VLSI Architectures for the Forward-Backward Algorithm

by

Warren Jeffrey Gross

A Thesis submitted in conformity with the requirements
for the degree of Master of Applied Science,
Department of Electrical and Computer Engineering,
University of Toronto

0-612-45609-9

VLSI Architectures for the Forward-Backward Algorithm

Warren Jeffrey Gross

Master of Applied Science, 1999

Department of Electrical and Computer Engineering

University of Toronto

Abstract

The forward-backward algorithm, also known as the BCJR or MAP algorithm, is a detection algorithm that provides soft reliability estimates. This thesis discusses issues relevant to the practical implementation of the forward-backward algorithm. Two applications are chosen for more detailed study: (i) turbo decoding and (ii) soft-output detection of class-IV partial response signalling. A novel circuit is introduced that eliminates the need for a lookup table in the computational kernel of the forward-backward algorithm. The design and implementation of an FPGA-based turbo decoder is presented. The difference-metric forward-backward algorithm is derived for class-IV partial response signalling. A test chip was designed in a 0.5 μm CMOS process and is expected to operate at speeds greater than 120 Mbps. The core area is 0.81 mm^2 and the overall silicon area is 7.3 mm^2.

Acknowledgments

This thesis would not have been possible without the help of many people and organizations. I would like to use this space to make special mention of my advisor, colleagues, friends and family who helped me towards my goals.

I wish to thank my advisor Professor Glenn Gulak for his support, ideas and encouragement throughout the course of this work. I hope to continue my collaboration with him in the future.

Financial support from NSERC as well as fabrication support from CMC are greatly appreciated.

I also wish to thank Professors Frank Kschischang, Emmanuel Boutillon, Amir Banihashemi and Saeed Gazor for their enlightening discussions.

Special mention goes out to Vincent Gaudet for the work which we did together on PRONTO and the interleaver project. Also, thanks to Jason Podaima for helping with the place and route of the PRONTO-1 chip.

Thanks to Dave, Marcus, Jaro and Peter for putting up with all my questions about the CAD tools.

I would like to thank all of my colleagues in the Communications Algorithms Laboratory at the U. of T.

Thanks to all my friends in LP392: Ali, Andy, Jason A., Javad, Jeff, Jordan, Ken, Khalid, Mark, Nirmal, Qiang, Rob, Sandy, Tooraj, Vaughn, Wai Ming and Yaska.

I am extremely grateful to my parents for their love, support and encouragement.

"Now it is a strange thing, but things that are good to have and days that are good to spend are soon told about, and not much to listen to; while things that are uncomfortable, palpitating, and even gruesome, may make a good tale, and take a deal of telling anyway."

J. R. R. Tolkien, *The Hobbit*

Contents

List of Figures

List of Tables

List of Symbols

$\alpha_k(s)$	Forward state probability at state s and time k
$\beta_{k-1}(s)$	Backward state probability at state s and time k
$\gamma_k(s', s)$	Branch probability between states s' and s at time k
Δ_k	Difference metric
$\eta_k(s', s)$	Sum of backward state metrics and branch metrics at time k
ν	Memory length of encoder or intersymbol interference channel
ρ	Weight of the delayed input to the adder in a PR4 filter
σ^2	Noise variance
a_k	Shift register input
A_k	Forward difference metric
$A_k(s)$	Forward state metric at state s and time k
B_k	Backward difference metric
$B_k(s)$	Backward state metric at state s and time k
$\mathbf{c}_k(s', s)$	Expected symbol along the branch from state s' to state s
D	Unit delay
D_L	Learning period
E_b/N_0	Bit energy-to-noise density ratio
f	Turbo decoder clock speed in Hz
$f_{\mathbf{y}}(\mathbf{y})$	Likelihood function.
G_{k+1}^{-1}, G_{k+1}^{+1}	Branch metrics for the log-difference-metric forward-backward algorithm
$G_k(s', s)$	Branch metric between states s' and s at time k

I	Interleaver
I^{-1}	Deinterleaver
k	Time index
$L(u)$	Log-likelihood ratio corresponding to transmitted symbol u
M_b	Number of symbols decoded per learning recursion
MAX	Maximum operator
MAX*	Maximum adjusted by a correction factor
n_k	AWGN noise sample
N	Block length
$P(u\|\mathbf{y})$	A-posteriori probability
$P_e(u, \mathbf{y})$	Probability of symbol error
R	Rate
s	Present state
s'	Predecessor state
S	Number of states
T	Symbol period
$\hat{u}_k$	Information bit
W_k	Extrinsic information
WL	Wordlength
WL_I	Number of integer bits in a fixed point word
WL_F	Number of fractional bits in a fixed point word
$x_{s,k}$	Systematic output of a turbo encoder
$x_{c1,k}$	First coded output of a turbo encoder

$x_{c2,k}$	Second coded output of a turbo encoder
$\mathbf{x}_k$	Modulated symbol
$y_{s,k}$	Received systematic value at the input of a turbo decoder
$y_{c1,k}$	Received first coded value at the input of a turbo decoder
$y_{c2,k}$	Received second coded value at the input of a turbo decoder
$\mathbf{y}_k$	Received symbol
z	A-priori information

List of Acronyms

ACS	Add-Compare-Select
APP	A-Posteriori Probability
AWGN	Additive White Gaussian Noise
BCJR	Bahl, Cocke, Jelinek and Raviv
BER	Bit Error Rate
CCS	Compare-Compare-Select
DMFB	Difference-Metric Forward-Backward algorithm
DMVA	Difference-Metric Viterbi Algorithm
FB	Forward-backward algorithm
FPGA	Field Programmable Gate Array
IAU	Interval Adjustment Unit
ISI	Intersymbol Interference
log-FB	Forward-Backward algorithm in the logarithmic domain
log-ML	Maximum Likelihood rule in the logarithmic domain
max-log-FB	Forward-Backward algorithm in the log domain using MAX instead of MAX^*
MAP	Maximum-A-Posteriori
MAX-DMFB	Difference-Metric Forward-Backward algorithm using MAX instead of MAX^*
ML	Maximum Likelihood
PR4	Class-IV Partial Response
RSC	Recursive Systematic Convolutional code
SOVA	Soft Output Viterbi Algorithm

SNR	Signal-to-Noise Ratio
TM-2	The Transmogrifier-2 field programmable system

Chapter 1

Introduction

1.1 Motivation

Error correcting codes in digital communications permit reliable transmission of data in noisy channels. The idea is to transmit some extra information along with the original message that in some way describes the intended message. The relationship between the extra and original information is known and can be exploited in the decoder to make reliable bit decisions.

Turbo codes, introduced in 1993 [BGT93] are the best known error correcting codes with a practical decoding algorithm. They have bit error rate performance close to the Shannon limit. The heuristic iterative decoding algorithm provides remarkable results at a very low complexity compared to codes with similar performance. In recent years, researchers have made a lot of progress in understanding these codes and in suggesting many promising applications. The focus of the research is now starting to turn towards practical hardware implementation issues.

The main component of a turbo decoder is the forward-backward algorithm. The forward-backward algorithm differs from the well-known Viterbi algorithm in that it provides 'soft' information about the reliability of each bit that it decodes. This soft information can be used in a subsequent use of the algorithm to improve its initial estimate. Although known for about thirty years, the forward-backward algorithm developed an undeserved reputation as being too difficult to implement. Even if it could be simplified somewhat, it was unlikely to beat the Viterbi algorithm for simplicity and there was no compelling need for soft outputs. The focus has suddenly shifted to the forward-backward algorithm as it provides optimal symbol-by-symbol soft outputs for turbo decoders. Initial implementations of turbo decoders used the SOVA algorithm, a soft-output variant of the Viterbi algorithm that was less complex than the forward-backward algorithm but did not provide optimum soft outputs. It would be interesting to design a turbo decoder using the forward-backward algorithm and address the implementation issues.

Another interesting area is the detection of signals in magnetic recording. The principles of detection are similar to error correcting coding: the correlation between transmitted signals is known and can be exploited in the receiver. A standard technique for detecting one of the popular signalling schemes, class-IV partial response, is called the difference metric Viterbi algorithm. Are soft outputs useful for this type of system? If so, can an analogous difference metric forward-backward algorithm be derived to compete with the difference metric Viterbi algorithm?

1.2 Objectives

The objective of this thesis is to present examples of practical implementations of the forward backward algorithm and to propose suitable VLSI architectures. Specifically, we have chosen to implement a turbo decoder and a detector for class-IV partial response signalling.

1.3 Organization of the Thesis

This thesis contains five chapters and one appendix. Chapter 2 is a review of the forward-backward algorithm and simplifications for its practical implementation. Chapter 3 presents the design of a turbo decoder for the reconfigurable TM-2 FPGA system. Chapter 4 describes the design and implementation of a soft-output decoder for class-IV partial response. Chapter 5 is a summary of the work and makes some suggestions for future research. Appendix A is a detailed listing of the equations of the forward-backward algorithm used to implement the turbo decoder in Chapter 3.

Chapter 2

The Forward-Backward Algorithm

The forward-backward algorithm is a detection algorithm that provides reliability estimates for each symbol that it decodes. Virtually ignored for many years because of its perceived complexity, interest in this algorithm has been reignited by the recent discovery of turbo codes. This chapter is a review of the forward-backward algorithm and techniques for its practical implementation. Section 2.1 defines the problem which the rest of the thesis is dedicated to solving: extracting the intended message from a noisy signal. A solution to this problem, the forward-backward algorithm is described in Section 2.2. A practical method for implementing the forward-backward algorithm is presented in Section 2.3. The main points of this chapter are summarized in Section 2.4.

2.1 Detection in the Presence of Noise

Signal *detection* is the attempt to recover a discrete-valued transmitted signal that has been corrupted by noise. In this thesis, we are only concerned with a discrete-time system and therefore will consider time in intervals of T seconds where T is called the symbol period. Fig. 2-1 is a block diagram of a basic digital communication system. The messages to be transmitted are equally likely binary symbols u_k where the subscript k refers to the time index. After transmission through an Additive White Gaussian Noise (AWGN) channel the input at the receiver, $\mathbf{y}_k$ is the sum of a noise sample and the modulated signal $\mathbf{x}_k$:

$$\mathbf{y}_k = \mathbf{x}_k + \mathbf{n}_k \tag{2-1}$$

The detector is a combined demodulator and symbol estimator that gives estimates $\hat{u}_k$ of the original message symbols.

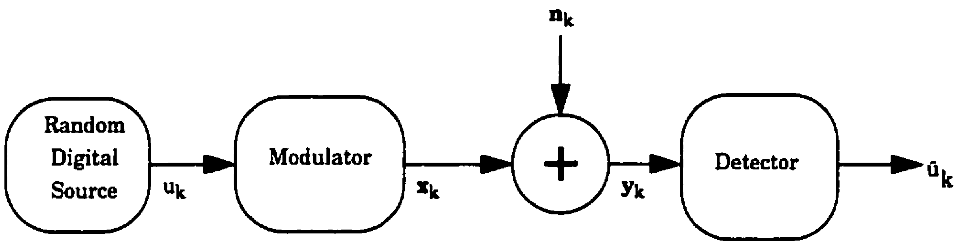


Fig. 2-1: The basic digital communication system.

2.1.1 The Maximum A-Posteriori and Maximum Likelihood Rules

The best possible detector is one that makes the fewest errors in estimating the message symbols. In other words, an optimum detector minimizes the probability of symbol error [Haykin88]:

$$P_e(u, \mathbf{y}) = 1 - P(u|\mathbf{y}) \tag{2-2}$$

As seen in the above equation, this is equivalent to maximizing $P(u|\mathbf{y})$. This quantity is called the *a-posteriori probability* and is the probability of the information symbol given that the noisy value has been received. A *maximum-a-posteriori* (MAP) decision rule can be implemented by calculating the a-posteriori probabilities of the two possible message symbols and then choosing symbol u (0 or 1) that results in the largest. Using Bayes' rule on Equation 2-2 we can write:

$$P(u|\mathbf{y}) = \frac{P(u)f_{\mathbf{y}}(\mathbf{y}|u)}{f_{\mathbf{y}}(\mathbf{y})} \tag{2-3}$$

where $f_{\mathbf{y}}(\mathbf{y}|u)$ is called the *likelihood function*. If the message symbols are equally likely then the MAP rule reduces to finding the message symbol that maximizes the likelihood function. It is often more convenient to apply the *maximum-likelihood* (ML) rule in the log domain by maximizing the logarithm of the likelihood function.

2.1.2 Sequence Detection and the Viterbi Algorithm

The error performance of the system can be improved if a constraint is imposed on the sequence of transmitted symbols. A channel in which the transmitted symbols are dependent on the past history of transmitted symbols is said to have *memory*. The channel memory can be naturally present or explicitly inserted. One example of a channel with memory is an *intersymbol interference* (ISI) channel. Convolutional coding involves inserting memory in the transmitter. Fig. 2-2 shows the different positions of the memory between the two techniques. Regardless of the source of the memory, the detector can use the constraint imposed on the received symbols to make more intelligent decisions. Convolutional codes will be described in greater detail in Chapter 3 while Chapter 4 deals with intersymbol interference channels.

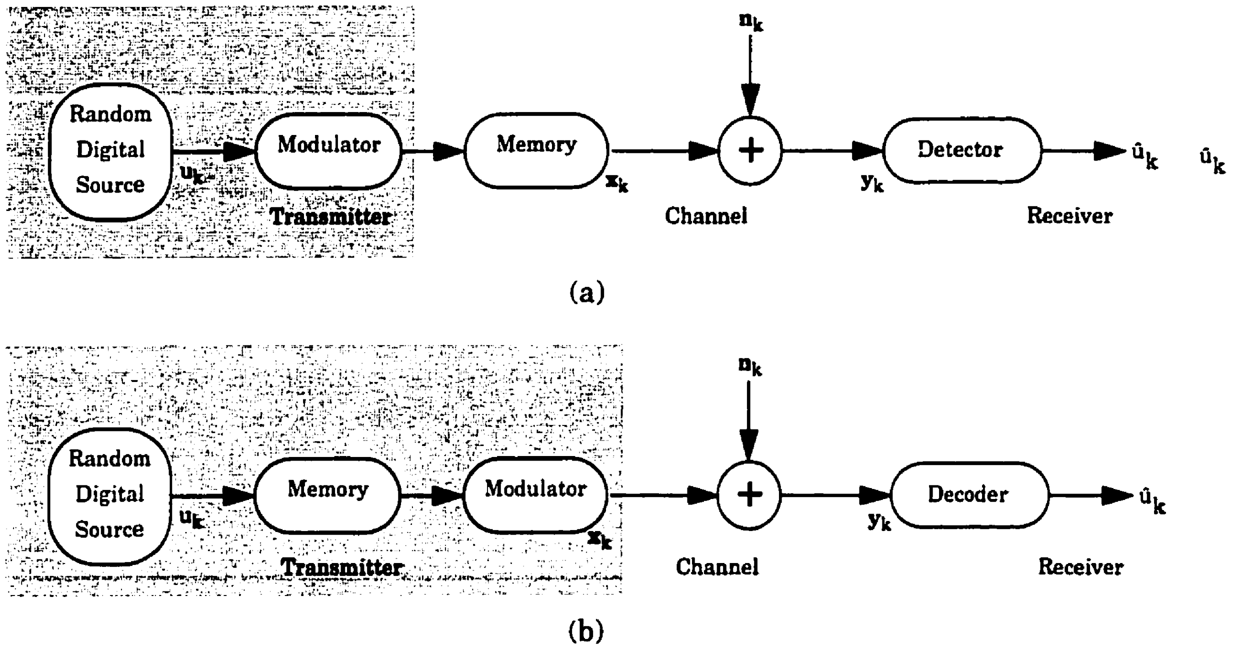


Fig. 2-2: Position of the memory in an ISI channel (a) and a convolutional coding scheme (b).

Consider the shift register structure in Fig. 2-3, which is the encoder for a convolutional code with a memory length (ν) of two. We say the *rate* (R) of the code is 1/2 since there are two output bits for every input bit. The *state* of the encoder at a given time is defined by the contents of the shift register. For example, the $\nu = 2$ shift register shown here has four possible states, namely "00", "01", "10" and "11". A state machine diagram which shows the valid state transitions is shown in Fig. 2-4. The edges of the diagram are labelled with the input bit / output bits. A more convenient representation can be derived by adding a time axis. Each path from the left hand side through the right hand side of the resulting *trellis* diagram in Fig. 2-5 corresponds to a unique transmitted sequence of bits.

The problem of detection (which we will call decoding in the case of a convolutional code) can now be cast in terms of operations on the trellis. The well-known Viterbi algorithm (VA) [Viterbi67][Omura69][Forney73] is an application of dynamic programming to digital

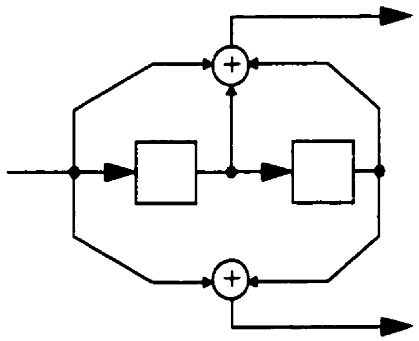


Fig. 2-3: A convolutional rate R =1/2 encoder with memory length v = 2.

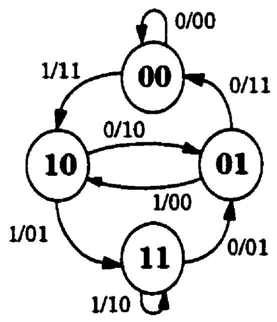


Fig. 2-4: A state transition diagram for the convolutional encoder shown in Fig. 2-3.

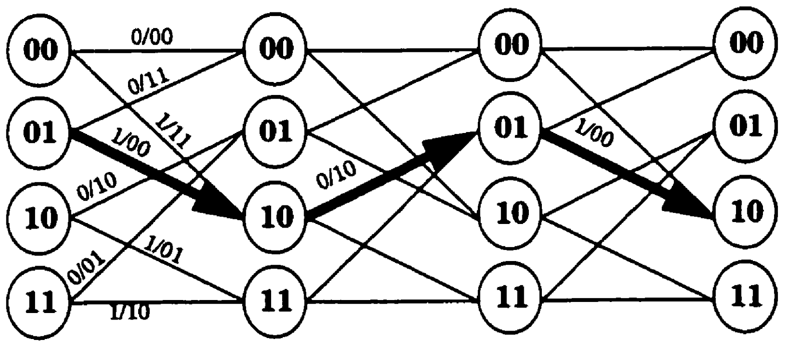


Fig. 2-5: A 3 stage trellis diagram corresponding to the encoder of Fig. 2-3 and the state diagram of Fig. 2-4. The highlighted path corresponds to a message sequence of {1,0,1} given the starting state 01

communications that operates on a trellis. If the edges (branches) in the trellis are labelled with a number that is proportional to the probability of that branch being taken, then the path with the largest accumulated number will be the most likely path to have occurred. In practice, the negative likelihood function is used instead, which reformulates the problem to finding the minimum length path through the trellis. For an AWGN channel, a *branch metric* is simply the Euclidean distance between the received and expected transmitted symbols. Therefore, the Viterbi algorithm, in choosing the sequence with the lowest accumulated metric, performs maximum likelihood sequence detection.

The key to the algorithm is the observation that at time k the shortest path must contain the shortest path up to time k-1 and therefore the other paths leading up to time k-1 can be thrown away. The procedure consists of two steps. In the first step, the accumulated metrics are calculated in the direction of increasing time in a *forward* recursion through the trellis. The accumulated metrics, called *forward state metrics* are calculated at each state at time k as

$$A_k(s) = \underset{s'}{\text{MIN}}\,(G_k(s', s) + A_{k-1}(s')) \tag{2-4}$$

where $A_k(s)$ is the forward state metric at state s at the current time interval, $A_{k-1}(s')$ is the forward state metric at a predecessor state s' at the previous time interval and $G_k(s', s)$ is the branch metric associated with the branch between states s' and s. The computational kernel of Equation 2-4 which is the critical path of the algorithm is the so-called Add Compare Select (ACS) operation (see Fig. 2-6). In the second step, the shortest path is determined by tracing back through the trellis according to the decisions made at each time interval by the forward recursion. Using the fact that competing paths are likely to have merged into the shortest path at about 5v time intervals back from the current time [ClarkC81], a practical real-time implementation can be built using a finite amount of memory. The two methods for performing traceback are the *register exchange* method and the *pointer traceback* method. The reader is directed to [ClarkC81][Rader81][FeyginG91] for descriptions of how to implement the two traceback methods.

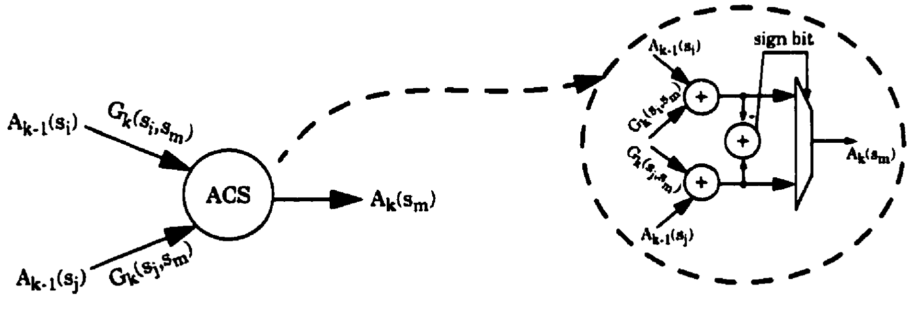


Fig. 2-6: The forward recursion in the Viterbi algorithm.

2.2 The Forward-Backward Algorithm

2.2.1 Soft-Output Algorithms

It can be useful for a decoding algorithm to provide an estimate of the reliability of the decoded bits. The '*soft*' reliability values can be used in some way to adjust the decoding algorithm to provide better performance. As far back as 1973, Forney proposed the use of 'augmented outputs' from the Viterbi algorithm as a measure of reliability of the decoding process [Forney 73]. Forney's heuristic idea to use the difference in state metrics between the best path and the next shortest path led to the soft-output Viterbi algorithm described by Batail [Battail87] and Hagenauer and Hoeher [HagenauerH89].

It is important to define what is meant by soft information. The reliability of a decoded bit is best described by the a-posteriori probability (APP) $P(u|\mathbf{y})$. For an estimate of bit u (-1/+1) having received symbol **y** we define the *optimum soft output* as:

$$L(u) = \ln\frac{P(u = 1|\mathbf{y})}{P(u = -1|\mathbf{y})} \tag{2-5}$$

which is called the *log-likelihood ratio* (LLR). The LLR is a convenient measure since it encapsulates both soft and hard bit information in one number. The sign of the number corresponds to the hard decision while the magnitude gives a reliability estimate.

The LLR can be easily computed by noticing that a decoding algorithm that uses the maximum a-posteriori (MAP) rule inherently calculates the APPs required. In fact, once determined and used to choose the most likely bit, the APPs are discarded, throwing away useful information. MAP algorithms have been proposed by several authors [ChangH66] [AbendF70][BCJR74] but were generally ignored because the Viterbi algorithm can provide nearly identical hard outputs with less computational effort.

In this thesis, we will focus on the implementation of what we call the "forward-backward algorithm" of [ChangH66][BCJR74]. In the literature, this algorithm is often referred to as the MAP algorithm or the BCJR algorithm, which are both somewhat inaccurate names. Abend and Fritchman proposed a MAP decoder that is optimal under the constraint of fixed decoding delay [AbendF70]. A simplified suboptimal detector (SPS) based on the Abend and Fritchman algorithm was derived in [ErfanianP89]. The Abend and Fritchman type algorithm has the drawback that the complexity of the decoder is exponentially related to the fixed decision delay. In [LVS95] a MAP detector is derived that has linear dependence on the delay. In [Battail87] and [HagenauerH89] the soft-output Viterbi algorithm (SOVA) is derived. The SOVA has received a lot of attention lately due to its ease of implementation even though it provides suboptimal soft outputs. In our opinion, the forward-backward algorithm is a feasible alternative to SOVA especially when optimum soft outputs are required (for example in turbo decoders). The rest of this thesis is devoted to presenting architectures and examples of the practical application of the forward-backward algorithm.

2.2.2 Description of the Forward-Backward Algorithm

This description of the algorithm is based on [BCJR74] and [HOP96] to which the reader is referred to for a detailed derivation. The algorithm is based on the same trellis as the Viterbi algorithm. The algorithm is performed on a block of N received symbols which corresponds to a trellis with a finite number of stages N. We will choose the transmitted bit u_k from the set of {-1,+1}. Upon receiving the symbol $\mathbf{y}_k$ from the AWGN channel with noise variance σ^2 we calculate the *branch probability* of the transition from state s' to state s as

$$\gamma_k(s', s) = \exp\left(\frac{-1}{2\sigma^2}\|\mathbf{y}_k - \mathbf{c}_k(s', s)\|^2\right) \tag{2-6}$$

where $\mathbf{c}_k(s', s)$ is the expected symbol along the branch from state s' to state s.

The algorithm consists of three main steps:

- **Forward Recursion.** The *forward state probability* of being in each state of the trellis at each time k given the knowledge of all the previous received symbols is recursively calculated and stored:

$$\alpha_k(s) = \sum_{s'}(\alpha_{k-1}(s')\gamma_k(s', s)) \qquad k = 1 \ldots N-1 \tag{2-7}$$

The recursion is initialized by forcing the starting state to state 0 and setting

$$\begin{aligned} \alpha_0(0) &= 1 \\ \alpha_0(s) &= 0 \qquad s \neq 0 \end{aligned} \tag{2-8}$$

- **Backward Recursion.** The *backward state probability* of being in each state of the trellis at each time k given the knowledge of all the future received symbols is recursively calculated and stored:

$$\beta_{k-1}(s') = \sum_{s}(\beta_k(s)\gamma_k(s', s)) \qquad k = N \ldots 2 \tag{2-9}$$

The recursion is initialized by forcing the ending state to state 0 and setting

$$\begin{aligned} \beta_N(0) &= 1 \\ \beta_N(s) &= 0 \qquad s \neq 0 \end{aligned} \tag{2-10}$$

The trellis termination condition requires the entire block to be received before the backward recursion can begin.

- **Log-Likelihood Ratio Calculation.** The output LLR for each symbol at time k is calculated as

$$L(u_k) = \ln\frac{P(u_k = +1|y_k)}{P(u_k = -1|y_k)}$$

$$= \ln\frac{\sum\limits_{\substack{(s',s)\\u_k=+1}} \alpha_{k-1}(s')\gamma_k(s',s)\beta_k(s)}{\sum\limits_{\substack{(s',s)\\u_k=-1}} \alpha_{k-1}(s')\gamma_k(s',s)\beta_k(s)} \tag{2-11}$$

where the upper summation is over all branches with input label +1 and the lower summation is over all branches with input label -1. The procedure is depicted graphically in Fig. 2-7 for a short 5-stage trellis.

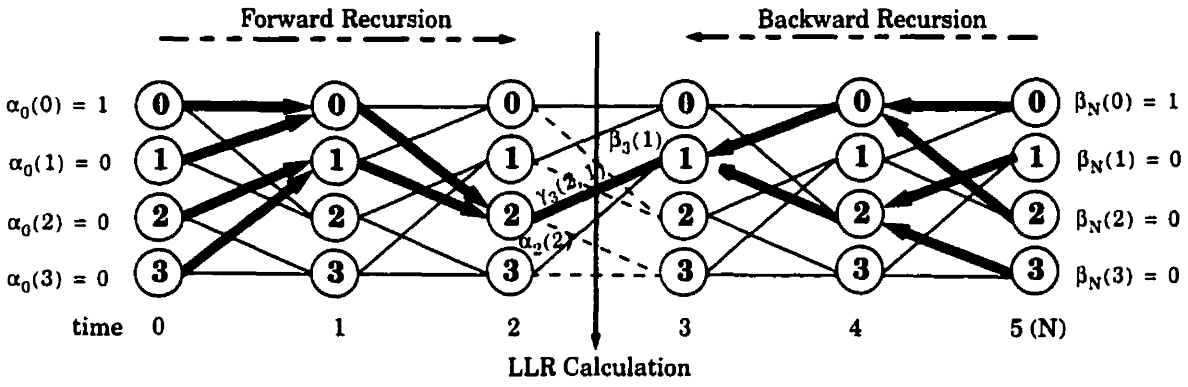


Fig. 2-7: Trellis processing in an N = 5 stage 4-state Forward-Backward algorithm corresponding to the encoder of Fig. 2-3. Shown is the calculation of one particular term in the lower summation of Equation 2-11. The terms in the upper summation are dashed while the terms in the lower summation are solid.

2.2.3 The Forward-Backward Algorithm in the Logarithmic Domain

The forward-backward algorithm was virtually ignored for many years because of the difficulty in implementing efficient exponentiation and multiplication. If the algorithm is implemented in the logarithmic domain like the Viterbi algorithm, then the multiplications become additions and the exponentials disappear. Addition is transformed according to the rule described in [KingsburyR71]. Following Erfanian and Pasupathy, who first applied this rule to the Abend and Fritchman MAP algorithm [ErfanianP89], the additions are replaced using the Jacobi logarithm:

$$\begin{aligned} \ln(e^X + e^Y) &= \mathrm{MAX}(X, Y) + \ln(1 + e^{-|X-Y|}) \\ &= \mathrm{MAX}^*(X, Y) \end{aligned} \tag{2-12}$$

which we call the MAX* operation to denote that it is essentially a maximum operator adjusted by a correction factor. The second term, a function of the single variable X-Y, can be precalculated and stored in a small lookup table with negligible effects on performance [ErfanianP89].

The forward-backward algorithm will now be restated in the logarithmic domain. As with the Viterbi algorithm, logarithms of probabilities are referred to as *metrics*. Define the new quantities:

- **Branch Metrics**:

$$G_k(s', s) = \ln(\gamma_k(s', s)) \tag{2-13}$$

- **Forward State Metrics**:

$$A_k(s) = \ln(\alpha_k(s)) \tag{2-14}$$

- **Backward State Metrics**:

$$B_k(s) = \ln(\beta_k(s)) \tag{2-15}$$

The branch metric calculation eliminates the exponential:

$$G_k(s', s) = \frac{-1}{2\sigma^2}\|\mathbf{y}_k - \mathbf{c}_k(s', s)\|^2 \tag{2-16}$$

The forward state metric recursion becomes:

$$A_k(s) = \underset{s'}{\mathrm{MAX}^*}(A_{k-1}(s') + G_k(s', s)) \qquad k = 1 \ldots N-1 \tag{2-17}$$

with initial conditions:

$$\begin{aligned} A_0(0) &= 0 \\ A_0(s) &= -\infty \qquad s \neq 0 \end{aligned} \tag{2-18}$$

The backward state metric recursion becomes:

$$B_{k-1}(s') = \underset{s}{\mathrm{MAX}^*}(B_k(s) + G_k(s', s)) \qquad k = N \ldots 2 \tag{2-19}$$

with initial conditions:

$$\begin{aligned} B_N(0) &= 0 \\ B_N(s) &= -\infty \qquad s \neq 0 \end{aligned} \tag{2-20}$$

The dynamic range of the metrics is much lower than the associated probabilities. Since each probability has a value of less than or equal to one, each multiplication results in a smaller and smaller number. The metrics tend to grow much more slowly than the probabilities.

The output LLR becomes:

$$L(u_k) = \underset{\substack{(s',s) \\ u_k = +1}}{\mathrm{MAX}^*}(A_{k-1}(s') + G_k(s',s) + B_k(s)) - \underset{\substack{(s',s) \\ u_k = -1}}{\mathrm{MAX}^*}(A_{k-1}(s') + G_k(s',s) + B_k(s)) \tag{2-21}$$

The computational kernel of the algorithm is now the MAX* operation, which is analogous to the MIN operation in the Viterbi algorithm adjusted by a correction factor (see Fig. 2-8). If the correction factor is ignored then MAX* reduces to MAX and the hard decisions are identical to those produced by the Viterbi algorithm and the soft decisions equivalent to those from the SOVA[1] [FBLH98].

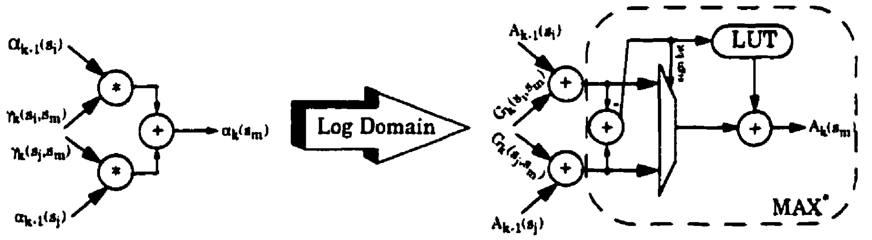


Fig. 2-8: Log-domain processing in the forward recursive equation. Multiplications become additions and additions become the MAX* operation. The shaded components are removed for the approximate algorithm.

2.3 The Sliding Window Forward-Backward Algorithm

The forward-backward algorithm as stated is a block processing algorithm. The constraint that the ending state be known implies that an entire block of data needs to be received before the backward recursion can begin. The memory requirements are therefore quite large. For a rate R code with memory length v (2^v states) and a block length of N, we have to store NR input words and 2^vN forward state metrics so that they can be used during the

1. The SOVA proposed in [Battail87] has higher complexity than the algorithm proposed in [HagenauerH89].

backward recursion to calculate the LLRs. The branch metrics may require additional storage, or they may be recalculated as needed. The basic amount of storage required is therefore $N(R+2^v)$ words. Unfortunately, turbo codes, one of the most important applications, require large values of N for good performance. In addition, high-speed applications may not be able to tolerate the delays of block processing. To overcome these problems, a modification of the algorithm that operates over a small sliding window allows continuous processing with a fixed latency [DawidM95]. The idea is to relax the criterion that the ending state be known and to consider each ending state as equally likely. Viterbi makes a comparison with the Viterbi algorithm's ability to "start cold" in any state [Viterbi98][2]. If the backward state probabilities are initialized to:

$$\beta_N(s) = \frac{1}{2^v} \qquad \forall s \tag{2-22}$$

then after a *learning* or *synchronization* period the state probabilities (or metrics) will be as good as those calculated starting from a known ending state. A learning period of 6 memory lengths is usually enough to achieve nearly the performance of the optimal algorithm [Viterbi98]. The memory requirements are reduced to D_L+2 words where D_L is the learning period. Fig. 2-9 shows the simplest form of the sliding window algorithm. The trade-off is that the new algorithm has to compute D_L backward stages for each decoded bit. A compromise between memory and computational complexity is given in [BDMP97] where the backward state metrics, determined by the learning recursion, are used to decode a block of M_b symbols. The structure of Fig. 2-9 which corresponds to $M_b = 1$ is still useful in some applications due to its ease of implementation. An example is given in Chapter 4 that is simple enough to efficiently utilize this architecture.

The more efficient sliding window algorithm may be derived by dividing the backwards recursion into two distinct parts. First, a learning recursion of D_L steps is performed to generate reliable backward state metrics. The generated metrics are then used to decode a block of $M_b = D_L$ symbols. The algorithm is best described by the time-space diagram in Fig. 2-10 introduced in [DawidM95] which divides the trellis into blocks of D_L stages. We will now describe the algorithm, referring to Fig. 2-10.

2. The Forward-Backward algorithm's recursive step is really just two Viterbi-like algorithms running in opposite directions with a slightly different computational kernel. In the case of the approximate algorithm, the kernels are identical.

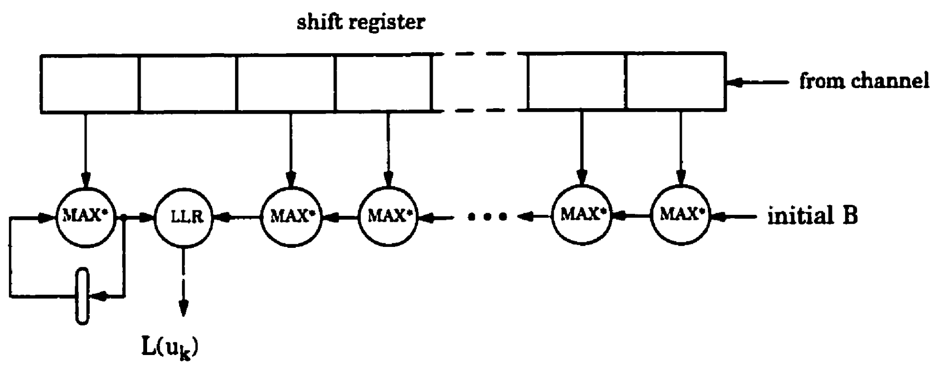


Fig. 2-9: The log-domain sliding window Forward-Backward algorithm for $M_b = 1$ [DawidM95]. The shaded boxes represent memory storage. The branch metric generators are not shown.

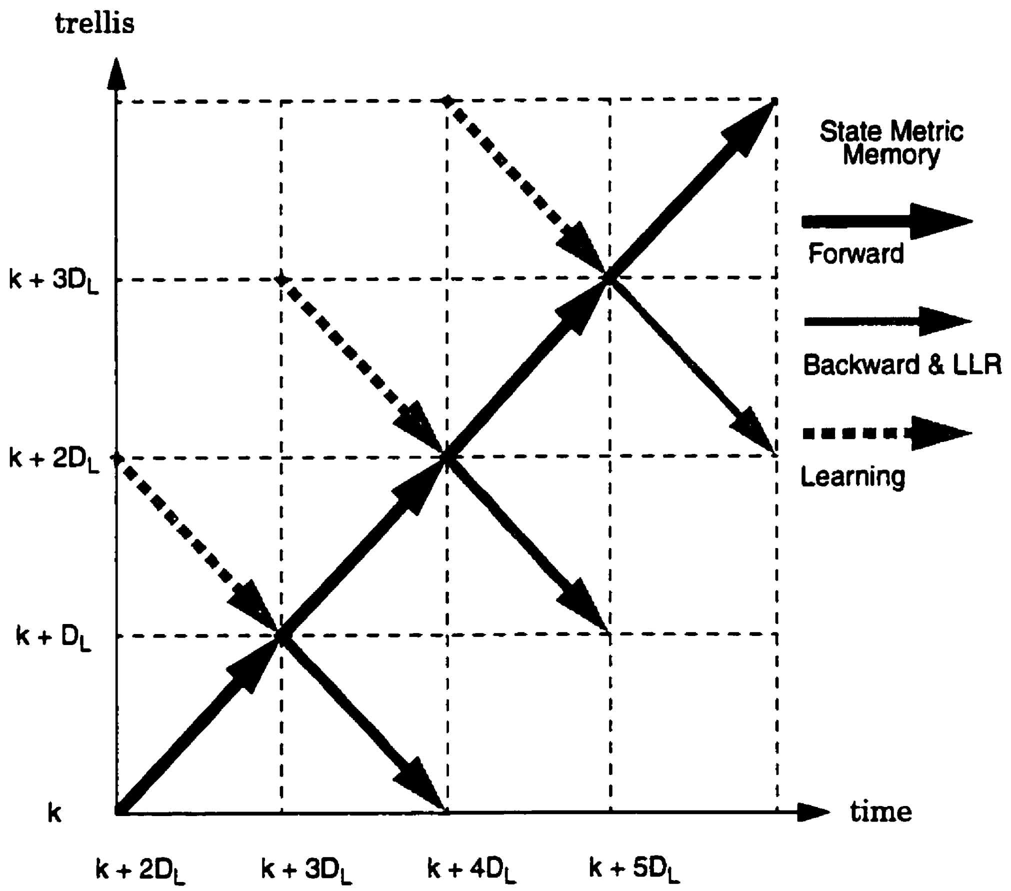


Fig. 2-10: Space-time diagram for the sliding window Forward-Backward algorithm with $M_b = D_L$ [DawidM95]. The shaded areas represent the memory for the forward state metrics.

The algorithm begins once the first $2D_L$ symbols have been received and stored in the input memory (not shown in the diagram). Let's consider what happens to a given stage of the trellis, say stage $k + D_L + j$ where $0 < j < D_L$. If we draw a horizontal line across the diagram, it will cut through one of each kind of arrow, forward, backward and learning. Therefore, we see the progression through time. The stage is visited by the learning recursion, then the forward recursion and lastly the backward recursion. Then, in the next block of time, the forward recursion proceeds using the results of the learning recursion. The soft outputs are generated during the backward recursion, in reverse order[3]. The hardware requirements can be found by drawing a vertical line, say at time $k + 3D_L + j$, $0 < j < D_L$. It can be seen that three recursion processors are required to run simultaneously, one each for the forward, backward, and learning recursions as shown in Fig. 2-11. Each processor consists of a branch metric generator, $S = 2^v$ MAX^* units and the appropriate trellis routing along with S registers. The fully parallel structure ensures a throughput of 1 decoded bit/cycle after an initial latency of $4D_L$. The memory requirements for the forward state metrics can also be found this way. At any point after $k + 3D_L$, the state metric memory required will be D_L at any point in time. Therefore, the state metric memory requirements for a S state trellis are SD_L. The structure of the decoder is shown in Fig. 2-12, which is derived from [DawidM95]. The input memory needs three independent banks, each of D symbols. The input symbols are directed to the correct branch metric generator (BMG) and then to the appropriate processing element (PE). A simplification in the soft output calculation may be obtained by noticing that the term $B_k(s) + G_k(s', s)$ in Equation 2-21 has already been calculated by the backward recursion in Equation 2-19 and may be supplied directly to the soft output unit.

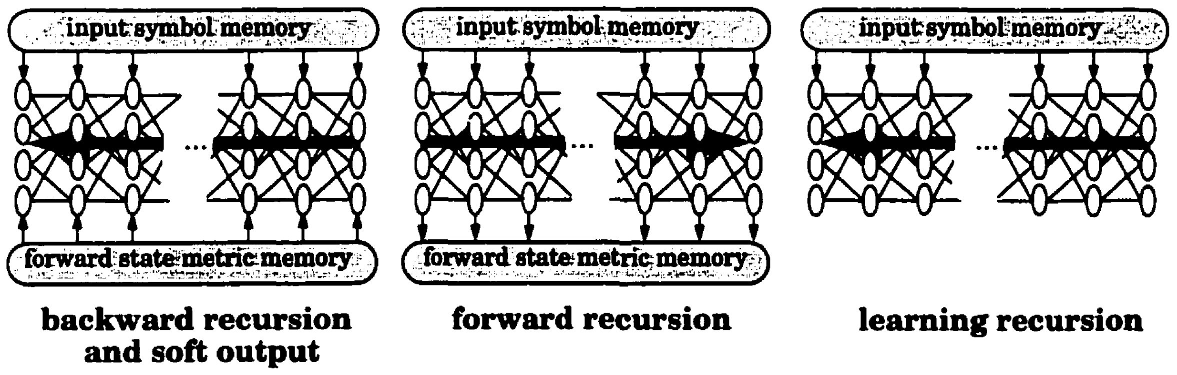


Fig. 2-11: A snapshot of computation in the trellis.

3. The outputs can be explicitly reversed using a LIFO. In a turbo decoder (Chapter 3), the forward-backward algorithm is followed by an interleaver which can perform the reversal for free.

The structure of the four memory banks and the memory access schedule remains to be defined. We will start with the forward state metric memory. Referring to Fig. 2-10, the forward recursion writes state metrics to the state metric memory that are read out in the opposite order by the soft output unit during the next time block. Since the overall memory requirements are never greater than D_L sets of state metrics, a S x D_L dual-port memory can be used to implement a bi-directional shift register (Fig. 2-13). To maintain proper shift register operation, the read operation should complete before the contents of the memory address are overwritten. The direction of the shifting should be reversed every D_L time steps.

The input buffers are each D_L rows long, but the width depends on the code rate R. For example, a rate R = 1/2 code requires three (2 word x D_L) buffers. In addition to the case where data is read and then replaced by new data like the state metric memory a non-destructive read is necessary. If an actual shift register is used instead of dual-port RAM, then a feedback path should be added to create a circular bi-directional buffer as shown in Fig. 2-14.

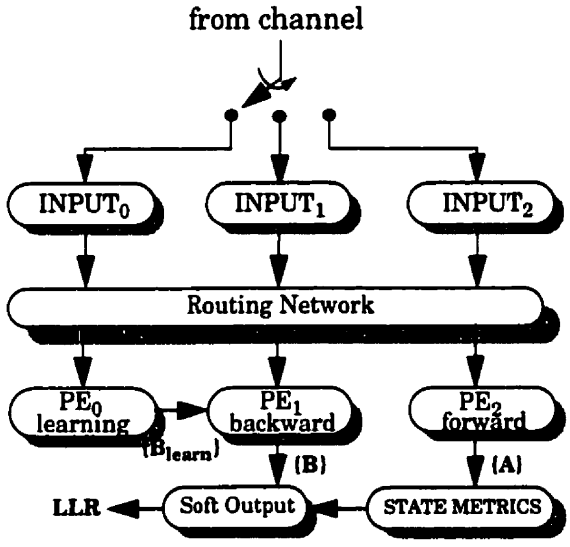


Fig. 2-12: Architecture of the sliding-window Forward-Backward algorithm with M_b=D_L (derived from [DawidM95])

The input buffer access schedule may be derived by considering the configuration of the architecture for each block of D_L clock ticks. During any given block, one buffer is selected to write to (destructive write), and two are operating in the cyclic reading mode. The direction of the accesses for each buffer reverses every D_L clock ticks. Finally, we see that there are three different routings from input buffer to processing element. The cycle of the six distinct combinations is shown in Fig. 2-16. From this figure, the control signals can be defined. We

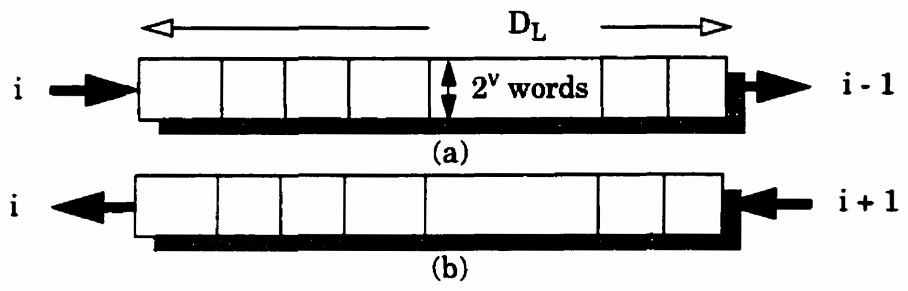


Fig. 2-13: Implementing the forward state metric memory with a bi-directional shift register. In (a) results from trellis block i are stored while results from the previous block i - 1 are sent to the soft output unit. In (b) results from block i + 1 are stored while the previously stored block i is read out in the opposite order in which it was written.

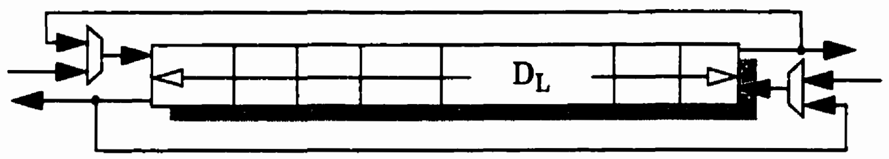


Fig. 2-14: Logical operation of the input buffers. The shift register is reversible in direction and either writes new data, destroying the old contents or is cyclic. A dual-port RAM implementation would not explicitly need the feedback.

note that the direction control is also used to control the forward state metric memory. The overall period of six is a product of the periods of the following two control signals:

- The direction signal DIR with a period of 2.
- The routing signal ROUTE, which controls both the input switch and routing network. ROUTE has a period of 3.

The algorithm is described by the pseudo-code in Fig. 2-15:

```
{A} := initial_A_values;
for j:= 0 to ∞ do
    {B} := {B_learn}
    {B_learn} := initial_B_values;
    DIR := j mod 2;
    ROUTE := j mod 3;
    for k = (jD_L + 1) to ((j + 1)D_L) do simultaneously
        if (DIR = 0) then
            INPUT_ROUTE[(j + 1)D_L - k + 1] := received_symbol[k];
            {B_learn} := smg({B_learn}, bmg(INPUT_(ROUTE + 2) mod 3[k - jD_L]));
            {B} := smg({B}, bmg(INPUT_ROUTE [k - jD_L]));
            {A} := SM_MEM[(j + 1)D_L- k + 1] := smg({A}, bmg(INPUT_(ROUTE +1 ) mod 3[k - jD_L]));
            LLR := soft_output({B}, bmg(INPUT_ROUTE [k - jD_L]), SM_MEM[(j+1)D_L - k + 1]);
        else
            INPUT_ROUTE[k - jD_L] := received_symbol[k];
            {B_learn} := smg({B_learn}, bmg(INPUT_(ROUTE + 2) mod 3[(j+1)D_L - k + 1]));
            {B} := smg({B}, bmg(INPUT_ROUTE [(j+1)D_L - k + 1]));
            {A} := SM_MEM[k - jD_L] := smg({A}, bmg(INPUT_(ROUTE +1 ) mod 3[(j+1)D_L - k + 1]));
            LLR := soft_output({B}, bmg(INPUT_ROUTE [(j+1)D_L - k + 1]), SM_MEM[k - jD_L]);
        end if;
    end for;
end for;
```

Fig. 2-15: Pseudo-code for the sliding window algorithm. {A} are the forward state metrics, {B} are the backward state metrics and {B_{learn}} are the backward state metrics in the learning recursion. The state metric processing elements are represented by smg() and the branch metric generators by bmg(). The soft output unit is represented by soft_output(). The forward state metrics are stored in SM_MEM. The soft output is LLR. Two assignment operators (:=) on the same line imply simultaneous assignment.

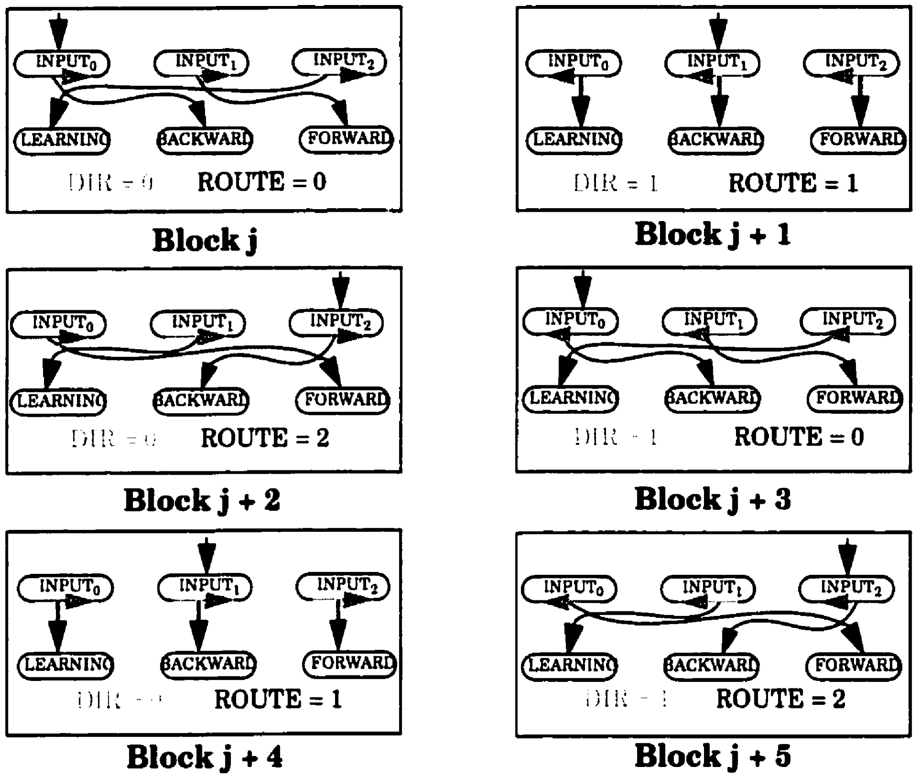


Fig. 2-16: The cyclic memory access sequence. The values of the control signals DIR and ROUTE are indicated. j= 0,6,12,...

2.4 Summary

In this chapter we have reviewed the basic theory of the forward-backward algorithm, which will be necessary for the rest of this thesis. The main points of this chapter are:

- The forward-backward algorithm is an algorithm for detecting a noisy signal in the presence of intersymbol interference (ISI) or decoding a convolutional code.
- The forward-backward algorithm provides optimum soft outputs, a measure of the decoding reliability.
- The main components of the forward-backward algorithm are a forward recursion, a backward recursion, and soft output calculation.
- The computational complexity can be reduced by working in the logarithmic domain.
- In the log-domain the recursions are just like the recursion in the Viterbi algorithm adjusted by a correction factor that is stored in a small lookup table.
- The sliding-window algorithm can be used to reduce the memory requirements and provide continuous decoding with a fixed delay.

Chapter 3

Implementation of a Turbo Decoder

Turbo codes, the best known error-correcting codes, use the forward-backward algorithm as the main component of their decoding algorithm. In addition to their near-optimum performance, turbo codes are relatively simple to decode. In this chapter, we describe the design of TORBO-TM2, an FPGA implementation of a turbo decoder on the Transmogrifier-2 reconfigurable system. In Section 3.1 we review the basic theory of turbo codes and their decoding algorithm. Section 3.2 is a survey of the existing turbo decoder implementations. In Section 3.3 we present the design of a general turbo decoder for hardware implementation. Section 3.4 describes the design of TORBO-TM2. Finally, Section 3.5 is a review of the main points and contributions of this chapter.

3.1 Turbo Codes

3.1.1 Concatenated Codes

Concatenated codes were introduced by Forney in [Forney66] as a way of combining the power of two relatively simple codes. The classic example of a concatenated code is the serial concatenated code shown in Fig. 3-1. The code is constructed by concatenating an inner and an outer code separated by an interleaver. The decoder is a mirror image of the encoder. The results of the inner decoder are deinterleaved and used as inputs to the outer decoder. The purpose of the interleaver/de-interleaver pair is to spread any errors caused by burst noise to make them appear as random error events. One popular technique is to use an inner Reed-Solomon code to correct the output of the convolutional code which typically contains burst errors. The advantage of a concatenated technique is that the combined complexity of the component decoders is less than that of a decoder for a single code with comparable performance.

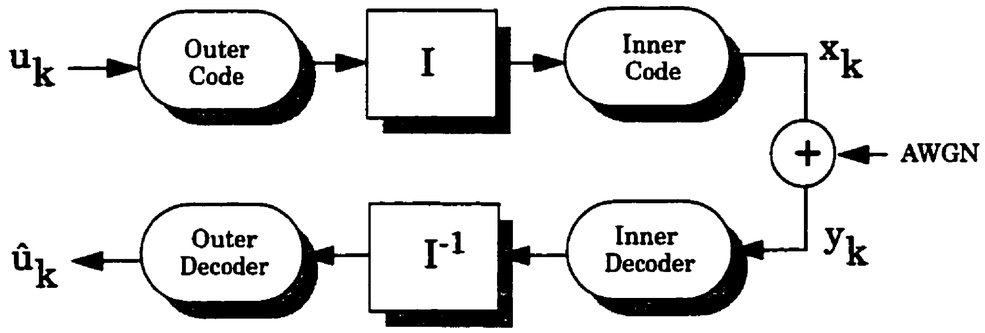


Fig. 3-1: A serial concatenated encoder and decoder. The interleaver is denoted by I and the deinterleaver by I^{-1}.

In recent years, the ideas behind concatenated codes have been extended and improved greatly. The two new ideas are:

- **Soft decoding**: Soft reliability information is exchanged between the decoders instead of hard bit decisions. Performance improves as the inner decoder has more information to work with.
- **Iterative decoding**: Instead of separately decoding the inner and outer codes, the decoder considers the combined code. Maximum likelihood decoding of the joint code is too complex because of the interleaver, so a heuristic *iterative* technique is used instead.

The three main categories of concatenated codes are serial [BDMP96], parallel [BGT93] and hybrid (both serial and parallel) [DivsalarP97] codes as shown in Fig. 3-2. Although parallel

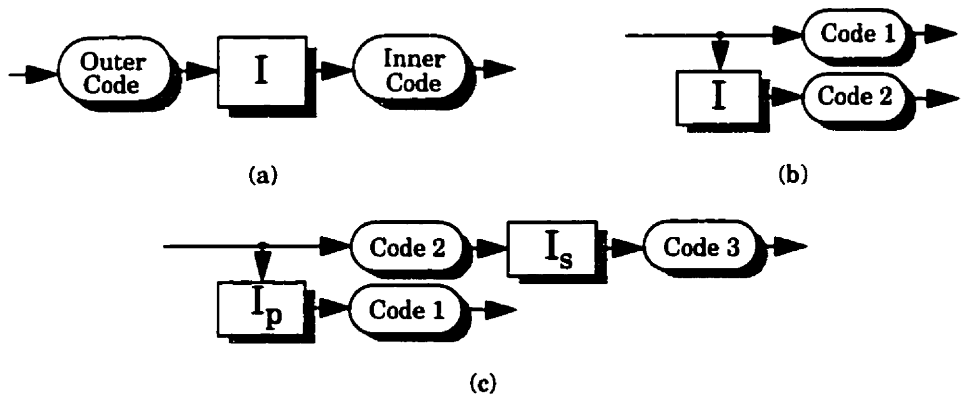


Fig. 3-2: Classes of concatenated codes. (a) Serial. (b) Parallel. (c) Hybrid

codes have been shown to achieve the performance closest to the Shannon limit they are outperformed by serial codes at very low bit error rates (10^{-7} or below) [BDMP96].

3.1.2 Encoders for Turbo Codes

A turbo code is the common name for a parallel concatenation of recursive, systematic, convolutional codes separated by a random interleaver. The idea of parallel concatenation of convolutional codes (and the iterative decoding algorithm) was published in [LYHH93] and [BGT93] simultaneously. The "turbo codes" in [BGT93] however, significantly outperformed the codes presented in [LYHH93] due to the use of *recursive*, *systematic* convolutional (RSC) codes. Fig. 3-3(a) gives an example of the recursive systematic version of the 4-state convolutional code introduced in Fig. 2-3. Systematic codes are codes in which the input bits are copied directly to the output. Recursive codes possess a feedback structure, like an IIR filter. The reason for using RSC codes is left to the thorough explanations of [BenedettoM96] and [BDMP96], which determine that they are necessary to achieve outstanding performance in a concatenated coding scheme. Turbo encoders use RSC codes with 4, 8 or 16 states as higher-order codes do not provide any significant performance improvement. Fig. 3-3(b) shows a turbo code composed of two 4-state recursive systematic convolutional codes. The natural rate of the encoder is 1/3 but higher rates can be achieved by *puncturing* or selecting only some of the bits to be transmitted. For example, a rate 1/2 code can be constructed by transmitting a bit alternating between the two coded bits each symbol interval.

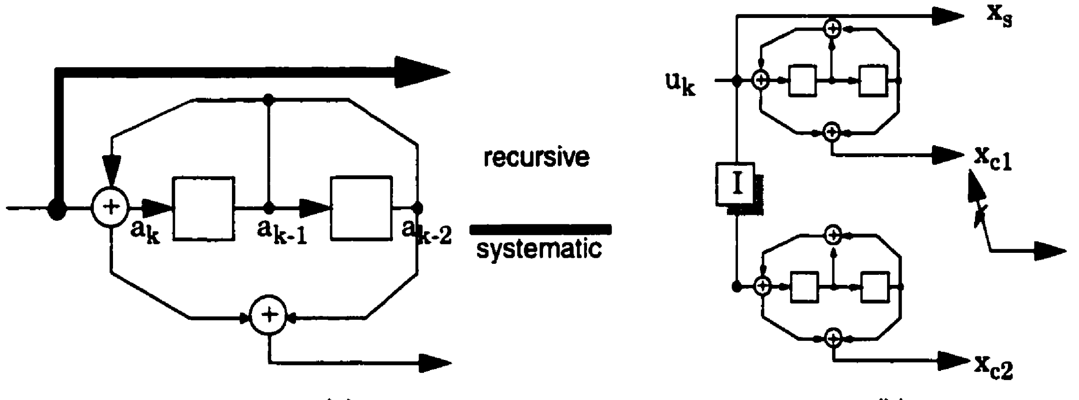


Fig. 3-3: (a) A 4-state recursive, systematic, convolutional code. The recursive nature of the encoder is highlighted by the shading and the systematic nature is indicated with a thick line. (b) A 4-state turbo encoder. The coded bits can be punctured to realize a rate 1/2 code.

3.1.3 Interleavers for Turbo Codes

The single most important factor in the performance of a turbo code is the interleaver length. Interleaver lengths range from about 100 bits to 64k bits. The choice of interleaver *permutation* determines the asymptotic performance of the turbo code. Turbo codes suffer from a flattening of the BER curve known as an error floor. Careful selection of a permutation can help alleviate this effect. The most successful technique aims to preserve a separation or *spread* of at least S symbols between any two symbols in the interleaved sequence that were adjacent in the non-interleaved sequence [DivsalarP95]. Once computed, a permutation is stored in a lookup table of size N. As we will show, a turbo decoder needs to access the inverse permutation as well (deinterleaver) and therefore the permutation storage required in a turbo decoder is 2N. One proposal is to restrict the choice of permutations to the class whose interleaver and deinterleaver are identical at the expense of potential performance losses. This reduces the permutation storage requirements in the decoder to N. Recent work has shown that it is indeed possible to find these "symmetric" interleavers that perform as well or outperform randomly chosen ones [TakeshitaC98] [HPG98]. Another solution is to use a linear feedback shift register (LFSR) with $\log_2 N$ memory spaces to generate a pseudo-random sequence. The interleaver is realized by writing the data in a memory sequentially and reading it out using the addresses generated by the LFSR. The deinterleaver writes the data in the permuted order and reads it out sequentially [HYS98].

3.1.4 The Iterative Decoding Algorithm

Maximum likelihood decoding of turbo codes is too complex to be practical due to the presence of the interleaver. Instead, a practical suboptimal decoding algorithm was proposed in [BGT93] that provides outstanding performance and low complexity. The idea is to use two forward-backward decoders to decode each of the component codes of the turbo code ($\{x_{s,k}, x_{c1,k}\}$ and $\{x_{s,k}, x_{c2,k}\}$) to get soft estimates of the information bits $\{u_k\}$. The soft estimates are then circulated to the other decoder to be used in the next iteration of decoding to improve its estimates. The new estimates are then distributed to the other decoder and so on. Berrou showed that the output of the forward-backward algorithm is a sum of the input scaled by a constant and the new information about the data bit extracted by the algorithm called the *extrinsic information* [BGT93]:

$$L(\hat{u}_k) = \frac{2}{\sigma^2} y_{s,k} + W_k \qquad (3\text{-}1)$$

Consider the turbo decoder shown in Fig. 3-4. The *soft demodulator,* which simply multiplies the input by $2/\sigma^2$, scales the received symbols so they can be combined with the extrinsic information. To compensate for this pre-scaling, we remove the multiplication by $2/\sigma^2$ in the branch metric calculations inside each forward-backward decoder. The first forward-backward decoder receives the channel information about the first code (y_s, y_{c1}) as well as *a-priori* information (z) about the information bits from the other decoder. The a-priori information is added to y_s, which is the noisy systematic bit, to bring the new information to the decoder. After decoding, the a-priori information is subtracted from the log-likelihood ratio to leave the sum of the extrinsic information and the systematic received value. After interleaving to compensate for the interleaver in the encoder, the updated estimate of the information bits are used as a-priori information for the second decoder, which uses the other coded information y_{c2} to decode the second code. The extrinsic information is extracted from the output, de-interleaved and then passed to the first decoder as a-priori information. One iteration of the process is defined as the use of two forward-backward decoders. An estimate of the information bits at the end of each iteration can be obtained by using a slicer on the log-likelihood ratio produced by the second forward-backward decoder. The complexity of the decoder is relatively independent of the power of the code since only small codes with 4 to 16 states are needed. The main factor in the code strength is the length of the interleaver, which increases the memory requirements and latency of the decoder. For high-speed operation, the feedback loop can be cut and the decoder can be unrolled into a pipelined system.

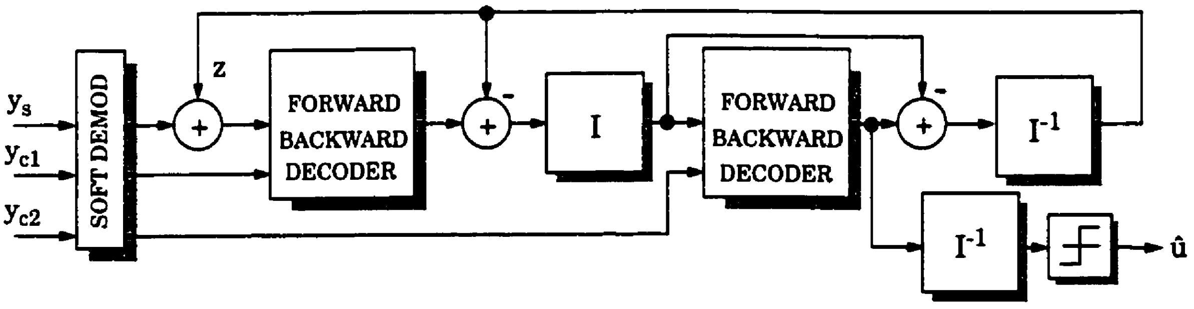


Fig. 3-4: The iterative turbo decoder.

3.2 Previous Turbo Decoder Implementations

In this section, we briefly review the existing hardware for turbo decoding parallel concatenated convolutional codes. We also include a recently developed high-speed software decoder. The important characteristics of the designs are summarized in Table 3-1.

3.2.1 CAS5093 [Comatlas95]

This chip was developed by Comatlas SA of France. The decoder integrates 2.5 iterations of 8-state turbo decoding into a single 0.8 μm CMOS ASIC. The SOVA algorithm is used instead of the forward-backward algorithm for lower complexity. The maximum decoding rate is 40 Mbps.

3.2.2 TURBO4 [BCPT95]

TURBO4 is a multi-chip turbo decoding solution built by France Telecom in 0.8 μm CMOS. Each ASIC, representing a single iteration, contains two 16-state SOVA decoders, an interleaver and deinterleaver. Multiple iterations can be achieved by cascading modules in a pipeline. The decoding rate is 40 Mbps.

3.2.3 JPL FPGA [JPL1997][BDMP97]

This FPGA turbo decoder consists of a single custom FPGA circuit board. A forward-backward decoder is implemented that is capable of decoding up to 64 states at interleaver lengths of up to 64k. Speed performance figures have not been published but personal communications with JPL indicate the decoder operates roughly at 10 Kbps.

3.2.4 University of Dresden FPGA [Koora98]

This FPGA turbo decoder implements an 8-state SOVA decoder in a single FPGA. Performance is 14 Mbps with a planned future version running at 30 Mbps. The maximum interleaver length is 448 which is slightly longer than the length of an ATM frame.

3.2.5 University of South Australia FPGA [Pietrobon98]

This FPGA design utilizes multiple circuit boards to realize a pipelined turbo decoder. The interleaver is programmable up to 64k, and a wide range of code rates can be selected. The turbo decoder modules are composed of log-forward-backward algorithms that calculate the state metrics in a serial manner. A 16 state code with an interleaver length of 64k can be decoded at 356 Kbps. Although a turbo decoder speed for a 4-state code was not quoted, a speed of 624 Kbps was reported when using the decoder as a 4-state stand-alone FB decoder. We therefore predict that this speed probably can be extended to turbo decoding as well.

3.2.6 University of Michigan ASIC [HYS98]

This ASIC integrates 4 iterations of a block-based turbo decoder in a 0.6 μm CMOS process. The 16-state component decoders use the MAX-LOG-FB algorithm. The highest reported speed is 1 Mbps.

3.2.7 University of California San Diego FPGA [HOCS98]

This decoder is implemented on a reconfigurable FPGA system called the ReConfigurable Processor Board (RCP). The number of states is configurable from 2 to 512. The maximum interleaver length is 64k. A nice feature of this decoder is that number of bits used to represent the data is programmable. The decoder has been implemented in VHDL and is currently being tested in hardware.

3.2.8 Communications Research Centre Software [CRC98]

The Canadian Communications Research Centre (CRC) has developed an ultra-fast turbo decoder for Pentium-II class PCs. The decoding algorithm reported is an "approximate APP" algorithm which we think is probably an integer arithmetic implementation of the forward-backward algorithm. The output of the MAX-LOG-FB algorithm is adjusted after each iteration by a correction factor. The decoding speed is 400 Kbps for 4 iterations on a 400 MHz Pentium II processor.

Table 3-1: Summary of Turbo Decoder Implementations

Design	Technology	Max Speed (Mbps)	# States	Interleaver Length	Component Decoder
[Comatlas95]	0.8 μm CMOS	40	8	1024	SOVA
[BCPT95]	0.8 μm CMOS	40	16	2048	SOVA
[JPL1997]	FPGA	0.01	up to 64	up to 64K	LOG-FB
[Koora98]	FPGA	14	8	up to 448	SOVA
[Pietrobon98]	FPGA	0.356 (16 state) 0.624 (4 state)	up to 512	up to 64K	LOG-FB
[HYS98]	0.6 μm CMOS	1	16	?	MAX-LOG-FB
[HOCS98]	FPGA	?	up to 512	up to 64K	LOG-FB
[CRC98]	Software 400 MHz PII	0.4	16	unlimited	MAX-LOG-FB
This Work	FPGA	0.75	4	up to 64K	LOG-FB

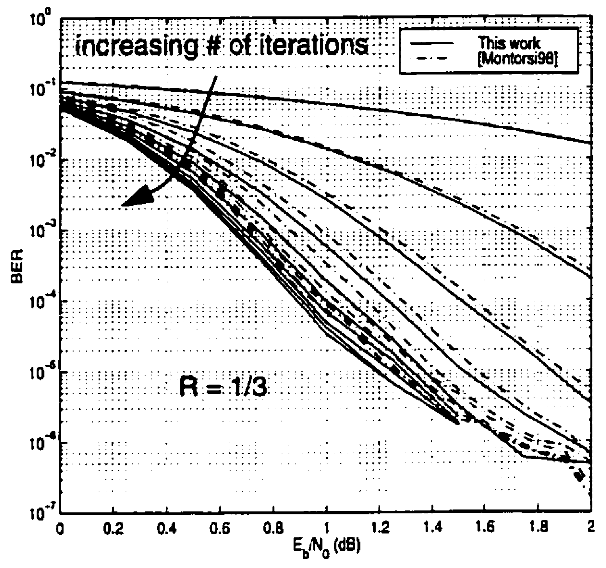
3.3 Design of a Hardware-Ready Turbo Decoder

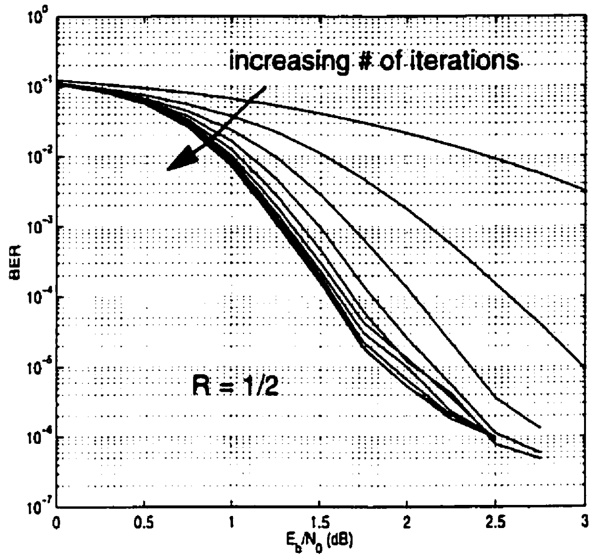
This section describes the design of a turbo decoder intended for hardware implementation. Previously published information on turbo decoder implementations has been very vague about the design choices and implementation issues. The goal of this section is to document the design process of a turbo decoder.

3.3.1 Simulation of the Turbo Code

The first step in the design of the turbo decoder is the selection of an appropriate turbo code. We decided to use the 4-state turbo code selectable between rate 1/2 or rate 1/3 with an interleaver length of 1024 (see Fig. 3-3(b)) on an AWGN channel. Interleaver lengths of 16K or 64K are only useful for squeezing out the last bit of performance near the Shannon limit. For more practical decoding applications, the interleaver length of 1024 represents a medium size interleaver that offers a reasonable mix of performance, memory requirements and latency. Some examples of applications that would benefit from a block size ranging from 100-1000 bits are IS-136 wireless transmission (162 bits) and ATM (424 bits). The four state codes were chosen to facilitate the design of a fully-parallel state metric unit. We will show in Section 3.4 that for our chosen architecture it is convenient to keep the number of states low to conserve memory bandwidth. For the turbo decoder we have naturally chosen the forward-backward algorithm for the constituent decoders. The decoder will be implemented in the logarithmic domain as described in Section 2.2.3.

Simulations were performed on a SUN workstation to determine the performance characteristics of the turbo code. Benchmark results are very scarce in the turbo decoding literature since it is necessary to know the exact interleaver permutation to perform an accurate comparison. Montorsi has published a set of turbo decoding results for the exact code that we have chosen to implement [Montorsi98]. The interleaver permutation was included with the results. Fig. 3-5 shows our benchmark floating point simulations of rate 1/3 and rate 1/2 turbo codes. We have found that the performance of the code does not improve beyond 10 iterations. Montorsi's results for the rate 1/3 turbo code have been superimposed in Fig. 3-5(a). As a comparison with conventional convolutional codes we have plotted the performance of a rate 1/2 64-state RSC decoded by the FB algorithm (blocksize 1024) in Fig. 3-6. The turbo decoder uses (2 x # iterations) FB decoders and so a 64-state FB decoder is a fair comparison to a turbo decoder with 1-10 iterations.





(a) (b)

Fig. 3-5: 4-state turbo code performance. Interleaver length 1024. (a) rate 1/3. (b) rate 1/2

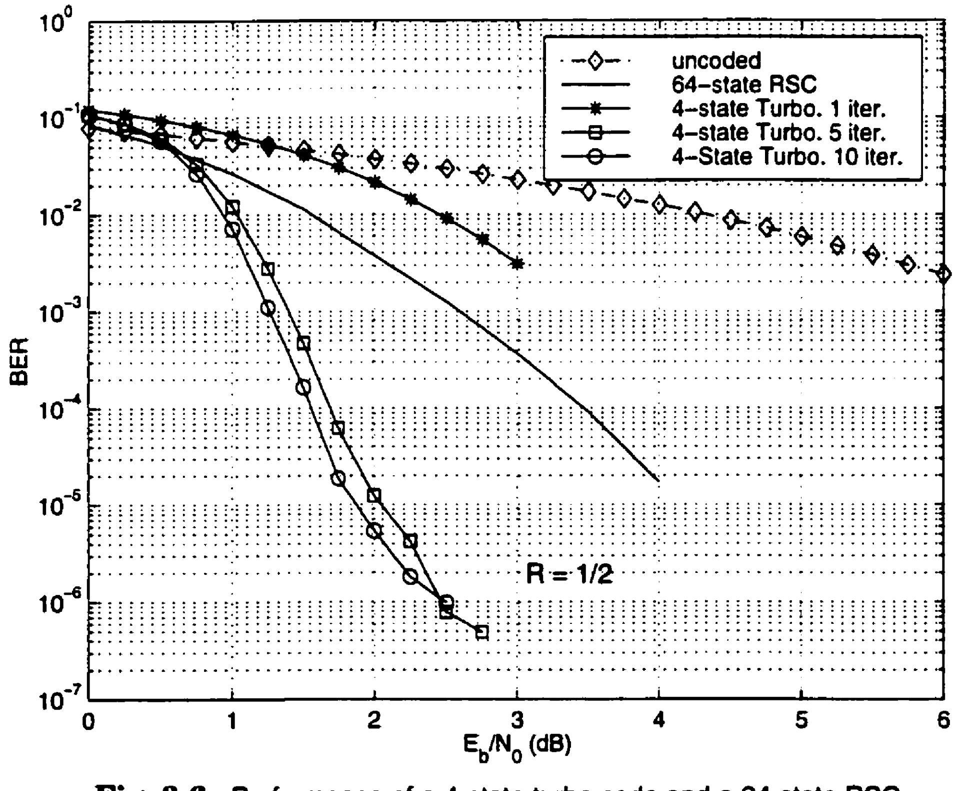


Fig. 3-6: Performance of a 4-state turbo code and a 64-state RSC

3.3.2 Approximating the Channel SNR

One of the difficulties in turbo decoding is estimating SNR of the channel in the soft demodulator. Hoeher has suggested that using a fixed estimate of the SNR does not degrade performance [Hoeher97]. Our simulations in Fig. 3-7 show that this is true on an AWGN channel using an estimate of 1.0 dB in the soft demodulator. We note however, that this result does not likely hold in fading channels where a more sophisticated estimation algorithm is probably required [ReedA97][SummersW98]. A fixed SNR estimate also helps control the dynamic range of the metrics

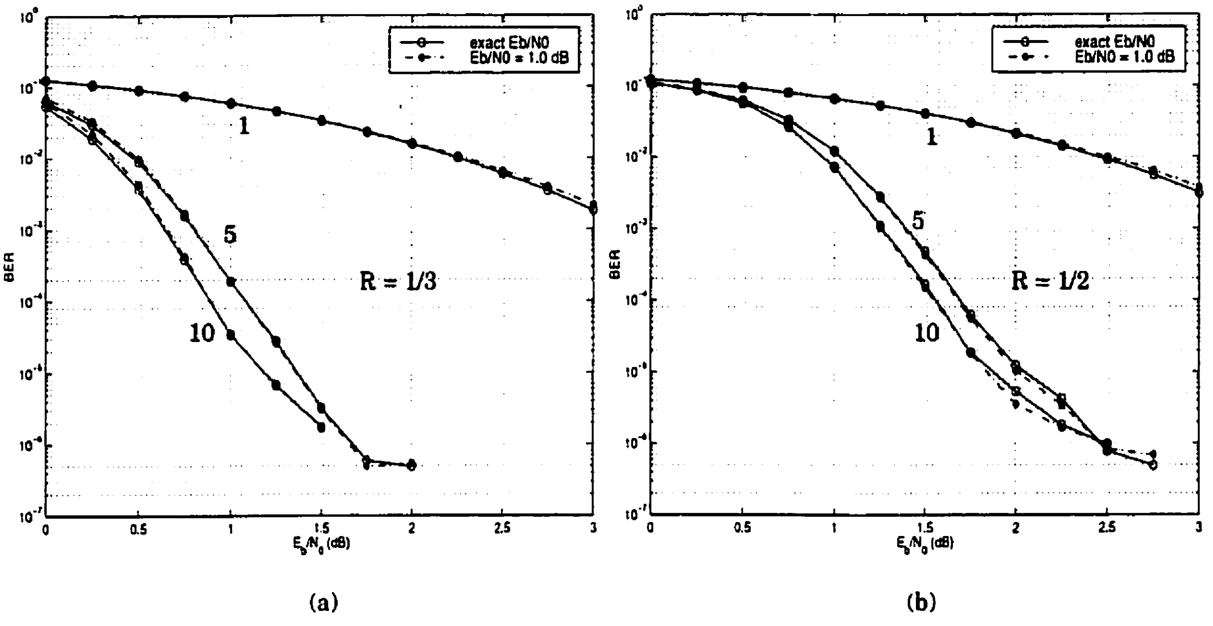


Fig. 3-7: Approximation of the noise variance in the soft demodulator. Shown are iterations 1,5 and 10. (a) rate 1/3. (b) rate 1/2.

3.3.3 Fixed-Point Turbo Decoding

The next step in the design of the decoder is to determine the wordlength and format required for a fixed-point integer implementation. Since both positive and negative numbers are required, two's complement notation was used as described in Fig. 3-8. We define the wordlength WL as:

$$WL = 1 + WL_I + WL_F \qquad (3-2)$$

for a word with one sign bit, WL_I integer bits and WL_F fractional bits. Saturating arithmetic operations are used so that a number that lies out of the range of the representation is assigned the most positive or most negative value as appropriate.

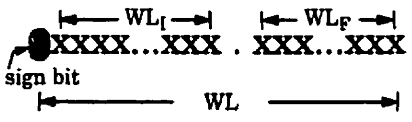


Fig. 3-8: Wordlength WL fixed-point representation with WL_I integer bits and WL_F fractional bits.

Simulations show that the FB decoders need an WL = 8 bit representation with $WL_I = 4$ and $WL_F = 3$ for the internal branch metrics, state metrics and soft-output calculations. All values external to the FB decoders need only a WL = 6 bit representation with $WL_I = 3$ and $WL_F = 2$. This includes the output of the A/D converter and the extrinsic info exchanged between decoders and in the interleavers. Not surprisingly, some previous work came to the same conclusions about the total number of bits required [Pietrobon98][1]. However, since the performance of the decoder was not compared to one with infinite precision then the trade-offs in making such a choice were not apparent. In [RHV97], such a comparison is made for a 16-state code with interleaver length of 1024 and 8 iterations. A loss of 0.2 dB at BER = 10^{-5} is reported using a quantized version of the log-FB algorithm relative to a floating-point FB algorithm. The authors report an overall loss of about 0.5 dB. Four bits were used for the input data and eight bits were used for the metrics in the FB decoders.

The next task is to determine the size of the LUT required to represent the correction factor $f(x) = \ln(1 + \exp(-|x|))$ in the MAX* operation. We first considered eliminating the table completely and implementing the max-log-FB algorithm as some authors have suggested [HYS98]. Simulations shown in Fig. 3-9 (floating-point) motivate the need to find an efficient way of implementing the complete log-FB algorithm so the focus shifted to minimizing the size of the lookup table. Previous work reports lookup tables of 2k x 16 [BDMP97] and 81 x 4 [Pietrobon98]. If more than one state metric generator is to be implemented in parallel then multiple copies of the table are needed, which is cumbersome for the LUT sizes reported. Good results are reported in [RHV97] for an 8-word LUT but it is not clear in their reference whether this refers to the FB algorithm alone or to turbo decoding. The reference is likely to turbo decoding since the max-log-FB and log-FB algorithms have identical hard decision performance. Since we use an 8-bit representation with 5 integer bits and 3 fractional bits, the smallest value of f(x) we can represent is 1/8 corresponding to a maximum value of x of

1. The scaling used in [Pietrobon98] is not the same as in this work.

about 2.0 Our simulations show that for +1/-1 signalling and using a fixed SNR estimation, the following rule provides excellent results [GrossG98]:

$$f(x) = \begin{cases} \frac{3}{8} & -2 \le x < 2 \\ 0 & \text{otherwise} \end{cases} \qquad (3\text{-}3)$$

Fig. 3-10 shows the function f(x) and the approximate function we have implemented. The implication of the above rule is that the lookup table for f(x) can be reduced to a simple logic circuit, which either adds or does not add a constant to the output of the maximum selection circuit. The simplified circuit we have implemented is shown in Fig. 3-11. The difference, d, from the maximum circuit is used as input to the correction function circuit. We check if the number is less than 2.0 by considering the upper 4 bits. For positive numbers, they will all be zero if the number is strictly less than 2.0 and for negative numbers they will all be ones if the number is less than or equal to -2.0. This is the reason for the asymmetry of the rule. The correction circuit generates a 1 or a 0 that is repeated and padded with zeros at the input to the final adder to form the 3 bit number 00000011 (0.375) or 00000000 (0.0). The new simplified circuit has the added advantage of only requiring the difference X-Y and does not need the difference Y-X. We briefly note that in full-custom VLSI implementations, the difference Y-X can be made available for free at the output of the subtractor and pass transistor logic can be used to implement an 8-LUT with 60 transistors. The new simplified circuit can be implemented with 20 transistors. Synthesis to standard cells in 0.5μm CMOS results in a 40% area savings. The savings are even more pronounced when considering FPGA or DSP applications. In these cases, the LUT may require storage in memory, possibly creating a memory bottleneck[2]. The simplified rule does not suffer from these restrictions. Fig. 3-12 shows the results of the simulations including all of the simplifications; fixed SNR estimate, quantization and simplified MAX^* operation. The simplified decoder has performance comparable to the ideal floating point decoder.

The simulations shown here were all performed with the soft-demodulator implemented in floating-point arithmetic, in other words, before the 6-bit A/D converter. It may be desirable however to place the soft demodulator after the A/D converter. We investigated the effect of implementing the soft demodulator as a 64 x 6 LUT and plot the results in Fig. 3-13. At a BER of 10^{-5} there is a small loss of 0.15 dB at R = 1/3 and 0.1 dB at R = 1/2.

2. Some FPGAs have configurable memory blocks that could be used to implement a LUT.

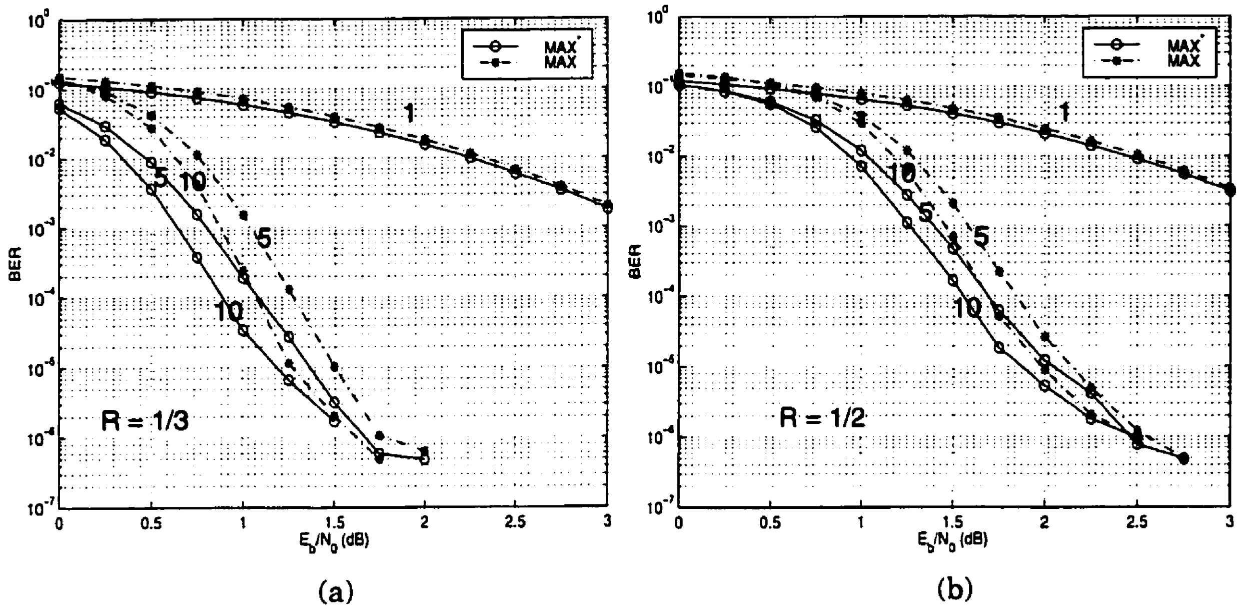


Fig. 3-9: Comparison of floating point turbo decoders using log-FB (MAX* operation) and the max-log-FB (MAX operation) constituent decoders. 4-state codes, interleaver length 1024. Iterations 1, 5 and 10 are shown. (a) rate 1/3. (b) rate 1/2.

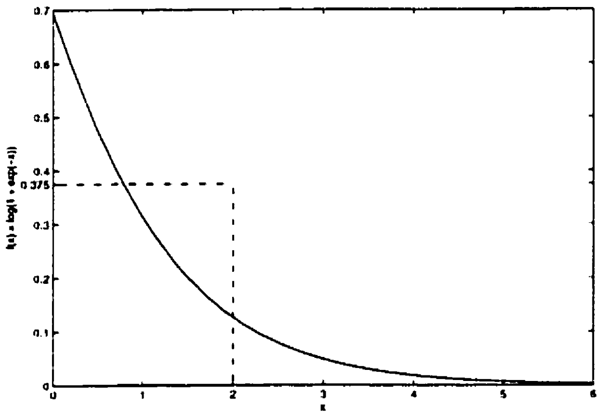


Fig. 3-10: Approximation of the correction function f(x) in the MAX* operator.

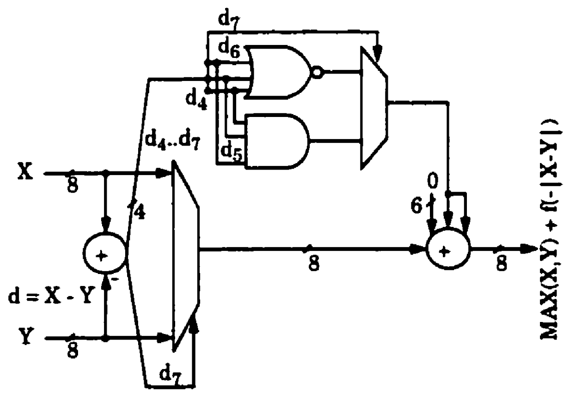


Fig. 3-11: Simplified MAX* circuit.

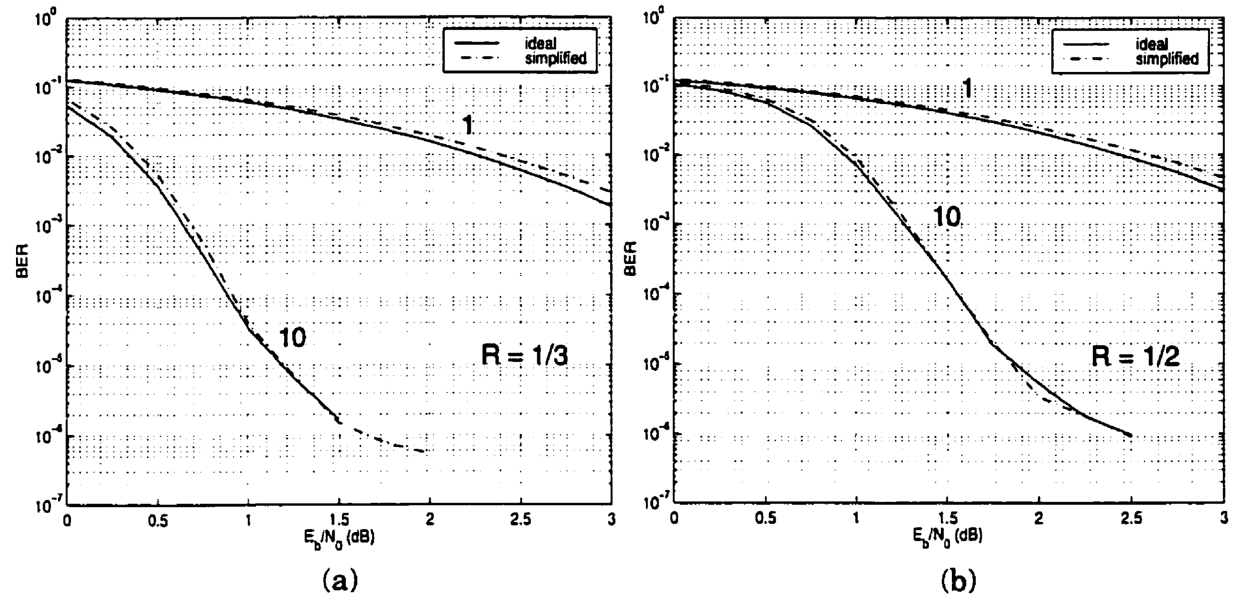


Fig. 3-12: Performance of the simplified hardware-ready turbo decoder compared to the ideal floating point turbo decoder. 4-state turbo code, interleaver length 1024. 1 and 10 iterations shown. (a) rate 1/3. (b) rate 1/2.

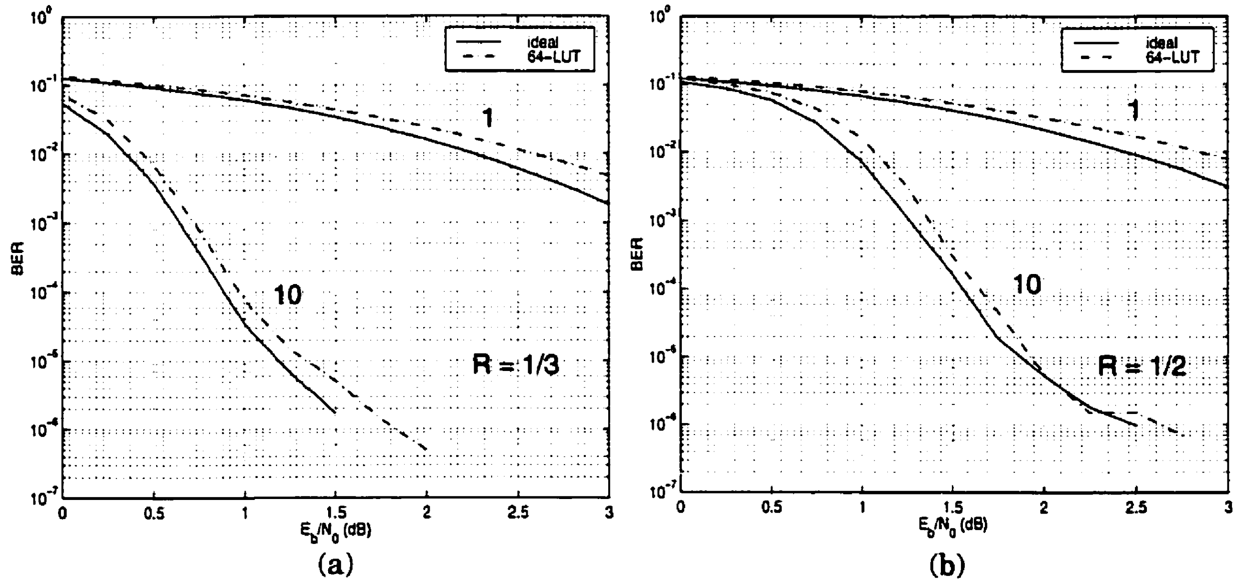


Fig. 3-13: Effect of implementing the soft demodulator after the A/D converter as a 64 x 6 LUT compared with the ideal floating point turbo decoder. 4-state turbo code, interleaver length 1024. 1 and 10 iterations shown. (a) rate 1/3. (b) rate 1/2.

3.4 TORBO-TM2: An FPGA Implementation

3.4.1 The Transmogrifier-2 (TM-2)

The Transmogrifier-2 (TM-2) is a multi-FPGA reconfigurable system [LGVRC98]. The goal of the TM-2 is to provide a scalable testbed for rapid prototyping. The TM-2 is made up of a number of circuit boards, each containing two large Altera 10k50 FPGAs, 4 I-Cube crossbar routing switches and up to 8 Mbytes of SRAM. The newer TM-2A revision uses 10k100 FPGAs. When complete, the TM-2A will consist of 16 such boards giving a total programming capacity of 2 million gates. Fig. 3-14 is a photograph of one board of the TM-2. The TM-2 can be programmed using HDLs such as VHDL, Altera's proprietary AHDL or the TM-2 specific C-based Transmogrifier C [Galloway95]. From the user's (designer) point of view, the details of the TM-2, such as routing between FPGAs, were not important. We essentially see it as a big collection of gates, with one exception. Currently, the TM-2 compiler software does not automatically partition a design among the FPGAs so that task has to be done manually. Communication with a host SUN workstation is via a 4-bit "nibble" bus that is handled on the FPGAs by an automatically generated circuit. Each of the two FPGAs can simultaneously access two 64-bit wide SRAM banks.

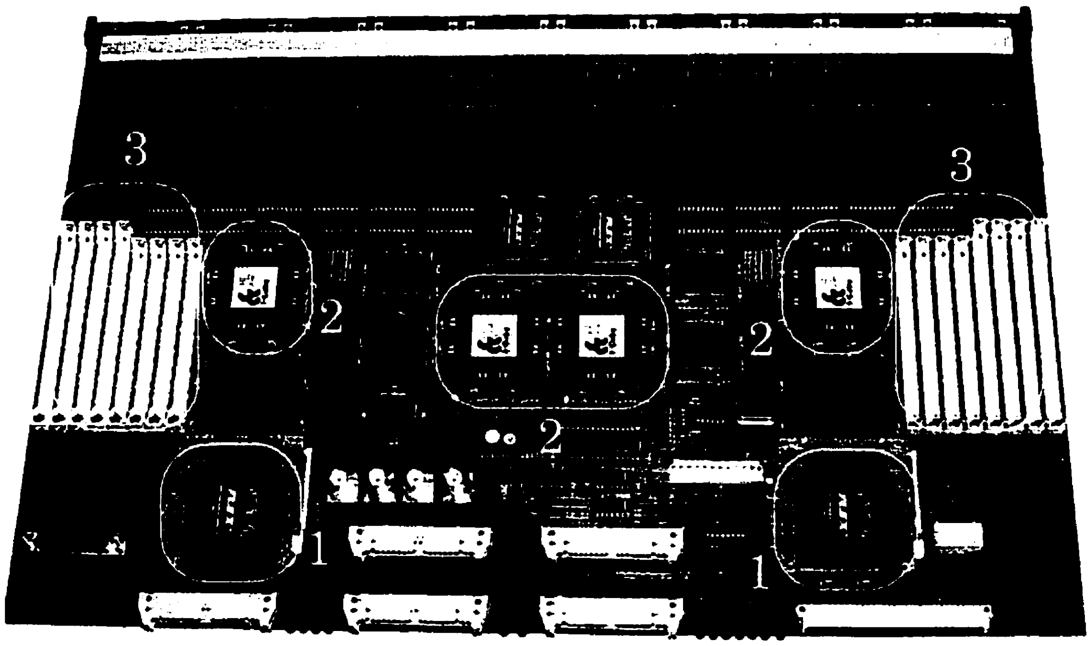


Fig. 3-14: Photograph of one board of the TM-2. 1. Altera 10k50 FPGAs. 2. I-Cube crossbar switches. 3. SRAM banks (unpopulated).

3.4.2 System Architecture

TORBO-TM2 is a configurable 4-state turbo decoder implementing one log-FB algorithm block that is reused to produce the desired number of iterations. Although the code construction is fixed to the 4-state code described above, the interleaver length, permutation and number of iterations are user selectable. The input/output parameters are described in Table 3-2. The TORBO-TM2 system architecture is shown in Fig. 3-15. The design is partitioned into two FPGAs. The entire turbo decoder easily fits into one 10k100 FPGA but the second FPGA is needed to provide access to a third bank of RAM. The SUN workstation prepares the 6-bit quantized input data by generating random bits, encoding them, and adding Gaussian noise. Currently, the soft-demodulator is also contained in the workstation software, but this could be migrated into the decoder by using the LUT described in Section 3.3.3. The LUT could be placed in the unused bank of RAM connected to FPGA_1. The data bits are sent to the decoder along with the soft input and compared to the hard decisions at the end of the decoding process. The number of errors detected is sent back to the workstation. The soft decisions themselves are not sent back to the SUN to conserve bandwidth on the communications link, which is currently very limited. The SUN workstation also generates the interleaver permutation and sends it to the decoder during its initialization phase. Almost all of the functionality of the decoder is in FPGA_0, which reduces the number of pins required to route signals between the two FPGAs. The RAM banks alone use more than half of the pin budget of FPGA0 (e.g. 174/304 user pins on a 10k50).

Table 3-2: TORBO-TM2 I/O parameters.

Signal	I/O	# bits	Value	Description
CLK	input	1	{0,1}	clock
RESET	input	1	{0,1}	reset
N_IN	input	16	1-65535	interleaver length
ITER	input	5	1-31	# iterations
ERROR_COUNT	output	16	0-65535	# decoding errors

The decoder is made up of four main blocks: A datapath, control unit, host interface and memory controllers. The data block transferred from the SUN workstation is stored in SRAM. Each iteration of decoding is divided into two half iterations (one FB algorithm), which in turn are divided into forward and backward passes for a total of four stages per iteration. In the first stage, the data is read from SRAM and the forward state metrics are calculated and stored in another SRAM. In the backward stage, the data is read again to calculate the backward state metrics that are combined with the stored forward state metrics to create the log-likelihood ratio and extrinsic information. The extrinsic information is stored in a permuted order back in the SRAM where it will be read out during the next two stages, which represent the second FB decoder in the turbo decoder. The second decoder proceeds in a similar fashion except that the results are stored using the inverse permutation so that it is in the original order. The interconnection and schedule that govern the decoder will be described in the following sections.

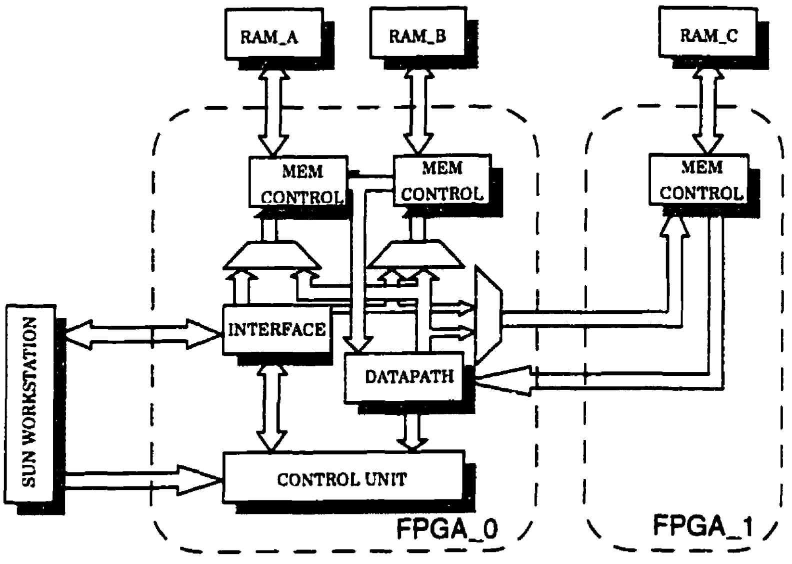


Fig. 3-15: TORBO-TM2 system architecture.

3.4.3 Interface to the SUN Workstation

The TM-2 is connected to the SUN workstation via a 4-bit "nibble bus". The TM-2 compiler automatically generates a circuit that handles the data transfers over the nibble bus and a software ports package is provided on the host side. Unfortunately, this bus was not designed for high-speed transfer and currently the turbo decoder is not expected to be able to decode at real-time speeds. An S-bus interface for the TM-2 is being planned and will help alleviate this problem.

The interface unit controls the transfer of the interleaver permutation and the input data to the decoder from the host as well as returning the block error count to the host. The interface unit is a state machine that implements a handshaking data protocol. The interface signals are summarized in Table 3-3.

Table 3-3: I/O signals to the interface unit.

Signal	I/O	# bits	Description
SUN_DATA_IN	input	32	shared between the interleaver permutation and input data
WANTED_IN	output	1	indicates the TM-2 wants to receive data
READY_IN	input	1	indicates that data for the TM-2 is valid
SUN_DATA_OUT	output	16	error count
READY_OUT	output	1	indicates that data for the host is valid
ACK_OUT	input	1	indicates that the host received the data

3.4.4 Memory Controller

The memory controller is a synchronous interface between the FPGA circuits and the SRAM banks. The main purpose of this unit is to covert the tri-state data bus to separate input and output signals. The TM-2 memory control register is 5 bits wide while the TM-2A uses a 10-bit register. The 10-bit control word used in the turbo decoder is translated to the correct format by the memory controller. This means that when migrating from the TM-2 to the TM-2A, only the memory controller needs to change.

3.4.5 Datapath

The datapath of the turbo decoder consists of: a branch metric generator (BMG), state metric generator (SMG) and a log-likelihood ratio generator (LLRG) as shown in Fig. 3-16.

The BMG is shown in Fig. 3-17. Appendix A lists the equations for the log-FB algorithm used in the decoder. For the log-FB algorithm we call the received data $\{y_{s,k}, y_{1,k}, y_{2,k}\}$ the a-priori information for the first decoder $z_{1,k}$ and the a-priori information for the second decoder $z_{2,k}$. For the upper RSC of the turbo code, the four possible branch metrics are 0, $y_{s,k} + z_{1,k}$, $y_{1,k}$ and $(y_{s,k} + z_{1,k}) + y_{1,k}$. The possible branch metrics for the lower RSC are 0, $z_{2,k}$,

y_{2k} and $z_{2,k} + y_{2k}$. Multiplexers select the appropriate input depending on whether the FB algorithm is decoding the upper or the lower constituent code.

The fully-parallel state metric unit that calculates both forward and backward state metrics is shown in Fig. 3-18. The critical path in the decoder is the state metric unit because of the feedback loop. When the backward metrics are being calculated, the branch metrics are added to the previous state metrics and these values are sent to the LLRG. The exact grouping of state and branch metrics is derived in Appendix A. This simplification eliminates repeating the same calculation in the LLRG. Alternately, the forward state metrics could be used in the grouping, but this would mean that 8 state metrics would have to be stored in memory instead of 4. The state metrics are normalized by subtracting the largest value at each time step from all of the metrics. Finding the maximum is time consuming especially considering that this operation is in the critical path of the decoder. Unfortunately, our simulations showed that alternative normalization techniques such as subtracting a constant value from all of the state metrics, were not effective. The LLRG that operates in parallel with the SMG is shown in Fig. 3-19.

Fig. 3-20 shows the circuit used as a saturating adder in all datapath operations. We used the Altera MAX-PlusII software to compare the speed and area of the saturating adder using three 8-bit component adders: a carry propagate adder (CPA), carry look-ahead adder (CPA) and an Altera provided module (LPM). The results shown in Table 3-4 indicate that there is no advantage to using the LPM adder or CLA adder over the CPA for 8-bit adders.

Table 3-4: Speed and area of the saturating adder using different component adders

Adder	# logic blocks	delay (ns)
8-bit CPA	24	29.3
8-bit CLA	24	29.4
8-bit LPM	28	29.7

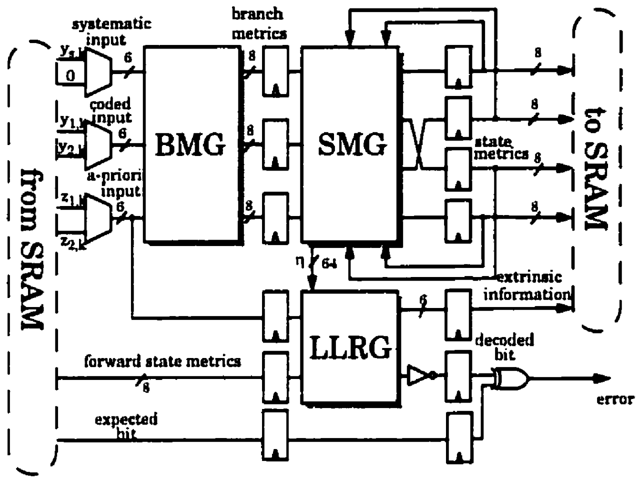


Fig. 3-16: TORBO-TM2 datapath.

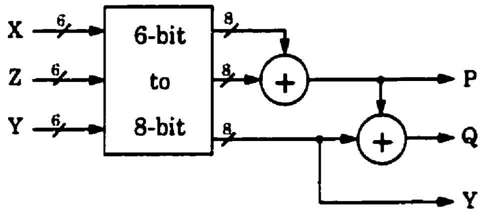


Fig. 3-17: Branch Metric Generator (BMG). X is the systematic input, Y is the coded input and Z is the a-priori information input.

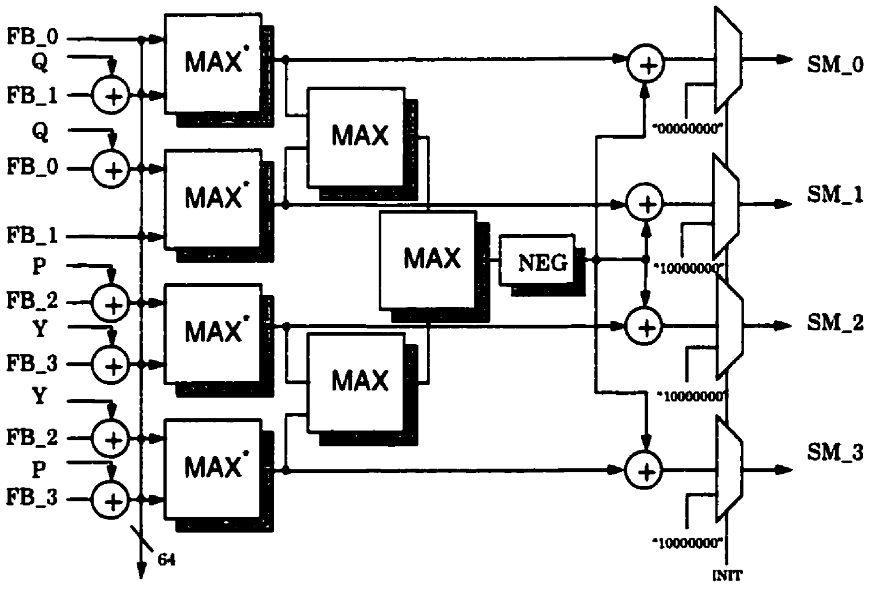


Fig. 3-18: State Metric Generator (SMG). FB_i are the values of the previous state metrics from the feedback path. The INIT signal selects the initial state metrics. SM_i are the new state metrics. All values not otherwise indicated are 8 bits.

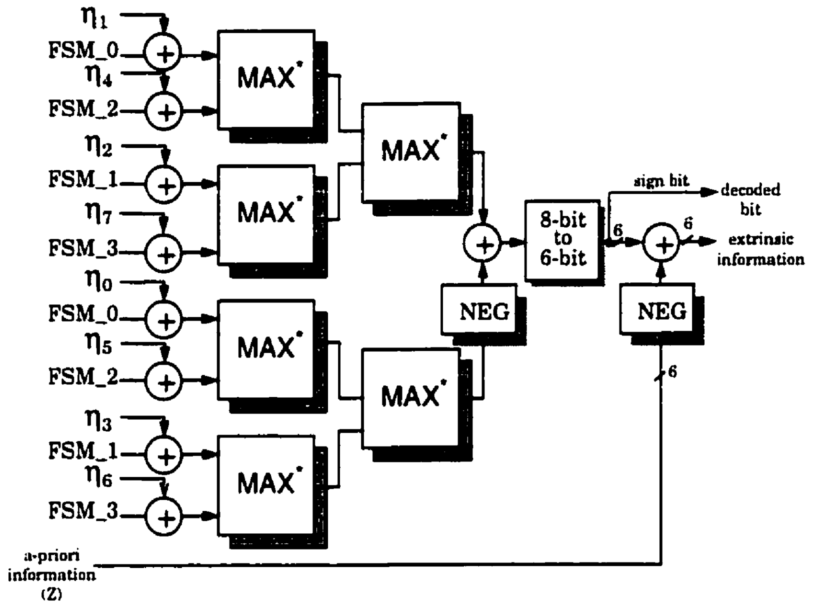


Fig. 3-19: Log-likelihood Ratio Generator (LLRG). FSM_i are the forward state metrics recalled from SRAM. η_i are the sum of the backward state metrics and the branch metrics as described in Appendix A. All values are 8-bit unless otherwise indicated.

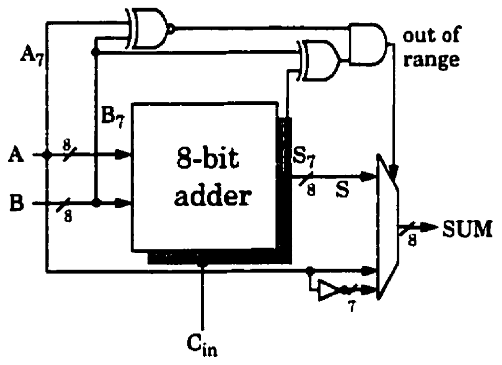


Fig. 3-20: 8-bit saturating adder. SUM = A+B if the result can be represented in the 8-bit format otherwise SUM = 01111111 (most positive number) or 10000000 (most negative number).

3.4.6 Control Unit

The control unit directs the decoding process by controlling the operation of the other units and the SRAMs. In this section we describe the processing schedule and use of the SRAMs implemented in the control unit. We note that the SRAMS are addressed starting at location 1 and not 0 because location 0 may be overwritten upon TM-2 initialization. The SRAMS are 64-bits wide and individual bytes can be selected for reading and writing. This is why we use multiples of 8-bits to hold data that is only quantized to 6-bits. A variable name in this section actually refers to a block of values. The interleaver length is N. Note that the transmitted data comes in blocks of N+2 words, the extra two words are used to force the encoder state to zero. The variables involved in the memory schedule are described in Table 3-5:

Table 3-5: Variables involved in the memory schedule for TORBO-TM2.

Name	# bits	block size	description
U	1	N	expected bit values
X_1	6	N+2	received systematic values
X_2	6	2	received terminating sequence for the lower code
Y_1	6	N+2	received upper coded values
Y_2	6	N+2	received lower coded values
Z	6	N	a-priori information.
W	6	N	extrinsic information
A	32	N+2	forward state metrics (4 x 8 bits)
I	16	N	interleaver permutation
I^{-1}	16	N	inverse interleaver permutation

The interleaver is implemented differently than is usually described. The standard procedure is to write an array of values in sequential order and then read them out in a permuted order. Our schedule uses sequential reads but out-of-order writes to simplify the control due to a limited number of independent memory ports. The only difference is that interleaving is done using the inverse permutation and de-interleaving is done using the regular permutation.

Assume the input data is initially in RAM_A. If this is the first iteration, Z will be set to zeros, otherwise it is the a-priori information provided by the previous iteration. The interleaver permutation and inverse permutation are stored in RAM_B. The "tail" X_2 is stored in the appropriate location in RAM_C. Each iteration can be broken down into two FB passes each with two recursions for a total of four stages, FORWARD0, BACKWARD0, FORWARD1 and BACKWARD1:

FORWARD0: the received data for the upper code (X_1, Y_1) and the a-priori information (Z) are read from RAM_A and used to calculate forward state metrics (A) that are stored in RAM_B. The coded information for the second code (X_2) is transferred to RAM_C.

BACKWARD0: the received data (X_1, Y_1) and a-priori information (Z) are read from RAM_A again and the backward state metrics are calculated. The extrinsic information (W) is calculated, interleaved, and stored in RAM_C. The expected bit decisions (U) are also read from RAM_A, interleaved, and stored in RAM_C.

FORWARD1: W, X_2 and Y_2 are read from RAM_C and used to calculate the forward state metrics (A) that are stored in RAM_B.

BACKWARD1: A, W, X_2 and Y_2 are used to calculate the backward state metrics and the log-likelihood ratios. The extrinsic information derived from the LLRs is deinterleaved and stored in RAM_A (Z) where it is available for the next iteration. The expected decisions, interleaved during BACKWARD0 are compared with the hard decisions (sign bit of the LLR) and the error count is updated.

The procedure is shown in Fig. 3-21. The forward stages take N+2+2 clock cycles and the backward stages take N+2+3 cycles due to the latency introduced by the datapath pipeline. The overall decoding efficiency of one iteration is therefore: N/(4N + 18) decoded bits/ clock cycle. The total memory required is 40(N+2) + 64(N+2) + 24(N+2) bits or 16(N+2) bytes. For N = 1024, this corresponds to just over 16Kbytes. With the current interface scheme, the total time to decode one block is actually dominated by the transfers to the TM-2 from the host. With an improved interface, the received blocks could be buffered in memory and the decoder could decode at its maximum speed.

3.4.7 Logic Synthesis and Performance

The decoder was described in VHDL and was verified against the C simulation as a benchmark. The VHDL simulations showed that the decoder provided the exact same bit-for-bit results as the C simulation and therefore the FPGA decoder is expected to perform as

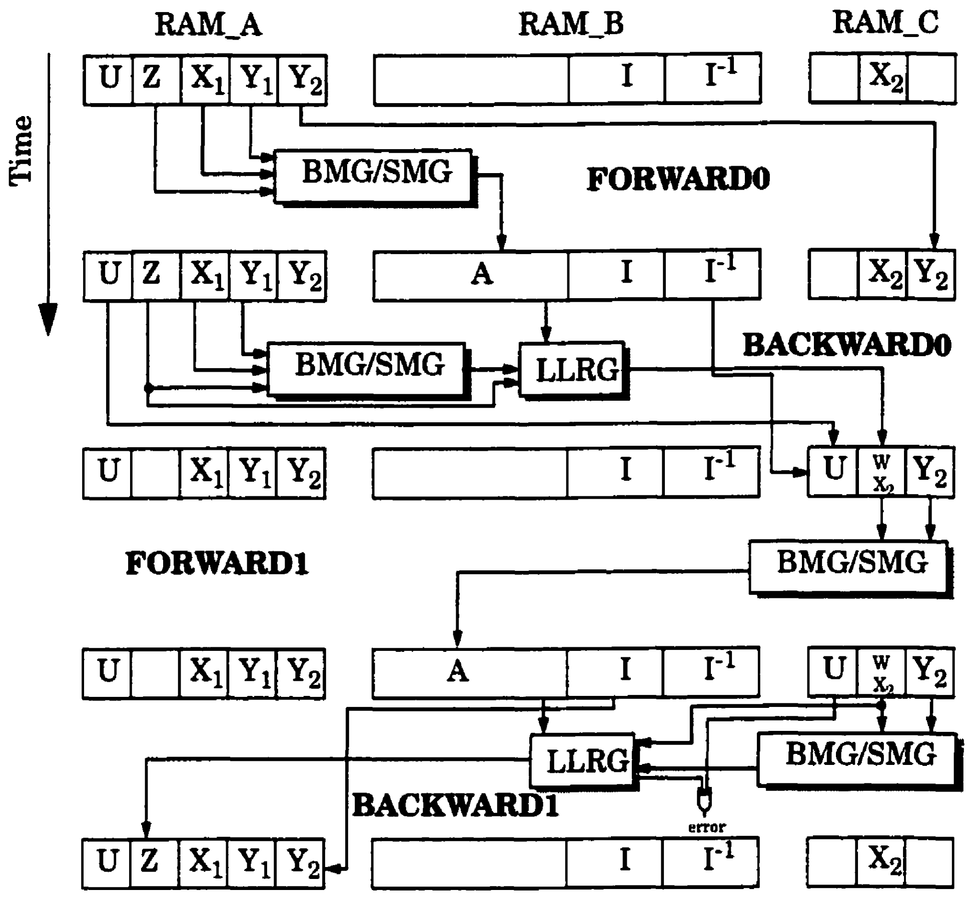


Fig. 3-21: TORBO-TM2 memory schedule.

the simulations in Section 3.3.3 predict. FPGA_0, which contains most of the logic, occupies 3343 logic blocks or 66% of a 10k100 FPGA on the TM-2A. FPGA1 uses 127 logic blocks (2%). The Altera MAX-PlusII software reported that the design would operate at 3.66 MHz. The actual speed of operation is expected to be lower due to the routing through the I-cube chips and miscellaneous circuits on the TM-2. If the data transfer bottleneck can be eliminated by developing an S-bus interface, the decoding rate is expected to be

$$\frac{fN/(4N+18)}{\#\ \text{iterations}}\ \text{bits/s} \qquad (3\text{-}4)$$

where f is the decoder speed in Hz. For example, if N = 1000 and f = 3 MHz, then a 5 iteration decoder would decode 149 kbps and a 10 iteration decoder would decode 74 kbps.

3.5 Summary

This chapter described the design of an FPGA turbo decoder. The main contributions of this chapter are:

- Turbo decoding can be implemented efficiently with small wordlength integer arithmetic (8-bit maximum) with low losses in BER performance.
- The MAX* operation can be implemented with a very small lookup table. A novel circuit realizing the simplified MAX* operation was introduced.
- An FPGA implementation of a turbo decoder, TORBO-TM2 was described.

Chapter 4
A Soft-Output Partial Response Detector

Magnetic disk drives are an example of a communication system that exhibits intersymbol interference (ISI). In recording, the information transmission is not through space as in most communication systems, but rather in time. Nonetheless, the same models of ISI and error-control coding can be applied. Modern disk drive read channels have adopted the Viterbi algorithm to combat ISI as recording densities increase. This chapter introduces a novel soft-output detector for magnetic recording read channels. Section 4.1 is a brief introduction to the PR4 signalling scheme and the special difference-metric Viterbi algorithm (DMVA) used to decode it. An analogous soft-output difference-metric forward backward (DMFB) algorithm is derived in Section 4.2. Section 4.3 describes VLSI architectures for the DMFB. PRONTO-1, an ASIC implementation of the DMFB is described in Section 4.4. Finally, Section 4.5 summarizes the main contributions of this chapter.

4.1 Partial Response for Magnetic Recording

4.1.1 Class-IV Partial Response for Magnetic Recording (PR4)

As hard disk recording densities increase, the pulses detected by the magnetic read head interfere with each other introducing intersymbol interference (ISI). Modern disk drive read channels use a known model for the ISI to combat it in the receiver with advanced signal processing. Partial response signalling is a technique where a controlled amount of intersymbol interference (ISI) is introduced into a communications system to shape the spectrum of the transmitted signal. For example, partial-response systems can be realized to eliminate frequency components that the channel cannot efficiently transmit, such as DC or high-frequencies. The ISI can be modeled by a FIR filter (tapped feedforward shift register), much like a convolutional encoder with real instead of modulo-2 addition. The output of the encoder is therefore multi-level. Partial-response FIR filters have the general transfer function

$$F(D) = 1 + \sum_{j=1}^{N-1} a_k D^j \tag{4-1}$$

shown in Fig. 4-1 where D is a unit delay.

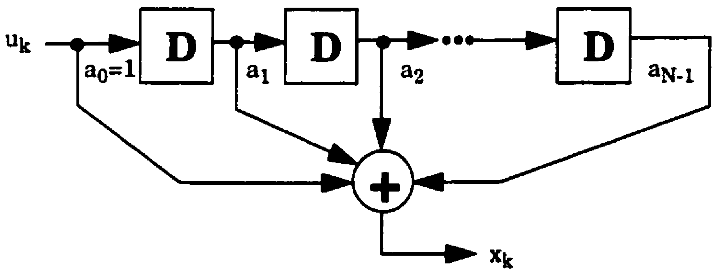


Fig. 4-1: The general structure of an FIR filter that models the partial response transfer function of Equation 4-1.

Class-IV partial response (PR4) signalling for magnetic recording was proposed by Kobayashi in [Kobayashi71][1] and has the transfer function:

$$F(D) = (1 + D)(1 - D) = 1 - D^2 \tag{4-2}$$

Fig. 4-2 illustrates the impulse and frequency responses of PR4 signalling. A consequence of F(D) being dependent only on D^2 is that the transfer function can be broken down into two independent time-interleaved 1-D functions. The detection problem therefore is reduced to decoding unit memory ISI which can be easily done with 2-state Viterbi detectors as shown in Fig. 4-3.

4.1.2 The Difference Metric Viterbi Algorithm

The Viterbi detector required to decode PR4 signalling has just two states. A well known property of the VA is that a constant value can be subtracted from all of the state metrics at

1. As recording densities increase, extended class-IV partial response (EPR4) with transfer function $F(D) = 1 + D - D^2 - D^3$ is becoming more popular.

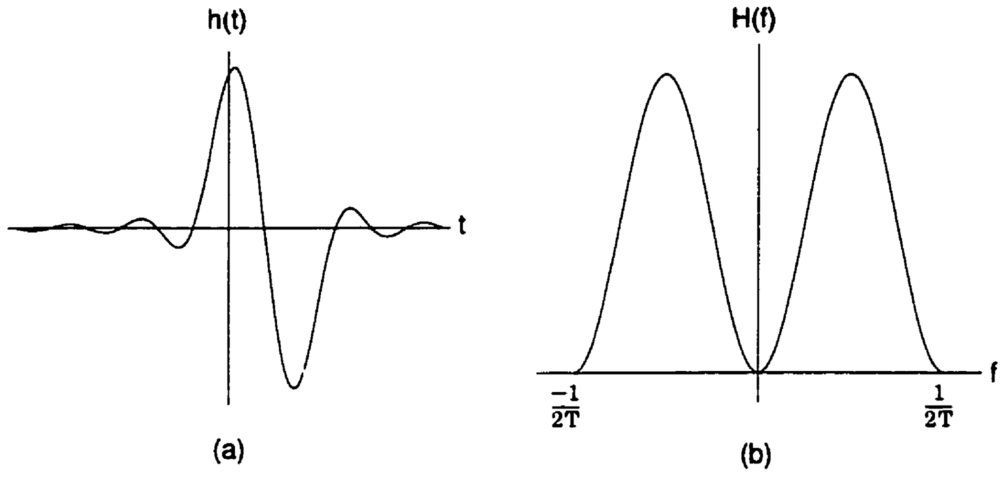


Fig. 4-2: PR4 impulse response (a) and frequency response (b).

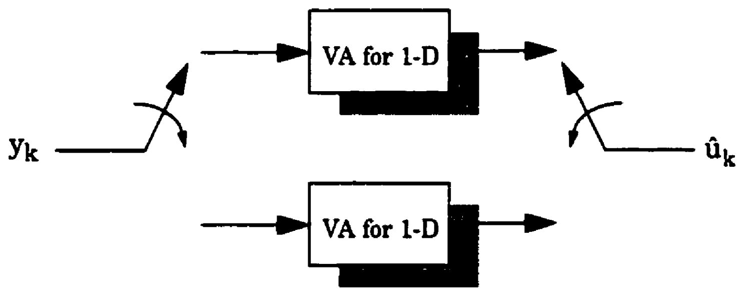


Fig. 4-3: A PR4 decoder can be built with two independent time-interleaved 1-D decoders. The Viterbi decoders are clocked at 1/2 the channel rate.

each time interval without affecting the correctness. This property is often used to limit the dynamic range of the state metrics through renormalizing. Applying this concept to a two-state trellis, it is possible to distribute only one value, the state metric difference between stages of the trellis instead of the individual state metrics. This idea is the basis of the difference metric Viterbi algorithm (DMVA) introduced in [Ferguson72].

Although the DMVA can be applied to any partial response system of the form:

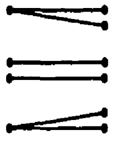
$$F(D) = 1 + \rho D \tag{4-3}$$

we present the specific case of PR4 signalling where $\rho = 1$ without loss of generality. Define the *difference metric* as:

$$\Delta_k = A_k(0) - A_k(1) \tag{4-4}$$

The DMVA compares the difference metric at each stage to the received signal y_k and updates it as:

$$\Delta_k = \begin{cases} y_k - 0.5 & \Delta_{k-1} < y_k - 0.5 \\ \Delta_{k-1} & y_k - 0.5 < \Delta_{k-1} < y_k + 0.5 \\ y_k + 0.5 & y_k + 0.5 < \Delta_{k-1} \end{cases} \qquad (4\text{-}5)$$



where the graphics on the right hand side correspond to the update required in the traceback unit. The operation in Equation 4-5 is a type of adaptive limiter that has efficient digital [FKST95] and analog implementations [SJM98]. A consequence of the limiter operation is that the value is either passed through to the next stage, or a new value is brought in, which eliminates cumulative calculation error and also the need to re-normalize the metrics.

4.2 The Difference Metric Forward-Backward Algorithm

The difference metric forward-backward algorithm (DMFB) is a soft-output detector for two-state partial response systems with the form of Equation 4-1 such as PR4 signalling.

4.2.1 Motivation

The first question to be answered is what is the use of soft outputs in the PR4 channel. We motivate the need for soft outputs by considering the classical serial concatenated system in Fig. 4-4 that was introduced in Chapter 3. The inner code in this case is the PR4 FIR filter and the outer code is chosen to be the 4-state RSC shown in Fig. 3-3(a). The choice of outer code is rather arbitrary and we could substitute a higher power code in its place.

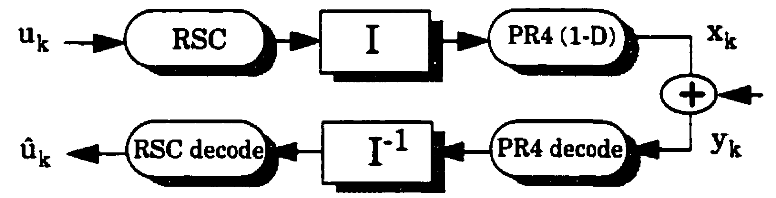


Fig. 4-4: Concatenating a convolutional code with PR4 signalling. The information shared between the two decoders can either be hard or soft.

The standard way of implementing the PR4 decoder is the difference metric Viterbi algorithm. Of course, since it can only produce hard outputs, some vital information has been thrown away. It would be nice to be able to provide soft information to the RSC decoder to improve the performance. In fact, the simulation results of Fig. 4-5 show that a soft PR4 decoder using the forward-backward algorithm provides a 1.8 dB improvement at a bit-error rate of 10^{-5}. Also plotted is the performance of the approximate forward-backward algorithm which uses MAX instead of MAX*. We will show in Section 4.3 that it may be much more convenient to implement the MAX forward-backward algorithm and these simulations show that the soft outputs generated are nearly as good as the optimum soft-outputs.

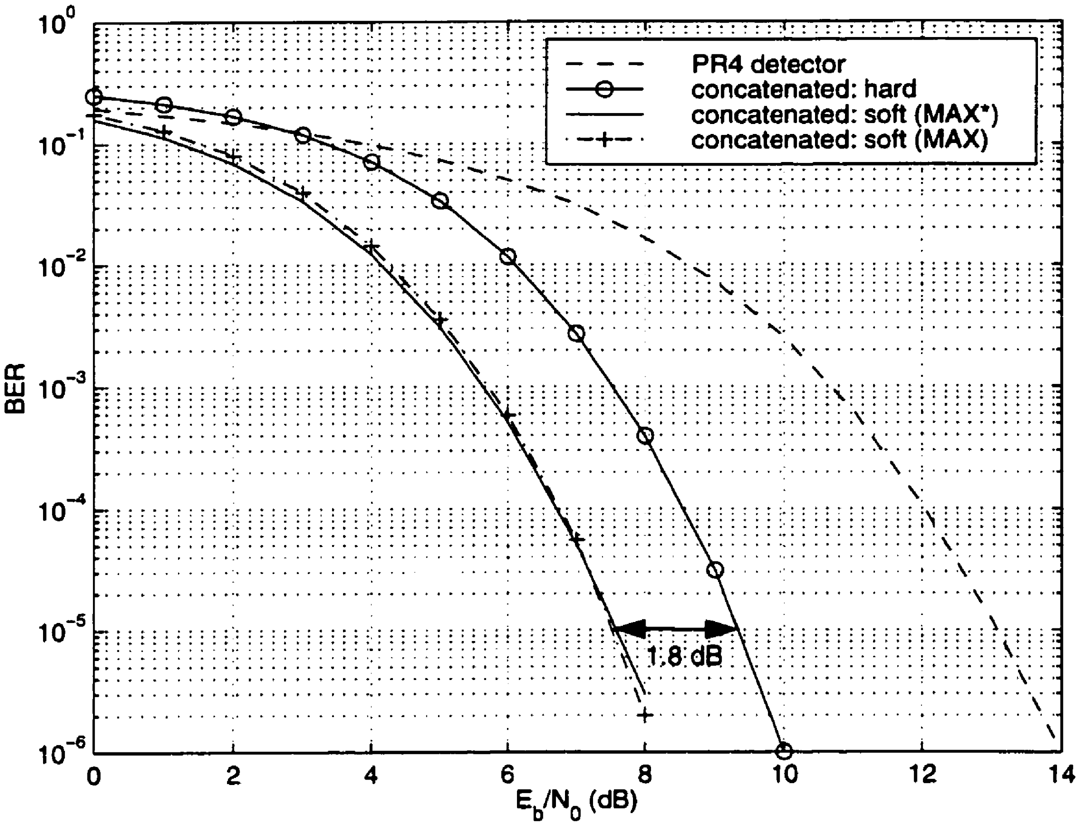


Fig. 4-5: Bit-error rate performance of a serial concatenation of a 4-state 7/5 RSC with PR4.The blocksize is 1K bits. A soft PR4 detector provides 1.8 dB of gain over a hard output detector at a BER of 10^{-5}. The approximate forward-backward algorithm using MAX as the arithmetic kernel performs nearly as well as the optimum MAX* algorithm.

The second question concerns complexity. The simplicity of the DMVA is very attractive in terms of VLSI implementation and leads us to ask if there is an analogous implementation for the forward-backward algorithm. Intuitively, the prospects are encouraging since the forward-backward algorithm is essentially two Viterbi-like algorithms running in opposite directions. In fact, a difference metric forward-backward algorithm (DMFB) does exist as we will show in Section 4.2.2. In Section 4.3, we will further develop the MAX version of the new DMFB algorithm and show that the same limiter structure from the DMVA can be used.

4.2.2 Derivation of the Algorithm

The difference metric forward-backward algorithm can be applied to any two-state partial response trellis where the information bit labels on the two branches entering each state are equal. This fact, noticed by Gaudet, will allow the algebraic manipulations that reveal the difference metric implementation [Gaudet98].

Consider the partial response system with transfer function

$$F(D) = 1 + \rho D \tag{4-6}$$

as show in Fig. 4-6. The two states are labelled -1 and +1, the input symbols are -1 and 1 and the time index is k.

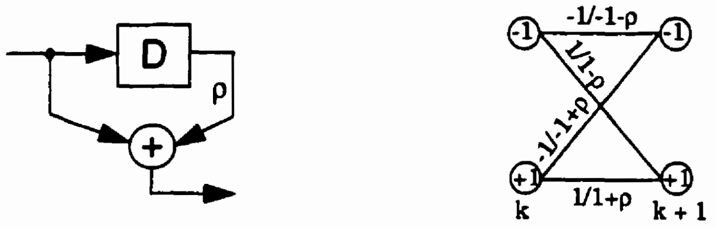


Fig. 4-6: The partial response system with transfer function F(D) = 1 + ρD.

The branch probabilities for received symbol y_k are:

$$\begin{aligned}
\gamma_k(-1,-1) &= \exp\left(-\frac{(y_k-(-1-\rho))^2}{2\sigma^2}\right)\\
\gamma_k(-1,+1) &= \exp\left(-\frac{(y_k-(1-\rho))^2}{2\sigma^2}\right)\\
\gamma_k(+1,-1) &= \exp\left(-\frac{(y_k-(-1+\rho))^2}{2\sigma^2}\right)\\
\gamma_k(+1,+1) &= \exp\left(-\frac{(y_k-(1+\rho))^2}{2\sigma^2}\right)
\end{aligned} \tag{4-7}$$

The forward recursion can be expressed as:

$$\begin{aligned}
\alpha_{k+1}(-1) &= \alpha_k(-1)\gamma_{k+1}(-1,-1)+\alpha_k(+1)\gamma_{k+1}(+1,-1)\\
\alpha_{k+1}(+1) &= \alpha_k(-1)\gamma_{k+1}(-1,+1)+\alpha_k(+1)\gamma_{k+1}(+1,+1)
\end{aligned} \tag{4-8}$$

and the backward recursion as:

$$\begin{aligned}\beta_k(-1) &= \beta_{k+1}(-1)\gamma_{k+1}(-1,-1) + \beta_k(+1)\gamma_{k+1}(-1,+1)\\ \beta_k(+1) &= \beta_{k+1}(-1)\gamma_{k+1}(+1,-1) + \beta_k(+1)\gamma_{k+1}(+1,+1)\end{aligned} \tag{4-9}$$

We write the expression for the LLR explicitly:

$$L_k = \ln\frac{\alpha_{k-1}(-1)\gamma_k(-1,+1)\beta_k(+1) + \alpha_{k-1}(+1)\gamma_k(+1,+1)\beta_k(+1)}{\alpha_{k-1}(-1)\gamma_k(-1,-1)\beta_k(-1) + \alpha_{k-1}(+1)\gamma_k(+1,-1)\beta_k(-1)} \tag{4-10}$$

The structure of the trellis lets us factor out β_k:

$$L_k = \ln\frac{\beta_k(+1)(\alpha_{k-1}(-1)\gamma_k(-1,+1) + \alpha_{k-1}(+1)\gamma_k(+1,+1))}{\beta_k(-1)(\alpha_{k-1}(-1)\gamma_k(-1,-1) + \alpha_{k-1}(+1)\gamma_k(+1,-1))} \tag{4-11}$$

The resulting expression contains the forward recursion equations. Substituting Equation 4-8 into Equation 4-11 we get:

$$\begin{aligned}L_k &= \ln\left(\frac{\beta_k(+1)\alpha_k(+1)}{\beta_k(-1)\alpha_k(-1)}\right)\\ &= \ln\left(\frac{\beta_k(+1)}{\beta_k(-1)}\right) + \ln\left(\frac{\alpha_k(+1)}{\alpha_k(-1)}\right)\end{aligned} \tag{4-12}$$

The LLR is now expressed in terms of a sum of the quotients of forward and backward state probabilities. Since all the state probabilities can be scaled by the same number without affecting the correctness of the algorithm, the recursions can propagate the quotient of the state probabilities instead of the two individual probabilities. In the log domain we would express Equation 4-12 in terms of the difference metrics A_k and B_k:

$$\begin{aligned}L_k &= \ln(\beta_k(+1)) - \ln(\beta_k(-1)) + \ln(\alpha_k(+1)) - \ln(\alpha_k(-1))\\ &= B_k + A_k\end{aligned} \tag{4-13}$$

where

$$\begin{aligned}A_k &= \ln(\alpha_k(+1)) - \ln(\alpha_k(-1))\\ B_k &= \ln(\beta_k(+1)) - \ln(\beta_k(-1))\end{aligned} \tag{4-14}$$

So the algorithm is reduced to recursively calculating a single forward metric, recursively calculating a single backward metric and then simply adding them together.

To derive the forward difference metric recursion equation, use Equation 4-8 to write:

$$\ln\left(\frac{\alpha_{k+1}(+1)}{\alpha_{k+1}(-1)}\right) = \ln\left(\frac{\alpha_k(-1)\gamma_{k+1}(-1,+1)+\alpha_k(+1)\gamma_{k+1}(+1,+1)}{\alpha_k(-1)\gamma_{k+1}(-1,-1)+\alpha_k(+1)\gamma_{k+1}(+1,-1)}\right) \tag{4-15}$$

Substituting the expressions for the branch probabilities from Equation 4-7, followed by dividing and simplifying we get:

$$\begin{aligned}\ln\left(\frac{\alpha_{k+1}(+1)}{\alpha_{k+1}(-1)}\right) &= \ln\left[\frac{\alpha_k(+1)}{\alpha_k(-1)}+e^{\frac{-2\rho}{\sigma^2}(y_{k+1}-1)}\right]-\ln\left[1+\frac{\alpha_k(+1)}{\alpha_k(-1)}e^{\frac{2\rho}{\sigma^2}(y_{k+1}+1)}\right]\\ &= \ln\left[e^{\ln\left(\frac{\alpha_k(+1)}{\alpha_k(-1)}\right)}+e^{\frac{-2\rho}{\sigma^2}(y_{k+1}-1)}\right]-\ln\left[e^0+e^{\frac{2\rho}{\sigma^2}(y_{k+1}+1)+\ln\left(\frac{\alpha_k(+1)}{\alpha_k(-1)}\right)}\right]\end{aligned} \tag{4-16}$$

In terms of the difference metric and the MAX* notation:

$$A_{k+1} = \mathrm{MAX}^*\left(A_k, \frac{-2\rho}{\sigma^2}(y_{k+1}-1)\right)-\mathrm{MAX}^*\left(0, A_k+\frac{2\rho}{\sigma^2}(y_{k+1}+1)\right) \tag{4-17}$$

Similarly, the backward recursion is:

$$B_{k-1} = \mathrm{MAX}^*\left(B_k, \frac{2\rho}{\sigma^2}(y_k+1)\right)-\mathrm{MAX}^*\left(0, B_k+\frac{-2\rho}{\sigma^2}(y_k-1)\right) \tag{4-18}$$

4.2.3 Initialization

For a block-based decoder, the starting and ending states are forced to -1. The initial values of the recursions are:

$$\begin{aligned}A_0 &= \ln\left(\frac{\alpha_0(+1)}{\alpha_0(-1)}\right)\\ &= \ln\left(\frac{0}{1}\right)\\ &= -\infty\end{aligned} \tag{4-19}$$

Similarly,

$$B_N = -\infty \tag{4-20}$$

We note that the initial values of the difference metrics are never actually used in the LLR calculation, which only needs to decode symbols at $k = 1\ldots N-1$.

Substituting A_0 and B_0 into the expressions for A_1 and B_{N-1} we get:

$$\begin{aligned} A_1 &= \lim_{\phi \to -\infty}\left(\mathrm{MAX}^*\left(\phi, \frac{-2\rho}{\sigma^2}(y_1 - 1)\right) - \mathrm{MAX}^*\left(0, \phi + \frac{2\rho}{\sigma^2}(y_1 + 1)\right)\right) \\ &= \frac{-2\rho}{\sigma^2}(y_1 - 1) + \lim_{\phi \to -\infty}\left(\ln\left(1 + e^{-\left|\phi + \frac{2\rho}{\sigma^2}(y_1 - 1)\right|}\right)\right) - \lim_{\phi \to -\infty}\left(\ln\left(1 + e^{-\left|\phi + \frac{2\rho}{\sigma^2}(y_1 + 1)\right|}\right)\right) \\ &= \frac{-2\rho}{\sigma^2}(y_1 - 1) \end{aligned} \tag{4-21}$$

and

$$B_{N-1} = \frac{2\rho}{\sigma^2}(y_1 + 1) \tag{4-22}$$

which eliminates the need for infinite values anywhere. For a sliding window implementation, the ending states are equiprobable and

$$B_N = \ln\left(\frac{1}{2}/\frac{1}{2}\right) = 0 \tag{4-23}$$

4.2.4 Normalization and Overflow

The DMFB is equivalent to explicitly calculating both states and then subtracting one from each to normalize the state metrics. The redundant value of zero does not need to be propagated. The DMFB is therefore self-normalizing. Overflow can occur at large values of the SNR, just as in the regular formulation of the FB algorithm since there is a division by a very small noise variance in the branch metrics. Although not a practical concern in floating point arithmetic for most channels, the growth of the state metrics with increasing SNR can affect the wordlength required for an integer implementation. This issue will be considered and a solution proposed in Section 4.3, which describes practical VLSI implementations.

4.2.5 Summary of the DMFB for Class-IV Partial Response

- **Branch Metrics**

$$\begin{aligned} G_k^{-1} &= \frac{2}{\sigma^2}(y_k - 1) \\ G_k^{+1} &= \frac{2}{\sigma^2}(y_k + 1) \end{aligned} \qquad k = 1 \ldots N-1 \tag{4-24}$$

- **Forward Recursion**

$$\begin{aligned} A_1 &= G_1^{-1} \\ A_{k+1} &= \mathrm{MAX}^*(A_k, G_{k+1}^{-1}) - \mathrm{MAX}^*(0, A_k - G_{k+1}^{+1}) \qquad k = 1 \ldots N-2 \end{aligned} \tag{4-25}$$

- **Backward Recursion**

$$\begin{aligned} B_{N-1} &= -G_{N-1}^{+1} \\ B_{k-1} &= \mathrm{MAX}^*(B_k, -G_k^{+1}) - \mathrm{MAX}^*(0, B_k - G_k^{-1}) \qquad k = N-1 \ldots 2 \end{aligned} \tag{4-26}$$

- **Soft Outputs**

$$L_k = A_k + B_k \qquad k = 1 \ldots N-1 \tag{4-27}$$

4.3 VLSI Architectures for Class-IV Partial Response

4.3.1 The Limiter Form of the DMFB: MAX-DMFB

We have shown that a difference metric formulation of the forward-backward algorithm does indeed exist. It then seems reasonable that if the MAX approximation of the FB algorithm is used, then the recursion update can be done using a limiter as in the DMVA. The simulations of Fig. 4-5 show that the approximation is reasonable. In fact, the soft decisions will be equivalent to those provided by Battail's SOVA [Battail87]. The MAX DMFB that we will derive can therefore be thought of as an alternative derivation for a difference metric SOVA.

If the MAX approximation is used, then the forward recursion becomes:

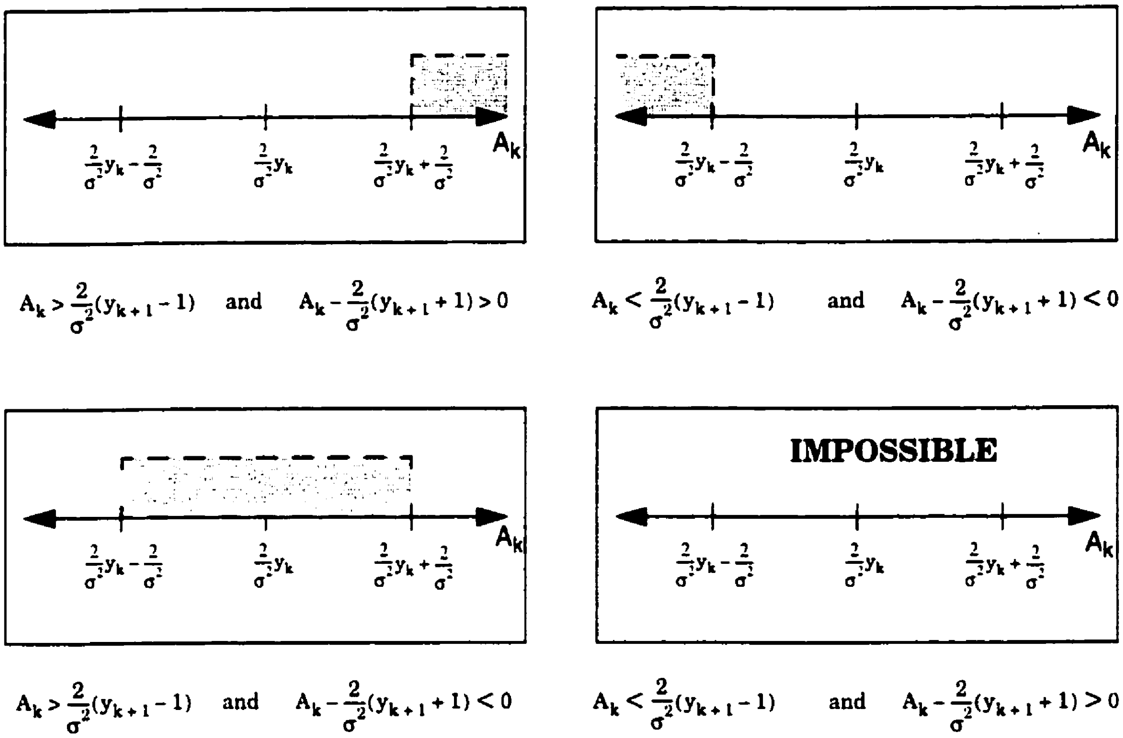
$$A_{k+1} = \mathrm{MAX}\left(A_k, \frac{2}{\sigma^2}(y_{k+1} - 1)\right) - \mathrm{MAX}\left(0, A_k - \frac{2}{\sigma^2}(y_{k+1} + 1)\right) \tag{4-28}$$

for which there are the four distinct cases as illustrated in Fig. 4-7. Since the inputs to each MAX term are related, this imposes a constraint that makes the fourth case impossible. By considering the three possible cases, the new recursion becomes:

$$A_{k+1} = \begin{cases} G_{k+1}^{-1}, & G_{k+1}^{-1} > A_k \\ A_k, & G_{k+1}^{-1} \le A_k \le G_{k+1}^{+1} \\ G_{k+1}^{+1}, & A_k > G_{k+1}^{+1} \end{cases} \tag{4-29}$$

Equation 4-29 can be interpreted as the limiter in Fig. 4-8. A similar expression can be derived for the backward recursion:

$$B_k = \begin{cases} -G_{k+1}^{+1}, & -G_{k+1}^{+1} > B_{k+1} \\ B_{k+1}, & -G_{k+1}^{+1} \le B_{k+1} \le -G_{k+1}^{-1} \\ -G_{k+1}^{-1}, & B_{k+1} > -G_{k+1}^{-1} \end{cases} \tag{4-30}$$



A_k
$\frac{2}{\sigma^2}y_k - \frac{2}{\sigma^2}$
$\frac{2}{\sigma^2}y_k$
$\frac{2}{\sigma^2}y_k + \frac{2}{\sigma^2}$
$A_k > \frac{2}{\sigma^2}(y_{k+1} - 1)$ and $A_k - \frac{2}{\sigma^2}(y_{k+1} + 1) > 0$
$A_k < \frac{2}{\sigma^2}(y_{k+1} - 1)$ and $A_k - \frac{2}{\sigma^2}(y_{k+1} + 1) < 0$
$A_k > \frac{2}{\sigma^2}(y_{k+1} - 1)$ and $A_k - \frac{2}{\sigma^2}(y_{k+1} + 1) < 0$
IMPOSSIBLE
$A_k < \frac{2}{\sigma^2}(y_{k+1} - 1)$ and $A_k - \frac{2}{\sigma^2}(y_{k+1} + 1) > 0$

Fig. 4-7: The possible combinations of inputs to the forward recursion equation.

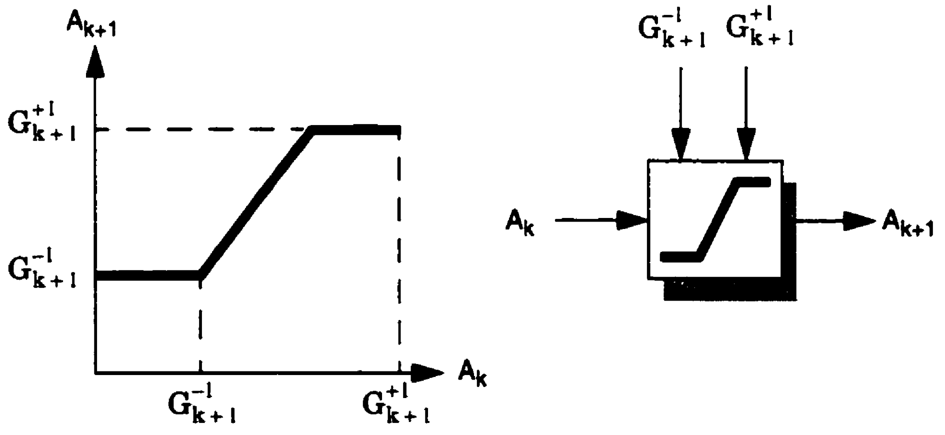


Fig. 4-8: The limiter used as the update rule for the forward recursion equation.

One of the advantages of the limiter formulation is that no arithmetic computation is required. At each recursion step, the input or one of the two threshold levels is copied to the output eliminating any calculation error propagation.

The limiter enables us to build very fast circuits. To see this more clearly, we can rederive the same result from a hardware viewpoint. Fig. 4-9 shows the hardware implementation of Equation 4-28 as well as the limiter of Equation 4-29. The 'naive' circuit in Fig. 4-9(a) uses redundant hardware to implement the impossible case from Fig. 4-7. The final subtractor is needed only when point Q is non-zero. i.e:

$$Q = A_k - \frac{2}{\sigma^2}(y_{k+1} + 1) \tag{4-31}$$

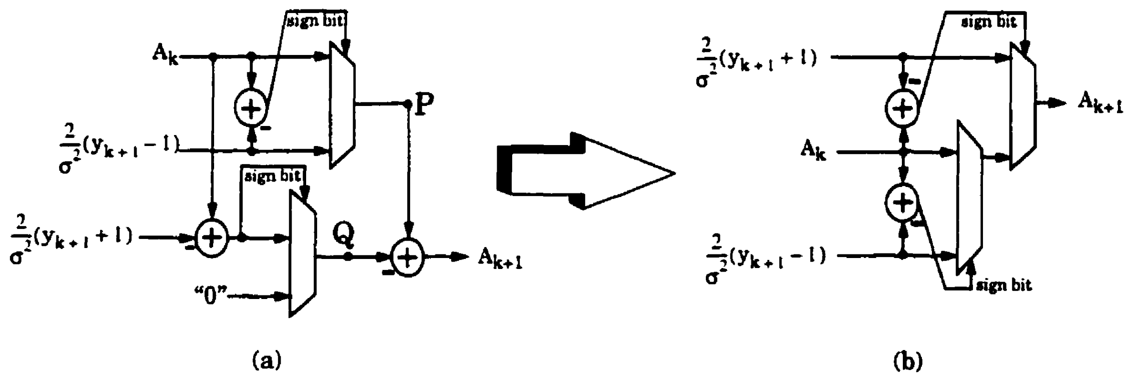
This implies that $A_k > \frac{2}{\sigma^2}(y_{k+1} - 1)$ i.e., node P will always be:

$$P = A_k \tag{4-32}$$

Therefore:

$$\begin{aligned} A_{k+1} &= P - Q \\ &= A_k - \left(A_k - \frac{2}{\sigma^2}(y_{k+1} + 1)\right) \\ &= \frac{2}{\sigma^2}(y_{k+1} + 1) \end{aligned} \tag{4-33}$$

which is just one of the inputs, eliminating the need for the final subtractor. The critical path of the resulting limiter is just one adder and two multiplexers. Since only the MSB of the subtractions are used in the limiter, the adders can be replaced by comparator circuits resulting in a *Compare-Select-Select* (CSS) operation.



sign bit
A_k
$\frac{2}{\sigma^2}(y_{k+1} - 1)$
$\frac{2}{\sigma^2}(y_{k+1} + 1)$
P
sign bit
Q
"0"
A_{k+1}
(a)
sign bit
$\frac{2}{\sigma^2}(y_{k+1} + 1)$
A_k
$\frac{2}{\sigma^2}(y_{k+1} - 1)$
A_{k+1}
sign bit
(b)

Fig. 4-9: A limiter saves an adder in the critical path of the recursion hardware.

4.3.2 Eliminating the Noise Variance Problem

The presence of the noise variance σ^2 in the branch metric expressions complicates the implementation of the DMFB. As mentioned in Section 4.2.4, the word length of the metrics needs to accommodate the smallest value of σ^2. Multiplication is also a highly undesirable operation. These problems can be eliminated by simply eliminating the term $\frac{2}{\sigma^2}$ from the branch metrics so:

$$\begin{aligned} G_k^{-1} &= y_k - 1 \\ G_k^{+1} &= y_k + 1 \end{aligned} \qquad k = 1 \ldots N-1 \tag{4-34}$$

This is a valid solution since the new state metric at each stage is either one of the current branch metrics or the previous state metric. Therefore, each state metric will take on the value of either the initial state metric or one of the 2(N-1) branch metrics each of the form in Equation 4-34. There is therefore no cumulative error in this approximation and the new soft output will simply be the desired value scaled by $\sigma^2/2$.

4.3.3 The Sliding Window Architecture

The sliding window architectures introduced in Section 2.3 can be easily applied to the MAX-DMFB. An efficient implementation (for M_b=D_L) requires only 3 limiters and 4D_L words of memory. Although the architecture for M_b =1 shown in Fig. 2-9 is not as efficient in the general case, it still provides a competitive alternative in the MAX-DMFB case where there is only one state metric propagated and D_L will generally be small. The pipelined version of the architecture is shown in Fig. 4-10. This architecture is ideal for rapid hardware prototyping. Since the whole process is self-regulating, with no control unit it is easy to implement quickly as a proof-of-concept.

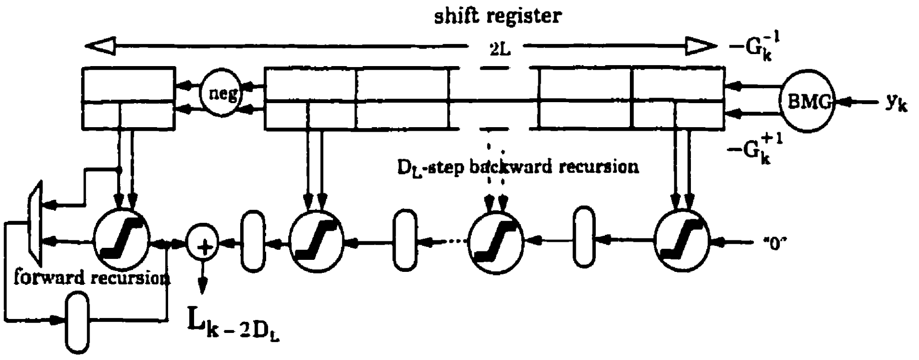


Fig. 4-10: The sliding window MAX-DMFB for M_b = 1.

An interesting computational challenge will now be considered. The backward recursive calculation is really not necessary in one sense because all of the inputs to the calculation (the branch metrics and the initial state metric) are always available in the shift register at once. Therefore, in theory at least, all possible inputs and outputs could be tabulated in a very large lookup table (see Fig. 4-11(a)) [TzouD81]. Of course, a lookup table that large could never be practically built, and if it could, the access time would be a still be a function of the table size. For example, if $D_L = 10$, then with say 6-bit input symbols the input address to the table would be 60 bits wide! A compromise is to use a tree-like structure with $\log_2 D_L$ stages as proposed in [FetweisM91] for the Viterbi algorithm (see Fig. 4-11(b)). The question remains of how to implement the new type of processor required. Fortunately, the limiter concept we introduced provides an interesting and simple solution.

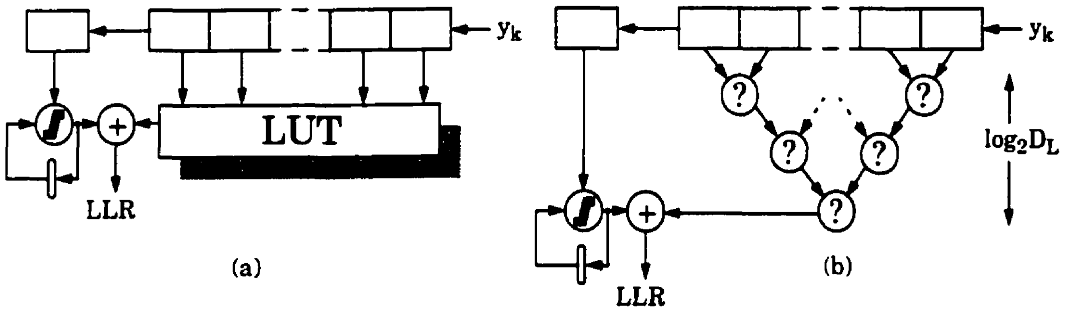


Fig. 4-11: Parallel processing of the backward recursion using (a) a lookup table or (b) A tree.

The concept can be derived by a simple analogy with a high-rise building that has developed a leak in its roof (see Fig. 4-12). If each floor also has a hole in it, then assuming the floor dips towards the hole, the water will run along the floor until it falls through the hole. If the hole on two adjacent floors overlap, then the water can possibly pass right through both holes without ever creating a puddle on the floor! The chain of limiters in our problem are like the stack of floors in the building. Each floor represents the number line, with the linear part of the limiter represented by the hole and the cutoff regions represented by the concrete floor. The relative location of the hole is determined by the value of the input symbols by way of the branch metrics. If all the floors of the building were torn out and replaced by a single floor with one hole in the correct place then the puddle in the basement would still be in the same place.

The lookup table approach calculates the equivalent overall limiter in one step while the tree approach uses $\lceil \log_2 D_L \rceil$ steps by considering pairs of adjacent limiters. The new limiter functions are then paired up and the calculation is repeated until the overall limiter function is known. We will call one of these new processors an *interval adjustment unit* or IAU as shown in Fig. 4-13. The IAU uses one more multiplexer than two limiters and the delay is one multiplexer less than that of two limiters in series.

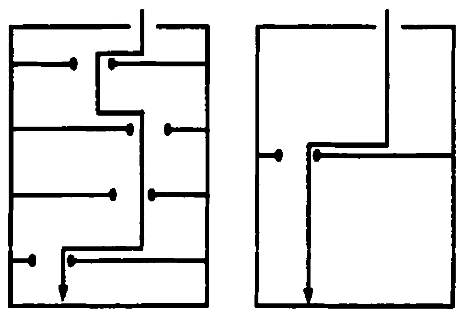
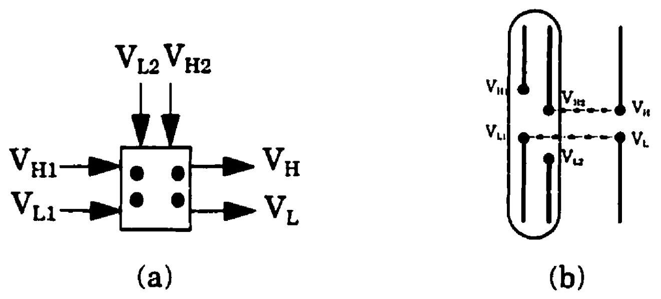


Fig. 4-12: A graphical analogy to a chain of limiters. The chain can be replaced by a single limiter which has the same overall effect.



```
$V_L$ = MAX($V_{L1}$, $V_{L2}$)
$V_H$ = MIN($V_{H1}$, $V_{H2}$)
if ($V_L$ > $V_H$)
    if ($V_{L2}$ > $V_{H1}$)
        $V_H$ = $V_L$
    else
        $V_L$ = $V_H$
```

(c)

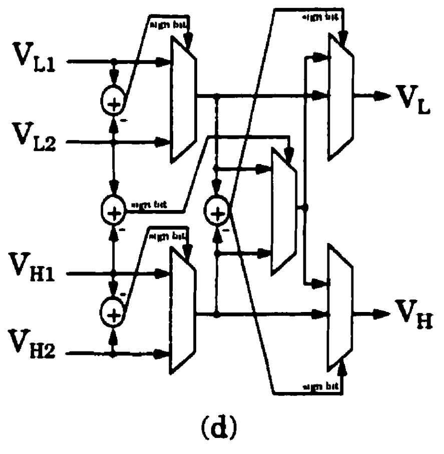


Fig. 4-13: The Interval Adjustment Unit (IAU). (a) Symbol. (b) Example of IAU operation. (c) Algorithm. (d) Hardware implementation.

The tree architecture offers a trade-off between complexity and latency. We note that significant savings in latency are only realized for large values of D_L. The length of the critical path for the tree architecture can be controlled by inserting pipeline registers where desired. A trade-off between memory and hardware is possible within the tree architecture by recognizing that some IAUs are performing redundant calculations since the same input was available to other IAUs at an earlier point in the shift register [Farhang-BoroujenyG94]. An example for $D_L = 10$ is shown in Fig. 4-14.

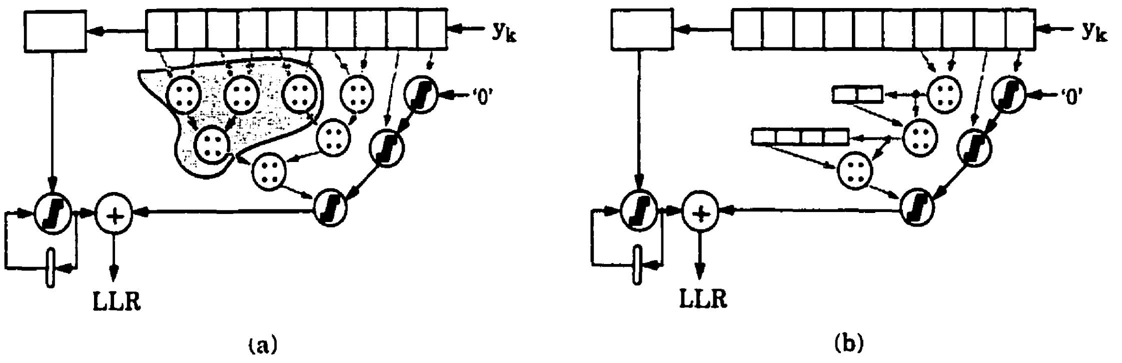


Fig. 4-14: The tree architecture for the sliding window MAX-DMFB using IAUs and limiters. (a) The full tree. Shaded arrows indicate intervals defined by two numbers. The shaded IAUs are redundant and can be pruned. (b) The pruned tree using memory to replace hardware.

4.4 PRONTO-1: An ASIC Implementation

In this section we describe the design and implementation of a test chip for verification of the MAX-DMFB: PRONTO-1. The goal was to design a chip capable of decoding at least 100 Mbps. For the test chip, only one MAX-DMFB detector was implemented. A complete PR4 detector would include two identical detectors as depicted in Fig. 4-3.

4.4.1 MAX-DMFB Detector Architecture

The sliding window architecture of Fig. 4-10 was chosen because of its ease of rapid implementation. If a memory module generator were available, the more efficient three-processor architecture of Fig. 2-12 could have been used to avoid implementing a shift register, reducing the power requirements.

Simulations were performed to determine the performance of the PRONTO-1 chip. A 6-bit two's complement format with 1 sign bit, 1 integer bit and 4 fractional bits was used for all variables. The window length was chosen to be L = 9. The projected performance of the chip plotted relative to a floating point block-based forward-backward detector (N = 1024) is plotted in Fig. 4-15. There is a 0.3 dB loss at low SNR which disappears as the SNR increases and the BER approaches the desired operating conditions. The architecture shown in Fig. 4-16 requires 10 limiters, 301 registers, a BMG and a two's-complementor, a MUX for initialization and a final 6-bit adder.

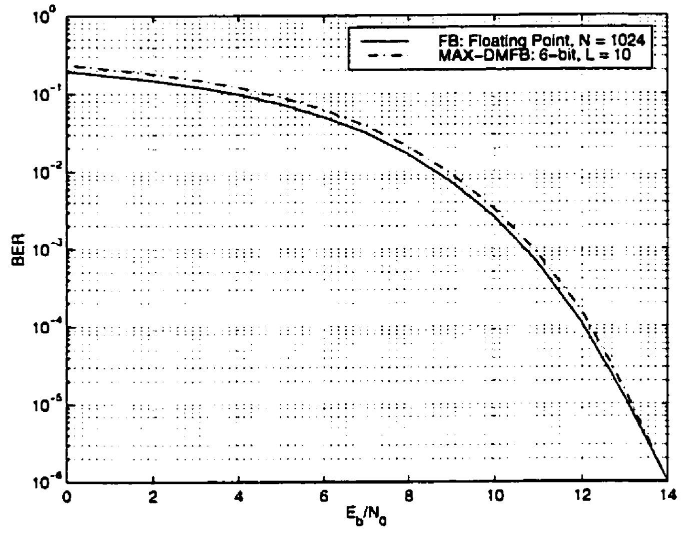


Fig. 4-15: Simulated performance of the PRONTO-1 chip with 6-bit quantization and window length of L = 9 compared to the optimum floating point block-based forward-backward algorithm with blocksize N = 1024.

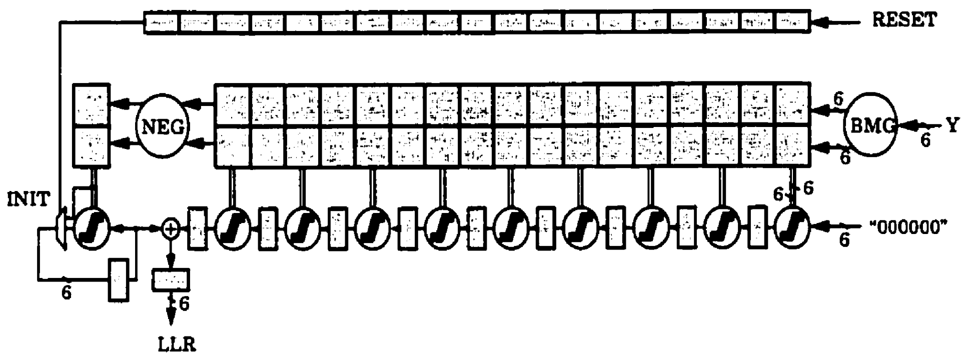


Fig. 4-16: The PRONTO-1 MAX-DMFB detector architecture.

4.4.2 Control Unit

The state of the detector can be reset to state 0 by asserting the RESET signal for one clock cycle, which is delayed by a shift register the appropriate amount to generate the INIT signal to the forward recursion unit. The initial design used a counter to implement this function but it was discovered that the counter required was in the critical path and therefore was replaced by the shift register scheme to improve the speed performance. The reset scheme can be eliminated if the first few decoded bits are ignored while the detector state is synchronized.

4.4.3 Limiter

The presence of a limiter in the forward feedback loop means that it has the potential to lie in the critical path if it is too slow. The limiter (see Fig. 4-9(b)) is designed using a logical comparator circuit instead of a subtractor for speed.

4.4.4 Adders

Initial simulations using the fast limiter design showed that the speed-limiting circuits were the adders in the BMG, NEG unit and the final soft-output adder. The saturation circuit introduced in Chapter 3 was used. A 6-bit carry-look-ahead adder (CLA) design was used for speed. The BMG and NEG units add a constant value to a variable and therefore their logic equations could be simplified greatly.

4.4.5 Clock Doubler

The IC tester available for chip verification is only capable of stimulating the I/O pins at a maximum speed of 60 MHz, which is below our performance targets for PRONTO-1. To work around this limitation, a clock doubler circuit was added to the chip to enable the core to be tested at speeds of up to 120 MHz. The new system can be operated in either single speed (DBL = 0) or double-speed mode (DBL = 1). In double-speed mode, the two parallel inputs and outputs (Y1,Y2) and (L1, L2) are used while in single-speed mode only Y1 and L1 are used. Fig. 4-17 shows the PRONTO-1 top-level architecture with the clock-doubler support circuits added around the MAX-DMFB detector.

4.4.6 Test Chip Synthesis and Layout

The test chip, PRONTO-1 was synthesized to a 0.5 μm CMOS standard cell technology using Synopsys tools. The synthesized netlist was verified by simulation with one million test vectors. Synopsys reported a speed estimate for the chip core of 300 MHz. Taking into consideration the effect of pad loading, clock skew and the fact that this estimate was performed before placement and routing, it would be reasonable to expect the chip to perform at the maximum testable speed (60 MHz at the pins, 120 MHz core). The placement and routing was performed using Cadence, which resulted in a core area of 0.81 mm^2 and an overall silicon area of 7.3 mm^2. After manually fixing some design rule violations the chip passed a design rule check with no errors. Table 4-1 is a summary of the specifications of the PRONTO-1 chip. Fig. 4-18 is a layout plot of the chip.

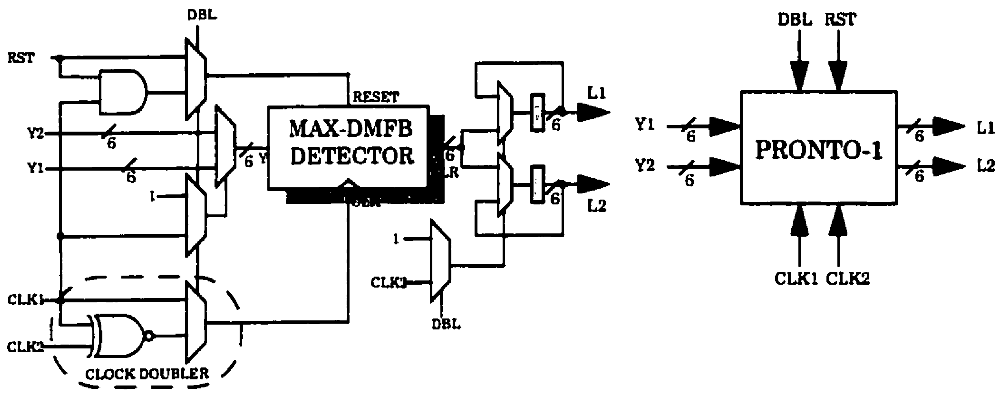


Fig. 4-17: PRONTO-1 top-level architecture.

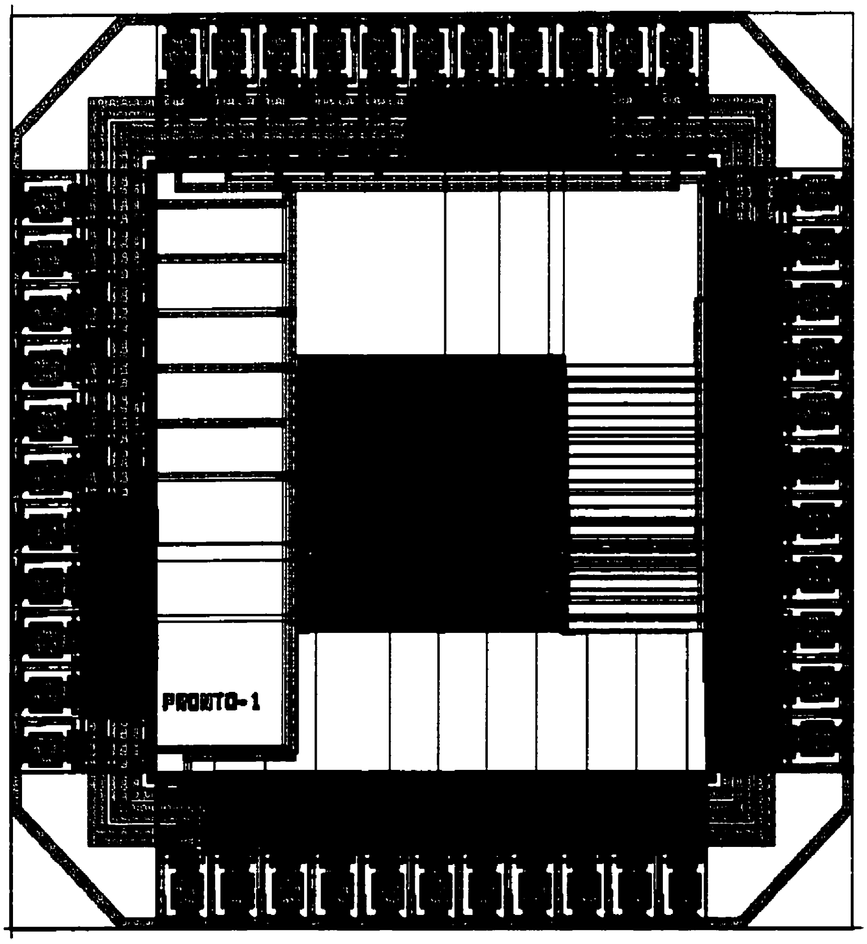


Fig. 4-18: Layout of the PRONTO-1 test chip.

Table 4-1: PRONTO-1 specifications.

Design name	PRONTO-1
Purpose	Soft-output detector for ISI with F(D) = 1-D
Algorithm	MAX-DMFB
Input/output quantization	6 bits
# states	2
Process	0.5 μm CMOS (3M1P)
Supply voltage	3.3 V
Core area	0.81 mm^2
Total area	7.3 mm^2
Pins	44
Estimated speed	> 120 Mbps

4.5 Summary

In this chapter we developed a soft-output detector based on the forward-backward algorithm for the detection of class-IV partial response signals. The contributions of this chapter are:

- The difference metric forward-backward algorithm (DMFB) was developed for a two-state partial response trellis.
- An approximation of the DMFB, the MAX-DMFB was developed. It was shown that the computational kernel of the MAX-DMFB is a limiter.
- A test chip of the MAX-DMFB, PRONTO-1 was implemented.

Chapter 5

Conclusions

5.1 Summary and Conclusions

In this thesis we have described two applications of the forward-backward algorithm and their hardware implementations.

Chapter 2 provided background for discussions of implementing the forward-backward algorithm. In Chapter 2 we first reviewed the soft-output forward backward (FB) algorithm and compared it to the well-known hard-output Viterbi algorithm. Then, the logarithmic form of the algorithm (log-FB) was reviewed to show that the log-FB algorithm is really just two Viterbi-like algorithms operating in opposite directions along the trellis. The sliding window FB algorithm was then described for continuous-time soft-output decoding.

Chapter 3 presented the implementation of a turbo decoder. After a brief introduction to concatenated codes, turbo codes were introduced as a parallel concatenation of recursive systematic convolutional codes. The iterative decoding algorithm, which uses the FB algorithm, provides near-optimal performance and low complexity. The existing collection of hardware turbo decoders were reviewed. Most of the work to date has focused on implementing turbo decoders using the SOVA which is a variant of the FB algorithm that only provides approximate soft output. The designs published in the past year or so are now turning to the FB algorithm using the simplifications discussed in Chapter 2. The design of a general hardware-ready turbo decoder was then presented. The most significant simplification described was a novel low-complexity circuit for the MAX* operation which replaces a traditional LUT and does not significantly affect BER performance. Using this new circuit, an FPGA turbo decoder, TORBO-TM2, was designed for the TM-2 reconfigurable FPGA system.

A soft-output detector for partial response signalling was introduced in Chapter 4. The new algorithm is a form of the log-FB algorithm specifically designed for the two-state PR4 trellis popular in magnetic recording. After a brief review of PR4 and the difference metric Viterbi algorithm, the difference metric forward-backward algorithm was derived. A simplification, the MAX-DMFB allows a significant hardware reduction by reducing the computational kernel to a limiter operation. Next was a discussion of VLSI architectures for the sliding window MAX-DMFB. Finally, the design of a test chip, PRONTO-1 was described.

5.2 Contributions of this Thesis

The contributions of this thesis are:

- A novel low-complexity MAX* circuit was introduced which eliminates the need for a lookup table.
- An architecture for turbo decoding on the TM-2, TORBO-TM2.
- The MAX-DMFB algorithm was introduced. The implementation uses a limiter as its computational kernel just as in the difference metric Viterbi algorithm.
- An ASIC implementation of the MAX-DMFB, PRONTO-1.

5.3 Suggestions for Future Research

Research in the area of soft-output decoding is now starting to emerge from the conceptual stage to practical realizations. We are certain of the existence of unpublished turbo decoder implementations in industry. There are many unanswered questions to solve, and we therefore provide a sample of interesting research projects as our final comment.

5.3.1 Serial and hybrid concatenated codes

These codes provide excellent performance down to very low bit error rates avoiding the error floor of turbo codes. It would be interesting to build hardware to enable experiments at bit error rates not possible by computer simulation.

5.3.2 Interleaver design

Interleaver design for hardware turbo decoders is still an open question. The shift register sequences described in Chapter 3 do not necessarily give the best possible performance.

Algorithms to produce random looking interleavers that can be easily implemented in hardware are needed. We have started a project on general interleaver design during the course of the thesis research but the results are not ready at the time of this writing.

5.3.3 Very high speed DSP or microprocessor turbo decoder

CRC's 400Kbps software decoder demonstrates that high speed microprocessor or DSP implementations are possible. Our software simulations already run at tens of Kbps with no specific optimizations for speed.

5.3.4 Hardware variance estimation

For fading channels, methods of implementing hardware variance estimation for turbo decoding should be explored.

5.3.5 Continuous time turbo decoding on the TM-2

The turbo decoder implemented in this thesis was a block decoder, which is area-efficient and suitable for applications where the transmission speed is low or the data is already available in block format. For continuous high-speed decoding, a pipelined turbo decoder can be implemented on the TM-2, using multiple boards with the goal of implementing one iteration/board. With two 10k100 FPGAs / board, we think this is a reasonable. A 16-board system is therefore capable of 16 iterations.

5.3.6 Turbo decoder ASIC

It would be interesting to apply the ideas described in this thesis to the design of a turbo decoder ASIC.

Appendix

TORBO-TM2 FB Algorithm

Consider the code in Fig. 3-3(a) whose trellis can be expressed as the two independent butterflies shown in Fig. A-1. The state is defined as $\{a_{k-2}, a_{k-1}\}$. We will explicitly state the equations used to implement the FB algorithm for this trellis. Define the input of the rate 1/2 code as $\{x_k, y_k\}$. In a turbo decoder, x_k is the sum of the systematic information and the a-priori information and y_k is the coded information from either the upper or lower codes. The blocksize is N and 2 tail bits are added to force the final state to 0.

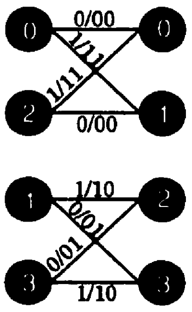


Fig. A-1: TORBO-TM2 trellis

The branch metrics are (k = 1..N+2) [Barbulescu96]:

$$\begin{aligned} \gamma_k(0,0) &= 0 & \gamma_k(1,2) &= x_k \\ \gamma_k(0,1) &= x_k + y_k & \gamma_k(1,3) &= y_k \\ \gamma_k(2,0) &= x_k + y_k & \gamma_k(3,2) &= y_k \\ \gamma_k(2,1) &= 0 & \gamma_k(3,3) &= x_k \end{aligned} \qquad \text{(A-1)}$$

The forward state metrics are (k = 1..N-1):

$$\begin{aligned}
A_k(0) &= \mathrm{MAX}^*(A_{k-1}(0), A_{k-2}(2) + (x_k + y_k)) \\
A_k(1) &= \mathrm{MAX}^*(A_{k-1}(0) + (x_k + y_k), A_{k-2}(2)) \\
A_k(2) &= \mathrm{MAX}^*(A_{k-1}(1) + x_k, A_{k-2}(3) + y_k) \\
A_k(3) &= \mathrm{MAX}^*(A_{k-1}(1) + y_k, A_{k-2}(3) + x_k)
\end{aligned} \tag{A-2}$$

subject to the initial conditions:

$$\begin{aligned}
A_0(0) &= 0 \\
A_0(1) &= -\infty \\
A_0(2) &= -\infty \\
A_0(3) &= -\infty
\end{aligned} \tag{A-3}$$

Grouping the backward state metrics and branch metrics, define:

$$\begin{aligned}
\eta_k(0,0) &= B_k(0) & \eta_k(1,2) &= B_k(2) + x_k \\
\eta_k(2,0) &= B_k(0) + x_k + y_k & \eta_k(3,2) &= B_k(2) + y_k \\
\eta_k(0,1) &= B_k(1) + x_k + y_k & \eta_k(1,3) &= B_k(3) + y_k \\
\eta_k(2,1) &= B_k(1) & \eta_k(3,3) &= B_k(3) + x_k
\end{aligned} \tag{A-4}$$

The backward state metrics are (k = N+2..2):

$$\begin{aligned}
B_{k-1}(0) &= \mathrm{MAX}^*(\eta_k(0,0), \eta_k(0,1)) \\
B_{k-1}(1) &= \mathrm{MAX}^*(\eta_k(1,2), \eta_k(1,3)) \\
B_{k-1}(2) &= \mathrm{MAX}^*(\eta_k(2,0), \eta_k(2,1)) \\
B_{k-1}(3) &= \mathrm{MAX}^*(\eta_k(3,2), \eta_k(3,3))
\end{aligned} \tag{A-5}$$

subject to the initial conditions:

$$\begin{aligned}
B_{N+2}(0) &= 0 \\
B_{N+2}(1) &= -\infty \\
B_{N+2}(2) &= -\infty \\
B_{N+2}(3) &= -\infty
\end{aligned} \tag{A-6}$$

Define:

$$\begin{aligned}
\lambda_k(0,1) &= A_{k-1}(0) + \eta_k(0,1) & \lambda_k(0,0) &= A_{k-1}(0) + \eta_k(0,0) \\
\lambda_k(2,0) &= A_{k-1}(2) + \eta_k(2,0) & \lambda_k(2,1) &= A_{k-1}(2) + \eta_k(2,1) \\
\lambda_k(1,2) &= A_{k-1}(1) + \eta_k(1,2) & \lambda_k(1,3) &= A_{k-1}(1) + \eta_k(1,3) \\
\lambda_k(3,3) &= A_{k-1}(3) + \eta_k(3,3) & \lambda_k(3,2) &= A_{k-1}(3) + \eta_k(3,2)
\end{aligned} \tag{A-7}$$

The log-likelihood ratio is then:

$$\begin{aligned}
L_k = {} & \mathrm{MAX}^*(\mathrm{MAX}^*(\lambda_k(0,1), \lambda_k(2,0)), \mathrm{MAX}^*(\lambda_k(1,2), \lambda_k(3,3))) \\
& -\mathrm{MAX}^*(\mathrm{MAX}^*(\lambda_k(0,0), \lambda_k(2,1)), \mathrm{MAX}^*(\lambda_k(1,3), \lambda_k(3,2)))
\end{aligned} \tag{A-8}$$

References

[AbendF70] K.Abend and B. Fritchman, "Statistical Detection for Communication Channels with Intersymbol Interference," *Proceedings of the IEEE*, Vol. 58, No. 5, pp. 779-785. May 1970.

[Barbulescu96] S. A. Barbulescu, "Iterative Decoding of Turbo Codes and Other Concatenated Codes, " *Ph.D. thesis*, University of South Australia, 1996.

[Battail87] G. Battail, "Ponderation des symboles decodes par l'algorithme de Viterbi," *Ann. Telecommun.*, vol. 42, pp. 31-38. Jan. 1987.

[BCJR74] L. R. Bahl, J. Cocke, F. Jelinek and J. Raviv, "Optimal Decoding of Linear Codes for Minimizing Symbol Error Rate," *IEEE Transactions on Information Theory*, pp.284-287. March 1974.

[BCPT95] C. Berrou, P. Combelles, P. Penard and B. Talibart, "An IC for Turbo-Codes Encoding and Decoding," *IEEE International Solid State Circuits Conference*, San. Francisco, CA, pp.90-91. Feb. 1995.

[BDMP96] S. Benedetto, D. Divsalar, G. Montorsi and F. Pollara, "Serial Concatenation of Interleaved Codes: Performance Analysis, Design and Iterative Decoding," *TDA Progress Report 42-126*, Jet Propulsion Lab, Pasadena CA, pp. 1-26. Aug. 15, 1996.

[BDMP97] S. Benedetto, D. Divsalar, G. Montorsi and F. Pollara, "Turbo Codes: Principles and Applications," *UCLA Short Course*, UCLA, Los Angeles CA, October 28-31, 1997.

[BenedettoM96] S. Benedetto and G. Montorsi, "Unveiling Turbo-Codes: Some Results on Parallel Concatenated Coding Schemes", *IEEE Transactions on Information Theory*, vol. 42, no. 2, pp. 409--429, Mar. 1996.

[BGT93] C. Berrou, A. Glavieux and P. Thitimajshima, "Near Shannon Limit Error-Correcting Coding and Decoding: Turbo Codes," *Proc. ICC'93*, Geneva, Switzerland, May 23-26, 1993. pp. 1064-1070.

[BlackM97] P. J. Black and T. H.-Y. Meng, "A 1-Gb/s, Four-State, Sliding Block Viterbi Decoder," *IEEE Journal of Solid State Circuits*, Vol. 32, No. 6, pp. 797-805. June 1997.

[ChangH66] R. W. Chang and J. C Hancock, "On Receiver Structures for Channels Having Memory," *IEEE Transactions on Information Theory*, Vol. IT-12, No. 4. pp. 463-468. October 1966.

[ClarkC81] G. C. Clark and J. B. Cain, *"Error-Correction Coding for Digital Communications"*. New York. Plenum Press, 1981.

[Comatlas95] *CAS 5093 Data Sheet*, Rev 4.1, Comatlas SA, May 1995.

[CRC98] Communications Research Centre. http://www.crc.ca/fec/tcintro.htm

[DawidM95], H. Dawid and H. Meyr, "Real-Time Algorithms and VLSI Architectures for Soft Output MAP Convolutional Decoding," *Proceedings of the IEEE International Symposium on Personal, Indoor and Mobile Radio Communications, PIMRC'95*, New York NY, pp. 193-197. 1995.

[DivsalarP95] D. Divsalar and F. Pollara, "Multiple Turbo Codes for Deep-Space Communications," *TDA Progress Report 42-121*, Jet Propulsion Lab, Pasadena CA, pp. 66-77. May. 15, 1995.

[DivsalarP97] D. Divsalar and F. Pollara, "Hybrid Concatenated Codes and Iterative Decoding," *TDA Progress Report 42-130*, Jet Propulsion Lab, Pasadena CA, pp. 1-23. Aug. 15, 1997.

[ErfanianP89]. J. A. Erfanian and S. Pasupathy, "Low-Complexity Parallel-Structure Symbol-by-Symbol Detection for ISI Channels," *IEEE Pacific Rim Conference on Communications, Computers and Signal Processing*, June 1-2 1989. pp. 350-353.

[Farhang-BoroujenyG94] B. Farhang-Boroujeny and S. Gazor, "Generalized Sliding FFT and its Application to Implementation of Block LMS Adaptive Filters," *IEEE Transactions on Signal Processing*, Vol. 42, No. 3. pp. 532-538. March 1994.

[FBLH98] M. P. C. Fossorier, F. Burkert, S. Lin and J. Hagenauer, "On the Equivalence Between SOVA and Max-Log-MAP Decodings," *IEEE Communications Letters*, Vol. 2. No. 5, pp. 137-139, May 1998.

[Ferguson72] M. J. Ferguson, "Optimal Reception for Binary Partial Response Channels," Bell Systems Technical Journal, Vol. 51, No. 2, pp. 493-505, Feb. 1972.

[FettweisM91] G. Fettweis, H. Meyr, "High-Speed Parallel Viterbi Decoding: Algorithm and VLSI-Architecture," *IEEE Communications Magazine*, Vol. 29, No. 5. pp 46-55. May 1991.

[FeyginG91] G. Feygin, P. G. Gulak, "Survivor Sequence Memory Management in Viterbi Decoders," *University of Toronto Computer Systems Research Institute Technical Report CSRI-252*. January 1991.

[FKST95] G. Fettweis, R. Karabed, P. H. Siegel and H. K. Thapar, "Reduced-Complexity Viterbi Detector Architectures for Partial Response Signalling," *Proceedings Globecom'95*. pp. 559-563. 1995.

[Forney66] G. D. Forney Jr., "*Concatenated Codes*," Cambridge, Ma: MIT Press, 1966.

[Forney73] G. D. Forney Jr., "The Viterbi Algorithm," *Proceedings of the IEEE*, vol. 61, pp. 268-278, Mar. 1973.

[Galloway95] D. Galloway, "The Transmogrifier C Hardware Description Language and Compiler for FPGAs," *IEEE Symposium on FPGAs for Custom Computing Machines*, FCCM '95, April 1995.

[Gaudet98] V. Gaudet, Private discussions, July 1998.

[GrossG98] W. J. Gross and P. G. Gulak, "Simplified MAP Algorithm Suitable for Implementation of Turbo Decoders," *Electronics Letters*, Vol. 34, No. 16. pp.1577-1578. 6'th August 1998.

[HagenauerH89] J. Hagenauer and P. Hoeher, "A Viterbi Algorithm with Soft-Decision Outputs and its Applications," *Proceedings IEEE Globecom Conference*, Dallas TX. pp. 1680-1686. Nov. 1989.

[Haykin88] Simon Haykin, "*Digital Communications*," New York. John Wiley & Sons. 1988.

[HOCS98] S. Halter, M. Obert, P. M. Chau and P. H. Siegel, "Reconfigurable Signal Processor for Channel Coding & Decoding in Low SNR Wireless Communications," *IEEE Workshop on Signal Processing Systems*, pp. 260-274, 1998.

[Hoeher97] P. Hoeher, "New Iterative ("Turbo") Decoding Algorithms," I*nternational Symposium on Turbo Codes*, Brest, France. pp. 63-70. 1997.

[HOP96] J. Hagenauer, E. Offer and L. Papke, "Iterative Decoding of Binary Block and Convolutional Codes," *IEEE Transactions on Information Theory*, Vol. 42, No. 2. pp. 429-445. March 1996.

[HPG98] M. S. C. Ho, S. S. Pietrobon and T. Giles, "Interleavers for Punctured Turbo Codes," *IEEE Asia-Pacific Conference on Communications and Singapore International Conference on Communications Systems*, Singapore, Nov. 1998.

[HYS98] S. Hong, J. Yi and W. E. Stark, "VLSI Design and Implementation of Low-Complexity Adaptive Turbo-Code Encoder and Decoder for Wireless Mobile Communication Applications," *IEEE Workshop on Signal Processing Systems*, pp. 233-242, 1998.

[JPL1997] Jet Propulsion Laboratory. http://www331.jpl.nasa.gov/public/TurboDecoder.html

[KingsburyR71] N.G. Kingsbury and P.J.W. Rayner, "Digital Filtering Using Logarithmic Arithmetic," *Electronics Letters*, Vol. 7, No. 2. pp. 56-58, Jan. 1971.

[Kobayashi71] H. Kobayashi, "Application of Probabalistic Decoding to Digital Magnetic Recording Systems," *IBM Journal of Research and Development*, vol. 15, pp. 64-74. Jan. 1971.

[Koora98] K. Koora. Dresden University of Technology. http://entnw2.et.tu-dresden.de/cgiwrap/tc.sh

[LeeM94] Edward A. Lee and David G. Messerschmitt, "*Digital Communication, Second Edition*, " Boston. Kluwer Academic Publishers, 1994.

[LGVRC98] D. Lewis, D. Galloway, M. van Ierssel, J. Rose and P. Chow, "The Transmogrifier-2: A 1 Million Gate Rapid Prototyping System," *IEEE Transactions on VLSI*, Vol. 6, No. 2, pp 188-198. June 1998.

[LVS95] Y. Li, B. Vucetic, and Y. Sato, "Optimum Soft-Output Detection for Channels with Intersymbol Interference," *IEEE Transactions on Information Theory*, Vol. 41, No. 3 pp.704-713. May 1995.

[LYHH93] J. Lodge, R. Young, P. Hoeher and J. Hagenauer, "Separable MAP "Filters" for the Decoding of Product and Concatenated Codes," *Proceedings ICC'93*, Geneva, Switzerland, May 23-26, 1993. pp. 1740-1745.

[Montorsi98] G. Montorsi. Politecnico Di Torino. http://www.tlc.polito.it/~montorsi/simulation.html

[Omura69] J. K. Omura. "On the Viterbi Decoding Algorithm," *IEEE Transactions on Information Theory*, Vol.IT-15, No.1. pp. 177-179. Jan. 1969.

[Pietrobon98] S. S. Pietrobon, "Implementation and performance of a turbo/MAP decoder," *International Journal of Satellite Communications*, vol.16, pp. 23-46, Jan.-Feb. 1998.

[RabinerG75] L. R. Rabiner and B. Gold, "*Theory and Application of Digital Signal Processing*," Englewood Cliffs, NJ, Prentice-Hall, p. 606. 1975.

[Rader81] C. M. Rader, "Memory Management in a Viterbi Algorithm," *IEEE Transactions on Communications*, 29:1399-1401, Sept. 1981.

[ReedA97] M. C. Reed and J. Asenstorfer, "A Novel Variance Estimator for Turbo-Code Decoding," *Proceedings International Conference on Telecommunications*, Melbourne, Australia, pp. 173-178, April 1997.

[RHV97] P. Robertson, P. Hoeher and E. Villebrun, "Optimal and Sub-Optimal Maximum a Posteriori Algorithms Suitable for Turbo Decoding," *European Transactions on Telecommunications*, Vol. 8, No. 2, pp. 119-125. March/April 1997.

[SJM98] M. H. Shakiba, D. A. Johns and K. W. Martin, "An Integrated 200-MHz 3.3-V BiCMOS Class-IV Partial-Response Analog Viterbi Decoder," *IEEE Journal of Solid State Circuits*, Vol. 33, No. 1. pp. 61-65. Jan. 1998.

[SummersW98] T. A. Summers and S. G. Wilson, "SNR Mismatch and Online Estimation in Turbo Decoding," *IEEE Transactions on Communications*, Vol. 46, No. 4. pp. 421-423. April 1998.

[TakeshitaC98] O. Y. Takeshita and D. J. Costello Jr., "New Classes of Algebraic Interleavers for Turbo-Codes," *International Symposium on Information Theory*, ISIT 1998, Cambridge MA, Aug 16-21, 1998.

[TzouD81] K. Tzou and J. G. Dunham, "Sliding Block Decoding of Convolutional Codes," *IEEE Transactions on Communications*, Vol. COM-29, No. 9. pp. 1401-1403. Sept. 1981.

[Viterbi67] A. J. Viterbi, "Error Bounds for Convolutional Codes and an Asymptotically Optimum Decoding Algorithm," *IEEE Transactions on Information Theory*, Vol. 13, No. 2, pp. 260-269, Apr. 1967.

[Viterbi98] A. J. Viterbi, "An Intuitive Justification and a Simplified Implementation of the MAP Decoder for Convolutional Codes," *IEEE Journal on Selected Areas in Communications*, Vol. 16, No. 2 pp. 260-264. Feb. 1998.