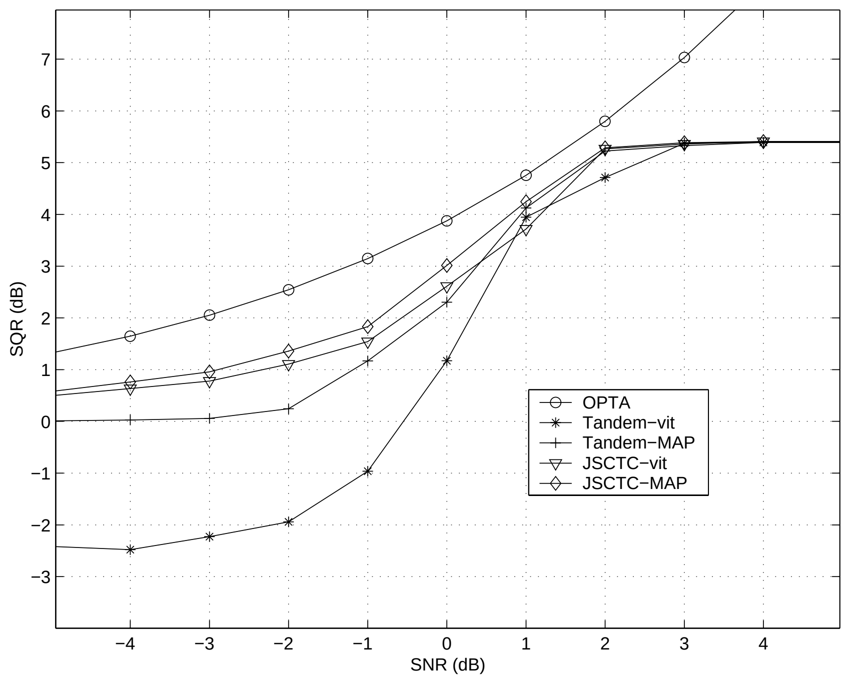


Figure 6.4: Performance comparison between the proposed system and a tandem scheme for a gaussian source and AWGN channel.

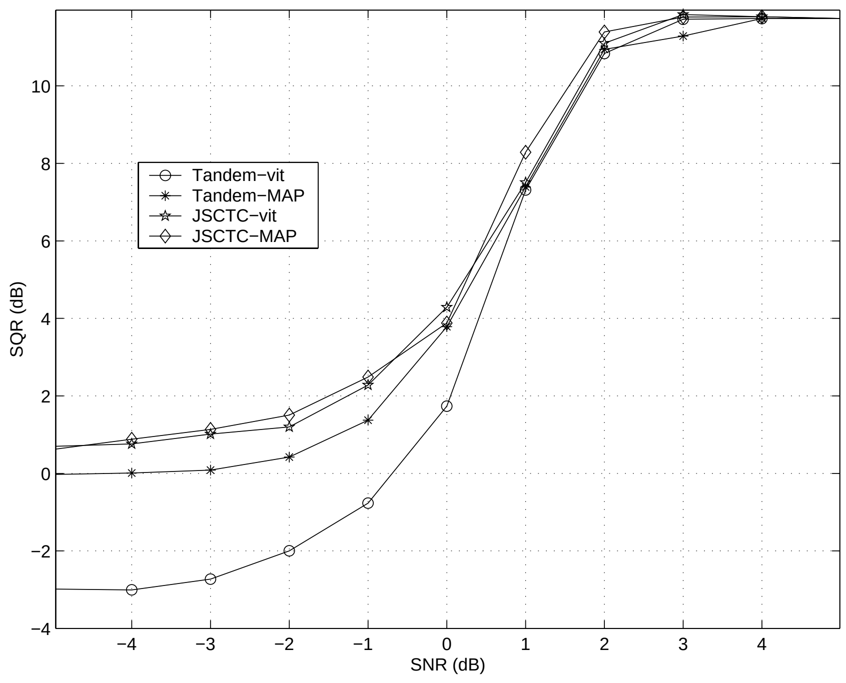
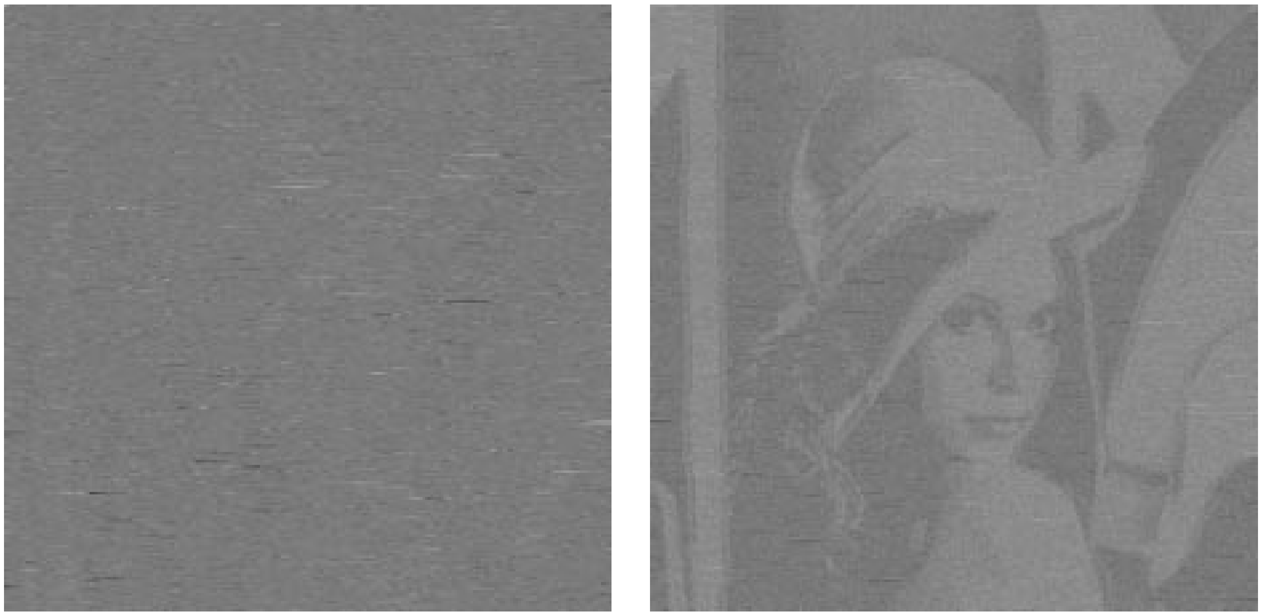
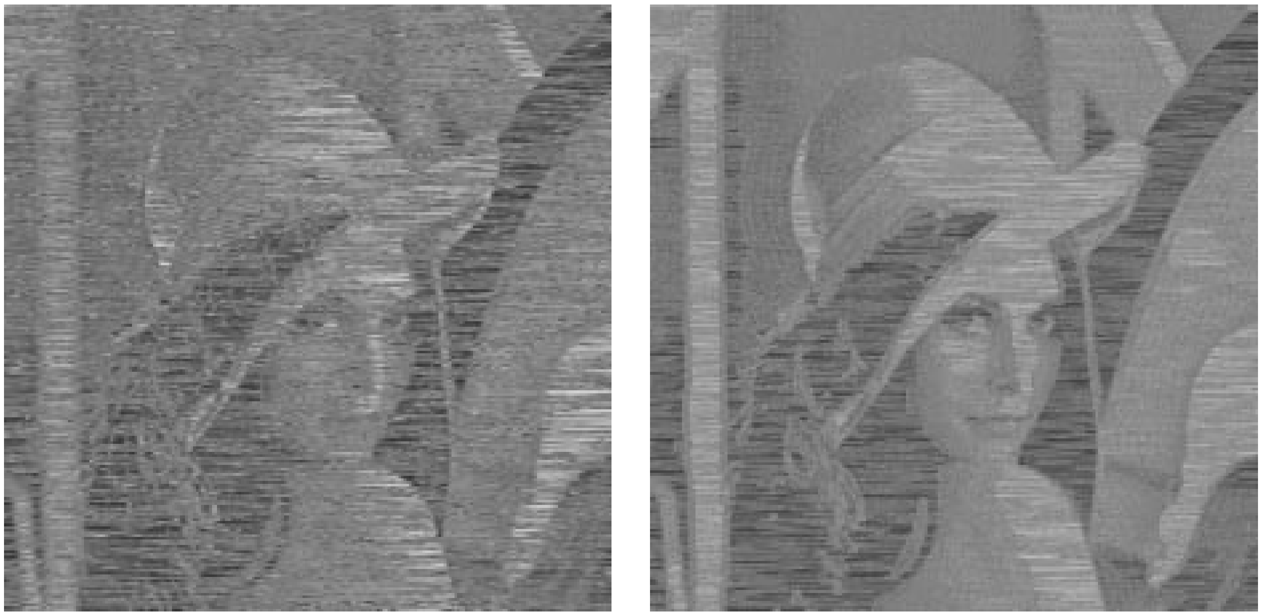


Figure 6.5: Performance comparison between the proposed system and a tandem scheme for a first order gauss-markov source and AWGN channel.



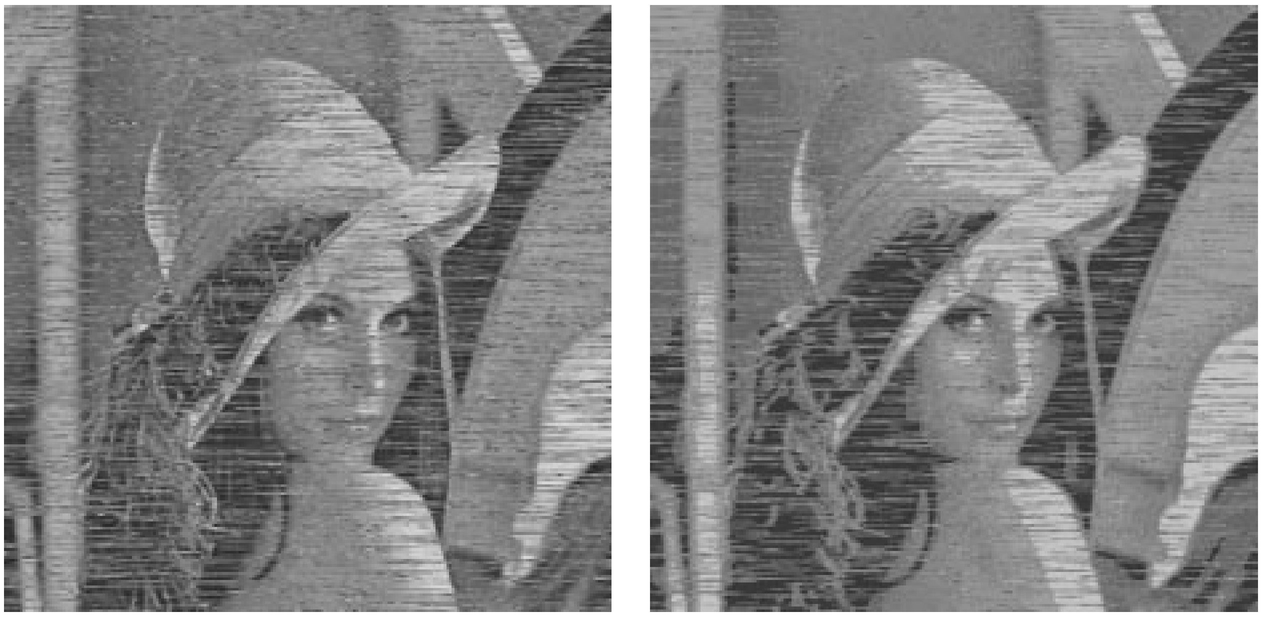
(a) Tandem-MAP; $K = 7$ *(b)* JSCTC-MAP; $K = 7$

Figure 6.6: Lenna images coded with JSCTC-MAP and Tandem-MAP systems at SNR=-3dB.



(a) Tandem-MAP; $K = 7$ *(b)* JSCTC-MAP; $K = 7$

Figure 6.7: Lenna images coded with JSCTC-MAP and Tandem-MAP systems at SNR=-1dB.



(a) Tandem-MAP; $K = 7$ *(b)* JSCTC-MAP; $K = 7$

Figure 6.8: Lenna images coded with JSCTC-MAP and Tandem-MAP systems at SNR=0dB.

6.7 Conclusion

A jointly optimized trellis source coder and convolutional code was presented. It was shown that the channel estimation for source encoding and codebook design can be tighly bounded with the transfer function of the convolutional code without the need of extensive monte-carlo simulations. On the other hand, the BCJR algorithm was modified to perform "soft source decoding" with the use of its a posteriori probability estimation. We saw, that by using MAP decoding, the performance of the tandem system can be largely improved. Finally, if the JSCTC approach is combined with this MAP decoding, further improvements can be obtained.

Conclusion and Perspectives

Conclusion

In this thesis, a first approach towards the hardware implementations of JSCC techniques was presented. In addition, new ideas for improving performance were presented. In the last decade, JSCC became a very important topic in Information Theory. Exciting recent developements have shown the potential advantages of JSCC over conventional communication systems where source and channel coders are designed separately. Nowadays, researches continue scanning the horizon for new ideas and solutions in order to approach Shannon's theoretical limits.

This thesis has served to demonstrate that JSCC may lead to the design of hardware architectures with a better trade off performance-computational load. In addition, we saw that the behaviour of these systems is "softer" than conventional tandem systems. By "softer" we mean systems that present a smooth performance degradation as the conditions of the transmission channel worsen. That is, these systems do not present the dramatic drop of performance presented in tandem systems when the transmission channel attains a certain threshold.

In a more specific way, we study a JSCTC technique [9] and designed its hardware architecture. This algorithm presents the following characteristics:

- a simultaneous compression and channel protection technique
- channel errors are not corrected, it is the impact of these errors in the decoded source which is minimized. As a result, channel coding for error correction is completely eliminated or very simple
- since channel coding may not be employed, the ressources that would have been used by this operation are now dedicated to the source coding operation. This way, a better performance on the decoded source can be obtained since more bits are used for its representation
- larger gains if correlated sources are coded, if the constraint length of the trellis is larger and when the transmission channel is very noisy

- when the transmission channel is noiseless, the performance of noiseless trellis quantization and JSCTC are identical.

In order to alleviate the hardware complexity of this algorithm, a suboptimum trellis search algorithm was used. Simulation results showed that this algorithm offers an excellent trade off hardware complexity-performance degradation. Then, new hardware architectures for the implementation of the M algorithm were proposed. These architectures are much more efficient than previous hardware implementations of this algorithm. In addition, it must be pointed out that they can be used for different applications sucha as channel equalization.

Finally, a joint source-channel coding technique consisting in the joint optimization of the JSCTC technique and a convolutional code was presented. At the receiver end, we explored the use of MAP estimation for decoding the source with successfull results. It must be noted that the MAP algorithm was used to decode the source, not to correct transmission errors. This corroborates an idea expressed in [91] indicating that in JSCC applications, evaluating the performance of a system in terms of its error probability does not provide a usefull indication of the system's performance.

Future Work

The are a number of areas considered in this thesis where further work could be pursued:

- JSCTC
  1. obviously, a short-term perspective would consist in the hardware implementation of the architectures proposed in this work.
  2. in the context of image coding, the images presented in this work presented an horizontal pattern of channel errors. This is due to the fact that the image was rasterized before being coded. In order to reduce or eliminate this error pattern, an intelligent solution would be to perform a bidimensional JSCTC.
  3. another interesting perspective would be the application of the JSCTC technique for speech coding, and then to implement its hardware architecture. In [87], a TCQ-CELP coder was designed; thus, we could merge the ideas presented in this thesis with those of [87] in order to implement a hardware architecture for robust TCQ-CELP coders.
- Architectures for the M algorithm
  1. to implement a parallel architecture of the M algorithm. So far a serial implementation on FPGAs has been considered

2. to evaluate the performance of these new architectures in the context of inter-symbol interference

- Joint optimization of JSCTC and convolutional coding

1. the use of punctured codes to increase the rate of channel coding
2. study the application of different channel coding schemes such as block codes or turbo codes
3. to study suboptimal trellis searches applied to the MAP algorithm. We have already started to work in this subject with the use of the M algorithm. Nevertheless, the results are quite dissapointing. In [45] it was indicated that when applied to turbo decoding, the M algorithm yields poor results since it has a fixed amount of survivors. The authors proposed to use a trellis search algorithm with adaptive effort such that the number of surviving paths depends on the conditions of the transmission channel. If the channel is very noisy, the number of surviving pahs can increase to its maximum capacity of 2^{K-1} states. On the other hand, in the opposite case, the number of surviving paths may decrease even to one searches path. As a result, we can use the same approach to evaluate the performance when applied the context of a posteriori estimation for source decoding

Appendix A

Trace-Back Techniques in Survivor Memory Management for the Viterbi Algorithm

This appendix deals with the usual methods employed to improve the efficiency of the Trace-Back algorithm used in Viterbi decoders. In Chapter 4, we highlighted the working operation of the TB algorithm and its advantages over the register exchange procedure (density, power consumption, wiring, etc). Nevertheless, a direct implementation of the TB algorithm is not possible since it is assumed that the TB memory stores **all** the decision bits from the ACS unit before the algorithm is started. Consequently, methods to simultaneously updating and reading the TB memory must be found.

In chapter 4 we also showed a naive method to perform the TB algorithm. Clearly, that method is not feasible since serious problems of decoding delay, storage management and read/write conflicts arise. In order to reduce the number of trace-back steps per decoded bit, we can increase the trace back memory from L to $L + l$ memory addresses. This way, for each trace back procedure, l bits are decoded and the number of trace-back steps per decoded bit is reduced from $L + 1$ to $\frac{L+l}{l}$. Clearly, a choice of longer l will further reduce the number of trace-back steps per decoded bit. Nevertheless, there is a price to pay since the decoding delay and the storage requirements both grow. Moreover, the throughput constraint of one decoded bit per decision vector generated by the ACS unit will never be reached. In addition, we have to take into account that during the trace back procedure more decision vectors are created by the ACS unit which need to be stored in the memory.

A.1 l-Pointer Trace-Back

To solve this problem, the conventional approach adopted is to divide the trace back memory in memory banks where three basic operations are performed:

1. **Merging trace-back**: This trace back procedure is performed on a memory bank of length L so as to attain the decoding depth needed to decode reliable segments of the output sequence.
2. **Decoding trace-back**: This trace-back operation uses the output pointer of the merging trace-back in order to decode the surviving path corresponding to the time interval $[k = k - L + 1, k = k - 2L]$. Then, the bits decoded in this trace back operation are sent to the LIFO for data reordering.
3. **Writing of new decision vectors**: New decision vectors generated by the ACS unit are stored in a free memory bank.

Figure A.1 shows the schedule of these operations on a 3-memory-bank trace-back. As the name suggests, the overall trace back memory has been divided into three memory banks. In general, while a *merging trace-back* is being performed on a given memory bank, a *decoding trace-back* is performed in other memory bank with an initial pointer given by a previous *merging trace-back* operation. At the same time, the memory space that is being freed by the *decoding trace-back* is used for *writing new decision vectors.* The detailed process is as follows:

- **initialization** (from $k = 0$ to $k = 2L - 1$)
  1. Memory banks 0 and 1 are filled in with the decision vectors of the ACS unit.
- **Reaching the first decoding depth** (from $k = 2L$ to $k = 3L - 1$)
  1. A *merging trace-back* is performed on memory bank 1 so that the decoding depth is reached and a reliable pointer can be used to trace memory bank 0 back.
  2. In the meantime, the *writing of new decision vectors* is performed on memory bank 2.
- **First decoding trace-back** (from $k = 3L$ to $k = 4L - 1$)
  1. A *decoding trace-back* on memory bank 0 is performed with the pointer given by the *merging trace-back* on memory bank 1 of the previous step.
  2. A *merging trace-back* is performed on memory bank 2 and
  3. The memory space freed by the *decoding trace back* is used for *writing new decision vectors*

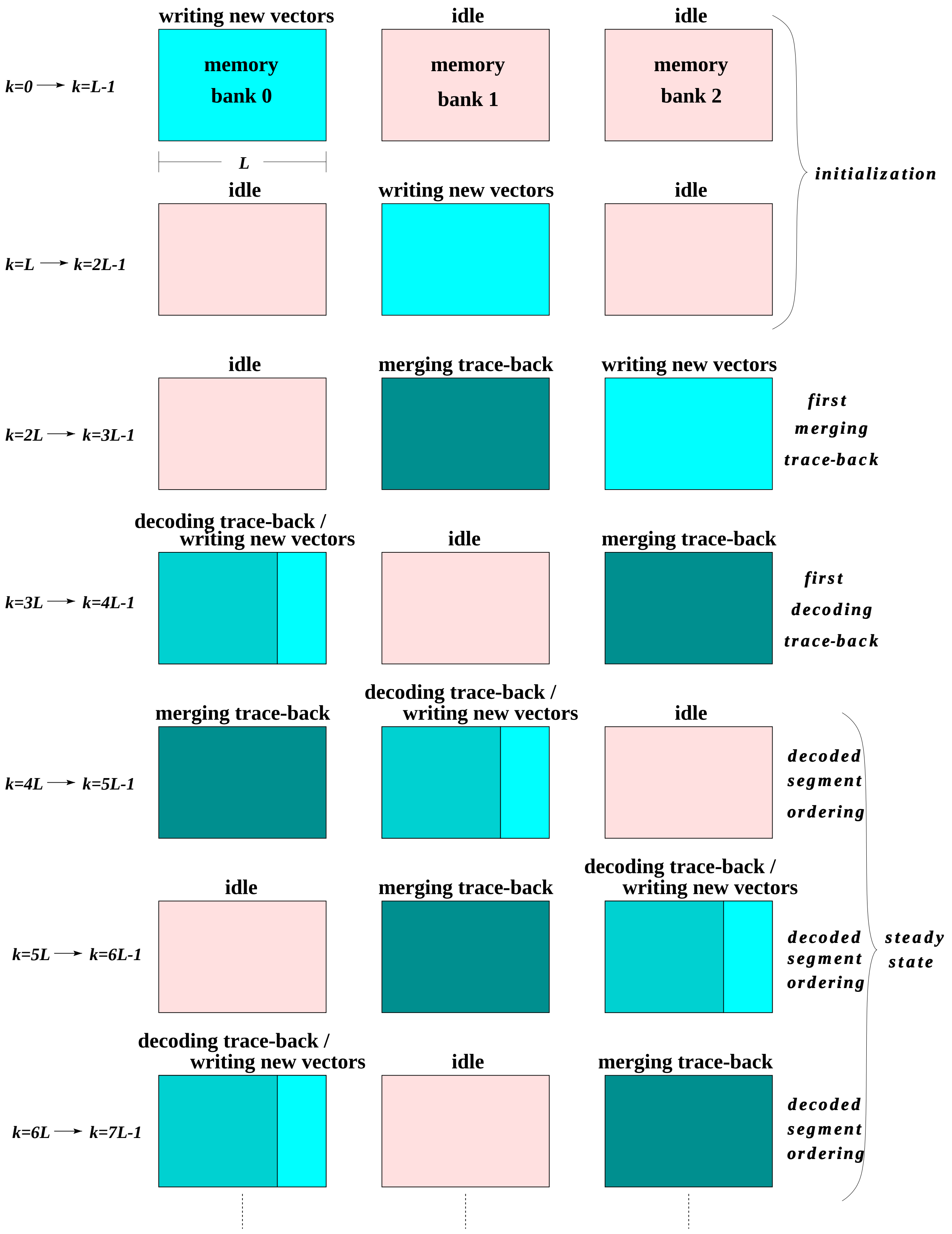


Figure A.1: Trace-back schedule for a 2-pointer trace-back.

- **Steady state** (for $k \geq 4L$; for i=0 to i=∞ in steps of L cycles.)
    1. The decoded segment of decision vectors from the previous *decoding trace-back* on memory bank $i \mod 3$ is arranged in reverse order,
    2. A *merging trace-back* operation is done on memory bank $i \mod 3$ that have just been filled in with decision vectors,
    3. A *decoding trace-back* is performed on memory $(i+1) \mod 3$,
    4. The *writing of decision vectors* is done in the spaced freed by the *decoding trace-back* on memory $(i+1) \mod 3$.
    5. Set $i = i+1$ and go to step 1.

Performing the trace-back algorithm in this way guarantees that the throughput constraint of one decoded bit per decision vector generated is observed.

The trace-back method presented above uses two pointers to implement the trace back algorithm, one pointer for the merging trace-back and another pointer for the decoding trace-back; nevertheless, generalizations to l-pointer trace-backs have been proposed which improve the efficiency of the architecture in different aspects such as latency, read/write conflicts and storage requirements [31, 41]. Other versions of the same approach propose the use of systolic architectures and different clock rates for storing and reading the decision vectors in order to improve the latency and throughput rate of the architecture [22, 75, 112].

The main drawback of this approach is the decoding delay which for the example described above was of $4L$ cycles. This delay can be reduced by increasing the number of memory pointers but the price to pay is an increased complexity of the control circuitry that manages the trace-back operations on different memory banks.

A.2 Improving Hardware Efficiency: Hybrid Survivor Memory Management Architectures.

In order to improve the hardware performance, it is possible to design architectures that mix both the register exchange and the trace-back algorithm [22, 21, 85]. By doing this, we can profit from the advantages of both survivor memory management approaches, namely the small latency and regularity of the register exchange algorithm and the low power dissipation and high storage density of the trace-back procedure. Consequently, we can design hardware architectures with a better trade off between latency and storage requirements.

A first observation leading to this hybrid approach is that in a segment of decision bits of length $(K-1)l$ corresponding to a given state, each subsegment i $(i = 1 \cdots l)$ of $K-1$ bits indicates the ancestor state visited by the surviving path at time $k = k-(K-1)\cdot i$, as indicated in figure A.2. This important feature of the surviving paths can be exploited to improve the hardware efficiency of the survivor memory management architecture.

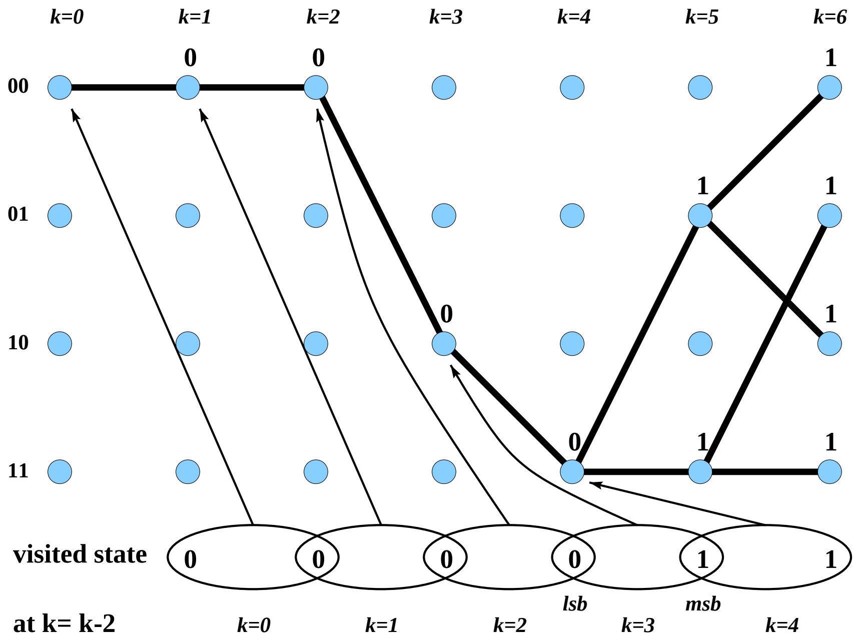


Figure A.2: Ancestor state property of the surviving paths.

The architecture for the survivor memory management can be improved in two major aspects by using the hybrid approach:

- Latency and
- Storage requirements

Two methods to improve this two parameters are treated in the following paragraphs.

A.2.1 Block Trace-Back

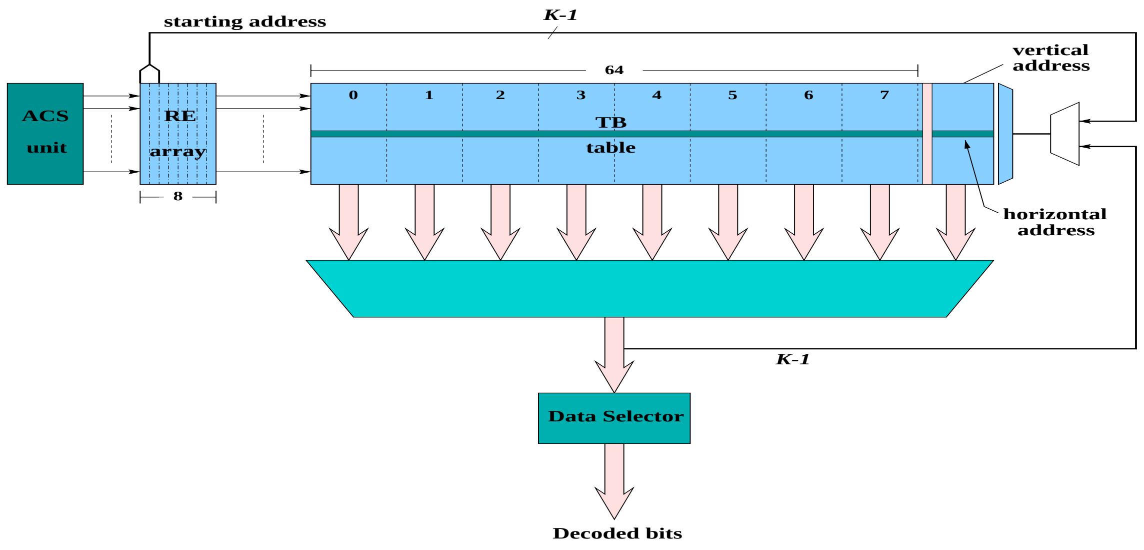
The idea behind the block trace-back (Block TB) is to generate l-bit segments of the surviving paths in a $2^{K-1} \times l$-size register exchange structure, and every l cycles the l-bit segment of the surviving paths is transferred to a trace-back table with 2^{K-1} horizontal addresses and $m \cdot l$ vertical addresses. The segments of the surviving paths are written in the vertical addresses. After $n \cdot l$ cycles, a trace-back operation is performed over the horizontal addresses at a rate of l bits per trace-back step such that in n cycles the surviving path is traced back over a $n \cdot l$-bit depth. In the meantime, the contents of the register exchange array are transferred to the $m - n$ columns of the trace-back table and new n decision vectors are shuffled in the register exchange structure. From the $n \cdot l$ decision vectors that are traced back, l bits belong to the final decoded sequence.

Figure A.3 shows an example of the block trace-back architecture for the special case $l = 8$, $n = 8$ and a total trace-back memory of $n \cdot l + l = 72$ columns. As explained above, at every 8 cycles, the surviving paths in the register exchange structure are shifted into the trace-back table. Upon the shift of 64 columns of the surviving paths into the trace-back memory, an 8-bit-step trace-back operation is performed during 8 cycles.

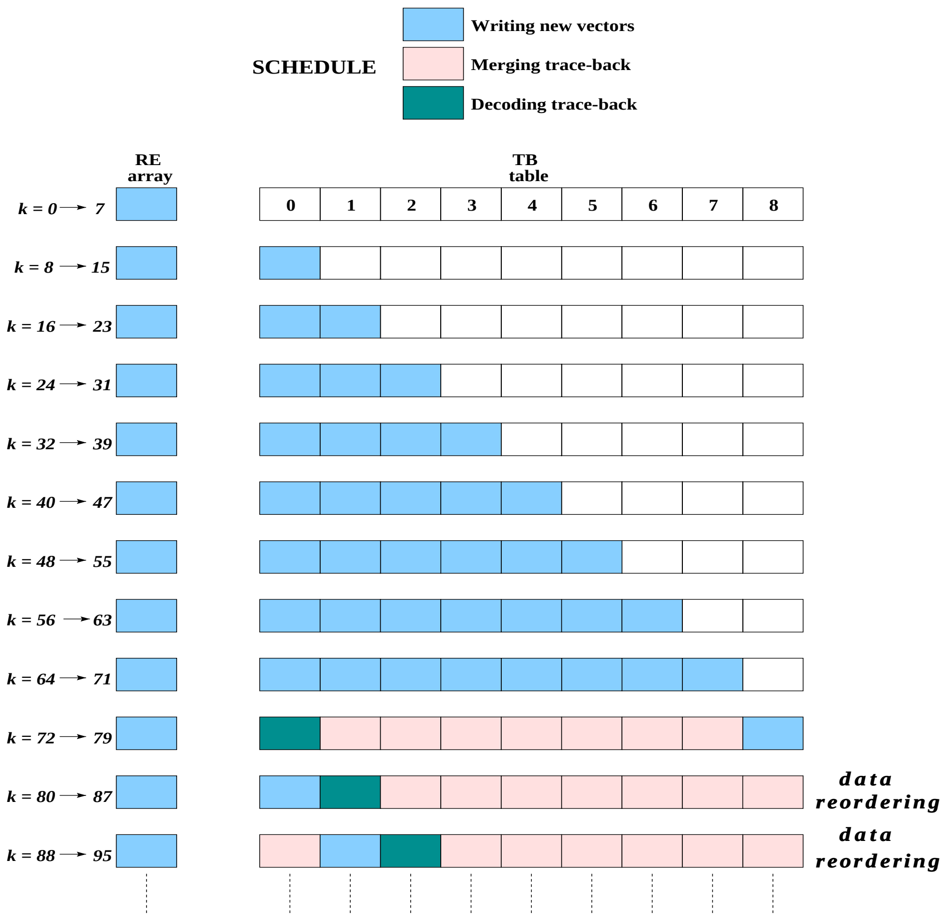
The reading operation of the trace-back table is done over the horizontal addresses and the starting address of the trace-back operation is given by the $K-1$ LSBs of the register exchange contents corresponding to the zero state (or any other state). From the 64-bit word pointed by these $K-1$ bits, only the first 8 bits are taken into account. This 8-bit word corresponds to the first packet of decision bits that were shifted into the trace-back memory. In the next clock cycle, the horizontal address pointed to by the $K-1$ LSBs of this 8-bit word are read; this time the second segment of 8 bits are taken, from which the $K-1$ LSBs indicate the next horizontal address that has to be read. The process is continued until the last segment of 8 bits is read, which is part of the final decoded sequence.

In the meantime, the contents of the register exchange structure are shifted into the trace-back table and l new decision vectors from the ACS unit are shuffled into the register exchange circuit. Since the total trace-back memory is $n + l$ columns, no data is over-written and the throughput constraint of one decoded bit per decision vector generated is respected.

We can see that in this architecture the decoding depth is $L = 64$ which make a total memory of 80 columns and a latency of 80 cycles. Compared to this method, the conventional 2-pointer trace-back approach of figure A.1 would need 192 columns and its latency would be of 256 cycles. Clearly, the hybrid architecture offers a significant improvement. Finally, notice that by reducing the size of register exchange depth l and increasing the trace-back memory in the same proportion, the memory requirements and the latency of the hybrid approach are further reduced.



(a) Block trace-back architecture



(b) Operation schedule.

Figure A.3: Block trace-back architecture and operations schedule.

A.2.2 Register-Exchange-Pointer Trace-Back

A special case of the above method which is able to reduce the storage requirements and latency to their lower bounds is the *register-exchange pointer trace-back* (RE-pointer TB). In this so called method, the architecture is also divided into a register exchange circuit and a trace-back memory. However, in this case the register exchange circuit is only $K-1$ columns wide.

Figure A.4 shows the architecture and operation principle of this method. In this example, the trace-back memory is divided into two banks of L columns each. The decision vectors generated by the ACS unit are transferred to the trace-back memory in the usual way. Regarding the register exchange array, during the first $K-1$ cycles, the curcuit works as usual; then, from $k = K-1$ to L, the decision vectors coming from the ACS unit are not shifted into the register exchange array but are only used to drive the shuffling operation. This way, according to the decoding depth property and the property described in figure A.2, after L cycles the register exchange contents indicate the ancestor state of the surviving paths L cycles earlier. As a consequence, this state can be used as the initial pointer of the decoding trace-back. This way, the merging trace-back of the l-pointer trace-back method is avoided and hence the decoding delay is reduced.

In the above example, the latency of the architecture has been reduced from $4L$, corresponding to the 2-pointer trace-back method, to $3L$. In addition, the storage requirements have been reduced from $3L$ to $2L + K-1$ decision vectors. Generalizations to several register exchange units are straightforward. Figure A.5 shows the register-exchange-pointer trace-back schedule for two register exchange units RE_1 and RE_2. During the first $\frac{L}{2}$ cycles, the first trace back memory is filled in. Then, at $k = \frac{L}{2}$, the second TB memory starts to be filled in and the RE_1 array is initialized during $K-1$ cycles. Afterwards, the decision bits control the shuffle of this array during L cycles in order to attain the decoding depth. In the meantime, at $k = L$, the RE_2 array is initialized and shuffled during L cycles. At $k = \frac{3L}{2}$, the first RE array attains the decoding depth and hence its stored valued indicates the visited state L cycles before. This values serves to perform a trace back on the first TB memory. Then, at $k = 2L - 1$ the second RE array attains the decoding depth and a trace-back is performed on the second TB memory. The process continues in this way until all the sequence is trated. Notice the significant reduction of the latency and the size of the trace-back memory with this method ($2L$ and $\frac{3}{2}L$, repectively).

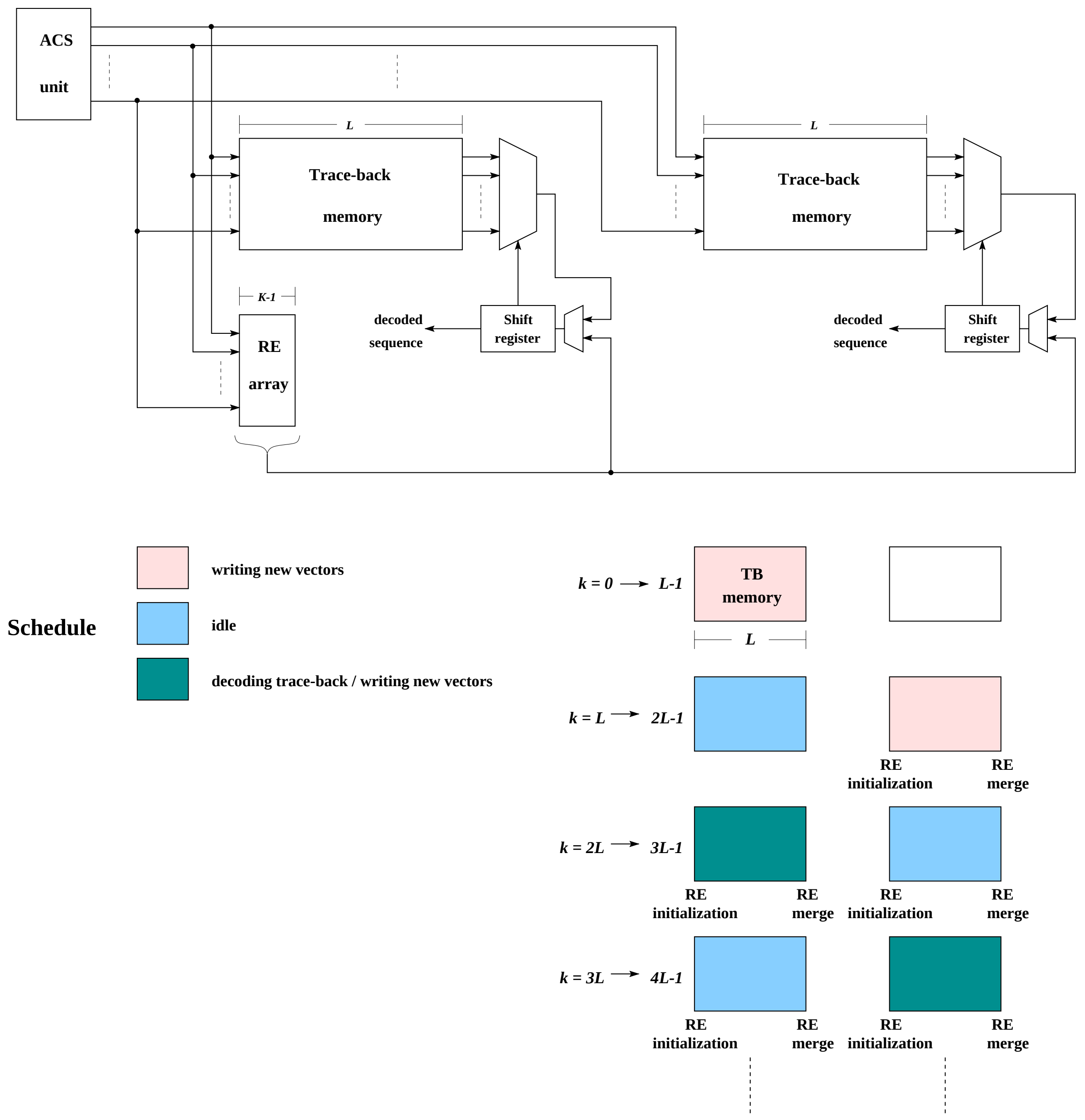


Figure A.4: Register-Exchange-Pointer Trace-Back Architecture and Operations Schedule.

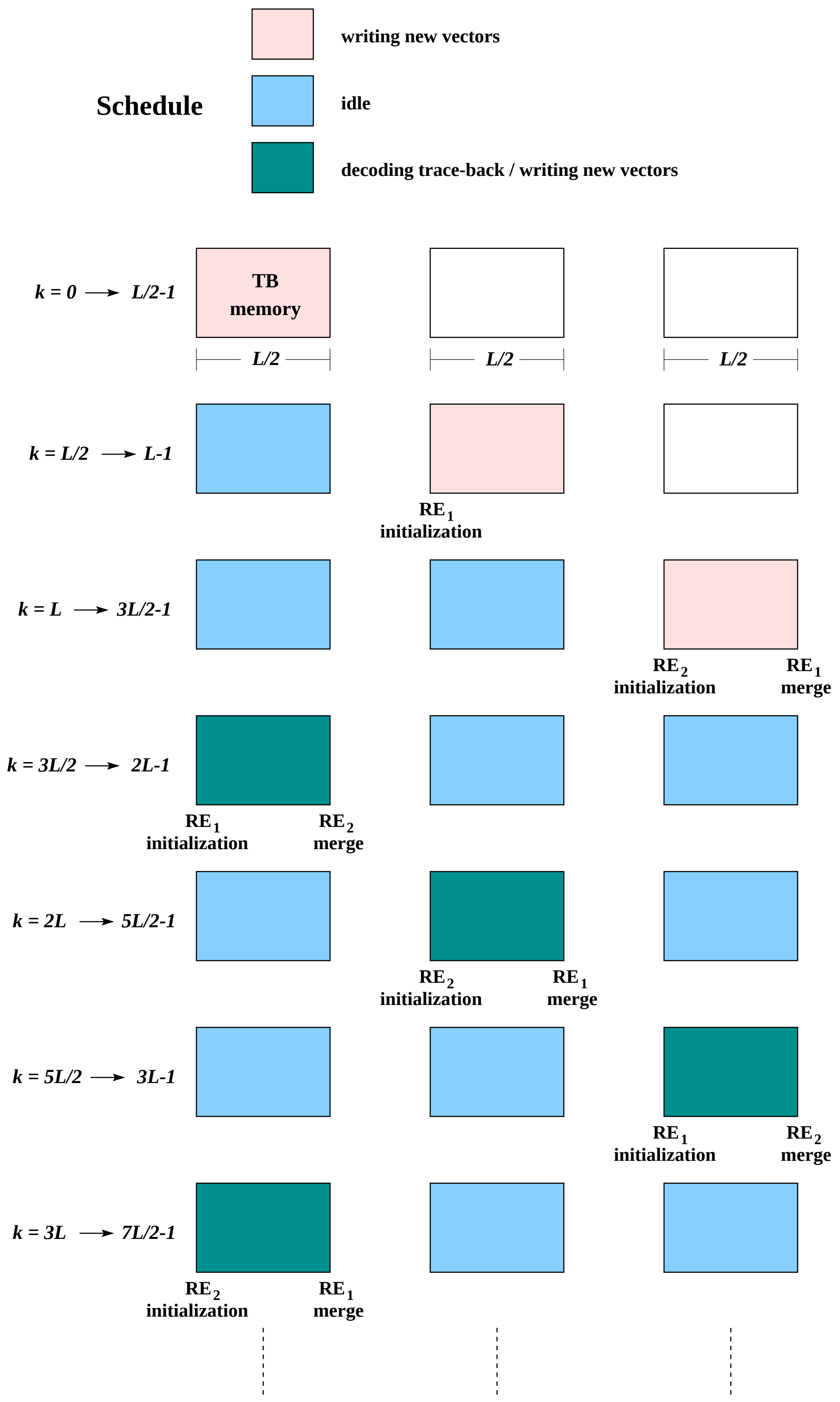

Schedule
writing new vectors
idle
decoding trace-back / writing new vectors
$k = 0 \longrightarrow L/2-1$
TB memory
L/2
L/2
L/2
$k = L/2 \longrightarrow L-1$
RE_1 initialization
$k = L \longrightarrow 3L/2-1$
RE_2 initialization
RE_1 merge
$k = 3L/2 \longrightarrow 2L-1$
RE_1 initialization
RE_2 merge
$k = 2L \longrightarrow 5L/2-1$
RE_2 initialization
RE_1 merge
$k = 5L/2 \longrightarrow 3L-1$
RE_1 initialization
RE_2 merge
$k = 3L \longrightarrow 7L/2-1$
RE_2 initialization
RE_1 merge

Figure A.5: Register-Exchange-Pointer Trace-Back Schedule with two RE pointers.

A.3 Complexity Models of the Different Survivor Memory Management Techniques

As a conclusion of this appendix, table A.1 summarizes the latency and storage requirements of the different survivor memory management techniques presented. The reader is referred to [22] and [41] for detailed derivations of these expressions.

Table A.1: Latency and storage requirements of the different survivor memory management techniques.

technique	Latency (cycles)	Storage requirements
TB	$\frac{2l_1}{l_1-1}L$	$2^{K-1} \times \frac{2l_1-1}{l_1-1}L$
Block TB	$nl_2 + l_2 + n$	$2^{K-1} \times (nl_2 + l_2 + n)$
RE-pointer TB	$L + 2\frac{L}{p}$	$2^{K-1} \times \left[p(K-1) + \left(L + \frac{L}{p}\right)\right]$

l_1: number of TB pointers
l_2: RE depth
n: TB depth
p: number of RE pointers

Appendix B

A Brief Survey on Sorting Algorithms

Sorting has attracted so much attention that bibliography on this subject is very extensive [13, 56, 61, 63, 88, 117]. One of the reasons of this is that sorting is an interesting problem with a great deal of practical applications, specially in computer science research; for example, computer programs such as compilers or editors often choose to sort tables and lists so as to enhance the speed and simplicity of the algorithms used to access them.

Since the main area of application of sorting algorithms is computer science, the characteristics of these sorting algorithms are suitable for software applications rather than for hardware implementations. Yet, with the advent of parallel processing, these algorithms evolved and hadware implementations have been reported [17, 18]

Nevertheless, in spite of the great effectiveness of these algorithms, their study and utilization are beyond the scope of this dissertation because those algorithms were designed for the processing of large amounts of information, whereas in our particular case, the number of values to sort is relatively small. Thus, by using one of these algorithms, we would be overestimating the problem, resulting in hardware designs that are more complex than they should be.

Sorting algorithms can be divided into two classes: parallel and serial. In the former, all the values to be sorted are processed together in an interconnection network fashion; conversely, in the latter it is assumed that the values arrive serially to the sorting circuit in such a way that the new value to sort is inserted in an already ordered list. In the following, different sorting algorithms which are the most suitable for our application are treated. Extensive studies can be found in [20, 46, 111]. In fact, most of the material presented here was taken from those references so as to ease the comprehension of the exposition.

B.1 Parallel Sorting

B.1.1 Batcher's Odd-Even Sorting Network

This algorithm is due to Batcher [11], and the principle of this algorithm is based on the merging of two ascendengly-ordered lists of numbers into one ascendengly-ordered list. To make this sorting network fast, it is necessary to have a number of camparison elements performing comparisons in parallel. A compaison-exchange element is shown in figure B.1. The two inputs entering the comparison-exchange element are compared, the smallest input is released by the upper output line (L output) and the largest input is released according to the arrow head (H output).

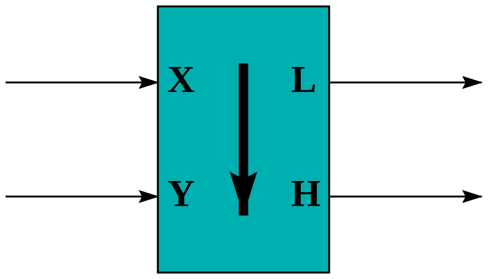


Figure B.1: Comparison-exchange element for sorting networks.

The odd-even sorting network is based on iterated merging, that is, an iterative rule is applied to an N-item list so as to create small sorted lists of size 2, 4, 8, $\cdots$, N during successive stages. The iterative rule of the odd-even merging network is illustrated in figure B.2. Starting from two sorted lists of numbers $A = (a_1, \cdots, a_s)$ and $B = (b_1, \cdots, b_t)$, two new lists C and D are generated. List C is formed by merging the odd-numbered terms of lists A and B, and list D is formed by merging the even-numbered terms. Finally, lists C and D are merged to obtain the sorted list E by performing the following comparison-exchanges

$$\begin{aligned} e_1 &= c_1 \\ e_{2i} &= min(c_{i+1}, d_i) \\ e_{2i+1} &= max(c_{i+1}, d_i) \quad i = 1, 2, \cdots \\ e_{s+t} &= d_t \end{aligned} \tag{B.1}$$

The iterative merge of numbers is done by noting that the merging network of two numbers is simply a comparison-exchange element. Then, a merging network of four elements is created from two 2-item merging networks and the iterative rule described in equations B.1, as shown in figure B.3(a). Notice that one condition to employ the merging network is that the two list are in sorted order. Thus, to implement a four item sorting

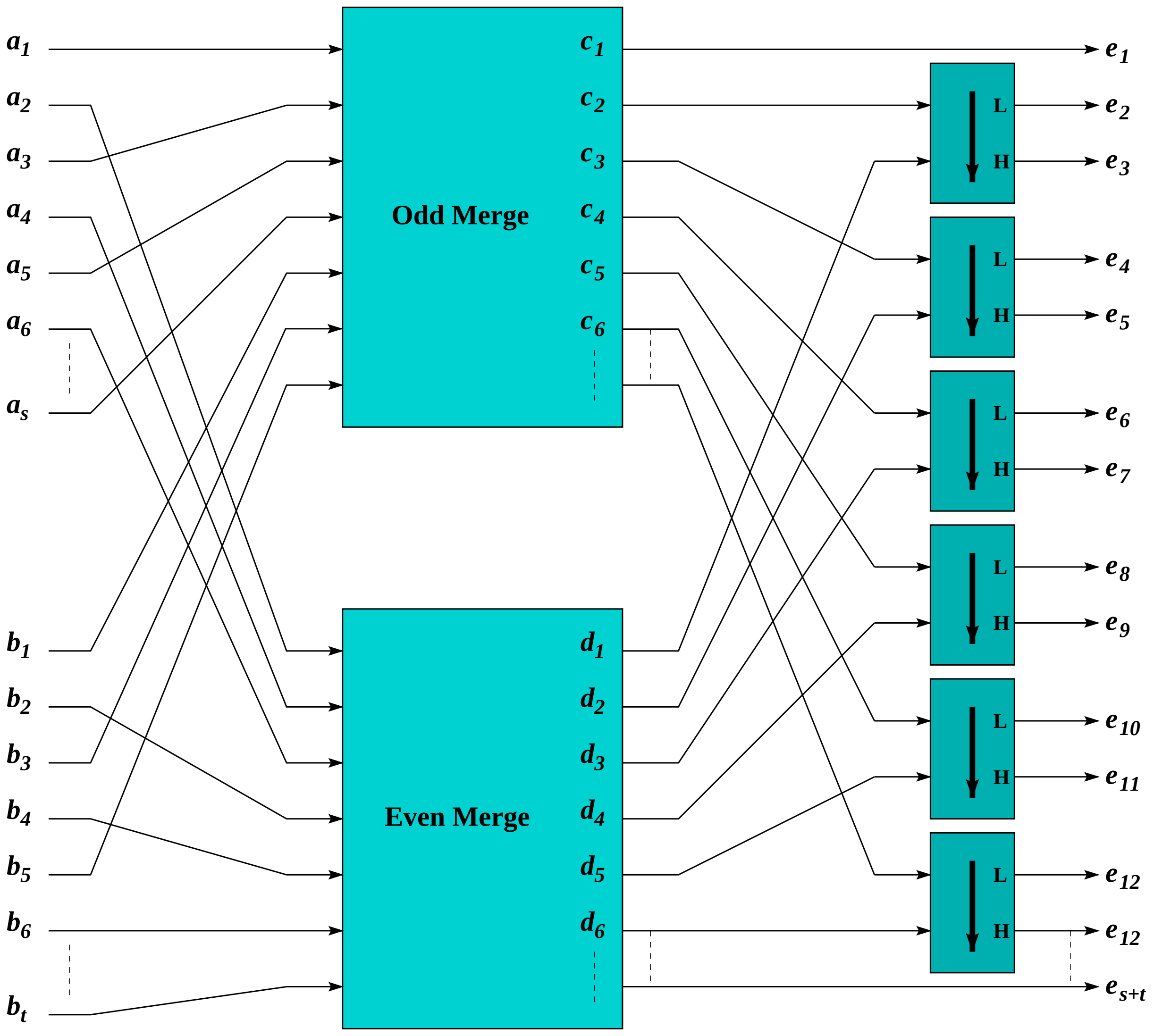
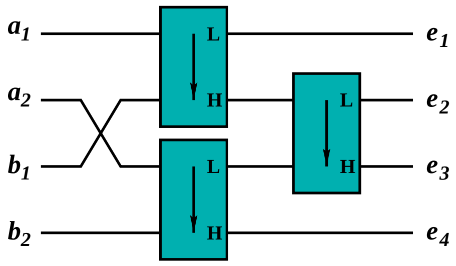
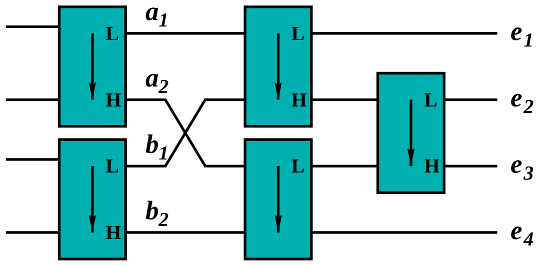


Figure B.2: Iterative rule of the odd-even merging network.

network, we need two 2-item sorting networks (that is, two comparators) followed by a 4-item merging network as indicated in figure B.3(b). The two comparators in the first vertical layer of the sorting network deliver two ordered lists of two items each to the 4-item merging network. Then, the 4-item merging network merge the two 2-item ordered list into a single ordered list.



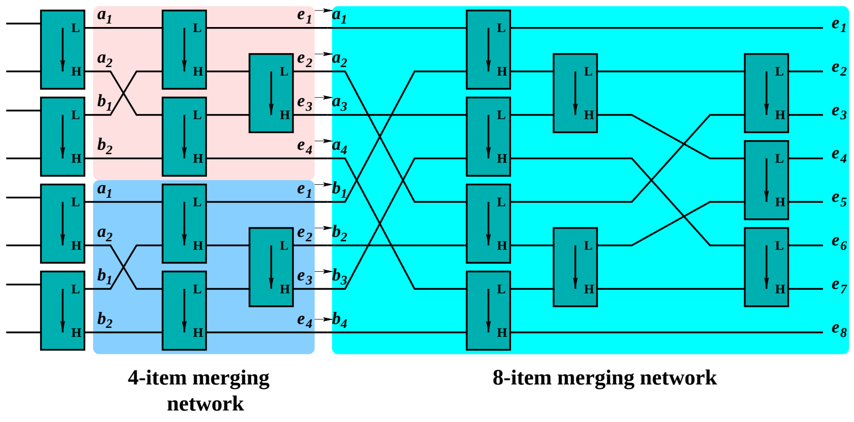
(a) odd-even merging network



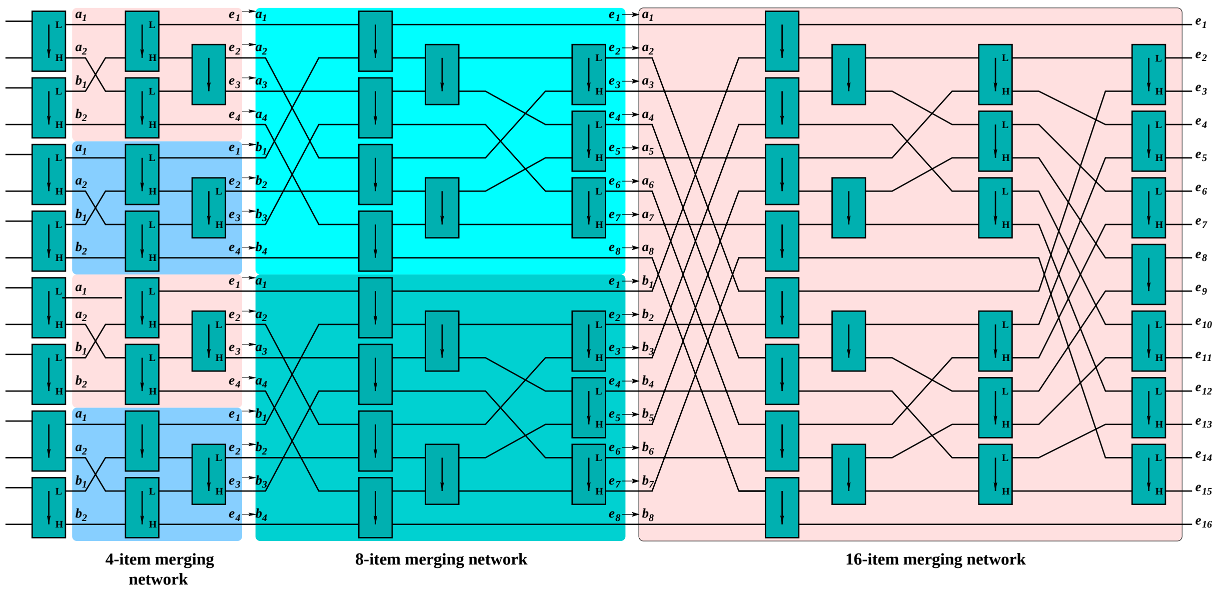
(b) odd-even sorting network

Figure B.3: Four-item merging and sorting networks.

A sorting network of $N = 8$ items is shown in figure B.4(a). Notice the recursivity of the algorithm; in order to produce two ordered lists entering the 8-item merging network, two 4-item merging networks are needed. These in turn need two 2-item merging networks (two comparators) so that the two lists entering each merging network are in sorted order. It is important to notice the difference between a **merging network** and a **sorting network**. The merging network requires two ordered lists at its inputs whereas the sorting network not. Finally, figure B.4(b) presents a sorting network for $N = 16$ items.



(a) 8-item sorting network.



(b) 16-item sorting network.

Figure B.4: Odd-even sorting networks for $N = 8$ and $N = 16$-item lists.

The features of the odd-even sorgin network are:

- Number of comparision-exchange elements employed in a N-item **merging network**:

$$\log_2\left(\frac{N}{2}\right) \cdot \frac{N}{2} + 1 \tag{B.2}$$

- Number of comparison-exchange elements used in a N-item **sorting network**:

$$\frac{N}{4} \cdot \left[(\log_2 N)^2 - \log_2 N + 4\right] - 1 \tag{B.3}$$

- Number of vertical comparison-exchange layers:

$$\frac{(1 + \log_2 N) \cdot \log_2 N}{2} \tag{B.4}$$

Notice that the smallest item takes $\log_2 N$ comparisons to be known. In addition, other important aspect of this sorting network is that pipelining can be introduced very easily.

B.1.2 Batcher's Bitonic Sorting Network

The iterative rule of the bitonic merging network is shown in figure B.5. A bitonic list is obtained by concatenating two monotonically increasing lists, one in ascending order and the other in descending order. The bitonic rule comes from the fact that if $A = (a_1, a_2, \cdots, a_N)$ is a bitonic list and we form two lists defined as

$$min(a_1, a_{\frac{N}{2}+1}), min(a_2, a_{\frac{N}{2}+2}), min(a_3, a_{\frac{N}{2}+3}), \cdots, min(a_{\frac{N}{2}}, a_N) \tag{B.5}$$

and

$$max(a_1, a_{\frac{N}{2}+1}), max(a_2, a_{\frac{N}{2}+2}), max(a_3, a_{\frac{N}{2}+3}), \cdots, max(a_{\frac{N}{2}}, a_N) \tag{B.6}$$

then, these two lists are also bitonic, and list B.5 has the smallest values of the N-item list.

In the same manner as in the odd-even merging network, a bitonic sorter of 2 items is a single comparison-exchange element. Figure B.6 shows bitonic merging and sorting networks for lists of four and eight items. Notice that the same construction of sorting networks as that of the odd-even case applies. That is, in order to sort a list of N items, these are combined two at a time to form ordered lists of size 2; these lists are merged two at a time to form ordered lists of length four and so on, until all the items are merged into a single list.

The features of the bitonic sorting network are:

- Number of comparision-exchange elements employed in a N-item **merging network**:

$$\frac{N}{2} \cdot \log_2 N \tag{B.7}$$

- Number of comparison-exchange elements used in a N-item **sorting network**:

$$\frac{N \cdot \log_2 N \cdot (\log_2 N + 1)}{4} \tag{B.8}$$

- Number of vertical comparison-exchange layers:

$$\frac{(1 + \log_2 N) \cdot \log_2 N}{2} \tag{B.9}$$

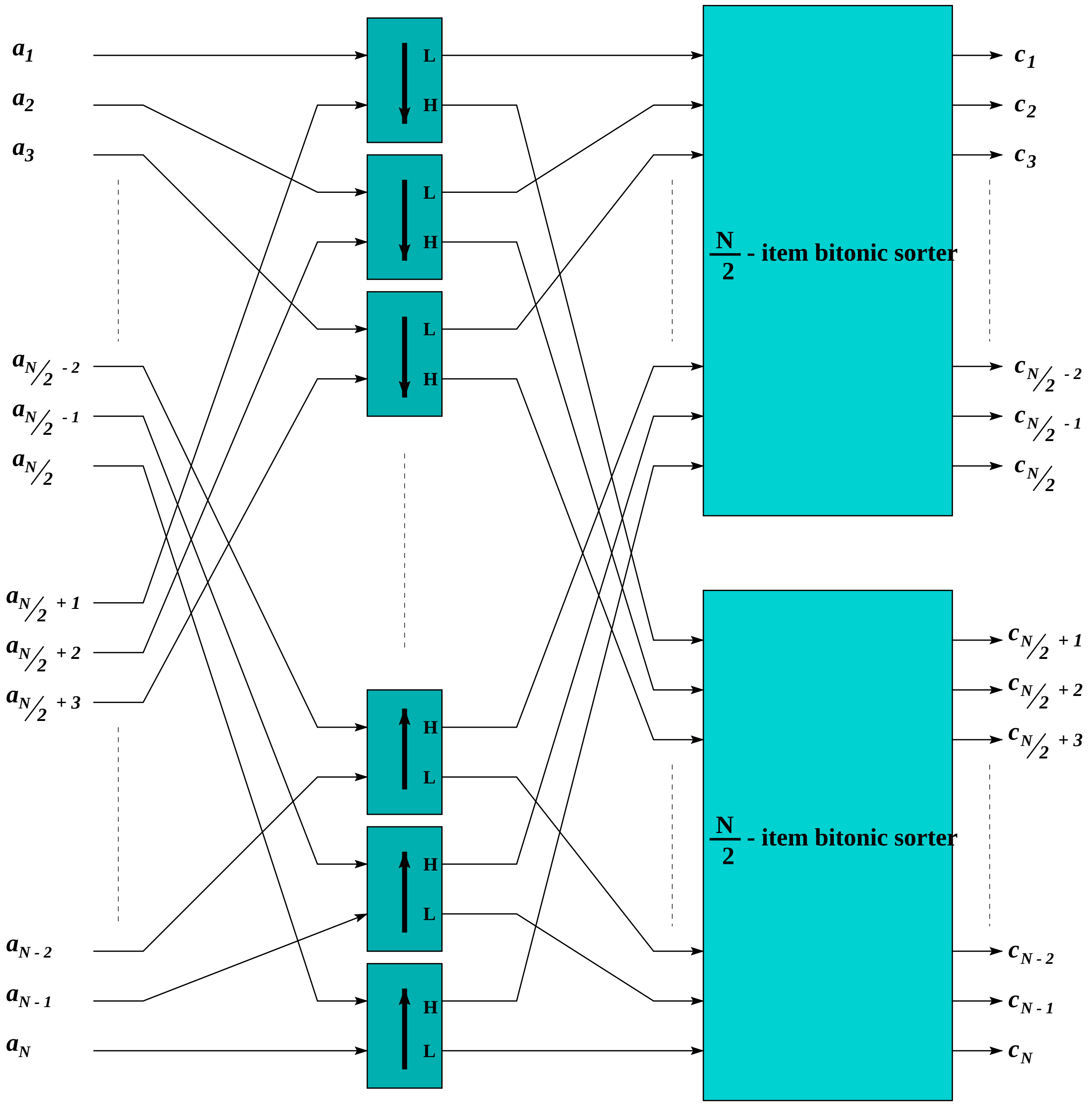
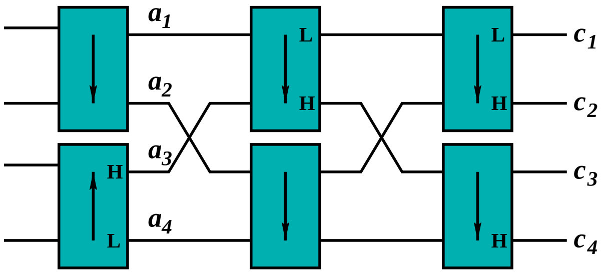
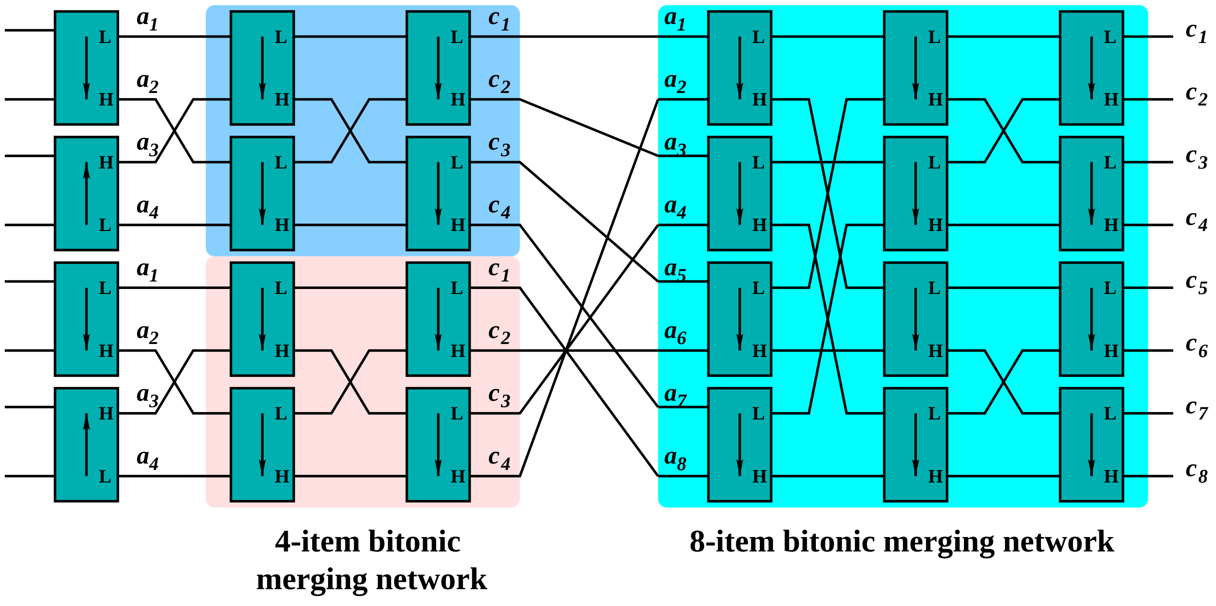


Figure B.5: Iterative rule for the bitonic merging network.



(a) 4-item sorting network.



(b) 8-item sorting network.

Figure B.6: Bitonic sorting networks for four and eight item lists.

B.1.3 Bubble Sort

The principle of this algorithm is based on the slow ascending of "light" values. Figure B.7 show a bubble sorter for $N = 8$ items. At each stage of the algorithm, the "heavy" values tend to go down along the network whereas the "light" values tend to go up. The features of this sorting network are:

- Number of comparison-exchange elements used in a N-item sorting network:

$$\frac{N \cdot (N-1)}{2} \tag{B.10}$$

- Number of vertical comparison-exchange layers:

$$N \tag{B.11}$$

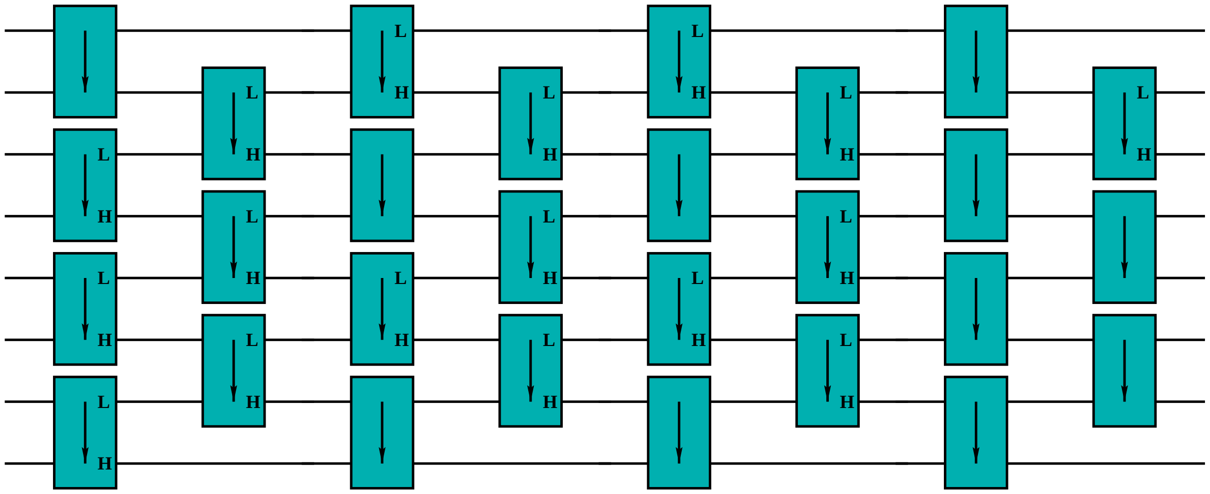


Figure B.7: Bubble sorting network for 16-item lists.

B.2 Serial Sorting

B.2.1 Single Insertion

This algorithm consists in the sequential comparison of the item to be inserted (*inserting element*) with each one of the items already inserted in the sorting circuit. Notice that the insertion is done on an already ordered list. Figure B.8 shows the architecture of this sorting algorithm. A first comparison is made with the first element of the ordered list and depending on the result, the inserting element is either inserted in that position or passed to the second comparison-exchange element. When the inserting element is inserted, the item that was previously stored in that position is compared to the right-hand-side items of the list producing a shifting operation. Notice that the comparison elements in the figure are not comparison-exchange elements anymore; they are simple comparator circuits whose output is the largest element of the two inputs and a control signal that indicates if the inserting element is larger than the inserted element or not.

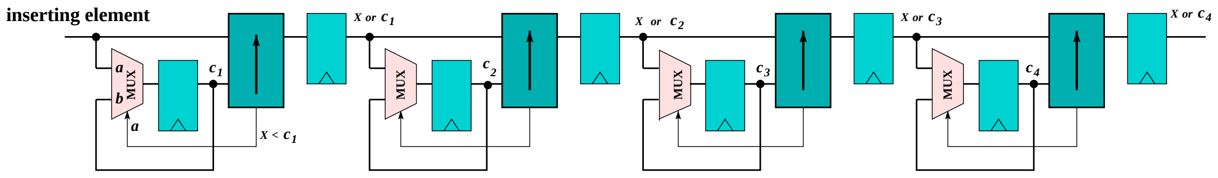


Figure B.8: Sorting architecture with the single insertion algorithm.

The features of this algorithm are:

- Processing time to sort N items:

$$\frac{N \cdot (N-1)}{2} \tag{B.12}$$

- Number of comparison-exchange elements:

$$N \tag{B.13}$$

B.2.2 Dichotomic Insertion

This algorithm is shown in figure B.9. The idea is to compare the inserting element to the middle element in an ordered list. Then, depending on the result of this comparison, the inserting element is either moved to the upper half of the N-item list or to the lower half. Next, the inserting element is compared to the middle element of the upper or lower half (second middle element in the figure) and depending on the result, the inserting element passes to the upper or lower half of the upper or lower half of the N-item list. This process is repeated until there is only one item left to compare with (third middle element). Finally, the previously inserted elements are shifted to the right and the inserting element is inserted. We must point out that in the comparison elements utilized in this structure the middle elements of the ordered list do not pass through the comparison element, only the inserting element in released as output either through the L or H outputs. The features of the dichotomic insertion are:

- Processing time to sort N items (number of comparisions):

$$N \cdot \log_2 N \tag{B.14}$$

- Number of comparison-exchange layers:

$$\log_2 N \tag{B.15}$$

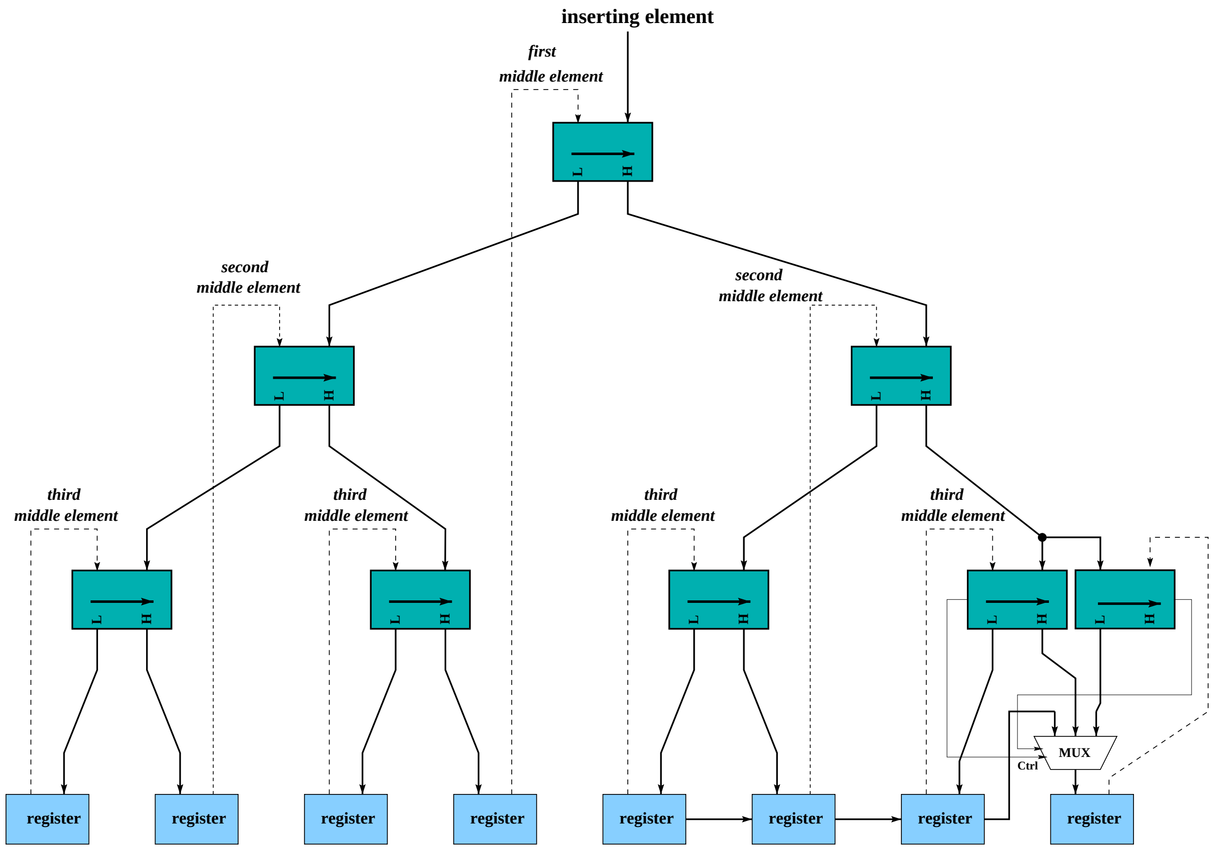


Figure B.9: Sorting architecture with the dichotomic insertion algorithm.

The main drawback of this algorithm is that we cannot introduce pipelining since we must wait for the current inserting element to be inserted before starting the insertion on the next element. This is due to the fact that the middle values of the list must be updated.

A final comment is in order. If we want to obtain the smallest N values from a list of M ($M > N$), the rightmost comparison-exchange element and the multiplexer of figure B.9 are required. Depending on the result of the two comparisons between the two largest values on the list and the inserting element, the inserting element is either inserted in the penultimate position, in the last position or not inserted at all.

B.2.3 Parallel Insertion

In this final algorithm, the inserting element is compared to all the previously inserted elements at the same time. Three cases may occur during this parallel comparision (see figure B.10.

1. The inserting element is larger than the inserted element of the i^{th} position. In this case, nothing in done.
2. The inserting element is larger than the inserted element of the i^{th} position but is smaller than the inserted element of position $i+1$. This means that the inserting element must be inserted at the $i+1^{th}$ position.
3. The inserting element is smaller than the inserted elements to its right. In this case, the inserted elements must be shifted to the right position.

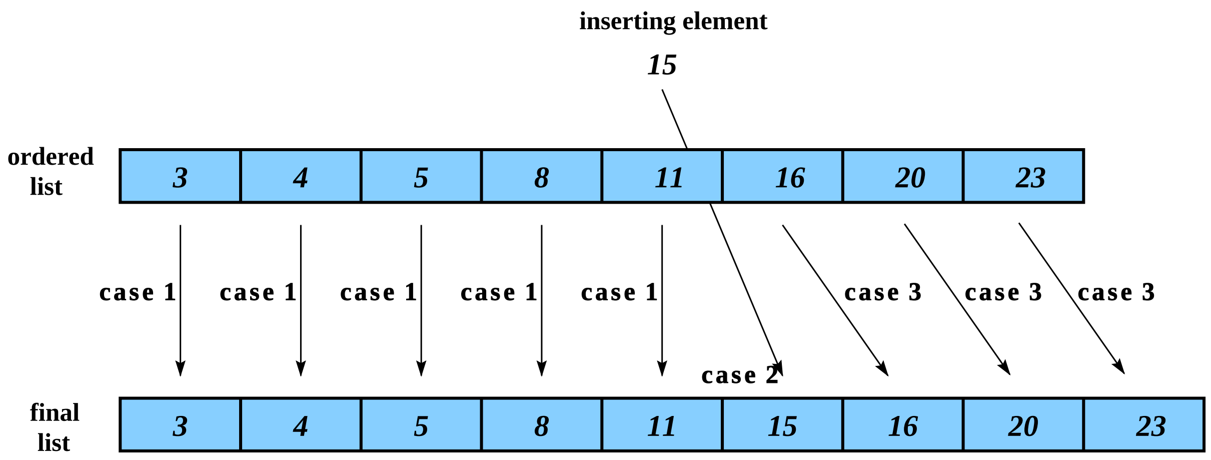


Figure B.10: Principle of parallel insertion.

This insertion takes only one processing time consisting of one comparison (actually, N parallel comparisons) followed by shifting operations. The architecture of this algorithm is shown in figure B.11. This time, each element is a simple comparator whose output only indicates which of the two inputs is the smallest one. The multiplixer on each register is controlled by the output of the current register $r = i$ and the output of the previous register $r = i-1$. When the inserting element is smaller than the element of register $r = i$, the ouput of the comparator commands the multiplexer to select the inserting element to be stored in register $r = i$. Then, the outputs of comparators i and $i+1$ select the element of the i^{th} position to be stored in register $r = i+1$. The same operation occurs with the other comparators, causing the elements stored at registers $r > i$ to be shifted to the right.

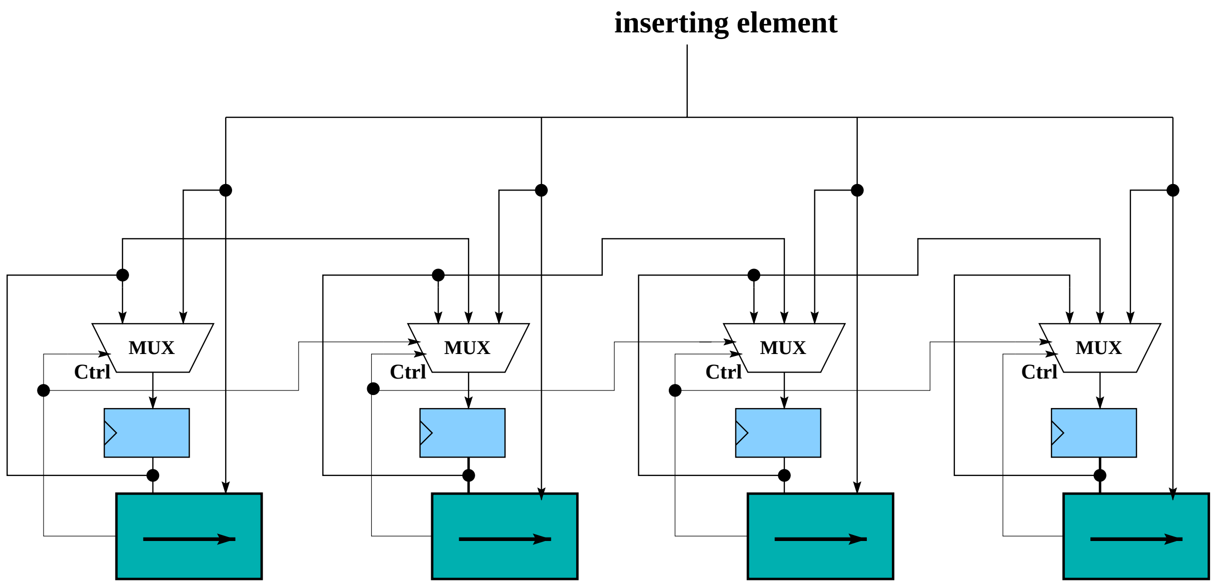


Figure B.11: Parallel insertion Architecture.

Bibliography

[1] F. Alajaji, N. Phamdo and T. Fuja, *"Channel Codes that Exploit the Residual Redundancy in CELP-encoded speech"*, IEEE Transactions on Speech, Audio and Signal Processing, Vol. 4, pp. 325-336, September 1996.

[2] John B. Anderson, *"A stack algorithm for source coding with a fidelity criterion"*, IEEE Transactions on Information Theory, Vol. IT-20, No. 3, pp. 211-226, March 1974.

[3] John B. Anderson, *"Limited Search Trellis Decoding of Convolutional Codes"*, IEEE Transactions on Information Theory, Vol. IT-35, No. 5, pp. 944-955, September 1989.

[4] John B. Anderson, *"Effectiveness of sequential tree search algorithms in speech digitization"*, In 1979 Internatinal Conference on Communciations Conference Record, pp. 8.1.1-8.1.6, Boston Mass., June 1979.

[5] John B. Anderson and C.-W. P. Ho, *"Architecture and Construction of a Hardware Sequential Encoder for Speech"*, IEEE Transactions on Communciations, Vol. COM-25, pp. 703-707, July 1977.

[6] John B. Anderson and F. Jelinek, *"A 2-Cycle Algorithm for Source Coding with a Fidelity Criterion"*, IEEE Transactions on Information Theory, Vol. IT-19, pp. 79-92, January 1973.

[7] John B. Anderson and Seshadri Mohan, *"Sequential Coding Algorithms: A Survey and Cost Analysis"*, IEEE Transactions on Communciations, Vol. COM-32, No. 2, pp. 169-176, February 1984.

[8] Tor M. Aulin, *"Breadth-First Maximum Likelihood Sequence Detection: Basics"*, IEEE Transactions on Communications, Vol. COM-47, No. 2, pp. 208-216, February 1999.

[9] Ender Ayanoğlu and Robert M. Gray, *"The Design of Joint Source and Channel Trellis Waveform Coders"*, IEEE Transactions on Information Theory, Vol. IT-33, No. 6, pp. 855-865, November 1987.

[10] A. Baier and G. Heinrich, *"Performance of M-Algorithm MLSE Equalizers in Frequency-Selective Fading Mobile Radio Channels"*, Proceeding of the 1989 International Conference on Communications (ICC'89), Boston, USA, pp. 281-285, 1989.

[11] K. E. Batcher, *"Sorting Networks and their Applications"*, Proceeding of the AFIPS 1968 Spring Joint Comput. Conf., Montvale, NJ., pp. 307-314, 1968.

[12] L.R. Bahl and J. Cocke and F. Jelinek and J. Raviv, *"Optimal Decoding of Linear Codes for Minimizing Symbol Error Rate"*, IEEE Transactions on Information Theory, Vol. IT-20, pp. 284-287, March 1974.

[13] G. Baudet and D. Stevenson, *"Optimal Sorting Algorithms for Parallel Computers"*, IEEE Transactions on Computers, Vol. C-27, No. 1, pp. 84-87, January 1978.

[14] Toby Berger, *"Rate Distortion Theory: A Mathematical Basis for Data Compression"*, Prentice-Hall, Englewood Cliffs, N.J., 1971.

[15] Benjamin Belzer, John D. Villasenor and Bernd Girod, *"Joint Source Channel Coding of Images with Trellis Coded Quantization and Convolutional Codes"*, Proceedings of the 1995 IEEE International Conference on Image Processing (ICIP'95), vol.2, pp. 85-88, October 1993

[16] Ezio Biglieri, Marvin K. Simon, D. Divsalar, Peter J. McLane and John Griffin, *"Introduction to Trellis Coded Modulation with Applications"*, Prentice Hall, April 1991.

[17] Gianfranco Bilardi and Franco P. Preparata, *"An Architecture for Bitonic Sorting with Optimal VLSI Performance"*, IEEE Transactions on Computers, Vol. C-33, No. 7, pp. 646-651, July 1984.

[18] Gianfranco Bilardi and Franco P. Preparata, *"A Minimum Area VLSI Network for O(log n) Time Sorting"*, IEEE Transactions on Computers, Vol. C-34, No. 4, pp. 336-343, April 1985.

[19] M. Biver, H. Kaeslin and C. Tommasini, *"In-Place Updating of Path Metrics in Viterbi Decoders"*, IEEE Journal of Solid-State Circuits, Vol. 24, No. 4, pp. 1158-1160, August 1989.

[20] Dina Bitton, David J. DeWitt, David K. Hsiao and J. Menon, *"A Taxonomy of Parallel Sorting"*, Computing Surveys, Vol. 16, No. 3, pp. 287-318, September 1984.

[21] Peter J. Black and Teresa H.-Y. Meng, *"Hybrid Survivor Path Architectures for Viterbi Decoders"*, Proceeding of the 1993 International Conference on Acoustics, Speech and Signal Processing (ICASSP'93), Minnesota, USA, vol. 1, pp. 433-436, 1993.

[22] Emmanuel Boutillon, *"Architecture et Implantation VLSI de Techniques de Modulation Codées Performantes Adaptées au Canal de Rayleigh"*, PhD dissertation, Ecole Nationale Supérieure des Télécommunications, Paris, France, December 1995.

[23] Mahieddine Bouzidi, *"Estimation de la Consommation de Circuits CMOS Numériques"*, PhD dissertation, Ecole Nationale Supérieure des Télécommunications, Paris, France, September 1999.

[24] C. Britton Rorabaugh, *"Error Coding Cookbook"*, McGraw-Hill, 1996.

[25] Bruno Calvo Chevillat, *"Implémentation VLSI de l'Algorithme ML"*, Mémoire de Fin d'Etudes, ENST, Paris, France, July 2000.

[26] François Chan and David Haccoun, *"Adaptive Viterbi Decoding of Convolutional Codes over Memoryless Channels"*, IEEE Transactions on Communications, Vol. COM-45, No. 11, pp. 1389-1400, November 1997.

[27] David Chase and Allen Gersho, *"Real-Time VQ Codebook Generation Hardware for Speech Processing"*, Proceeding of the 1988 International Conference on Acoustics, Speech and Signal Processing (ICASSP 88), New York, USA, vol. 3, pp. 1730-1733, April 1988.

[28] Kwok-Hung Chei and Keang-Po Ho, *"Design of Optimal Soft Decoding for Combined Trellis Coded Quantization/Modulation in Rayleigh Fading Channel"*, Proceeding of the 2000 International Conference on Acoustics, Speech and Signal Processing (ICASSP2000), Istambul, Turkey, June 2000.

[29] G. C. Clark and J. B. Cain, *"Error Correction Coding for Digital Communications"*, Plenum Press, New York, 1981.

[30] D. Comstock and J. D. Gibson, *"Hamming coding of DCT compressed images over noisy channels"*, IEEE Transactions on Communications, Vol. COM-32, No. 7, pp. 856-861, July 1984.

[31] Robert Cypher and C. Bernard Shung, *"Generalized Trace-Back Techniques for Survivor Memory Management in the Viterbi Algorithm"*, Journal of VLSI Signal Processing, 5, pp. 85-94, 1993.

[32] Grant A. Davidson, Peter R. Capello and Allen Gersho, *"Systolyc Architectures for Vector Quantization"*, IEEE Transactions on Acoustics, Speedh and Signal Processing, Vol. 36, No. 10, pp. 1651-1664, Oct. 1988.

[33] R. DeMarca and N. Jayant, *"An Algorithm for Assigning Binary Indices to the Codevectors of a Multidimensional Quantizer"*, Proceedings of the IEEE ICC-87, pp. 1128-1132, 1987.

[34] James G. Dunham and Robert M. Gray, *"Joint Source and Noisy Channel Trellis Coding"*, IEEE Transactions on Information Theory, Vol. IT-27, No. 4, pp. 516-519, July 1981.

[35] R. M. Fano, *"A Heuristic Discussion of Probabilistic Decoding"*, IEEE Transactions on Information Theory, Vol. IT-9, pp. 64-74, April 1963.

[36] Wade Nicholas Farell, *"Robust Speech Coding"*, PhD dissertation, Institute for Telecommunications Research University of South Australia, Australia, July 1999.

[37] Nariman Farvardin and Vinay Vaishampayan, *"Optimal Quantizer Design for Noisy Channels: An Approach to Combined Source-Channel Coding"*, IEEE Transactions on Information Theory, Vol. IT-33, no. 6, pp. 827-838, November 1987.

[38] Gerhard Fettweis, *"Algebraic Survivor Memory Management Design for Viterbi Detectors"*, IEEE Transactions on Communications, Vol. 43, No. 9, pp. 2458-2463, 1995.

[39] Gerhard Fettweis and Heinrich Meyr, *"High-Speed Parallel Viterbi Decoding: Algorithm and VLSI Architecture"*, IEEE Communications Magazine, pp. 46-55, 1991.

[40] G. Feygin, P. G. Gulak and P. Chow, *"A Multiprocessor Architecture for Viterbi Decoders with Linear Speedup"*, IEEE Transactions on Signal Processing, Vol. 41, No. 9, pp. 2907-2917, September 1993.

[41] G. Feygin and P. G. Gulak, *"Architectural Tradeoffs for Survivor Sequence Memory Management in Viterbi Decoders"*, IEEE Transactions on Communications, Vol. 41, No. 3, pp. 425-429, March 1993.

[42] Thomas R. Fischer and Michale W. Marcellin, *"Joint Trellis Coded Quantization/Modulation"*, IEEE Transactions on Communications, Vol. 39, No. 2, pp. 172-176, February 1991.

[43] G. D. Forney Jr., *"The Viterbi Algorithm"*, Proceedings of the IEEE, Vol. 61, pp. 268-278, March 1973.

[44] G.D. Forney, *"On Iterative Decoding and the Two-Way algorithm"*, Proceedings of the 1st Int. Symp. on Turbo-Codes, Sept. 1997, Brest, France.

[45] Volker Franz and John B. Anderson, *"Concatenated Decoding with a Reduced-Search BCJR Algorithm"*, IEEE Journal on Selected Areas in Communications, Vol. 16, No. 2, pp. 186-195, February 1998.

[46] Arnaud Galisson, *"Réseaux de Tri et Réseaux d'Interconnexion: Performances et Implantation VLSI"*, PhD dissertation, Ecole Nationale Supérieure des Télécommunications, Paris, France, July 1991.

[47] Luis González Pérez and Emmanuel Boutillon, *"Simplified Path Metric Updating in the M Algorithm for VLSI Implementation"*, Proceeding of the 2000 International Conference on Acoustics, Speech and Signal Processing (ICASSP2000), Istambul, Turkey, June 2000.

[48] Emmanuel Boutillon and Luis González Pérez, *"Trace Back Techniques Adapted to the Surviving Memory Management in the M Algorithm"*, Proceeding of the 2000 International Conference on Acoustics, Speech and Signal Processing (ICASSP2000), Istambul, Turkey, June 2000.

[49] D.J. Goodman and C.E. Sundberg, *"Transmission errors and forward error correction in embedded differential pulse code modulation"*, The Bell System Technical Journal, Vol. 62, pp. 2735-2764, November 1983.

[50] Robert M. Gray, *"Sliding-Block Source Coding"*, IEEE Transactions on Information Theory, vol. IT-29, no. 4, pp. 357-368, July 1975.

[51] Allen Gersho and Robert M. Gray, *"Vector Quantization and Signal Compression"*, Kluwer Academic Publishers, 1992.

[52] Robert M. Gray, *"Entropy and Information Theory"*, Springer-Verlag, 1990.

[53] P. G. Gulak and T. Kailath, *"Locally Connected VLSI Architectures for the Viterbi Algorithm"*, IEEE Journal on Selected Areas in Communications, vol. 6, no. 3, pp. 527-537, April 1988.

[54] David Haccoun and M.J. Ferguson, *"Generalized Stack Algorithms for Decoding Convolutional Codes"*, IEEE Transactions on Information Theory, vol. IT-21, pp. 638-651, November 1975.

[55] Joachim Hagenauer, *"Source-Controlled Channel Decoding"*, IEEE Transactions on Communications, vol. COM-43, no. 9, pp. 2449-2457, September 1995.

[56] D. S. Hirschberg, *"Fast Parallel Sorting Algorithms"*, Communciations of the ACM, Vol. 21, No. 8, pp. 657-661, August 1978.

[57] H. Imai, S. Hirakawa, *"A New Multilevel Coding Method Using Error Correction Codes"*, IEEE Transactions on Information Theory, vol. IT-23, pp. 371-377, may 1977.

[58] F. Jelinek, *"Tree Encoding of Memoryless Time-Discret Sources with a Fidelity Criterion"*, IEEE Transactions on Information Theory, vol. IT-15, pp. 584-590, Septembert 1969.

[59] F. Jelinek, *"A Fast Sequential Decoding Algorithm Using a Stack"*, IBM J. Res. Develop., Vol. 13, pp. 675-685, Nov. 1975.

[60] F. Jelinek and J.B. Anderson, *"Instrumentable tree encoding of information sources"*, IEEE Transactions on Information Theory, vol. IT-17, pp. 118-119, June 1971.

[61] D. E. Knuth, *"The Art of Computer Programming, Vol. 3: Sorting and Searching"*, Addison-Wesley, 1973.

[62] Ravi K. Kolaglota, Shu-Sun Yu and Joseph F. JáJá, *"VLSI Implementation of a Tree Searched Vector Quantizer"*, IEEE Transactions on Signal Processing, Vol. 41, No. 2, pp. 901-905, Feb. 1993.

[63] Clyde P. Kruskal, *"Searching, Merging and Sorting in Parallel Computation"*, IEEE Transactions on Computers, Vol. C-32, No. 10, pp. 942-946, October 1983.

[64] H. Kumazawa, M. Kasahara and T. Namekawa, *"A Construction of Vector Quantizers for Noisy Channels"*, Electronics and Engineering in Japan, Vol. 67-B, January, 1984.

[65] A.J. Kurtenback and P.A. Wintz, *“Quantizing for Noisy Channels”*, IEEE Transactions on Communication Technology, Vol. COM-17, No. 2, pp. 291-302, April 1969.

[66] Pierre Lavoie, David Haccoun and Yvon Savaria, *“A Systolic Architecture for Fast Stack Sequential Decoders”*, IEEE Transactions on Communications, Vol. COM-42, No. 2/3/4, pp. 324-335, February/March/April 1994.

[67] Yoseph Linde, Andrés Buzo and Robert M. Gray, *“An Algorithm for Vector Quantizer Design”*, IEEE Transactions on Communications, Vol. COM-28, No. 1, pp. 80-95, January 1980.

[68] Zheng-Li and Lei-Wei, *“Breadth-first Algorithm with Adaptive Forney Structure for ISI Channels”*, GLOBECOM 1998, NJ, USA, pp. 2853-2858, 1998.

[69] Shu Lin and Daniel J. Costello Jr., *“Error Control Coding: Fundamentals and Appications”*, Prentice-Hall, 1983.

[70] S. P. Lloyd, *“Least Squares Quantization in PCM”*, IEEE Transactions on Information Theory, vol. 28, no.2, pp.129-137, 1982.

[71] Michael Marcellin and Thomas R. Fischer, *“Trellis Coded Quantization of Memoryless and Gauss-Markov Sources”*, IEEE Transactions on Communications, Vol. COM-38, No. 1, pp. 82-93, January 1990.

[72] J. Max, *“Quantization for Minimum Distortion”*, IEEE Transactions on Information Theory, vol. 6, no.1, pp.7-12, 1960.

[73] J. L. Massey, *“Joint Source and Channel Coding in Communications Systems and Random Process Theory”*, pp. 279-293, J. K. Skwirzynski, Ed. The Netherlands; Sijthoff and Noordhoff, 1978.

[74] R. Mehlan and H. Meyr, *“Soft Output M-algorithm Equalizer and Trellis-Coded Modulation for Mobile Radio Communication”*, Vehicular Technology Society 42nd VTS Conference. Frontiers of Technology. From Pioneers to the 21st Century, vol.2, pp. 586-591, 1992.

[75] Byoung-Ki Min, *“Architecture VLSI pour le Décodeur de Viterbi”*, PhD dissertation, Ecole Nationale Supérieure des Télécommunications, Paris, France, June 1991.

[76] R.J. McEliece, *“The Theory of Information and Coding”*, Addison Wesley, 1977.

[77] James W. Modestino, D.G. Daut, *“Combined Source-Channel of Images”*, IEEE Transactions on Communications, Vol. COM-27, No. 11, pp. 1644-1659, November 1979.

[78] James W. Modestino, D.G. Daut and Acie L. Vickers *“Combined Source-Channel of Images Usign the Block Cosine Transform”*, IEEE Transactions on Communications, Vol. COM-29, No. 9, pp. 1261-1274, September 1981.

[79] James W. Modestino, Vaseud Bhaskaran and John B. Anderson, *"Tree Encoding of Images in the Presence of Channel Errors"*, IEEE Transactions on Information Theory, Vol. IT-27, No. 6, pp. 677-697, November 1981.

[80] James W. Modestino, Vaseud Bhaskaran, *"Robust Two-Dimensional Tree Encoding of Images"*, IEEE Transactions on Communications, Vol. COM-29, No. 12, pp. 1786-1798, December 1981.

[81] James W. Modestino, Vaseud Bhaskaran, *"Adaptive Two-Dimensional Tree Encoding of Images Using Spatial Masking"*, IEEE Transactions on Communications, Vol. COM-32, No. 2, pp. 177-189, February 1984.

[82] Seshadri Mohan and Arun K. Sood, *"A Multiprocess Architecture for the (M,L)-algorithm Suitable for VLSI Implementation"*, IEEE Transactions on Communications, Vol. COM-34, No. 12, pp. 1218-1224, December 1986.

[83] J. Omura, *"A source coding theorem for discrete-time sources"*, IEEE Transactions on Information Theory, Vol. IT-19, pp. 490-498, July 1973.

[84] Keshab K. Parhi, *"Pipelining in Dynamic Programming Architectures"*, IEEE Transactions on Signal Processing, Vol. 39, No. 6, pp. 1442-1450, June 1991.

[85] E. Paaske, S. Pedersen and J. Sparso, *"An Area-Efficient Path Memory Structure for VLSI Implementation of High Speed Viterbi Decoders"*, Integration, the VLSI Journal, No. 12, pp. 79-91, 1991.

[86] Christine Pépin, *"Quantification Vectorielle et Codage Conjoint Source-Canal par les Réseaux de Points"*, PhD dissertation, Ecole Nationale Supérieure des Télécommunications, Paris, France, December 1997.

[87] Andrei Popescu, *"Codage de Parole CELP á Excitation Vectorielle Codée par Treillis"*, PhD dissertation, Ecole Nationale Supérieure des Télécommunications, Paris, France, July, 1995.

[88] Franco P. Preparata, *"New Parallel Sorting Schemes"*, IEEE Transactions on Computers, Vol. C-27, No. 7, pp. 669-673, July 1978.

[89] J. G. Proakis, *"Digital Communications"*, McGraw-Hill, 1995.

[90] Charles M. Rader, *"Memory Management in a Viterbi Decoder"*, IEEE Transactions on Communications, Vol. COM-39, No. 9, pp. 1399-1401, September 1981.

[91] Jorge Rodríguez Guisantes, *"Codage Conjoint de Source et de Canal"*, PhD dissertation, Ecole Nationale Supérieure des Télécommunications, Paris, France, December 1997.

[92] G. Sauer, *"Suboptimum MLSE for Severe Echos Using Metric Forecast"*, Proceedings of the 6th Mediterranean Electrotechnical Conference, vol.1, pp. 578-581, 1991.

[93] Khalid Sayood, *"Introduction to Data Compression"*, Morgan Kaufmann Publishers, August 1995.

[94] K. Sayood and J.C. Borkenhagen, *"Use of Residual Redundancy in the Design of Joint Source-Channel Coders"*, IEEE Transactions on Communications, Vol. 39, No. 6, pp. 838-846, June 1991.

[95] K. Sayood, F. Liu, J. D. Gibson, *"A Constrained Joint Source/Channel Coder Design"*, IEEE Journal on Selected Areas in Communications, Vol. 12, No. 9, pp. 1584-1593, December 1994.

[96] Robert Sedgewick, *"Algorithms in C++"*, Addison Wesley, 1998.

[97] P.Secker and A. Perkins, *"Joint Source and Channel Trellis Coding of Line Spectrum Pair Parameters"*, Speech Communication, Elseiver Science Publisher 1992

[98] C. E. Shannon, *"A Mathematical Theory of Communications"*, Bell System Technical Journal, vol. 27, no. 3-4, pp.379-423; 623-656, 1948

[99] C. E. Shannon, *"Coding Theorems for a Discrete Source with a Fidelity Criterion"*, IRE Nat. Conv. Re. vol 4, pp. 142-163, 1959.

[100] C. B. Shung, H-D. Lin, R. Cypher, P. H. Siegel and Hermant K. Thapar, *"Area-Efficient Architectures for the Viterbi Algorithm – Part I: Theory"*, IEEE Transactions on Communications, Vol. COM-41, No. 4, pp. 636-644, April 1993.

[101] Stanley J. Simmons, *"Breadth-First Trellis Decoding with Adaptive Effort"*, IEEE Transactions on Communications, Vol. COM-38, No. 1, pp. 3-12, January 1990.

[102] Stanley J. Simmons, *"A Nonsorting VLSI Structure for Implementing the (M,L) Algorithm"*, IEEE Journal on Selected Areas in Communications, Vol. 6, No. 3, pp. 538-546, April 1988.

[103] S. J. Simmnons, *"A Bitonic-Sorter Based VLSI Implementation of the M-Algorithm"*, IEEE Pacific Rim Conference on Communications, Computers and Signal Processing, pp. 337-340, 1989.

[104] Peter A. Bengough and Stanley J. Simmnons, *"Sorting-Based VLSI Architecture for the M-Algorithm and T-Algorithm Trellis Decoders"*, IEEE Transactions on Communications, Vol. 43, No. 2/3/4, pp. 514-522, February/March/April 1995.

[105] S. J. Simmnons and P. Senyshyn, *"Reduced-Search Trellis Decoding of Coded Modulations over ISI Channels"*, GLOBECOM '90, vol.1, pp. 393-396, 1990.

[106] H. Skinnemoen, *"Combined Source-Channel Coding with Modulation Organized Vector Quantization (MOR-VQ)"*, GLOBECOM '94, 1994.

[107] Mikael Skoglund and Per Hedelin, *"Hadamard-Based Soft Decoding for Vector Quantization over Noisy Channels"*, IEEE Transactions on Information Theory, Vol. IT-45, No. 2, pp. 515-532, March 1999.

[108] Mikael Skoglund, *"Soft Decoding for Vector Quantization over Noisy Channels with Memory"*, IEEE Transactions on Information Theory, Vol. IT-45, No. 4, pp. 1293-1307, May 1999.

[109] J. Sparso, H. N. Jorgensen, E. Paaske, E. Pedersen and T. Rubner-Petersen, *"An Area-Efficient Topology for VLSI Implementation of Viterbi Decoders and Other Shuffle-Exchange Type Structres"*, IEEE Journal of Solid-State Circuits, Vol. 26, No. 2, pp. 90-97, February 1991.

[110] Lawrence C. Stewart, Robert M. Gray and Yoseph Linde, *"The Design of Trellis Waveform Coders"*, IEEE Transactions on Communications, Vol. COM-30, No. 4, pp. 702-710, April 1982.

[111] Clark D. Thompson, *"The VLSI Complexity of Sorting"*, IEEE Transactions on Computers, Vol. C-32, No. 12, pp. 1171-1184, December 1983.

[112] T. K. Truong, Ming-Tang Shih, Irving S. Reed and E. H. Satorius, *"A VLSI Design for a Trace-Back Viterbi Decoder"*, IEEE Transactions on Communications, Vol. COM-40, No. 3, pp. 616-624, March 1992.

[113] Vinay A. Vaishampayan and Nariman Farvardin, *"Joint Design of Block Source Codes and Modulation Signal Sets"*, IEEE Transactions on Information Theory, Vol. IT-38, No. 4, pp. 1230-1248, July 1992.

[114] A. Viterbi, *"Error Bounds for Convolutional Coding and an Asymptotic Optimum Decoding Algorithm"*, IEEE Transactions on Information Theory, Vol. IT-13, pp. 260-269, April 1967.

[115] A. Viterbi and J. Omura, *"Trellis encoding of memoryless discrete-time sources with a fidelity criterion"*, IEEE Transactions on Information Theory, Vol. IT-20, pp. 325-332, May 1974.

[116] Andrew J. Viterbi and Jim K. Omura, *"Principles of Digital Communications and Coding"*, McGraw-Hill, 1979.

[117] Leslie G. Valiant, *"Parallelism in Comparison Problems"*, SIAM J. Comput., Vol. 4, No. 3, pp. 348-355, September 1975.

[118] M. Yan, J.V. MacCanny and Y. Hu, *"VLSI Architectures for Vector Quantization"*, Journal of VLSI Signal Processing, Vol. 10, pp. 5-23, 1995.

[119] Min Wang and Thomas R. Fischer, *"Trellis-Coded Quantization Designed for Noisy Channels"*, IEEE Transactions on Information Theory, Vol. IT-40, No. 6, pp. 1792-1802, November 1994.

[120] Wen Xu, Joachim Hagenauer and Jürgen Hollmann *"Joint Source-Channel Coding Using the Residual Redundancy in Compressed Images"*, Proceedings of the IEEE International Conference on Communications, pp. 142-148, 1996.

[121] Seyed Bahram Zahir Azami, *"Codage Conjoint Source/Canal, Protection Hiérarchique"*, PhD dissertation, Ecole Nationale Supérieure des Télécommunications, Paris, France, May 1999.

[122] K. Zeger and A. Gersho, *"Pseudo-Gray Coding"*, IEEE Transactions on Communications, Vol. COM-38, No. 12, pp. 2147-2156, 1979.