VLSI IMPLEMENTATION OF A TURBO ENCODER / DECODER

SAINATH PADINJARE

Your file Votre référence
ISBN: 0-612-93047-5
Our file Notre référence
ISBN: 0-612-93047-5

VLSI IMPLEMENTATION OF A TURBO ENCODER/DECODER

BY

© SAINATH PADINJARE, B.Eng

A Thesis submitted to the

School of Graduate Studies

in partial fulfillment of the

requirements for the degree of

Master of Engineering

FACULTY OF ENGINEERING AND APPLIED SCIENCE

MEMORIAL UNIVERSITY OF NEWFOUNDLAND

2003

MASTER OF ENGINEERING THESIS

OF

SAINATH PADINJARE, B.Eng

APPROVED:

Thesis Committee

Major Professors _____

_____

_____

DEAN OF THE SCHOOL OF GRADUATE STUDIES

MEMORIAL UNIVERSITY OF NEWFOUNDLAND

2003

Abstract

This thesis describes a hardware implementation of a Turbo encoder/decoder.

Turbo codes, introduced in 1993, enable reliable communications over power-constrained communications channels close to the Shannon limit. Since turbo codecs are employed in battery-powered devices such as cellular phones and laptop computers, power dissipation, along with speed and area, are major concerns in *very large scale integrated circuit* (VLSI) design.

There is thus the need for low power, modular, and parallel *application-specific integrated circuits* (ASICs) for turbo code encoders/decoders using VLSI techniques.

Possible algorithms for turbo decoding are the soft-output Viterbi algorithm (SOVA) and the Bahl, Cocke, Jelink, and Raviv (BCJR) Algorithm. In this research, the hardware implementation of a low-power turbo encoder and SOVA based decoder for wireless communications applications is investigated.

SOVA, which is a modification of the Viterbi algorithm(VA) has lower computational and implementation complexity. The turbo decoding process is sequential in

nature, so very high speed implementations are difficult to obtain.

By means of software simulation, the basic turbo encoding/decoding performance is first studied, then the effect of fixed-point arithmetic on the performance of the decoder is analyzed.

The power dissipation depends mainly on the switching of signal values in the path management unit of the decoder and on the number of iterations of decoding. Certain known power reduction techniques are implemented.

The performance of the implemented decoder is analyzed and finally conclusions and recommendations for future work are presented. The implementation has been carried out using custom ASIC 0.18 μm CMOS technology.

Acknowledgments

I am deeply indebted to my supervisors Dr. R. Venkatesan and Dr. P. Gillard for their guidance, advice, useful discussions, criticisms, and help in preparing this manuscript. I sincerely thank Dr. R. Venkatesan, Dr. P. Gillard, the Faculty of Engineering and Applied Science, the Department of Computer Science and the School of Graduate Studies for the financial support provided to me during my M.Eng. program.

I would like to thank Dr. C. Jablonski, Dean of the School of Graduate Studies, Dr. M. Haddara, Interim Dean of the Faculty of Engineering and Applied Science, Dr. R. Gosine, Interim Associate Dean for Graduate Studies and faculty members of the Engineering and Applied Science for their support during my study in Canada.

I sincerely acknowledge Mr. Reza Shahidi for his help and useful discussions. I would like to thank Mr. Nolan White of the Computer Science Department for his help with the VLSI CAD tools. I thank all my fellow graduate students for their friendship and constant encouragements. Finally, I would like to express my profound gratitude to my family especially my wife Deepa for their support and understanding during the course of my study.

Contents

Abstract ii

Acknowledgments iv

Table of Contents v

List of Figures x

List of Tables xv

List of Abbreviations xvi

1 Introduction 1

1.1 History of Coding 2

1.2 Code Types 4

1.2.1 Block Codes 4

1.2.2 Convolutional Codes 5

1.2.3 Concatenated Codes 6

1.3 Turbo Codes 8

1.4 Turbo Encoder 10
1.5 Turbo Decoder Block 11
1.5.1 Principle of Operation 11
1.6 Viterbi Algorithm 13
1.6.1 An Example of VA Decoding 17
1.6.2 Hard and Soft Decision Decoding 18
1.7 Soft Output Viterbi Algorithm 18
1.8 Objective of Research 22
1.9 Organization of Thesis 22

2 Review of Related Work **24**

2.1 Mathematics of the SOVA 25
2.2 Trellis Termination 33
2.3 Sliding Window Implementation 35
2.4 Effect of Interleaver on Decoder Performance 37
2.4.1 Interleaver Types 38
2.5 Effect of Puncturing 40
2.6 Stopping Criteria for Iterative Decoding 42
2.7 Review of Improvements in the SOVA 43
2.8 Design for Low Power 45
2.9 Hardware Implementations 48

2.10 Summary 50

3 Software Simulation of a Turbo Codec **52**

3.1 Introduction 52

3.2 Model of Digital Communication System 53

3.3 Flow Chart of a Turbo Decoder 56

3.3.1 Add Compare Select 57

3.3.2 Path Management and Storage Unit 58

3.3.3 Soft Values Generation 58

3.3.4 Interleaving and Deinterleaving 59

3.3.5 Iterative Decoding 59

3.4 Modelling the AWGN Channel 60

3.5 Turbo Encoding/Decoding Example 61

3.6 Software Implementation 70

3.6.1 Window Decoding 70

3.6.2 Path Metric Normalization 71

3.6.3 Fixed Point Implementation 71

3.7 Simulation Results 73

3.7.1 Influence of Iteration Number 73

3.7.2 Influence of Windowed Implementation 75

3.7.3 Influence of Frame Size 76

3.7.4 Influence of Interleaver Type 77
3.7.5 Influence of Code Rate 79
3.7.6 Influence of Fixed Point Implementation 79
3.7.7 Summary 81

4 Hardware Design and Implementation **82**

4.1 Motivation 82
4.2 Design Flow 83
4.3 Hardware Implementation of the Encoder 84
4.4 Hardware Implementation of the Decoder 86
4.5 The Input Buffer 88
4.6 The SOVA unit 88
4.6.1 The Viterbi Decoder(VA block) 89
4.6.2 Path Metric Difference Generation 97
4.6.3 The SOVA Multiple Traceback Unit 100
4.6.4 Soft Value Update Unit 100
4.6.5 The Complete Picture 101
4.6.6 Interleaver and Deinterleaver Unit 103
4.7 Decoder Control Unit 103
4.7.1 Top Level Controller 103
4.7.2 Input Buffer Controller 105

4.7.3 SOVA Unit Controller 106
4.8 Low Power Modifications 107
4.8.1 Gating the Clock 107
4.8.2 Disabling Unused Registers 108
4.8.3 Implementation of a Stopping Criterion 109
4.9 Testing of the Design 110
4.10 Performance and Synthesis Summary 111
4.11 Conclusions 115

5 Conclusions and Future Work **117**
5.1 Conclusions 117
5.2 Future Work 119
5.2.1 Multiple Bit Release for SOVA 119
5.2.2 Improvements in the Present Implementation 120
5.2.3 Implementation of the Improved SOVA 121
5.2.4 Power Consumption Analysis 121
5.2.5 Simulation Under the Influence of Fading 122

List of References **123**

Appendix A **128**

A Drawings and Simulation Waveforms **129**

List of Figures

1.1 Rate 1/2 convolutional encoder 6

1.2 Concatenated code system 7

1.3 Block diagram of SCCC encoder 7

1.4 Trellis-based decoding algorithms 9

1.5 Turbo encoder 11

1.6 Turbo decoder block 12

1.7 State diagram of a rate 1 convolutional encoder 14

1.8 Trellis diagram for a rate 1/2 convolutional encoder 14

1.9 Viterbi decoding example 17

2.1 SOVA block 27

2.2 Trellis section 28

2.3 Simplified section of the trellis for a 4 state RSC code 31

2.4 Trellis termination 33

2.5 Sliding window decoding 36

2.6 Row-Column interleaver 39

2.7 Pseudo-random interleaver 40

2.8 Punctured turbo code 41

3.1 Model of digital communication system 54

3.2 Flow chart of SOVA based turbo decoding 56

3.3 Encoding operation 62

3.4 Transmitted (a) and received sequence(b) 63

3.5 Weighted received sequence 63

3.6 SOVA1 decoder ML path 64

3.7 Survivor and competing partial path metrics for the trellis 64

3.8 Competing paths for reliability updates 65

3.9 SOVA1's updated reliability values 65

3.10 Input sequence to SOVA2 decoder 67

3.11 SOVA2 decoder ML and competing paths 67

3.12 Survivor and competing partial path metrics for the trellis of Decoder2 67

3.13 Competing paths for reliability updates for Decoder2 68

3.14 SOVA2's updated reliability values 68

3.15 Fixed point representation of SOVA decoder parameters 72

3.16 BER vs Signal to Noise ratio for different iterations (1024 bit frame) . 74

3.17 Windowed implementation vs Continuous decoding performance . . . 76

3.18 BER vs Signal to Noise ratio for different frame sizes 77

3.19 BER vs Signal to Noise ratio for random and block interleavers . . . 78

3.20 Influence of code rate on turbo code performance 79

3.21 Floating point vs Fixed point implementation performance 80

4.1 ASIC design flow 83

4.2 Encoder architecture 85

4.3 State diagram for the 2 bit encoder1 85

4.4 Decoder architecture 87

4.5 Viterbi algorithm architecture 89

4.6 Butterfly structure of trellis 90

4.7 ACS unit structure showing the branch metrics (bm) and the path metric (pm) 91

4.8 ACS unit symbolic view 92

4.9 Sliding window architecture 93

4.10 Register exchange method 95

4.11 Traceback scheme 96

4.12 Traceback unit symbolic view 97

4.13 Generation of path metric differences for ML states 98

4.14 Symbolic view of the interface unit 99

4.15 Soft value update 101

4.16 SOVA unit data path 102

4.17 Top level controller for turbo decoder 104

4.18 Input buffer controller for turbo decoder 105

4.19 Controller for SOVA unit 106

4.20 Clock gating for low power 107

4.21 Disabling unused registers 108

4.22 Early termination of iterations 110

4.23 SOVA unit functionality 112

A.1 Module structure 129

A.2 Timing diagram for 2 iterations 131

A.3 Timing diagram for 7 iterations 132

A.4 SDR stopping scheme 133

List of Tables

3.1 BER vs Signal to Noise ratio for different iterations (1024 bit frame) 74

3.2 BER vs Signal to Noise ratio for windowed vs continuous decoding . 75

3.3 BER vs Signal to Noise ratio for different frame sizes 76

3.4 BER vs Signal to Noise ratio for different interleaver types 78

3.5 BER vs Signal to Noise ratio for floating point & fixed point implementation 80

4.1 Summary of area, timing and power report for the components of the turbo decoder 113

4.2 Gate count for the SOVA block 115

4.3 Gate count estimate for the turbo decoder block 115

List of Abbreviations

3GPP	:	Third Generation Partnership Project(Wireless)
3G	:	Third Generation(Wireless)
ACS	:	Add Compare Select
ASIC	:	Application-Specific Integrated Circuit
AWGN	:	Additive White Gaussian Noise
BCJR	:	Bahl, Cocke, Jelinek, and Raviv (algorithm)
BER	:	Bit Error Rate
BPSK	:	Binary Phase Shift Keying
BCH	:	Bose-Chaudhuri- Hocquenghem
CAD	:	Computer Aided Design
CE	:	Cross Entropy
CRC	:	Cyclic Redundancy Check
CMC	:	Canadian Microelectronics Corporation
DSP	:	Digital Signal Processing
FPGA	:	Field Programmable Gate Array

GF	:	Galois Field
ML	:	Maximum Likelihood
RS	:	Reed-Solomon
RSC	:	Recursive Systematic Convolutional
SCCC	:	Serial Concatenated Convolutional Code
SCR	:	Sign Change Ratio
SDR	:	Sign Difference Ratio
SNR	:	Signal to Noise Ratio
SMU	:	Survivor Management Unit
SST	:	Scarce State Transition
VA	:	Viterbi Algorithm codes
LDPC	:	Low Density Parity Check Codes
PCCC	:	Parallel concatenated convolutional codes
XOR	:	Exclusive OR
SOVA	:	Soft Output Viterbi Algorithm
MAP	:	Maximum A Posteriori (Algorithm)
LLR	:	Log-Likelihood Ratio
VHDL	:	Very-high-speed Hardware Description Language
VLSI	:	Very Large Scale Integrated circuits

Chapter 1

Introduction

The last decade has seen a revolution in the telecommunications industry with major innovations and commercial products in wireless communication. With the advent of the third generation (3G) telecommunication standard, the convergence of voice and multimedia communication has become a reality. Computing power has increased with improved very large scale integrated (VLSI) techniques, and improved microelectronics fabrication has reduced the cost, size and power requirements of communication integrated circuits (IC's). Developments in digital signal processing (DSP), field programmable gate arrays (FPGA), spread spectrum systems, smart antennas and error correcting codes have all aided the wireless revolution. This thesis describes work in the area of error correction coding, specifically, about a recent class of error correcting codes called ***turbo codes***. They have been proposed as the channel coding scheme for the third generation (3G) cellular communication standard

due to their superior error correction performance. These codes have been found to have near Shannon limit error correcting capacity. Although error control coding is mainly used in digital communication systems like satellite communication, digital cellular telephony, they also have applications in digital television and high-density digital storage [1]. Error control coding is an area of science lying in the intersection of Electrical Engineering, Computer Science and Mathematics. The area emerged in the 1940s from the study of methods to transmit information reliably over noisy channels of communication. One of the central ideas arising from this study was the notion of *error-correcting encoding* which meant a clever way to add redundancy to data that enables recovery of information even after it has been corrupted. This concept also led to the name of the field, **coding theory**.

This introductory chapter will describe briefly the history of coding, explain the different types of codes, and describe the characteristics and performance of turbo codes. The decoding algorithm for convolutional and turbo codes are explained in detail. The objectives of this research are highlighted towards the end of the chapter.

1.1 History of Coding

Coding theory draws extensively from topics in algebra and probability, and has applications in theoretical computer science and cryptography as well as its original motivations, namely, storage and communication of information. Coding theory was

born with the work of Hamming who created a family of error correcting codes each able to correct one error in a block length of N bits [2]. This along with Shannon's celebrated 1948 paper "A Mathematical Theory of Communication" [3] laid the foundation for error control coding. The main result of Shannon's paper was the Noisy Channel Coding Theorem [4] which states that every communication channel is characterized by the *channel capacity* C, such that R randomly chosen bits per second can be transmitted with arbitrarily low bit error rate if and only if $R < C$ (R is called the data transmission rate). Shannon showed that the specific value of signal-to-noise ratio is not important so far as it is large enough that $R < C$ holds. The theorem guaranteed the existence of codes that could achieve a small probability of error if the data rate is smaller than the channel capacity.

Golay in 1949 was the first to publish a paper in coding theory [5]. In this paper he gave a generalization of Hamming codes to non binary fields. Since then most of the codes developed were generalization of the Hamming codes, starting with Bose-Chaudhuri-Hocquenghem (BCH) codes. Reed-Solomon (RS) codes are a generalization of BCH codes and Goppa codes are a generalization of RS codes.

Convolutional codes were invented by Elias in 1955 [2]. Sequential decoding for convolutional codes was first suggested by Wozencraft in 1957 [2]. In 1967 Viterbi presented his algorithm as a "new probabilistic non sequential algorithm" [6]. This later became the most popular decoding algorithm for convolutional codes. Forney,

in 1967, drew the first trellis, which simplified the understanding of the Viterbi algorithm (VA) and he coined the term "trellis". Trellis coded modulation proposed in 1982 by Ungerboeck, provided higher coding gains than before without sacrificing data rate and bandwidth [7]. In 1993 at the International Communications Conference in Geneva, turbo codes were presented by C. Berrou, A. Glavieux and P. Thitimajshima. They claimed that a combination of parallel concatenation and iterative decoding can provide reliable communication that is close to the Shannon limit. This was a revolutionary result and meant a big leap for telecommunications.

More recently, low density parity check (LDPC) codes, discovered by Gallagher in 1963, have attracted attention as achieving comparable performance to turbo codes but with reduced decoding complexity.

1.2 Code Types

1.2.1 Block Codes

Error-correcting codes have been classified basically into *block codes* or *convolutional codes*, depending on whether there is a memory in the encoder or not. In block coding, an encoder generates an n-bit code word with a k-bit message block, so code words are produced on a block-by-block basis. However if the need is to process the incoming bits serially rather than in large blocks, then convolutional coding is used.

For an (n, k) block code, to transmit k bits of data we convert it to an n-bit

string according to a specific rule, and send out the n bits. Thus there is some redundancy added. The addition of a parity-check bit to a character or a sequence of data bits is an example of a block coding technique. Consider now the entire sequence of information bits to be divided into blocks of k bits each. These blocks are called messages and denoted $\mathbf{u} = u_0u_1u_2\cdots u_{k-1}$. A binary (n,k) block code is a set of $M = 2^k$ binary n–tuples (or row vectors of length n) $\mathbf{v} = v_0v_1\cdots v_{n-1}$ called codewords. n is called the block length and the quantity $R = \log M/n = k/n$ is called the code rate. RS codes are the most popular block codes used by the communication industry. The decoding of RS codes is done by the Berlekamp-Massey algebraic method.

1.2.2 Convolutional Codes

Convolutional codes approach error control in a different manner from that of block codes. A convolutional encoder converts an entire data stream regardless of its length into a single code word. On the other hand block encoders break the data stream into fixed length k and map them into codewords of fixed length n.

Figure 1.1 shows the schematic of a typical rate 1/2 encoder. In the figure, binary data stream $\mathbf{u} = u_0u_1u_2\cdots$ is fed into a shift register consisiting of a series of flipflops. With each successive input to the shift register, the values of the memory elements are tapped off and added according to a fixed pattern, creating a pair of

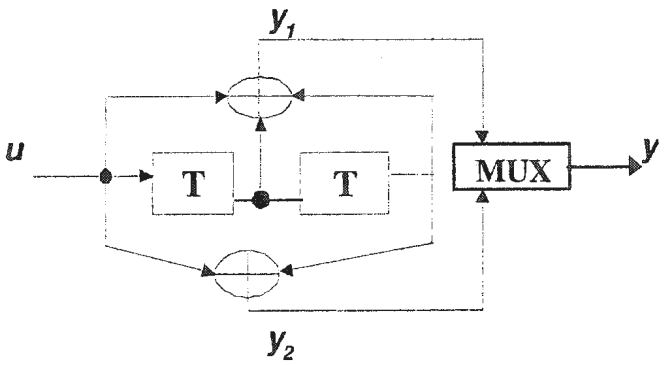


Figure 1.1: Rate 1/2 convolutional encoder

output coded streams. Decoding of convolutional codes is done bit-by-bit or symbol-by-symbol and uses a trellis. The most famous decoding algorithm is the Viterbi algorithm, which is a maximum likelihood sequence estimation.

1.2.3 Concatenated Codes

Satellite systems have been using convolutional encoding with VA decoding for several years. One of the drawbacks of VA decoding process is that uncorrectable errors tend to collect together and look like burst errors. RS codes have good burst error correction characteristic however these codes when used alone have poor bit error rates compared to convolutional codes. Concatenating RS codes with convolutional codes solved this problem. Figure 1.2 shows the structure of a concatenated coding system.

Figure 1.2: Concatenated code system

Concatenated codes can be classified into two broad categories: parallel concatenated convolutional codes (PCCC) and serial concatenated convolutional codes (SCCC). PCCC perform exceptionally well at low signal-to-noise ratios (SNRs) but develop rather large error floors at high SNRs [8].

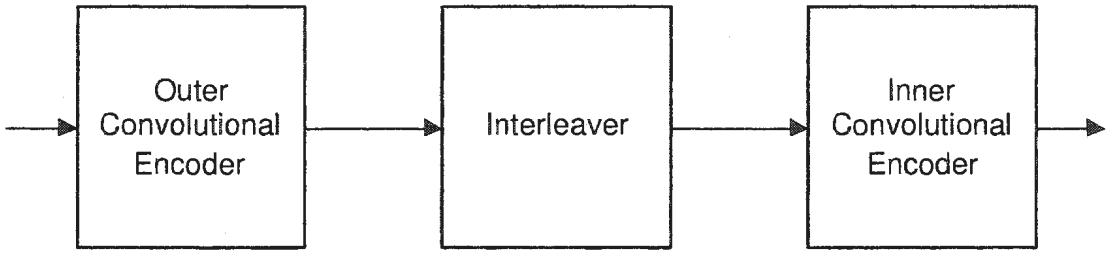


Figure 1.3: Block diagram of SCCC encoder

On the other hand, SCCC can achieve extremely low bit error rates at high SNRs, although this comes at the cost of inferior performance (relative to PCCC) at very low SNRs [9]. Figure 1.3 shows the block diagram of a serial concatenated convolutional encoder.

1.3 Turbo Codes

The performance of convolutional codes can be increased by increasing the memory size (the number of flipflops in the linear-shift register) but this leads to exponentially increasing decoding complexity. Since large block length codes were found to increase performance, concatenated codes were the subject of investigation as early as 1980. In that year Forney [10] used concatenation of an inner convolutional code and an outer RS code serially to obtain a code whose performance was better than convolutional codes alone. In this case it was found that the decoding complexity increased algebraically.

Turbo codes or PCCCs introduced by Berrou et al. [8] represented a way of concatenating two convolutional codes separated by an interleaver to produce a powerful code.

The turbo code consists of outputs of each recursive systematic convolutional (RSC) encoder, as well as the original input sequence. Recursive refers to the feed back of the register outputs and the XOR addition with the input stream as shown in Figure 1.5. Since turbo codes are built on convolutional codes, turbo decoding algorithms are modifications of the ones used for decoding of convolutional codes.

The VA is unsuitable for turbo decoding because it is a soft-input (explained later) hard-output algorithm. Turbo decoding is done by modifying the VA to the soft-output Viterbi algorithm (SOVA) introduced by Hagenauer and Hoeher in [11].

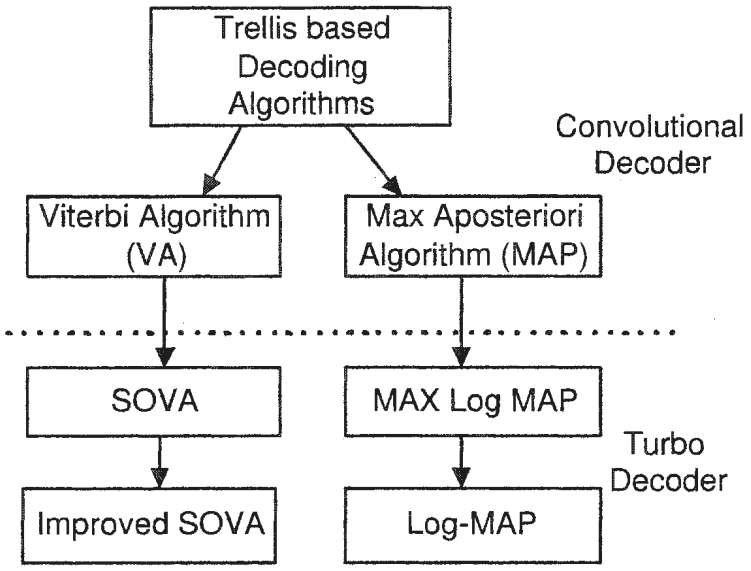


Figure 1.4: Trellis-based decoding algorithms

The SOVA is a soft-input, soft-output algorithm that retains information about competing paths and determines reliability of bits which differ from the surviving path. The SOVA is about twice as complex as the VA but the coding gains obtained compensates for the extra complexity. The other popular decoding algorithms for turbo codes are the maximum *a posteriori* (MAP) algorithm and its modifications namely Max-Log-MAP and Log-MAP. Other variations of these algorithms exist as shown above in Figure 1.4. The MAP algorithm, also known as BCJR algorithm, is a forward-backward recursion algorithm that minimizes the probability of bit error. One main difference from the VA is that the path MAP algorithm traces in the trellis need not be connected. The Log-MAP and Max-Log-MAP have slight performance improvement over SOVA but their complexities are higher [12]. The MAP algorithm

has the best performance of the lot and the highest complexity. The Battail's version of improved SOVA [13] is the most attractive as it has lower complexity and performance close to the Max-Log-MAP. The basic SOVA is implemented in this thesis with a long term goal of implementing the improved SOVA.

1.4 Turbo Encoder

A turbo code is formed by two constituent convolutional encoders in parallel, separated by an interleaver. A turbo encoder is specified as (k, n, N) where k/n is the code rate, and N is the block size. For each k information bits, n encoded bits are output.

Figure 1.5 shows a rate 1/3 turbo code encoder operating as a linear block encoder of length N. The connections between the shift register elements and the modulo 2 adders are described by generator sequences $g^{(1)} = (101)$ and $g^{(2)} = (111)$ and the overall generator polynomial $g(7, 5)$. It first divides the input bit stream u into blocks of length N bits, which go to each of the two encoders as shown. The upper encoder(henceforth called Encoder1) operates on the block as it is and generates N parity bits x_2 that it outputs along with the N original information bits shown above as x_1. The lower encoder(henceforth called Encoder2) operates on the same data block but with the bits permuted (shuffled) in a fixed pattern as determined by the interleaver P to generate the second set of N parity bits x_3. The system then

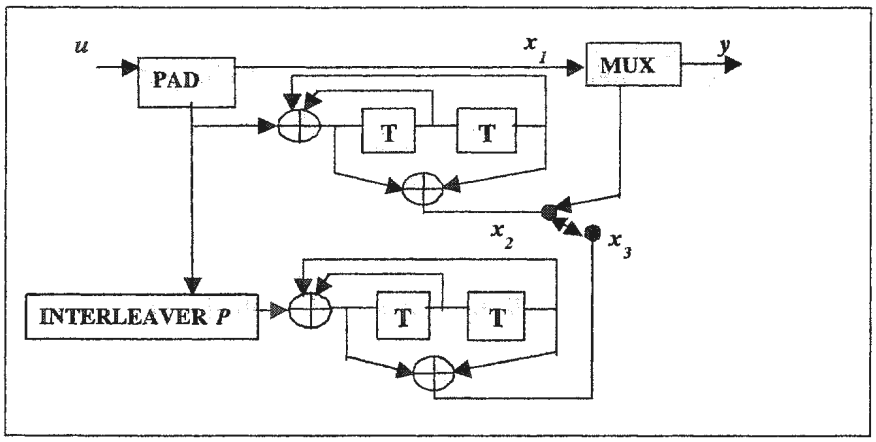


Figure 1.5: Turbo encoder

transmits y, the multiplexed result of x_1, x_2 and x_3. These $3N$ bits form the output codeword the system produced from the original N bits of input stream u. Example: input u = [10101], x_2 = [11011], x_3 = [01100], y = [110011101010110], interleaver matrix P = [25413].

1.5 Turbo Decoder Block

1.5.1 Principle of Operation

Figure 1.6 shows the block diagram of an iterative turbo decoder. The SOVA decoder block shown in the block diagram essentially consist of two blocks: the first block is the Viterbi block where the algorithm traces back over one path called the maximum likelihood (ML) path. The second block traces backwards over the ML path and its

next competitor (if the ML approaches a state with a '1' input, the competitor traces back the '0' path).

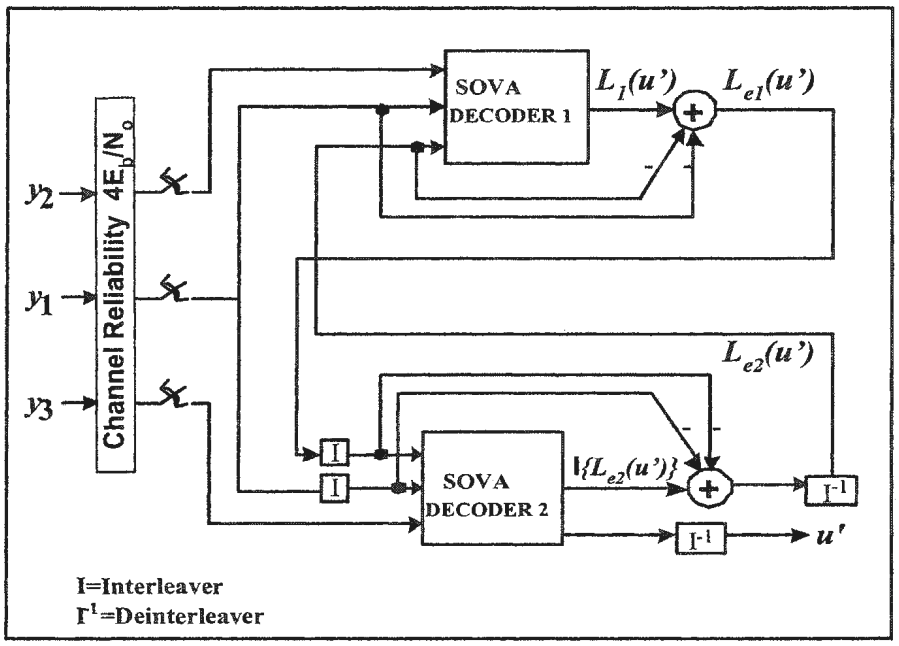


Figure 1.6: Turbo decoder block

SOVA Decoder1 computes a soft-output from the systematic data y_1, code information of Encoder1 y_2 and *a priori* information (Le_2). From this soft-output $L_1(u')$, the systematic data y_1 and *a priori* information (Le_2) are subtracted. The result is thus uncorrelated with y_1 and is denoted as $Le_1(u')$, for extrinsic data from Decoder1. Decoder2 takes the interleaved version of Le_1, the code information of the second decoder y_3 and the interleaved version of systematic data y_1 which is $\alpha(y_1)$ as inputs and generates a soft output Le_2, following the same process as described

above. α is the interleaving pattern and same as the interleaver used in the encoder. The output is then fed back to Decoder1. The sign of the soft values (Le_2) shown as u' gives the estimate of the decoded information bits. The iterative process continues until convergence is achieved and then the decision unit outputs the decoded bits u'. After a few iterations it has been found that there is not much change in the decoded bits and leads to the conclusion of convergence.

1.6 Viterbi Algorithm

For a convolutional code, the input sequence $\boldsymbol{u}$ is convoluted to the encoded sequence $\boldsymbol{c}$. Sequence $\boldsymbol{c}$ is transmitted across a noisy channel and the received sequence $\boldsymbol{r}$ is obtained. The VA computes a maximum likelihood estimate on the estimated code sequence $\boldsymbol{y}$ from the received sequence $\boldsymbol{r}$ such that it maximizes the probability $\boldsymbol{p(r \mid y)}$ that sequence $\boldsymbol{r}$ is received conditioned on the estimated code sequence $\boldsymbol{y}$. Sequence y must be one of the allowable code sequences and cannot be any arbitrary sequence. A convolutional encoder can be represented as a state diagram or a trellis diagram.

Figure 1.7 shows the state diagram for the convolutional encoder of Figure 1.1. This encoder has 2^2 states corresponding to the two memory elements (D flipflops or T flipflops). The state diagram indicates the outputs corresponding to various state transitions for input of 1 or 0. The trellis diagram is another representation of

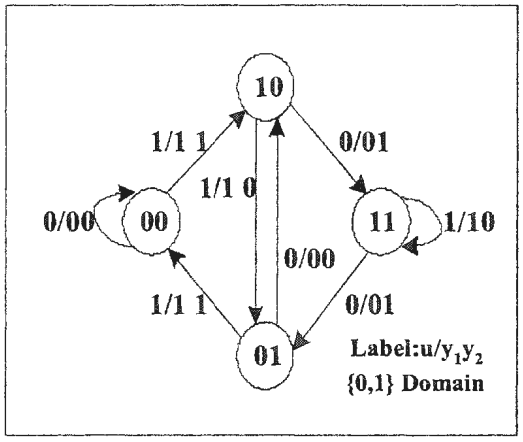


Figure 1.7: State diagram of a rate 1 convolutional encoder

the state diagram and is helpful for the decoding of convolutional and turbo codes. Figure 1.8 shows the trellis diagram for the single encoder, with 4 states- $00, 01, 10, 11$ shown on the vertical axis. The horizontal axis shows the different states for four input bit intervals. By adding padding bits the encoder can be made to start and end in the same state.

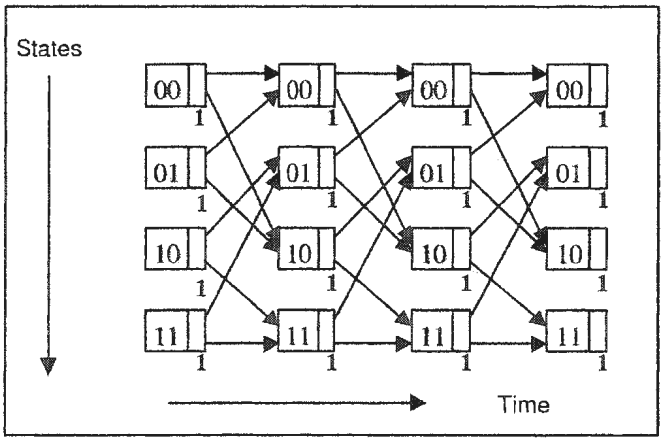


Figure 1.8: Trellis diagram for a rate 1/2 convolutional encoder

The VA accepts hard or soft values of the received signal to compute the branch

metric for every state of each stage in the trellis. The path metrics for the first stage are initialized to zero, and are added to the branch metrics values to form new path metric values. This is done recursively until the last stage of the trellis is reached. The path with the largest path metric value is selected as the survivor path and a traceback is done starting from the state with the highest path metric value in the last stage. If the encoder is terminated (say, starts and ends in the zero state) then the traceback starts from state zero.

The VA in mathematical form [4] is as follows: Assume that the convolutional encoder is in the zero state initially and ends in the same state at time $t = L+m-1$. L is the block size of the input stream and m represents the number of memory elements of the encoder. $\boldsymbol{y}$ is the estimated sequence from the received sequence $\boldsymbol{r}$. Let $S_{k,t}$ be the state in the trellis diagram that corresponds to state S_k at time t. Every state in the trellis is assigned a value denoted by $V(S_{k,t})$.

1. Let $V(S_{k,t})$ be the path metric value of the state.

2. Initialize time $t = 0$, $V(S_{0,0}) = 0$.

3. LOOP: Set time $t = t + 1$.

4. Compute Partial Path metrics for all paths going to state S_k at time t. To do this, first, find the t^{th} branch metric

$$M(\mathrm{r}_t|\mathrm{y}_t) = \sum_{j=1}^{n} M(r_t{}^{(j)}|y_t{}^{(j)}). \tag{1.1}$$

from the Hamming distance given by

$$\sum_{j=1}^{n} |(r_t{}^{(j)} - y_t{}^{(j)})|. \tag{1.2}$$

Second, compute the t^{th} partial path metric by the equation

$$M^t(\mathbf{r}|\mathbf{y}) = \sum_{i=0}^{t} M(\mathbf{r}_i|\mathbf{y}_i). \tag{1.3}$$

This is calculated from $V(S_{k,t-1}) + M(r_i|y_i)$.

5. Set $V(S_{k,t})$ to be the partial path metric, with the highest value. If there is a tie, choose any one. Store the best partial path metric (the one with the highest value or the lowest value depending on the branch metric used), associated survivor bits and state paths.

6. If $t < L + m - 1$, Return to LOOP.

7. If $t = L + m - 1$, start at state zero and follow the surviving state paths backwards through the trellis and output the survivor bits which correspond to the ML code word.

1.6.1 An Example of VA Decoding

Consider transmission of a sequence $\boldsymbol{u}$ = 1,0,1,0,1,0,0. Encoding using a rate 1/2 convolutional encoder results in the codeword $\boldsymbol{c}$= 11, 10, 00, 10, 00, 10, 11. Let us assume that second bit is corrupted when transmitted through the channel and is received as $\boldsymbol{r}$ = 10,10,00,10,00,10,11 as shown in the Figure 1.9. For received

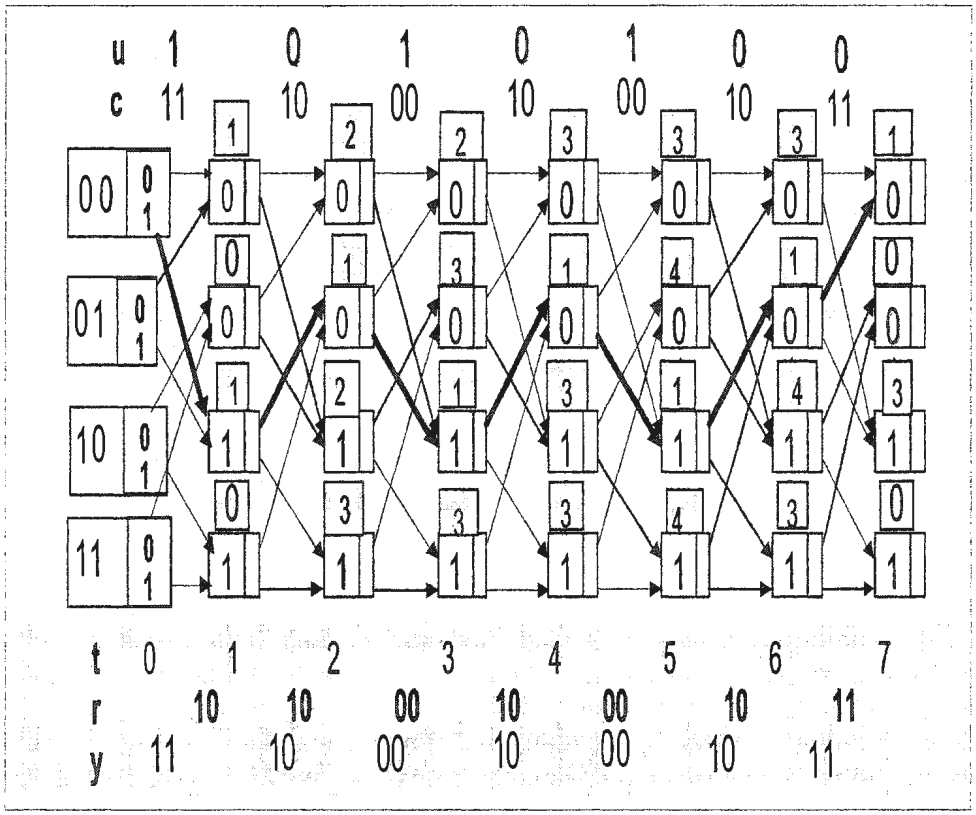


Figure 1.9: Viterbi decoding example

sequence $\boldsymbol{r}$, decoding using VA with lowest partial path metrics stored as best partial path metrics at each stage results in a survivor path $\boldsymbol{y}$ = 11,10,00,10,00,10,11 (shown

in bold), which is the same as the transmitted code word.

1.6.2 Hard and Soft Decision Decoding

If the decoder in the receiver makes a binary decision then it is called a hard decision, such as in the case of BPSK the demodulator decides whether the signal phase corresponds to a $+1$ (bit value 1) or -1 (bit value 0). A soft decision decoder in the receiver not only receives the binary value of '1' or '0', but also a confidence value associated with the bit. Soft-decision demodulators in the receiver quantize their output to 2^m levels where m is an integer ($m = 1, 2, \cdots$). It is represented by 1 bit for the sign, and the rest for magnitude. The larger the magnitude, the more confidence that the sign bit is correct. This is then passed as input to the decoder. A soft input decoder can produce hard estimate outputs or soft estimate outputs. A soft-input/soft-output (SISO) decoder receives soft (multibit digital) inputs and outputs soft decision.

1.7 Soft Output Viterbi Algorithm

The VA produces the ML output for convolutional codes. This algorithm provides optimal sequence estimation for one-stage convolutional codes. For concatenated systems as in turbo coding this algorithm is not effective against burst errors. The

performance is significantly improved if the Viterbi decoder produces reliability (soft-output) values. The reliability values are passed onto subsequent decoders as *a priori* information to improve the decoding performance as is seen in turbo decoders, which use iterative concatenated decoders. In this algorithm, to find the soft output non-surviving paths are considered. In Figure 1.9, we consider the non-surviving path the one that is produced by making a different decision in each stage of the trace-back. The SOVA algorithm can be implemented using the following steps [14, 15, 16].

Branch Metric Computation

The encoder state diagram is represented by means of a trellis diagram. It shows all the possible state transitions at each time step. In this stage the following computations are done:

1. (a) Initialize time $t = 0$.

   (b) Initialize $M_0^{(m)} = 0$ only for the zero state in the trellis diagram.

   (c) Initialize $M_i^{(m)}$ for all i (metrics corresponding to all other states) to ∞.

2. (a) Set time $t = t + 1$.

   (b) Compute the SOVA metric

$$M_t^{(m)} = M_{t-1}^{(m)} + u_t^{(m)} L_c y_{t,1} + \sum_{j=2}^{N} u_{t,j}^{(m)} L_c y_{t,j} + u_t^{(m)} L(u_t). \tag{1.4}$$

   for each state in the trellis diagram where m denotes the allowable binary

trellis branch/transition to a state $m = 1, 2$.

$M_t^{(m)}$ is the accumulated metric for time t on branch m.

$u_t^{(m)}$ is the systematic bit (first bit of N bits) for time t on branch m.

$x_{t,j}^{(m)}$ is the parity bit (j-th bit of N bits) at the encoder end for time t on branch $m(2 \leq j \leq N)$.

$y_{t,j}^{(m)}$ is the received value from the channel corresponding to $x_{t,j}^{(m)}$.

$L_c = 4 * (E_b/N_0) * a$ is the channel reliability value where a is the fading amplitude. For non fading AWGN channel $a = 1$.

$L(u_t)$ is the *a priori* sequence value from the second decoder.

Add Compare and Select

The heart of the Viterbi decoder is the ACS unit. It computes the maximum likelihood path at each node by first adding the branch metric to the value of the previous node/state for each of the two paths entering the current node and then selecting the maximum weight path. This is also called the path metric computer as it computes the partial path metrics at each node in the trellis. The surviving path at each node is identified and the traceback unit notified.

3. Find $\max(M_t^{(m)})$ for each state. Let $M_t^{(1)}$ denote the survivor path metric and $M_t^{(2)}$ denote the competing path metric.

4. Store $M_t^{(1)}$ and its associated survivor bit and state paths.

5. Compute $\Delta_{i=t}^{0} = 1/2|(r_t{}^{(i)} - y_t{}^{(i)})|$ and its associated survivor bit and state paths.

Traceback and Reliability values

6. Compare the survivor and competing paths at each state for time t and store the states where the estimated binary decisions of the two paths differ.

7. Go back to Step 2. until the end of the received sequence.

8. Output the estimated bit sequence u' and its associated "soft" or L-value sequence $(L_1(u') = u'.D)$, where the "." operator defines element-by-element multiplication operation and D is the final updated reliability sequence. $L(u')$ is then processed and passed on as the *a priori* sequence $L(u)$ for the succeeding decoder.

Extrinsic or *a priori* values The "soft" or L-value $L(u')$ can be decomposed into three distinct terms as given in [2].

$$L(u'_t) = L(u_t) + L_c y_{t,1} + L_e(u'_t) \tag{1.5}$$

$L(u_t)$ is the *a priori* value from the preceding decoder. $L_e(u'_t)$ is the extrinsic value produced by the present component decoder. The information that is passed between SOVA component decoders is the extrinsic value $L_e(u'_t) = L(u'_t) - L(u_t) - L_c y_{t,1}$.

The *a priori* value $L(u_t)$ is subtracted from the "soft" or L-value $L(u'_t)$ to prevent

passing information back to the decoder from which it was produced. Also, the weighted received systematic channel value $L_c y_{t,1}$ is subtracted to remove common information in the SOVA component decoders.

1.8 Objective of Research

The main goals of this research are

1. To gain a thorough understanding of the turbo coding and decoding algorithm.
2. To study through simulations,their behavior and performance in an additive white gaussian noise (AWGN) channel. This was done through software simulation.
3. To implement in hardware a turbo encoder/decoder in order to demonstrate the feasibility using currently available design tools.
4. To investigate modifications leading to a low power realization.

1.9 Organization of Thesis

Chapter 2 presents the various characteristics of turbo codes and explains techniques to achieve practical realizations of turbo codes. A review of the turbo code research literature and hardware implementations is also done in this chapter.

Chapter 3 starts with an example of the turbo encoding and decoding scheme and gives details of the software implementation of the turbo codec. Software simulation results that show the influence of various parameters such as the number of iterations, frame size, code rate, fixed point representation is presented in this chapter. In Chapter 4, integer and fixed point implementations of the SOVA algorithm are discussed. It is found that 5-bit representation of the input soft values and 8-bit representation of the internal datapath are sufficient to implement the algorithm with a little degradation in performance. Hardware architecture and implementation details of the different units of the turbo decoder are then presented. The performance of the implementation is also analyzed in this chapter. Conclusions and suggestions for future work are presented in Chapter 5. Module structures for the turbo decoder and timing diagrams of the hardware simulation are presented in Appendix A.

Chapter 2

Review of Related Work

In the beginning of this chapter, the mathematics involved in the SOVA is explained and more details about the algorithm are presented. Then, a discussion of the various parameters like trellis termination, interleaver structure and size, and puncturing is presented. All these parameters influence the turbo decoder performance. Some notable work that led to improvements in the SOVA algorithm is then reviewed. VA and SOVA have been investigated very well in the last 15 years and efficient implementations proposed in various papers. Some of the recent papers on this subject are discussed. The last section is devoted to a discussion of low power hardware design issues in general, some of which can be applied to the SOVA implementation. At the end of the chapter the main points of the chapter are presented as a summary.

2.1 Mathematics of the SOVA

In this section we derive the mathematics involved in SOVA using log-likelihood algebra [11, 15, 17, 16]. As described in Chapter 1 the VA algorithm is a soft input hard output type and for decoding turbo codes soft outputs are required for reliability information passed between decoders. There are two modifications to the VA algorithm that resulted in the SOVA: first, the path metrics are modified to take account of *a-priori* information when selecting the ML path through the trellis. Second, the VA is modified to provide soft outputs in the form of *a-posteriori* log likelihood ratios (LLR). The log-likelihood algebra used for SOVA decoding of turbo codes is based on a binary random variable u in GF(2) with elements $+1, -1$, where $+1$ is the logic 0 element or null element and -1 is the logic 1 element under *modulo*2 addition.

The log-likelihood ratio $L(u)$ for a binary random variable is defined as

$$L(u) = ln\frac{P(u=+1)}{P(u=-1)}. \tag{2.1}$$

$L(u)$ is denoted as the "soft" value or L-value of the binary random variable u. The sign of $L(u)$ is the hard decision of u and the magnitude of $L(u)$ is the reliability of this decision. Also the following relations hold:

$$P(u_1 \oplus u_2 = +1) = P(u_1 = +1)P(u_2 = +1) + P(u_1 = -1)(P(u_2 = -1) \tag{2.2}$$

$$P(u = -1) = 1 - P(u = +1) \tag{2.3}$$

where $\oplus$ indicates addition in the Galois field GF(2). The probability that u takes on a value of 1 is

$$P(u = +1) = \frac{e^{L(u)}}{1 + e^{L(u)}} \tag{2.4}$$

From Equations (2.2), (2.3) and (2.4) it can be found that the probability of the sum of two binary random variables being '1' and '0' is given by the following two expressions as

$$P(u_1 \oplus u_2 = +1) = \frac{1 + e^{L(u_1)}e^{L(u_2)}}{(1 + e^{L(u_1)})(1 + e^{L(u_2)})}, \tag{2.5}$$

and

$$P(u_1 \oplus u_2 = -1) = \frac{e^{L(u_1)} + e^{L(u_2)}}{(1 + e^{L(u_1)})(1 + e^{L(u_2)})}. \tag{2.6}$$

It follows from the definition of log-likelihood ratio (Equation (2.1)) that

$$L(u_1 \oplus u_2) = \ln \frac{P(u_1 \oplus u_2 = +1)}{P(u_1 \oplus u_2 = -1)}. \tag{2.7}$$

Using Equations (2.5) and (2.6) we have

$$L\left(u_1 \oplus u_2\right) = \ln \frac{1 + e^{L(u_1)}e^{L(u_2)}}{e^{L(u_1)} + e^{L(u_2)}}. \tag{2.8}$$

It is shown by Hagenauer in [15] that the above LLR can be approximated by the following:

$$L(u_1 \oplus u_2) \approx sign(L(u_1))\, sign(L(u_2)) \min(|L(u_1)|, |L(u_2)|). \tag{2.9}$$

where sign(x) represent the sign of variable x. The accuracy of this aproximation compared to the exact solution is shown in [14]. The inputs to the SOVA block in the turbo decoder are the channel outputs (encoded information and parity bits). The SOVA component shown in Figure 2.1 processes the log-likelihood ratios $\mathbf{L(u)}$ and $\mathbf{L_c y}$. $\mathbf{L(u)}$ is the *a-priori* sequence of the information sequence $\mathbf{u}$ and is produced from the preceding SOVA component decoder. In the first iteration it is zero. $\mathbf{L_c y}$ is the channel weighted received sequence $\mathbf{y}$.

Figure 2.1: SOVA block

The SOVA component produces outputs $\mathbf{u'}$, which is the estimated information sequence and "Soft-value" $\mathbf{L(u')}$ which is the log-likelihood ratio sequence.

The derivation of the modified Viterbi metric [12] which is used in the SOVA

component is presented here. The modified Viterbi metric takes into account *a-priori* information when selecting the ML path through the trellis.

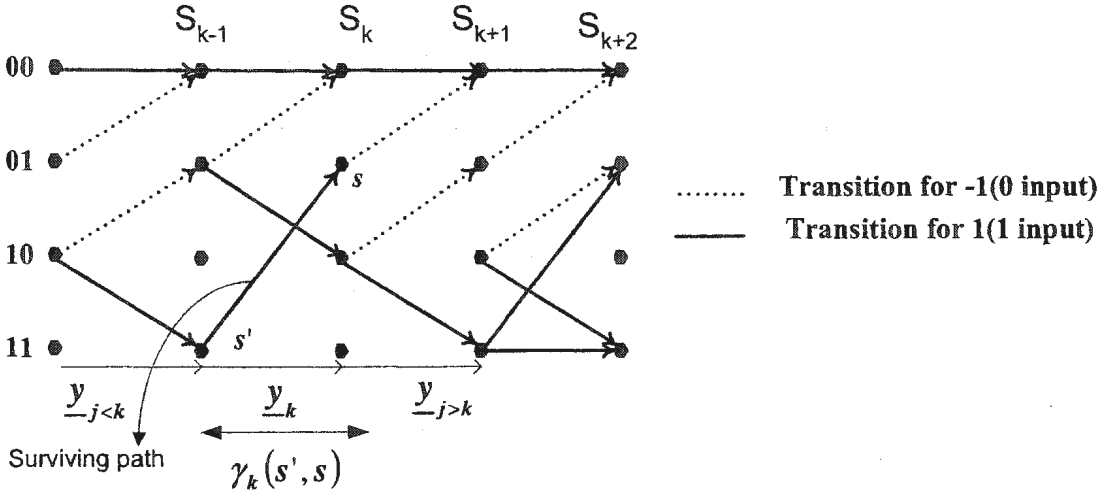


Figure 2.2: Trellis section

Consider the state sequence $\underline{s}_k^s$ which gives the states along the surviving path at state $S_k = S$ at stage k in the trellis shown in Figure 2.2. The probability that this is the correct path through the trellis is given by

$$p\left(\underline{s}_k^s \middle| \underline{y}_{j\leq k}\right) = \frac{p\left(\underline{s}_k^s \wedge \underline{y}_{j\leq k}\right)}{p\left(\underline{y}_{j\leq k}\right)}. \tag{2.10}$$

Now since the probability of the received sequence $p\left(\underline{y}_{j\leq k}\right)$ is constant for all paths $\underline{s}_k$ through the trellis to stage k, the probability that the path $\underline{s}_k^s$ is the correct one is proportional to $p\left(\underline{s}_k^s \wedge \underline{y}_{j\leq k}\right)$. Therefore the SOVA metric has to be defined to maximize $p\left(\underline{s}_k^s \wedge \underline{y}_{j\leq k}\right)$. If the path $\underline{s}_k^s$ has the path $\left(\underline{s}_{k-1}^s\right)$ for its first $k-1$ transitions then assuming a memoryless channel, we will have

$$\begin{aligned} p\left(\underline{s}_k^s \wedge \underline{y}_{j\leq k}\right) &= p\left(\underline{s}_{k-1}^{s'} \wedge \underline{y}_{j\leq k-1}\right) \cdot p\left(s \wedge \underline{y}_k \,|s'\right) \\ &= p\left(\underline{s}_{k-1}^{s'} \wedge \underline{y}_{j\leq k-1}\right) \cdot \gamma_k\left(s', s.\right). \end{aligned} \tag{2.11}$$

where $\gamma_k(s', s)$ is the probability that given the trellis was in state s' at time $k-1$ it moves to state s and the received channel sequence will be $\underline{y}_k$. The maximization is not changed if logarithm is applied to the whole expression and multiplied by 2 and the constant terms omitted. Then we have a suitable metric for the path $\underline{s}_k^s$

$$\begin{aligned} M\left(\underline{s}_k^s\right) &\underline{\underline{\Delta}} \ln\left(p\left(\underline{s}_k^s \wedge \underline{y}_{j\leq k}\right)\right) \\ &= M\left(\underline{s}_{k-1}^s\right) + \ln\left(\gamma_k\left(s', s\right)\right). \end{aligned} \tag{2.12}$$

Now $\gamma_k(s', s)$ is given as below

$$\gamma_k\left(s', s\right) = p\left(\underline{y}_k \,|\underline{x}_k\right) \cdot p\left(\underline{u}_k\right) \tag{2.13}$$

where $\underline{u}_k$ is the input bit necessary to cause the transition from state $S_{k-1} = s'$ to state $S_k = s$, $p(u_k)$ is the *a-priori* probability of this bit and $\underline{x}_k$ the transmitted codeword associated with this transmission.

Also the conditional received sequence probability $p\left(\underline{y}_k \,|\underline{x}_k\right)$ with (BPSK) modulation and matched filter demodulation [?] for Gaussian channel is given as

$$
\begin{aligned}
p\left(\underline{y}_k \left| \underline{x}_k \right.\right) &= \prod_{l=1}^{n} p\left(y_{kl} \left| x_{kl} \right.\right) \\
&= \prod_{l=1}^{n} \frac{1}{\sqrt{2\pi\sigma}} e^{\left(-\frac{E_b R}{2\sigma^2}\left(ykl - xkl\right)^2\right)}
\end{aligned}
\tag{2.14}
$$

where

xkl and ykl are individual bits with the transmitted and received code word

n is the number of bits in each code word,

E_b is the transmitted energy per bit,

σ^2 is the noise variance,

R is the fading amplitude.

Using Equations (2.13), (2.14) and omitting constant terms we thus have for the metric of SOVA

$$
M\left(\underline{s}_k^s\right) = M\left(\underline{s}_{k-1}^{s'}\right) + u_k L\left(u_k\right) + L_c \sum_{l=1}^{n} ykl \; xkl. \tag{2.15}
$$

The second modification to the Viterbi algorithm is to give the soft outputs. Consider the trellis shown in Figure 2.3. There are two paths reaching state $S_k = s$ at stage k in trellis, the SOVA algorithm calculates the metric for both paths according to Equation (2.15) and discards the path with lower path metric.

If $M\left(\underline{s}_k^s\right)$ and $M\left(\hat{\underline{s}}_k^s\right)$ are the path metrics for the two paths $\underline{s}_k^s$ and $\hat{\underline{s}}_k^s$ reaching state $S_k = s$ and path $\underline{s}_k^s$ is selected as the survivor, then we can define the metric

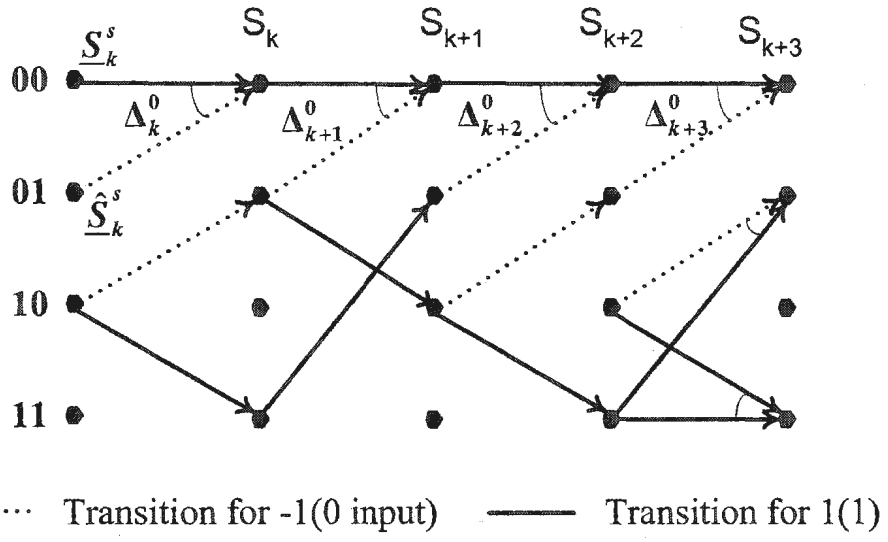


Figure 2.3: Simplified section of the trellis for a 4 state RSC code

difference as

$$\Delta_k^s = M\left(\underline{s}_k^s\right) - M\left(\hat{\underline{s}}_k^s\right) \geq 0. \tag{2.16}$$

The probability that we made the correct decision when we selected $\underline{s}_k^s$ as the survivor and discarded path $\hat{\underline{s}}_k^s$ is then

$$P(\text{correct decision at } S_k = s) = \frac{P\left(\underline{s}_k^s\right)}{P\left(\underline{s}_k^s\right) + P\left(\hat{\underline{s}}_k^s\right)}. \tag{2.17}$$

Thus from Equation (2.17) we see that the the metric in SOVA has the additional *a-priori* term. Figure 2.3 [12] shows a simplified section of the trellis of the $K = 3$ RSC code, with the metric differences Δ_k^s marked at various points in the trellis. Here K refers to the constraint length (one more than the number of memory elements in the encoder).

When we reach the end of the trellis and have identified the ML path through the trellis, we need to find the LLRs giving the reliability of the bit decisions along the ML path. Observations of the Viterbi algorithm have shown that all the surviving paths at a stage l in the trellis will normally have come from the same path at some point before l in the trellis. This point is taken to be at most δ transitions before l, where δ usually is set to be about three to five times the constraint length of the convolutional code. Therefore, the value of the bit u_k associated with the transition from state $S_{k-1} = s'$ to state $S_k = s$ on the ML path may have been different if, instead of the ML path, the Viterbi algorithm had selected one of the paths which merged with the ML path up to δ transitions later. Thus, when calculating the LLR of the bit u_k, the soft output Viterbi algorithm (SOVA) must take account of the probability that the paths merging with the ML path from stage k to stage $k+1$ in the trellis were incorrectly discarded. This is done by considering the values of the metric difference $\Delta_k^{s_i}$ for all states s_i along the ML path from trellis stage $i = k$ to stage $i = k + \delta$. It is shown by Hagenauer in [15, 16] that this LLR can be approximated by

$$L\left(u_k \left| \underline{y} \right.\right) \approx \min_{\substack{i=k,\ldots k+\delta \\ u_k \neq u_k^i}} \Delta_i^{s_i} \tag{2.18}$$

where u_k is the value of the bit given by the ML path, and u_k^i is the value of this bit for the path which merged with the ML path and was discarded at trellis

stage i. Note that this equation is of the form presented earlier as Equation (2.9). Thus the minimization in Equation (2.18) is carried out only for those paths merging with the ML path which would have given a different value for the bit u_k if they had been selected as the survivor. The paths which merge with the ML path, but would have given the same value for u_k as the ML path, do not affect the reliability of the decision of u_k.

2.2 Trellis Termination

Trellis termination normally refers to driving the encoder to the all-zero state. This is required at the end of each block to make sure that the initial state of the encoder before it encodes the next block of data is the all zero state.

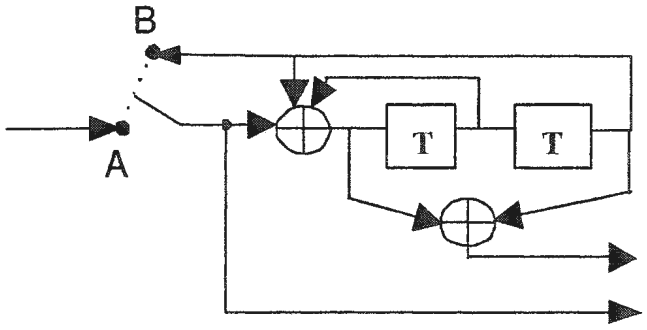


Figure 2.4: Trellis termination

Usually, for a component encoder with ν memory elements (2^ν states) operating on a block of N information bits, the state of the encoder is first found out after the N bits are encoded and then ν appropriate tail bits added to drive the encoder to the

all-zero state. A simple solution is shown in Figure 2.4. A switch in the encoder is in position A for the first N clock cycles and in position B for ν additional cycles. This will flush the encoder register with zeros after ν shifts and drive the encoder to the all-zero state. Note that with pseudo-random interleaver, this method will not terminate both component encoders of the turbo encoder to the same state. The performance degradation brought about by an unknown final state of the second encoder for large interleaver size N is shown to be negligible by Robertson [18]. When turbo codes were presented by Berrou et al. [8], the issue of trellis termination was not considered. The trellises for the interleaved and non-interleaved sequences were not terminated, and the decoder worked without any prior knowledge of the final state. While this was not considered a major problem with the large (65536-bit) frame sizes used, the performance improvement of trellis termination on smaller block sizes was seen in speech transmission by Jung and Nassahan [19].

A termination scheme was suggested by Jung and Nassahan [19], where only the second encoder is terminated and it is stated that this scheme gives better error performance than terminating the first encoder. The reason given for the improvement is that terminating the second sequence results in better extrinsic information as feedback for subsequent iterations. Barbulescu and Pietrobon [20] introduced a novel restriction on the interleaver pattern which forces the two trellises to end in the same state. This allows the use of a single tail sequence for both encoders. This type of interleaver is called a 'simile' interleaver. Divsalar and Pollara [21] suggested

a technique where both sequences are terminated (with different tails), and only the tail bits for the non-interleaved sequence are transmitted. An important observation is that the performance of a trellis termination method is the result of the combination of the termination method and the edge effects present for the particular interleaver used [22].

2.3 Sliding Window Implementation

In case of directly implementing infinite length decoding, for example in convolutional codes, very large amounts of memory would be required to store the systematic and parity bits and also the interleaver. This becomes an excessive burden in implementing to VLSI. However, in the continuous decoding using sliding-window technique, the required memory size is reduced by first partitioning the stream of information into frames. The end points of the frame are determined by the trellis termination technique previously discussed.

This frame is further divided into smaller windows and using the characteristics of convolutional coding that the path history nearly converges if we set the length of path history to several times the constraint length of the code, we set the small window to the length of path history and run the VA and SOVA. The Figure 2.5 provides a clearer picture. In this figure the first sliding window is $R + C$ symbols long. Once the path metrics are accumulated to $R + C$ stages then a Viterbi traceback

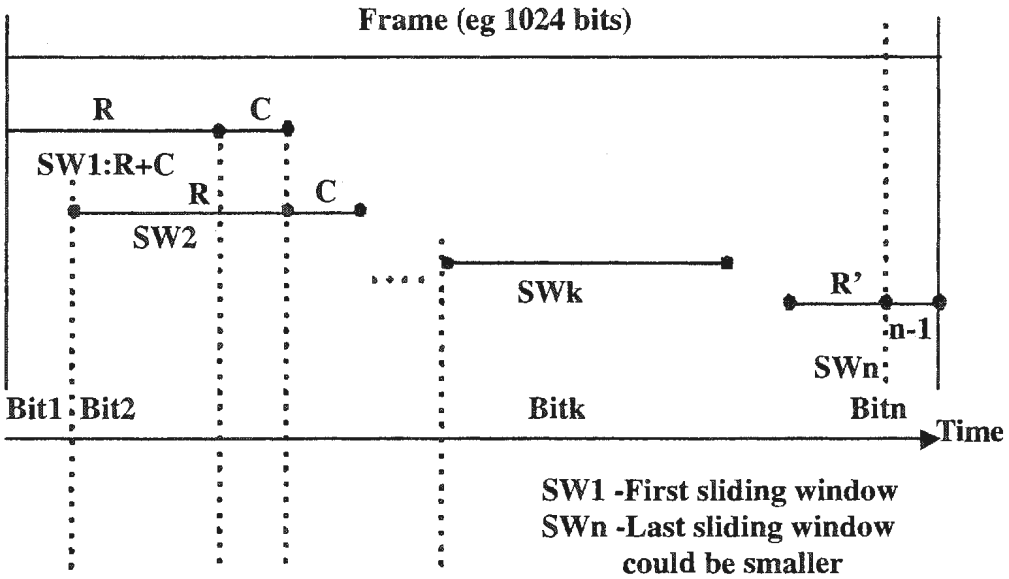


Figure 2.5: Sliding window decoding

is done to the first stage (time $t = 0$) and the VA bit released, then depending on whether we use the basic SOVA or modified SOVA, the SOVA multiple traceback is done from $R+C$th stage or the Rth stage and the soft value of the corresponding VA bit released. Using the latter approach reduces the hardware with a small penalty in terms of accuracy. Overall by using sliding-window technique, it is possible to achieve a smaller latency and higher throughput in comparison to infinite length decoding without noticeable degradation in the BER performance [23]. This degradation becomes considerable when the frame size increases. Further, very small size memory compared with normal convolutional Viterbi decoding can be achieved. Also the hardware size in terms of gate count is considerably reduced and thus reducing

overall power consumption. For a frame size of 1024 symbols, a window length of 30 symbols is good enough to produce comparable performance to the no window implementation case.

2.4 Effect of Interleaver on Decoder Performance

Interleaving is the rearrangement or permutation of a sequence of symbols such that two symbols which are close in time at the input get separated by a larger time unit at the output. Mainly, an interleaver transforms burst errors in a sequence into single errors and thus help in decoding.

In the original paper introducing turbo codes, Berrou and Glavieux [8] showed some of the parameters making a good interleaver. In particular:

1. Increasing the block size (and hence the size of the interleaver) results in improved performance.

2. The interleaver should randomize the input sequence in order to reduce particular low weight patterns mapping onto themselves, reducing the effective free distance of the resulting turbo code.

In [23], Branka et al. discuss three important roles played by interleavers. First, that the basic role of the interleaver is to construct a long block code from small memory convolutional codes, as long codes approach the Shannon capacity limit. Second, interleaving spreads out burst errors and decorrelates the inputs to the two component

decoders, so that an iterative decoding algorithm based on uncorrelated information exchange between component decoders can be applied. Finally, interleaving is done so that low weight code words (code words that have small Hamming distance from other code words) produced by one encoder are transferred into global high weight code words, thus increasing the code free Hamming distance. The input stream coming into encoder1 is shuffled and fed to the second encoder. If the encoder1 fails to produce high weight code words, then the second encoder will compensate with a higher weight code word and their combination with original sequence can result in a global higher weight code word. For turbo codes, it has been found that pseudorandom interleavers result in the best error correction performance among those interleavers studied in [2].

Although the effect of the interleaver on turbo code performance was recognized early by Divsalar and Pollara [21], the problem of choosing the optimum interleaver is still the subject of active research. It was shown, that randomly chosen interleavers perform better than structured interleavers as mentioned in the previous section. The importance of randomization was demonstrated by Divsalar and Pollara in [24], by considering the effect of the interleaver on critical low weight input patterns.

2.4.1 Interleaver Types

Interleavers are broadly divided into block or convolutional interleavers. The simplest type is the *row-column interleaver* in which the input sequence is written into a row

and the output taken from the column. It makes use of two memories of I rows and J columns, in order that data access is fast. When one memory is written into, data can be read from the other. For example for an input $x_0, x_1 \cdots x_7$, the corresponding interleaved outputs are $x_0, x_2, x_1, x_3, x_4, x_5 \cdots$, as shown in Figure 2.6.

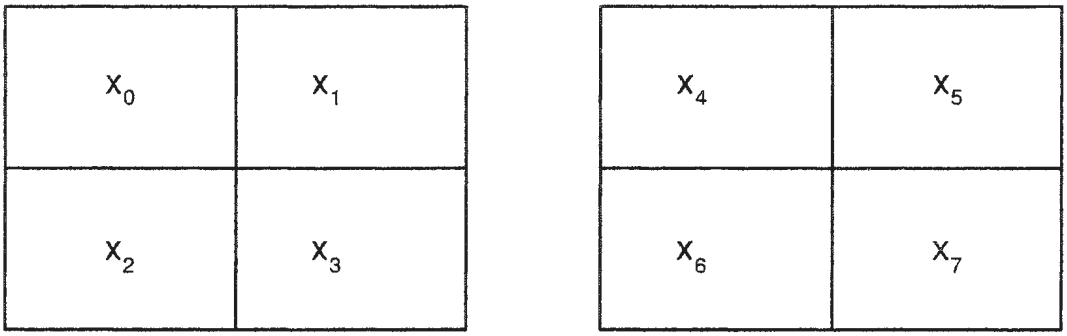


Figure 2.6: Row-Column interleaver

In the *pseudo-random* block interleaver the input symbols are written in a pseudo-random manner into the rows and output data read from the columns. It also has two memories. Example for an input $x_0, x_1 \cdots x_7$, the input sequence is written in a pseudo-random way into the rows and the outputs read from the columns. The outputs in this case are $x_3, x_1, x_2, x_0, x_7, x_5 \cdots$, as shown in Figure 2.7.

Sometimes the requirement is to drive both encoders of a turbo encoder to the same state after encoding the original sequence and the interleaved sequence as explained in the previous section. The *simile* interleaver has to perform the interleaving of the bits within each particular sequence in order to drive the encoder to the same state as that which occurs without interleaving. Since both encoders end in the same

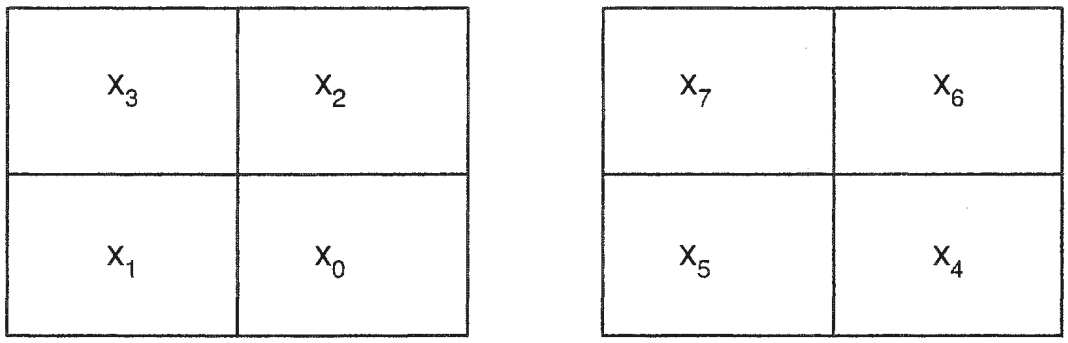


Figure 2.7: Pseudo-random interleaver

state, we need only one tail to drive both encoders to state zero at the same time.

The design of an interleaver for use in a punctured turbo code was first considered by Barbulescu and Pietrobon [25]. The odd-even interleaver proposed has the useful property that if the parity bit directly associated with any information bit is punctured in the non-interleaved sequence, then the corresponding parity bit in the interleaved sequence is not punctured. Random interleaver and block interleaver are employed and compared in the software and hardware implementations.

2.5 Effect of Puncturing

Conventional turbo codes are low rate codes. This implies that they require considerable bandwidth expansion, which may not be acceptable in some applications. It is a common practice [23] to puncture the output of the encoder (normally rate 1/3 turbo code) in order to increase the code rate to 1/2. For a rate 1/2 punctured turbo code,

the first output stream is the input stream itself (plus the necessary padding), while the second output stream is generated by multiplexing the M nonsystematic outputs of the RSC encoders. An example of a rate 1/2 punctured turbo encoder is shown in Figure 2.8. In this example, the multiplexer chooses its odd indexed outputs from the output of the upper RSC encoder and its even indexed outputs from the output of the lower RSC encoder. Based on the above example the default code rate is 1/3. When output puncturing is implemented the code rate will be increased to 1/2. This is particularly useful to increase the bandwidth efficiency as the message sent has less information and is compact. However, as the code rate increases, redundancy decreases, hence degrading the BER performance of the turbo code. This means non-puncturing will have better BER performance as more redundant information was sent along the faded channel. Therefore we trade the bandwidth efficiency for bit error rate.

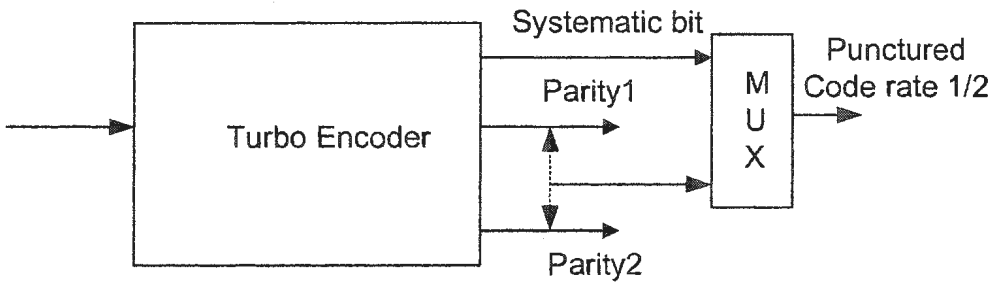


Figure 2.8: Punctured turbo code

It would be interesting to study to what increasing the number of decoding iterations can reduce the performance gap between the punctured and non-punctured code. In this research a non-punctured turbo code is implemented.

2.6 Stopping Criteria for Iterative Decoding

It is found that as the number of iterations increases the BER reduces and then after a point the reduction is marginal. Turbo decoding processing and delay can be improved if the decoder terminates the iterations after each individual frame bits are correctly estimated. The following are four stopping criteria that exist currently in the literature [1].

1. Cyclic Redundancy Check (CRC): CRC bits are appended by a CRC encoder before the frame is sent to the turbo encoder. In the decoder, CRC bits are used to check for frame errors after each iteration. When the CRC detects no errors, the iterative decoding is stopped. This method is attractive since most communication standards use some form of CRC for error detection.

2. Cross Entropy(CE): In this technique the approximate cross entropy between LLRs of the component decoders is calculated and the iterative decoding stopped if $CE1 < m * CE2$ where $m \approx 10^{-2}$ to 10^{-4}.

3. Sign Change Ratio (SCR): This technique is related to the CE technique. The

number of sign changes of the extrinsic information between successive iterations is computed and decoding terminated when this is a small value (usually $q_c N$ where q_c is a small number and N is the frame size).

4. Sign Difference Ratio (SDR): This is a new stopping criterion proposed by [1] to remove the need for storage of values from the previous iterations in the SCR method. It compares the sign of the extrinsic values from the two decoders and stops the iterations if the number of mismatches are a small fraction of the frame length.

2.7 Review of Improvements in the SOVA

In this section we review some notable changes to the SOVA proposed over the years since it was discovered in 1989. In order to get the reliability values for each output bit, an updating procedure is required after the hard decision bits are produced. There are two updating rules for SOVA, one proposed by G.Battail in [13] and the other by Hagenauer in [17]. The Battails's rule can be written as

$$u_j^s \neq u_j^c \Rightarrow L_j^s = \min(L_j^s, \Delta_k^m). \tag{2.19}$$

Equation (2.19) means that if at time k, the jth ($j < k$) bit in the survivor sequence u_j^s is not the same as the corresponding bit in the competitor sequence u_j^c, the new

reliability value L_j^s should be the minimum value between itself and Δ_k^m. Otherwise, the new log-likelihood value L_j^s should be the minimum value between itself and the sum of Δ_k^m and L_j^c,

$$u_j^s = u_j^c \Rightarrow L_j^s = \min(L_j^s, \Delta_k^m + L_j^c). \tag{2.20}$$

Lin and Cheng [26] proposed modifications to the original SOVA by limiting the reliability values to take care of the defect brought about by overestimating those values in the original SOVA. They also proposed a new interleaver to combat the tail effect of SOVA decoding. Fossorier et al. [27] theoretically proved the equivalence of the Battail's version of SOVA and the MAX-log-MAP algorithm. This improvement allowed the lower complexity Battail's algorithm as a candidate for implementation in third generation cdma2000 systems. Duanyi [28], verified through computer simulations that Battail's version of SOVA has a 0.4 dB performance improvement over the conventional SOVA and is comparable to the Max-Log-MAP algorithm.

In [1] the influence of quantization and fixed point arithmetic upon the BER performance of turbo codes was examined and an optimal scaling factor was found for the received signal so that it fit into the full scale range and the resolution of the quantizer. Also the upper bounds on the path metrics were found and an analytic bound on the minimum internal data width determined.

2.8 Design for Low Power

In the mid-80s, the increasing level of power dissipation in a chip made the industry shift from bipolar and NMOS technologies to CMOS technology. CMOS was a more energy efficient technology and also improved the integration level.

Low-power design can be performed at the architecture level, the logic/circuit level and at the device/process level [29]. At the architecture level, different architectures, such as parallel and pipelined architecture or transformations may be considered for low power dissipation. Also power management techniques like, shutdown of unused blocks, memory partition with selectively enabled blocks and reduction of the number of global busses, come under this category.

The reduction of switching activity at nodes of a circuit, use of more static style logic over dynamic style logic, optimization of clock and bus loading, reduction in V_{DD} (supply voltage) in non-critical paths and proper transistor sizing are the major focus at the circuit/logic level. Different transistor configurations in forming cells can also be applied at this level. For a standard cell approach, there is no control over internal circuitry (i.e. transistors) of a cell. Therefore low power design can be considered only at the architecture and other circuit/logic levels. At the device/process level voltage scaling, improved device characteristics, improved interconnect technology, higher density of integration, reduced capacitances through small geometries are some of the things that lead to power savings.

Any static CMOS chip has three sources that contribute towards power consumption. The first is called static power consumption and corresponds to the leakage values of CMOS transistors during stable states

The second source is called dynamic power dissipation. Dynamic power dissipation accounts for the major component of the power dissipation. Dynamic power dissipation of a full CMOS circuit is given by the Equation 2.21 [30],

$$P = \alpha C_l V^2 f. \tag{2.21}$$

Here, α is the switching activity, C_l is the parasitic capacitance, V is the supply voltage, and f is the clock frequency. Every time a gate changes its state (switches) it charges or discharges the parasitic capacitance. The amount of energy dissipated depends on the capacitance C_l and the source voltage V from which the capacitor is charged. The rate at which a gate switches in a circuit depends on the clock frequency. The switching activity α corresponds to the average percentage of gates which switch for each clock cycle. Given the formula for power dissipation, we can manipulate some parameters to reduce the power dissipation. The supply voltage and the clock frequency are determined at the system level, and they are beyond control of a circuit designer. The switching activity α and the parasitic capacitance C_l are affected by the circuit design. For the standard cell approach, it is possible to instruct some design tools, such as a place-and-router, to reduce the overall interconnect

length and hence to reduce the parasitic capacitance C_l. However, a major reduction in power dissipation can be achieved by reducing the switching activity α on which a designer has more direct control.

The third mechanism of power dissipation in static CMOS is short-circuit dissipation. This occurs because both the NMOS and PMOS transistors are on in a circuit for a short instant of time dt (related to the rise (t_r) and fall (t_f) times of a gate) during a state change. Simulations have shown that the amount of power dissipation caused by this short-circuit effect is related to both the input and output rise and fall times. A careful description of the circuit in a high-level hardware description language (such as VHDL) can yield a circuit with a lower switching activity. Some general techniques that can be employed for low power dissipation under the standard cell design approach are:

Elimination of redundant logic: A redundant logic, which does not contribute to the function of the circuit, dissipates power and should be eliminated. If a logic synthesis tool is used to synthesize a gate level circuit, elimination of redundant logic is performed during the logic synthesis.

Clock Gating: The clock gating is a powerful low power design techniques. Some blocks of a circuit are used only during a certain period of time. The clock of these blocks can be disabled to eliminate unnecessary switching when the blocks are not in use.

Toggle filtering: If signals arrive at the inputs of a combinational block at different times, the block may go through several intermediate transitions before it settles down. The intermediate transitions and consequently the dynamic power dissipation can be reduced, by blocking early signals until all input signals arrive. Bit-width minimization obviously also reduces power consumption. For example, it has been shown that reducing the bit-width by just one bit can reduce area and power consumption by up to about 25% [31]. Other special techniques for specific cases like, logic encoding, reduced swing clock, delay balancing, and adaptive filtering can also be used for power reduction [32].

2.9 Hardware Implementations

The SOVA was first proposed for serial concatenated coding scheme [11]. It was revised further in [15, 16] for use in turbo decoding. Since the VA forms the backbone of the SOVA implementation, it is useful to start with the study of existing hardware implementations of the VA and then build the SOVA implementation from there.

Hagenauer's rule presented in a previous section is simpler by updating only when $u_j^{(s)} \neq u_j^{(c)}$, hence it leads to lower complexity SOVA implementations as in [33].

Seki, Kubota, Mizoguchi and Kato [34] suggested a scarce state transition (SST) scheme to reduce the switching activity of a Viterbi decoder. The input is pre-decoded by a simple and power efficient decoder. The pre-decoded sequence, which

is not optimal under a noisy channel, is reprocessed by a Viterbi decoder to improve performance. The pre-decoded sequence is re-encoded using the same polynomial expression as used by the transmitter. It is then summed with the delayed received data in a modulo-2 operation. The summed data consists mainly of transmission errors. In the SST decoder, the input signals to the branch metric calculator and path memory circuits are all '0' unless many errors occur. Therefore the ML state of the decoding signal is distributed mostly around the '0' state reducing switching.

Oh and Hwang [35] proposed a traceback scheme that cut down the switching activities incurred while tracing back. Their scheme is designed for a decoder, where the traceback starts before the end of the code word. The main idea was to reuse the information from the previous trace to shorten the traceback. They achieved power reduction by a factor of ~ 5 over conventional traceback with increased hardware complexity by a factor of ~ 1.25

Joeressen et al., proposed a high speed VLSI architecture for the SOVA in [33]. In this paper they achieved high speed for the VA implementation by means of releasing D bits after an initial traceback of D steps. So the effective traceback depth is $2 * D$. In this paper several interesting ideas are presented, notably:

1. The Viterbi decoder add compare select unit (ACSU) must output metric differences to incorporate the SOVA.

2. Modifications are required only in the survivor memory unit (SMU) of the

Viterbi decoder.

3. Register exchange technique implementation of SMUs are superior in terms of regularity of design and decoding latency.

4. Trace back implementation of SMU achieve higher implementation efficiency, consume lower power.

Garrett and Stan [36] suggested a low-power architecture of the soft-output Viterbi decoder for turbo codes. They proposed an orthogonal access memory structure, which enables parallel access of sequentially received data. Use of such a memory structure reduces the switching activity for read and write of survivor path information

2.10 Summary

In this chapter the results of an extensive literature review are presented. The key concept of reliability updates was presented in a mathematical form in the first section, followed by discussions on the effect of trellis termination, sliding window implementation, effect of interleaver, interleaver types and puncturing in the subsequent sections. In order to reduce decoding latency, stopping criteria are presented highlighting their relative merits. Simplifications and improvements of the SOVA

that resulted in improved hardware implementation and better turbo decoding performance are then discussed. Some key issues in power reduction for digital design, that are being used in this research are also presented. Finally, some previous implementations are reviewed.

Chapter 3

Software Simulation of a Turbo Codec

3.1 Introduction

In order to implement a turbo decoder efficiently in digital hardware, software simulations of turbo decoding performance under floating point and fixed point conditions are necessary. Also software outputs of each unit can be used for comparison with outputs of corresponding units in hardware while testing the overall hardware. The turbo encoding and decoding systems were simulated using a C/C++ program. Figure 3.1 shows the functional block diagram of the software implementation. The main blocks are the data generator, turbo encoder, additive white gaussian noise (AWGN) generator, turbo decoder and the interleaver/deinterleaver. The encoder is made of two parallel recursive systematic convolutional encoders with generator polynomial $g(7,5)$. Encoder1 has been terminated while Encoder2 is not terminated.

The AWGN channel was used as it is the most commonly used model for communication channel and the noise generator was modelled using the Box Muller method. The decoding process is iterative serial decoding. The decoder is a 30-stage 4-state trellis that employs the SOVA. The flow chart of the SOVA decoding function is shown in Figure 3.2. The details of the software simulation along with results are presented in the following sections.

3.2 Model of Digital Communication System

Modern digital communication systems use source coding followed by channel coding and modulation. In the simulation we modelled the channel coding/decoding part. The first step was the generation of frames of 1024 bits and their turbo encoding. The software was made flexible so that frame length could be variable although most of the simulations were made with a frame length of 1024 bits. Noise was introduced after the turbo code generation and modulation. Our simulation models the binary phase shift keying (BPSK) modulation scheme although the simulation can be extended for any kind of modulation schemes. The binary phase shift keying modulation scheme is the conversion of the 1's and 0's into $+1$ and -1 domain and the modulation on to a sinusoidal carrier. The modulation and demodulation part is not actually implemented but assumed to exist. The AWGN was simulated for different signal to noise ratios. We assume a matched filter demodulator that outputs

soft values (weights depending on the magnitude of received estimates along with sign) rather than hard decision outputs ('0' or '1' which are based only on the sign of the received estimates) and the soft values are presented as inputs to the turbo decoder. The shaded area in Figure 3.1 shows the scheme described. The complete figure shows the main software blocks.

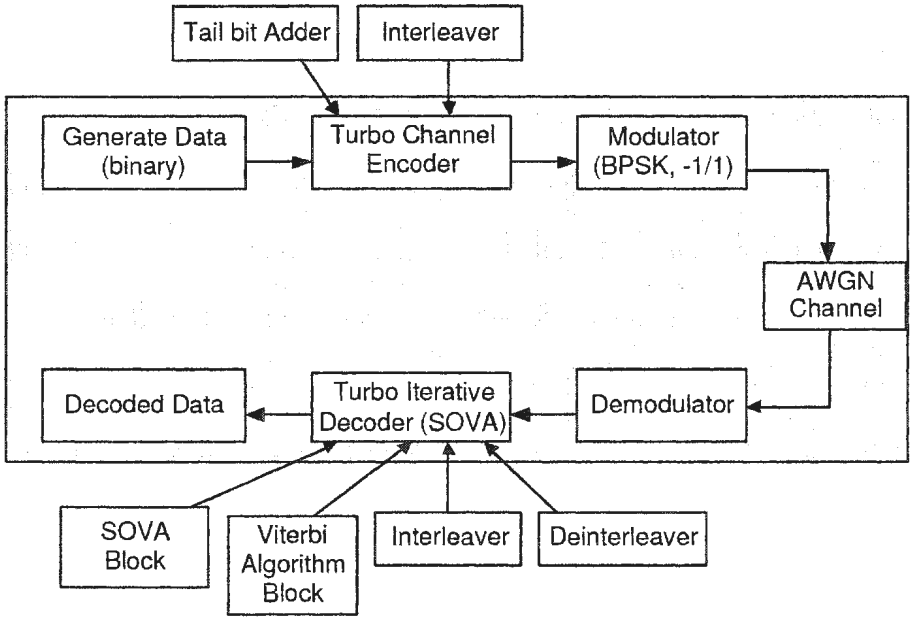


Figure 3.1: Model of digital communication system

Subsequent to the generation of the frame, tail bits are added by the *Tail bit adder* block. The tail bit addition requires knowledge of the current state of the memory elements in the encoder. The *Interleaver* shuffles the information stream

before providing it as input to the second decoder. The VA block does the ACS computations followed by the path management and storage of the state history. The SOVA block uses the VA block decoded bits, computes the soft values and updated reliability and extrinsic values. The deinterleaver block rearranges the shuffled inputs into the original order. The details of all these operations are presented in the following sections.

3.3 Flow Chart of a Turbo Decoder

The Figure 3.2 shows the flow chart of the SOVA algorithm for turbo decoding.

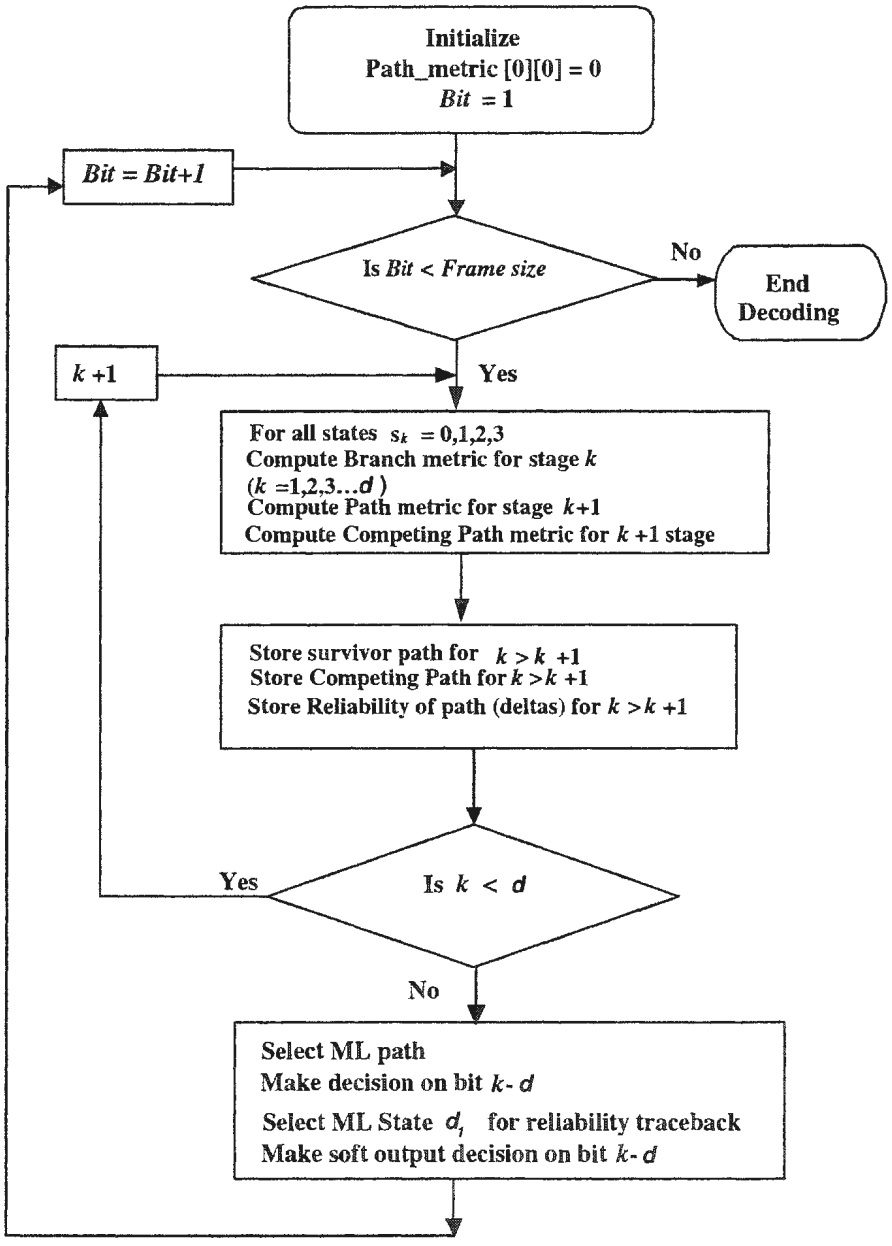


Figure 3.2: Flow chart of SOVA based turbo decoding

3.3.1 Add Compare Select

The heart of the Viterbi decoder is the ACS unit. It first computes the branch metrics. For branch metric computation, a modified Euclidean distance metric called *SOVA* metric is used for comparing the received symbols to the encoded symbols. Then it computes the maximum likelihood path at each node by first adding the branch metric to the value of the previous node/state for each of the two paths entering the current node and then selecting the maximum weight path. These nodes are stored in memory for traceback.

For this purpose the path metrics are initialized to zero. This unit is also called the path metric computer as it computes the partial path metrics at each node in the trellis. For our simulation a 4-state trellis was used due to its simplicity and wide usage. It is also easily implementable in hardware. Once the path metrics are computed for all the stages, the traceback is done from the zeroth state in the last stage. In the turbo decoder, Decoder1 traceback is started from the zeroth state while Decoder2 traceback is started from the state with highest path metric in the last stage. This is because the turbo encoder consists of two parallel convolutional encoders and Encoder1 is made to start and end in the same state (normally zero state). During traceback, the surviving path at each node stored in memory is identified and corresponding transition bits are released as decoded bits. Good accuracy is obtained if decoded bits are released after a traceback depth of 5 times or more of

the constraint length [4]. The constraint length K is given by $K = n + 1$ where n is the number of memory elements in each encoder. In this case the number of memory elements is 2 and thus a traceback depth of 15 would provide good accuracy. Bit by bit decoding or multiple bit release decoding are possible.

3.3.2 Path Management and Storage Unit

The Path management and storage unit is responsible for keeping track of the information bits associated with the surviving paths. There are two design approaches: register exchange and traceback. The traceback method is used in this implementation as it is faster and consumes less power. In the traceback method each state is assigned a register. Each state's register contains the history of the surviving branches entering the state. Information bits are obtained by tracing back through the trellis as per the history. In both techniques a shift register is associated with every trellis node throughout the decoding operation. Details of both approaches are presented in Chapter 4.

3.3.3 Soft Values Generation

The competing path metrics are also computed at the same time when the path metrics are computed, the competing path for each node is the path that has the lower weight amongst the two paths entering the node. The difference between the path metrics and competing path metrics is called the path metric difference and is

stored in memory for each of the states for all stages of the trellis.

The SOVA traceback is performed by tracing back from each of the stages through the competing path and determining the decision for the bit. Those stages that result in a different bit decision to the VA bit on traceback have their path metric differences stored in a register. Once the competing path tracebacks for a window are done a reliability update is performed. The reliability update corresponds to selecting the minimum of the path metric differences stored earlier. This minimum value is the reliability or soft value of the hard decision bit.

3.3.4 Interleaving and Deinterleaving

Two kinds of interleavers were used in the software simulation: Random interleavers and Block interleavers. The random interleaver takes in the input block of 1024 bits or symbols and rearranges them in a random position as given by the *Rand_Max* built-in function of the C++ compiler. For the block interleaver, the input block ($1024 = 8 * 128$) was written serially into 8 rows of 128 columns row-by-row and read out column-by-column.

3.3.5 Iterative Decoding

One iteration corresponds to decoding by the two serial decoders. The extrinsic values from Decoder2 are fed back into Decoder1 and serve as inputs for the second iteration. It is found that the BER keeps reducing with the number of iterations. The

magnitudes of the reliability values also increase with iteration and it is important to consider the number of bits necessary for fixed point implementation so that the reliability value does not overflow after a few iterations. With 5-bit representation it has been found through simulations that there is no overflow after 8 iterations.

3.4 Modelling the AWGN Channel

An AWGN noise generator was modelled using the Box Muller method [37]. The method generates a random sample n of Gaussian variables $N(0,1)$ (zero mean and standard deviation 1) from two random variables uniformly distributed over $[0,1]$. The method uses the fact that a Rayleigh-distributed random variable R, with the probability distribution

$$F(R) = \begin{cases} 0 & \text{if } R < 0 \\ 1 - exp(-R^2/2 * \sigma^2) & \text{if } R >= 0 \end{cases}$$

is related to a pair of Gaussian variables C and D through the transformation $C = R * cos(\theta)$ and $D = R * sin(\theta)$, where θ is a uniformly distributed variable in the interval $(0, 2 * pi)$. It is implemented by the function *addgauss* (mean, sigma) in the software and the pseudo code is described below.

- generate a uniformly distributed random number u_1 between 0 and $1 - 10^{-6}$.

  u_1 = `(double)lrand() / (LRAND_MAX +1)`

- generate a Rayleigh-distributed random number R with u_1 using the following transformation.

  `R = sigma * sqrt( 2.0 * log( 1.0 / (1.0 -` u_1`) ) )`

- generate another uniformly-distributed random number u_2 as before.

  u_2 `= (double)lrand() / (LRAND_MAX +1)`

- generate and return a Gaussian-distributed random number using R and $\theta = 2 * pi * u_2$.

  `return( (float) ( mean + R *` $\cos(\theta)$ `) )`

3.5 Turbo Encoding/Decoding Example

In order to have a clearer picture of the encoding and decoding processes, an example of a 6 bit data stream is presented below. The example refers to the turbo decoder block diagram of Figure 1.6 and uses the label SOVA1 component decoder to refer to the SOVA decoder1 of the block diagram and SOVA2 component decoder to refer to the SOVA decoder2. Consider the input information sequence $\mathbf{u} = \mathbf{0\ 0\ 1\ 1\ 0\ 1}$ and with tail bits added the information sequence is 0 0 1 1 0 1 1 0. It is first encoded by the parallel concatenated encoders. Encoder1 produces two output sequences, the

original sequence with two tail bits added (systematic bits) $x_1 = \{0\ 0\ 1\ 1\ 0\ 1\ 1\ 0\}$ and the encoded bits $x_2 = \{0\ 0\ 1\ 0\ 0\ 0\ 1\ 0\}$.

The encoding operation of the first encoder is shown in Figure 3.3.

INDEX	1	2	3	4	5	6	7	8
x_1	0	0	1	1	0	1	1	0
Random Interleaver	8	5	1	6	7	4	3	2
Interleaved data	0	0	0	1	1	1	1	0

x_1	D1	D2	Feedback x	x_1	x_2	
0	0	0	0	0	0	
0	0	0	0	0	0	
1	0	0	1	1	1	
1	1	0	0	1	0	
0	0	1	1	0	0	
1	1	0	0	1	0	
1	0	0	0	1	1	Tail
0	0	0	0	0	0	bits
	0	0				

INDEX	1	2	3	4	5	6	7	8
Interleaved data I(x_1)	0	0	0	1	1	1	1	0
Encoder2output x_3	0	0	0	1	0	1	1	1

Figure 3.3: Encoding operation

The original sequence with tail bits is interleaved and the interleaved output $I(x_1) = \{0\ 0\ 0\ 1\ 1\ 1\ 1\ 0\}$ is fed as input to Encoder2. The Encoder2 in turn encodes and outputs a single sequence $x_3 = \{0\ 0\ 0\ 1\ 0\ 1\ 1\ 1\}$. The encoding operation is same as that of Encoder1. Thus for every single input bit, 3 turbo coded bits are output resulting in a code rate of 1/3.

The transmitted data after modulation and received data after demodulation

and demultiplexing is shown in Figures 3.4 (a) and (b). '0' indicates an erasure, to indicate the reception of a signal whose corresponding symbol value is in doubt. Assuming $E_b/N_0 = 1$, the weighted received sequence $L_c = 4\frac{E_b}{N_0}a$ where $a = 1$ is the fading amplitude for AWGN channel and is shown in Figure 3.5.

x_1	-1	-1	1	1	-1	1	1	-1
x_2	-1	-1	1	-1	-1	-1	1	-1
x_3	-1	-1	-1	1	-1	1	1	1

(a)

y_1	-1	-1	1	1	-1	1	1	-1
y_2	**0**	-1	1	-1	-1	-1	1	-1
y_3	**0**	-1	-1	1	-1	1	1	1

(b)

Figure 3.4: Transmitted (a) and received sequence(b)

$L_c y_1$	-4	-4	4	4	-4	4	4	-4
$L_c y_2$	**0**	-4	4	-4	-4	-4	4	-4
$L_c y_3$	**0**	-4	-4	4	-4	4	4	4

Figure 3.5: Weighted received sequence

The trellis diagram and state diagram are required for decoding. The SOVA1 decoder is used to decode the RSC1 code. The SOVA1 decoder's input sequences are $L_c y_1$, $L_c y_2$ and $L_{e2}(u')$. For the first iteration the initial values of $L_{e2}(u')$ are all zeros.

The metric $M_t^{(m)} = M_{t-1}^{(m)} + u_t^{(m)} L_c y_{t,1} + x_{t,2}^{(m)} L_c y_{t,2} + u_t^{(m)} L_{e2}(u_t')$ is used to build the trellis as explained in section 1.7.

The first SOVA decoder's ML path is shown in Figure 3.6. It uses the metrics described in Figure 3.7. Traceback is done from the zeroth state of the last stage.

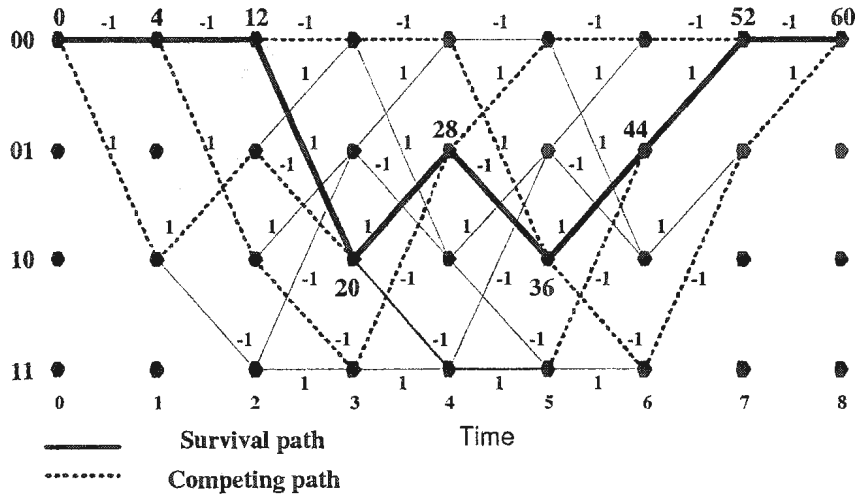


Figure 3.6: SOVA1 decoder ML path

Survivor and competing partial path metrics for the trellis diagram are shown in Figure 3.7.

	Time0	Time1	Time2	Time3	Time4	Time5	Time6	Time7	Time8
State 0	**0**	**4**	**12**	4/4	4/-4	12/20	20/12	12/**52**	**60**/20
State 1			-4	-4/-4	**28**/-12	4/12	**44**/4	20/28	
State 2		-4	-4	**20**/-12	4/-4	-4/**36**	20/12		
State 3			-4	-4/-4	12/4	4/12	28/20		

Figure 3.7: Survivor and competing partial path metrics for the trellis

From the state diagram and the ML path (Figure 3.6) the SOVA1 component decoder produces the estimated sequence and state sequence

$$u' = -1\ -1\ +1\ +1\ -1\ +1\ +1\ -1,$$

$$s = 00(0)\ 00(0)\ 10(2)\ 01(1)\ 10(2)\ 01(1)\ 00(0)\ 00(0).$$

The competing paths (bit and state sequences) for reliability updates are shown in Figure 3.8. SOVA1's calculated and updated (in bold) reliability values are represented in Figure 3.9. They are calculated from the difference of the survivor path metric and the competing path metric by the equation $\Delta_t^0 = 1/2|(M_t{}^{(1)} - M_t{}^{(2)})|$ explained in section 1.7.

	Time0	**Time1**	**Time2**	**Time3**	**Time4**	**Time5**	**Time6**	**Time7**	**Time8**
Time3		1/10	1/01	1/10					
Time4		-1/00	1/10	-1/11	-1/01				
Time5		-1/00	-1/00	-1/00	-1/00	1/10			
Time6		-1/00	-1/00	1/10	-1/11	1/11	-1/01		
Time7		-1/00	-1/00	1/10	1/01	1/00	-1/00	-1/00	
Time8		-1/00	-1/00	1/10	1/01	-1/10	-1/11	-1/01	1/00

Figure 3.8: Competing paths for reliability updates

	Time0	**Time1**	**Time2**	**Time3**	**Time4**	**Time5**	**Time6**	**Time7**	**Time8**
Time3		**16**	**16**	**16**					
Time4		20	**20**	**20**	**20**				
Time5		20	20	**20**	**20**	**20**			
Time6		20	20	20	**20**	**20**	**20**		
Time7		20	20	20	20	**20**	**20**	**20**	
Time8		20	20	20	20	20	**20**	**20**	**20**

Figure 3.9: SOVA1's updated reliability values

SOVA1 component decoder produces the updated reliability sequence and estimate.

$$\begin{aligned}\Delta &= 16\ 16\ 16\ 20\ 20\ 20\ 20\ 20,\\ u' &= -1\ -1\ +1\ +1\ -1\ +1\ +1\ -1.\end{aligned}$$

The SOVA1 component decoder outputs the *soft* or L-value sequence as

$$L_1(u') = -16\ -16\ +16\ +20\ -20\ +20\ +20\ -20.$$

The extrinsic value sequence, obtained by subtracting SOVA1's inputs from the *soft* or L-value sequence is given by

$$L_{e1}(u') = L_1(u') - L_c y_1 - L_{e2}(u') = -12\ -12\ +12\ +16\ -16\ +16\ +16\ -16.$$

The SOVA2 component decoder is used to decode the RSC2 code. The SOVA2 component decoder's input sequences are $I(L_c y_1)$, $L_c y_3$ and $I(L_e 1(u'))$ and the values are shown in Figure 3.10.

The trellis in Figure 3.11 shows the ML path and the competing paths for Decoder2. The metric used is : $M_t^{(m)} = M_{t-1}^{(m)} + u_t^{(m)} I(L_c y_{t,1}) + x_{t,2}^{(m)} L_c y_{t,3} + u_t^{(m)} I(L_{e1}(u_t'))$.

Survivor and Competing partial path metrics for the trellis diagram of Decoder2

INDEX	1	2	3	4	5	6	7	8
Random Interleaver	8	5	1	6	7	4	3	2
$L_c y_1$	-4	-4	4	4	-4	4	4	-4
$I(L_c y_1)$	-4	-4	-4	4	4	4	4	-4
$L_{e1}(u')$	-12	-12	12	16	-16	16	16	-16
$I(L_{e1}(u'))$	-16	-16	-12	16	16	16	12	-12
$L_c y_3$	0	-4	-4	4	-4	4	4	4

Figure 3.10: Input sequence to SOVA2 decoder

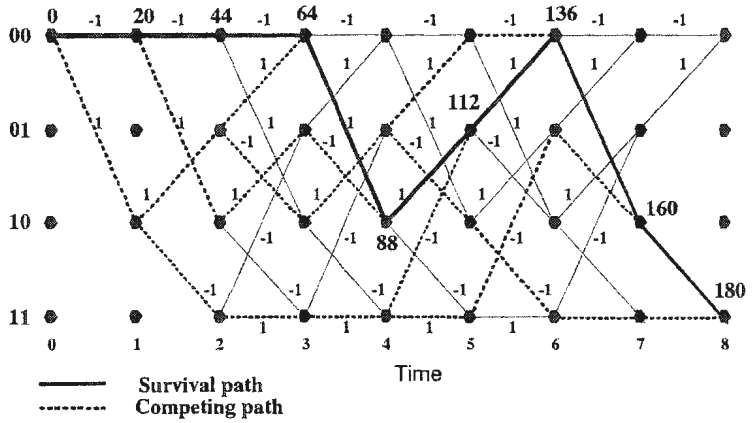


Figure 3.11: SOVA2 decoder ML and competing paths

are given in Figure 3.12. With the knowledge of the ML path, the second SOVA

	Time0	Time1	Time2	Time3	Time4	Time5	Time6	Time7	Time8
State 0	0	20	44	64/56	40/32	24/56	32/136	120/92	130/88
State 1			-36	-16/8	-8/40	0/112	72/48	100/160	110/140
State 2		-20	-4	24/-16	-16/88	56/24	88/80	52/160	110/100
State 3			-4	-16/8	8/24	48/64	80/40	76/92	180/72

Figure 3.12: Survivor and competing partial path metrics for the trellis of Decoder2

decoder (SOVA2 component decoder) recomputes the SOVA metric and also calculates and updates the reliability values. From the state diagram and the ML path

(Figure 3.11), the SOVA2 component decoder produces the estimated bit sequence and state sequence as

$$I(u') = 52\ 56\ 52\ 52\ 52\ 52\ 52\ 52,$$

$$I(s) = 00(0)\ 00(0)\ 00(0)\ 10(2)\ 01(1)\ 00(0)\ 10(2)\ 11(3).$$

The competing paths (bit and state sequences) for reliability updates of Decoder2 are given in Figure 3.13.

	Time0	**Time1**	**Time2**	**Time3**	**Time4**	**Time5**	**Time6**	**Time7**	**Time8**
Time3		**1/10**	**1/01**	**1/00**					
Time4		-1/00	**1/10**	**1/01**	**-1/10**				
Time5		-1/00	-1/11	**1/11**	1/11	**-1/01**			
Time6		-1/00	**1/01**	-1/10	1/01	1/00	**-1/00**		
Time7		**1/11**	-1/11	**1/11**	1/11	1/11	**-1/01**	**-1/10**	
Time8		-1/00	**1/01**	-1/10	1/01	**-1/10**	**-1/11**	1/11	1/11

Figure 3.13: Competing paths for reliability updates for Decoder2

SOVA2's calculated and updated reliability values are in Figure 3.14.

	Time0	**Time1**	**Time2**	**Time3**	**Time4**	**Time5**	**Time6**	**Time7**	**Time8**
Time3		**60**	**60**	**60**					
Time4		52	**52**	**52**	**52**				
Time5		56	56	**56**	56	**56**			
Time6		52	**52**	52	52	52	**52**		
Time7		**54**	54	**54**	54	54	**54**	**54**	
Time8		54	**54**	54	54	**54**	**54**	54	**54**

Figure 3.14: SOVA2's updated reliability values

SOVA2 decoder produces the estimate and final reliability sequence as

$$\begin{aligned} I(\Delta) &= 54\ 52\ 52\ 52\ 54\ 52\ 54\ 54, \\ I(u') &= -1\ -1\ -1\ +1\ +1\ +1\ +1\ -1. \end{aligned}$$

The SOVA2 component decoder outputs the *soft* or L-value sequence

$$I(L_2(u') = -54\ -52\ -52\ +52\ +54\ +52\ +54\ -54.$$

The extrinsic value sequence, obtained by subtracting SOVA2's inputs from the *soft* or L-value sequence is

$$I(L_{e2}(u')) = I(L_2(u')) - I(L_c y_1) - I(L_{e1}(u')) = -34\ -32\ -36\ 32\ 34\ 32\ 34\ -34.$$

The extrinsic value is deinterleaved to get

$$I^{-1}(I(L_{e2}(u')) = -36\ -34\ +34\ +32\ -32\ +32\ +34\ -34.$$

and is used to decode the RSC1 code for the next decoding iteration.

The estimated bit sequence $I(u') = -1\ -1\ -1\ +1\ +1\ +1\ +1\ -1$ is also deinterleaved to get $I^{-1}(I(u')) = -1\ -1\ +1\ +1\ -1\ +1\ +1\ -1$. This includes the tail bits and is the estimated information sequence. After mapping from the

$(-1, +1)$ domain to the $(0, 1)$ domain, the estimated information sequence is $u' = u = 0\ 0\ 1\ 1\ 0\ 1$ excluding the tail bits. This is indeed the transmitted 6-bit sequence.

3.6 Software Implementation

3.6.1 Window Decoding

The decoding is done frame by frame. Each frame is composed of several bits (1024 for 3G). There are two approaches to building the trellis, either build a window of fixed trellis length and use it repeatedly to decode the whole frame or build a trellis for the whole length of the frame. The first approach is faster and consumes less hardware. A 30-stage sliding window trellis is used for decoding the frame. The window uses the first set of 60 symbols received and builds the 30-stage trellis, then the Viterbi algorithm traces back to the first stage and releases the first hard decision bit. Now the SOVA traces back to the same depth and releases the soft value for the decoded first bit. Then the trellis is extended to the next stage and the window slides to run the VA and SOVA again releasing the second and consecutive bits. A modified SOVA where the SOVA multiple tracebacks are shortened is also implemented for study purpose. In this case a Viterbi traceback window of 30 and SOVA traceback window of 20 is used. This choice is arbitrary but has to fulfil the criteria that the traceback length is $\geq 3*$ constraint length. This choice also is a tradeoff between hardware complexity and decoding speed and offers some savings from the hardware

implementation point of view.

3.6.2 Path Metric Normalization

The path metric computation in the trellis chain is cumulative. At each node of the trellis the current branch metric value is added to the previous stage path metric to get the new current path metric. Also since the Decoder2 extrinsic values are added to path metric computation for Decoder1 from iteration2 onwards, the path metric values tend to increase to a high value after a few iterations. In a fixed point implementation it is thus important to ensure that the path metric value does not overflow. This is done by subtracting the path metric value of all the nodes (4 nodes or states in our case) at that stage from a fixed number once the path metric reaches a particular maximum value. Another approach is to subtract the value of the smallest path metric at each stage. The first method was adopted in the implementation as it involved only a subtraction as compared to the latter approach which involved a comparison to determine the least path metric and then subtraction.

3.6.3 Fixed Point Implementation

The turbo coded bits after addition of noise are seen as soft inputs by the turbo decoder in contrast to '1' or '0' detection by earlier hard decision receivers. Observation of the soft values for several frames for different signal to noise ratios in the simulation, led to the conclusion that 5 bits were sufficient for their 2's complement

representation. In Figure 3.15, they are indicated as $L_c y_{t,1}$, $L_c y_{t,2}$. The branch metric $BM_t^{(m)}$ is a sum of these soft values and the reliability value $L_{e2}(u_t')$. Since the reliability value is proportional to the path metric differences (Path metric - Competing path metric), it can also be represented as a 5-bit value without much degradation. In view of the above facts we can conclude that the branch metric values will not exceed 6-bit values.

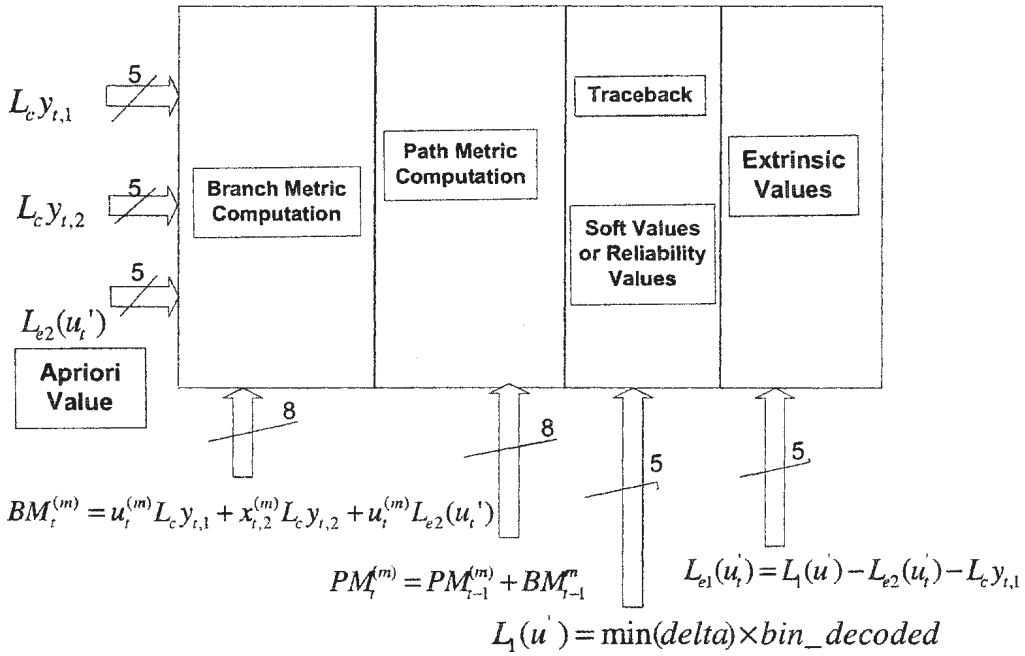


Figure 3.15: Fixed point representation of SOVA decoder parameters

It is shown in [38] that the maximum spread of path metrics is given by the formula:

$$PM_{max} - PM_{min} \leq (K-1)(BM_{max} - BM_{min}). \tag{3.1}$$

For our case with constraint length $K = 3$, maximum and minimum branch metrics

$BM_{max} = 31, BM_{min} = -32$, the maximum difference between largest and smallest path metric is 126. This implies that at least 7 bits are required to represent the path metrics. In the software implementation soft values were represented as 5-bit values and path metrics (with some margin) were represented as 8-bit values.

3.7 Simulation Results

The Turbo encoding/decoding performance was simulated and studied by varying different parameters like iteration number, frame size, code rate and interleaver type. The Bit Error Rate (BER) is an indicator of decoding performance and it is plotted on the X-axis while the Y-axis indicates the signal to noise ratio of the transmission. The results are discussed with tables and graphs in the following subsections.

3.7.1 Influence of Iteration Number

The influence of iteration number on turbo code performance is shown in Figure 3.16. Here the continuous turbo decoding was adopted and a floating point implementation was tested. A random interleaver is the default interleaver used for the simulation. It is seen that the BER decreases as the number of iterations increases. Also it was found that as the number of iterations increases, the improvement in the BER diminishes. The 3G specifications (BER -10^{-5}) [39] at SNR 2 dB can be obtained after 7 iterations of decoding.

S/N(dB)	Iterations	BER
1	0	0.181
1.5		0.167
2.0		0.152
1	1	0.075
1.5		0.046
2.0		0.232
1	3	0.023
1.5		0.006
2.0		0.0004
1	5	0.0114
1.5		0.0022
2.0		0.00015
1	7	0.0077
1.5		0.0011
2.0		0.000081

Table 3.1: BER vs Signal to Noise ratio for different iterations (1024 bit frame)

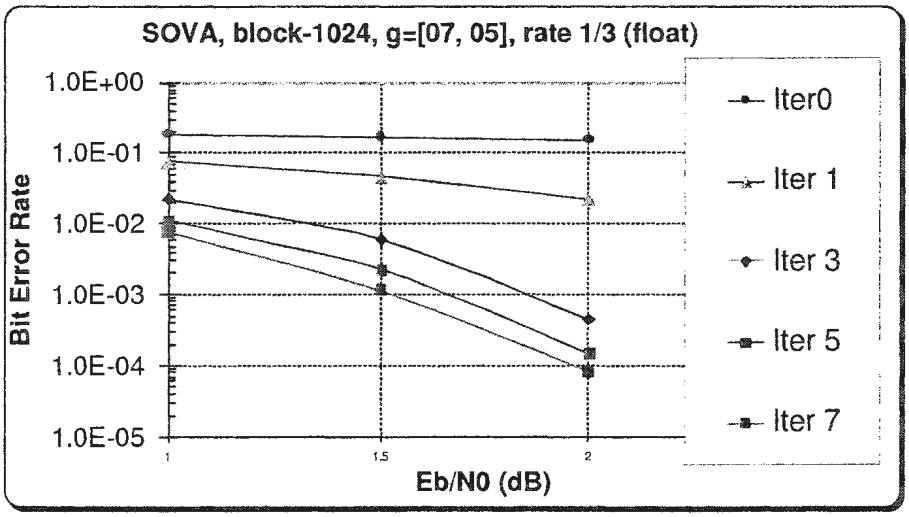


Figure 3.16: BER vs Signal to Noise ratio for different iterations (1024 bit frame)

3.7.2 Influence of Windowed Implementation

The influence of a windowed floating point implementation of the turbo decoder versus that of a continuous decoding floating point implementation on turbo decoding performance is shown in Figure 3.17. The continuous lines indicate the case of windowed implementation and the dashed lines indicate the performance of continuous decoding. There is very little degradation compared to continuous decoding for a windowed implementation (window length of 30, 1024 bit frame) as shown by the figure and data in Table 3.2.

S/N(dB)	Iterations	BER(window)	BER(continuous)
1	1	0.0759	0.0755
1.5		0.0439	0.0448
2.0		0.0222	0.0191
1	3	0.02438	0.0240
1.5		0.0036	0.00355
2.0		0.00048	0.00035
1	5	0.0132	0.0142
1.5		0.0011	0.00109
2.0		0.00014	0.00011
1	7	0.00852	0.0091
1.5		0.00056	0.0006
2.0		0.000098	0.000059

Table 3.2: BER vs Signal to Noise ratio for windowed vs continuous decoding

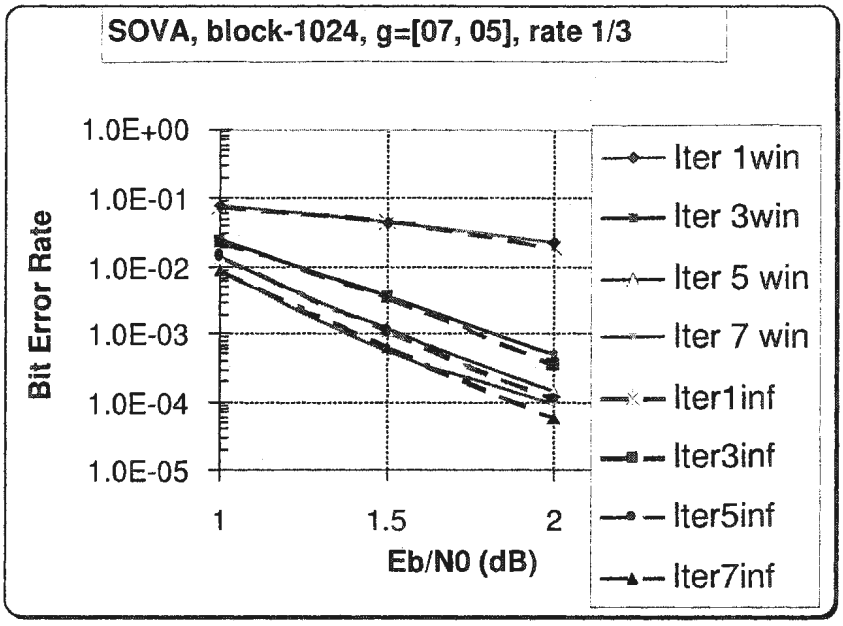


Figure 3.17: Windowed implementation vs Continuous decoding performance

3.7.3 Influence of Frame Size

The influence of frame size on turbo decoding performance was simulated with three different frame lengths (512, 1024, 4096).

S/N(dB)	Frame size	BER
1	512	0.031
1.5		0.00656
2.0		0.00083
1	1024	0.025
1.5		0.00272
2.0		0.00022
1	4096	0.01568
1.5		0.00034
2.0		0.00004

Table 3.3: BER vs Signal to Noise ratio for different frame sizes

As the frame size increased, the interleaver size also increased. The BER performance (after 8 iterations) increased with frame size as shown in Figure 3.18. This is attributed to the interleaver and is called the *interleaving gain* [23].

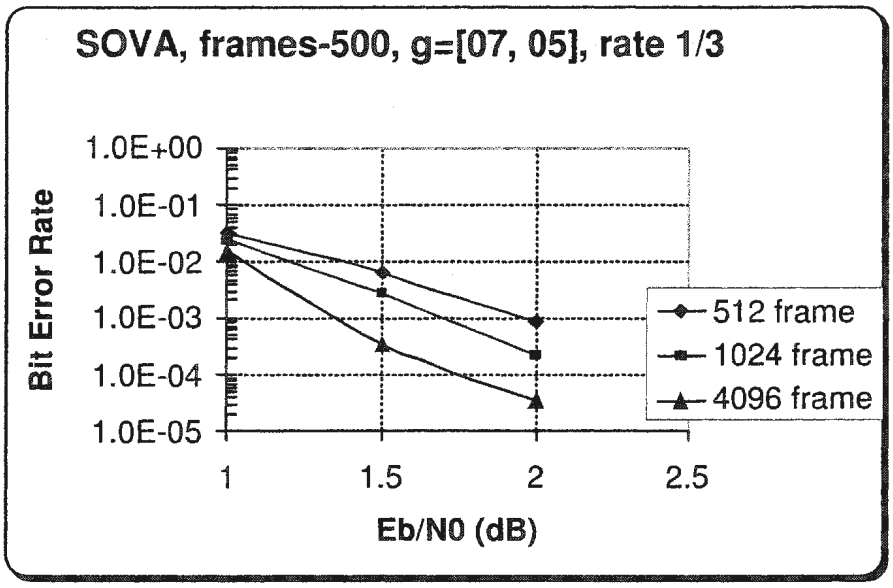


Figure 3.18: BER vs Signal to Noise ratio for different frame sizes

3.7.4 Influence of Interleaver Type

The influence of random and block interleavers $8X128$ were compared in the simulation. Random interleavers clearly out performed rectangular interleavers at all signal to noise ratios as shown in Figure 3.19.

S/N(dB)	Interleaver	BER
1	Random	0.0227
1.5		0.0030
2.0		0.0002
1	Block	0.0419
1.5		0.0177
2.0		0.0069

Table 3.4: BER vs Signal to Noise ratio for different interleaver types

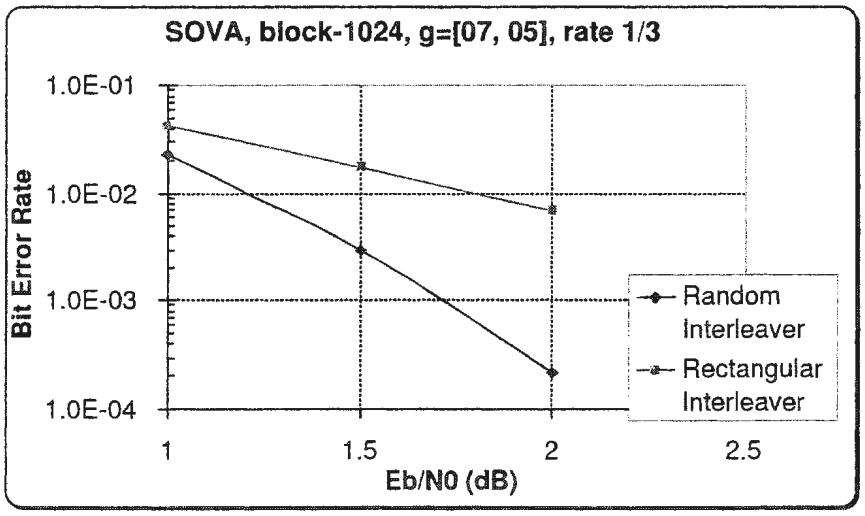


Figure 3.19: BER vs Signal to Noise ratio for random and block interleavers

3.7.5 Influence of Code Rate

The influence of code rate (1/3 and 1/2) on turbo code performance is shown in Figure 3.20 [14]. It is found that for a fixed constraint length, a decrease in code

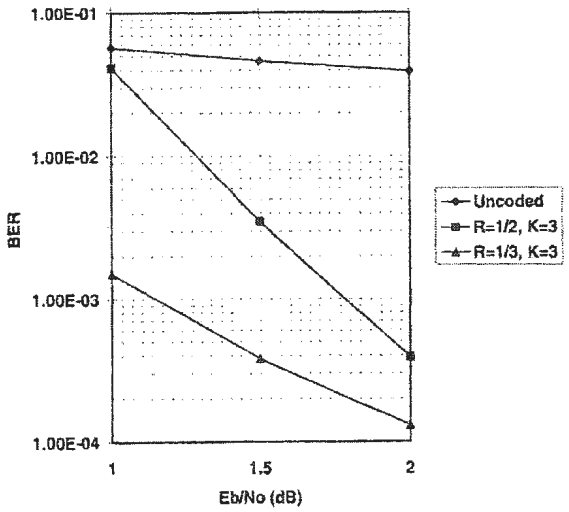


Figure 3.20: Influence of code rate on turbo code performance

rate increases the turbo code BER performance. The code rate of 1/2 is obtained by puncturing the output of the encoder (normally rate 1/3 turbo code).

3.7.6 Influence of Fixed Point Implementation

The simulations in the previous sections were floating point simulations. A software simulation that resembled the hardware architecture using fixed-point variables was also performed. This yielded decoding performance with little degradation as represented in Figure 3.21. Soft inputs with 5-bit quantization and internal data word of 10 bits were sufficient to obtain close to original decoding performance.

S/N(dB)	Iterations	BER(float)	BER(fixed point)
1	0	0.181	0.195
1.5		0.167	0.181
2.0		0.152	0.169
1	1	0.075	0.101
1.5		0.046	0.069
2.0		0.0232	0.041
1	3	0.023	0.052
1.5		0.006	0.0175
2.0		0.0004	0.0015
1	5	0.014	0.038
1.5		0.0022	0.008
2.0		0.00015	0.00061
1	7	0.0077	0.033
1.5		0.0011	0.005
2.0		0.000081	0.000195

Table 3.5: BER vs Signal to Noise ratio for floating point & fixed point implementation

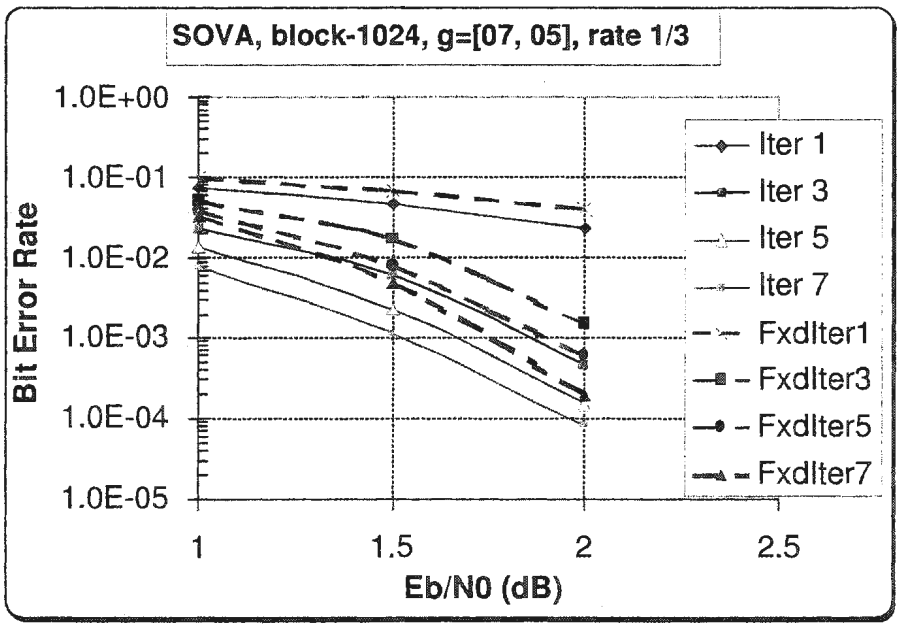


Figure 3.21: Floating point vs Fixed point implementation performance

3.7.7 Summary

This chapter described the details of the software simulation of the turbo coding/decoding system by means of a flow chart and description of the blocks therein. The encoding decoding mechanism for one iteration was illustrated with an example of a 6-bit stream. The generation of noise for AWGN channel was explained. At the end the simulation results along with graphs were presented. The BER converges after some number of iterations. In our case 7 iterations normally led to acceptably low BER. It is seen that the BER decreases sharply after the first iteration but this decrease becomes progressively smaller for the successive iterations. The windowed implementation is attractive from the hardware savings point of view and increased throughput. Simulations were carried out to find the degradations and very little degradation was seen compared to the continuous decoding implementation. Also the concept of *interleaving gain* was seen by decoding frames of different sizes. It was also seen that random interleavers outperform block interleavers. Finally, the fixed point version of the turbo decoder was simulated and found to have a little degradation compared to the floating point version. This was done with input soft values to the decoder set to 5 bits and internal data path with 8 bits. In the next chapter the hardware implementation of the turbo encoder/decoder is presented.

Chapter 4

Hardware Design and Implementation

4.1 Motivation

Turbo codes have became part of the third generation wireless systems standard [1] which offers high data-rate services (up to 2Mbits/s) for Internet and multimedia applications. Since turbo decoders are part of the base band receiver in these mobile systems, power consumption is very crucial and thus low power architectures are in demand. Since complexity is often directly proportional to power consumption, it was decided to use the lower complexity SOVA. Of the two popular turbo decoding algorithms, namely Log-MAP and SOVA, the latter is half as complex as the former but with a 0.5 dB degradation in performance. The main focus of this chapter is the design and VLSI implementation of a low power turbo decoder. An ASIC implementation is targeted, as opposed to realizing the design on an FPGA, as the

main objective is to reduce the power consumption and area. The ASIC is targeted to be implemented using 0.18μm CMOS technology.

4.2 Design Flow

The first step was to identify the word size for the various signals in the encoder and decoder. This was done using software simulations as explained in Chapter 3. The next step was the identification of various components required for the encoder and decoder. The combinational and sequential blocks were identified and the data path

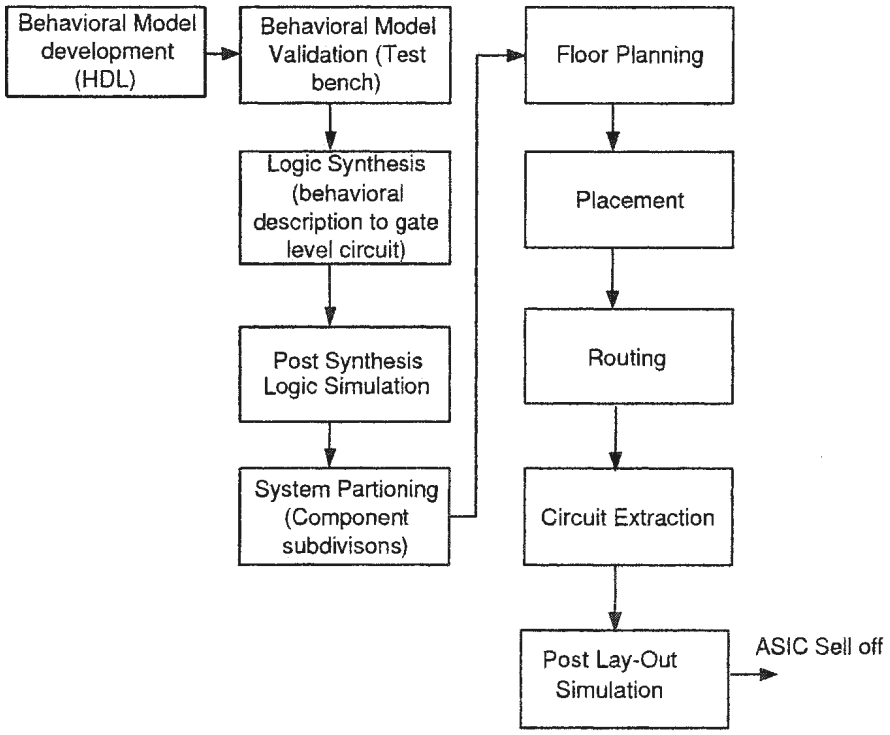


Figure 4.1: ASIC design flow

and control units constructed structurally, employing the paradigm *design top-down*

and implement bottom-up. The components were coded in VHDL and were functionally simulated using Synopsys VHDL System Simulator (VSS). The simulated working design was then synthesized using Synopsys Design Compiler. A general flow diagram for the ASIC design process is shown in Figure 4.1 [40]. This flow diagram was used as the reference for our design. Only the first four steps in this flow have been attempted in this thesis.

4.3 Hardware Implementation of the Encoder

The encoder implementation shown in Figure 4.2 is straightforward and has much lower power requirements compared to the decoder. It generates a rate 1/3 turbo code with generator g [5,7] and uses the encoder state diagram of Figure 4.3. The individual RSC encoders are implemented by means of T flipflops and XOR gates. An XOR-mux combination with taps from the flipflops generates the tail bits. The random interleaver needed to shuffle the input data before feeding into the second encoder is implemented by a linear feedback shift register (LFSR) in combination with a multiplexer. The position of the interleaver output is given by the LFSR count. As the performance improves with a larger interleaver, a large interleaver is usually employed and so this is the largest hardware block in the encoder. If a fixed interleaver is implemented using simple fixed wires connecting the input to the output, the size (in terms of number of gates) can be reduced.

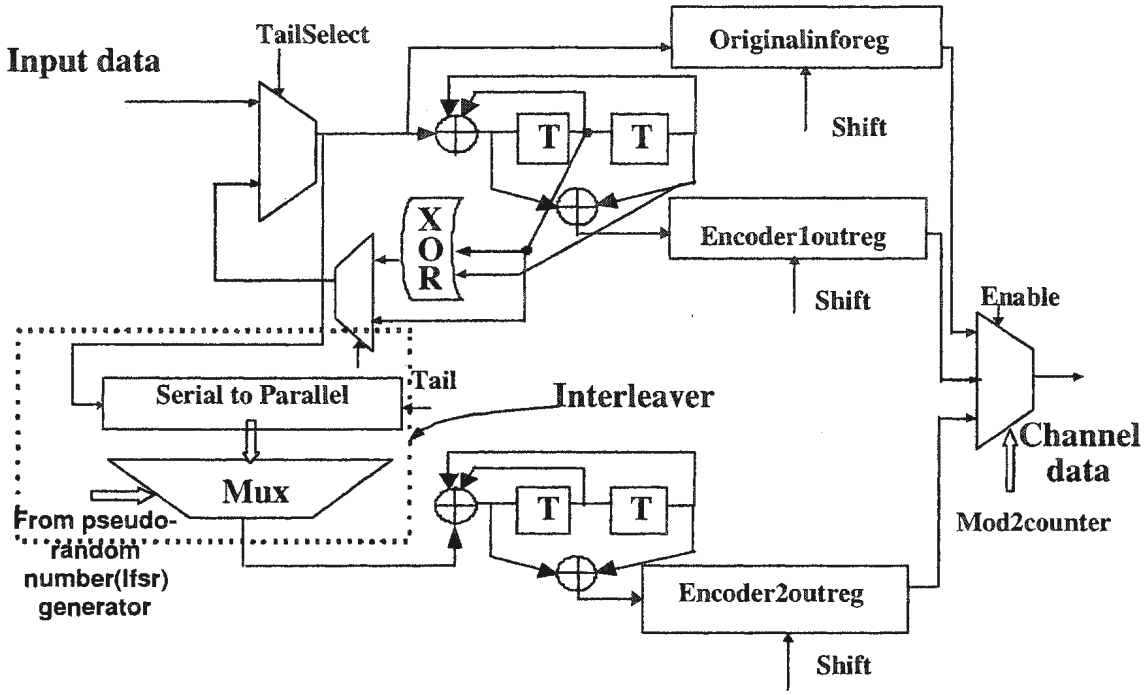


Figure 4.2: Encoder architecture

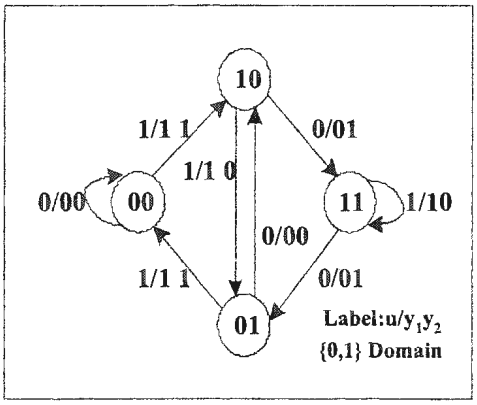


Figure 4.3: State diagram for the 2 bit encoder1

4.4 Hardware Implementation of the Decoder

The major components of the decoder are:

1. The Input Buffer: These are circular shift registers and are required for the storage of the incoming frame of symbols to be decoded.

2. The SOVA unit: It is the main unit of the turbo decoder and implements the Viterbi algorithm and the soft-in soft-out version of the Viterbi algorithm (SOVA) to generate the reliability values.

3. Adder/Subtractor unit: This unit subtracts the inputs from the reliability values generated by the SOVA unit. This has the effect of decorrelating the reliability values from the inputs. The outputs from this unit are called extrinsic values.

4. Interleaver and Deinterleaver unit: The extrinsic values from Decoder1 are interleaved, while the outputs of Decoder2 are deinterleaved.

5. The control unit: Generation of the control signals that activate/deactivate the above units is done here.

The components of the turbo decoder are shown in Figure 4.4.

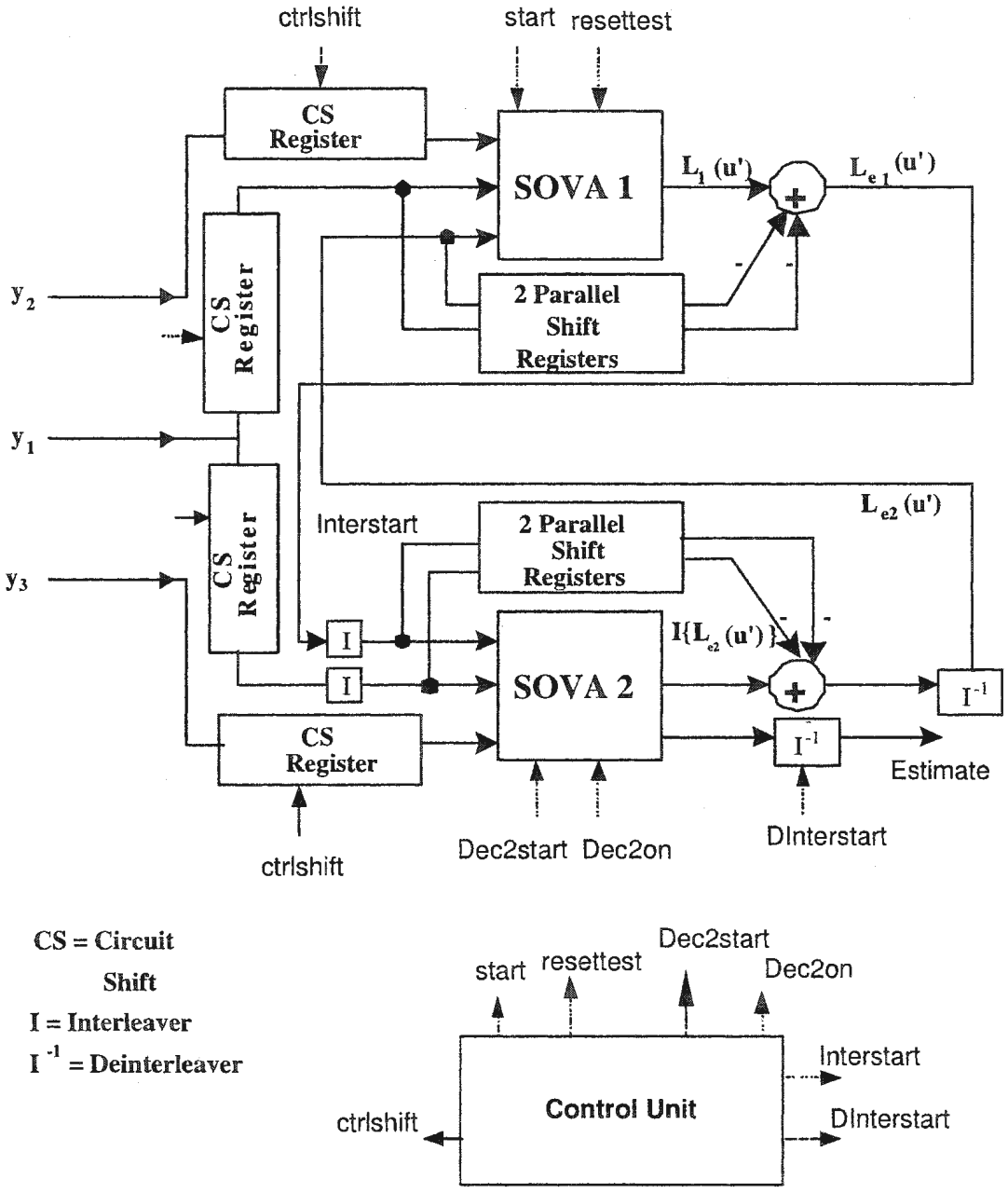


Figure 4.4: Decoder architecture

4.5 The Input Buffer

The serial soft inputs from the channel are stored in circular shift registers. The circular shift register is needed in order to reuse a set of values after being used for Decoder1 processing. Moreover they are needed to store the input values for the successive iterations. In total, four circular shift registers are required, as shown in Figure 4.4.

4.6 The SOVA unit

The SOVA block consists of the VA block and Reliability update unit. The components of the SOVA block are as follows:

1. The Branch metric computing unit (VA block)
2. The Add Compare Select unit (VA block)
3. Path management and storage unit of hard values (VA block)
4. Path management and storage unit of soft values
5. Soft values update unit

4.6.1 The Viterbi Decoder(VA block)

The Viterbi (VA) block) is shown in Figure 4.5.

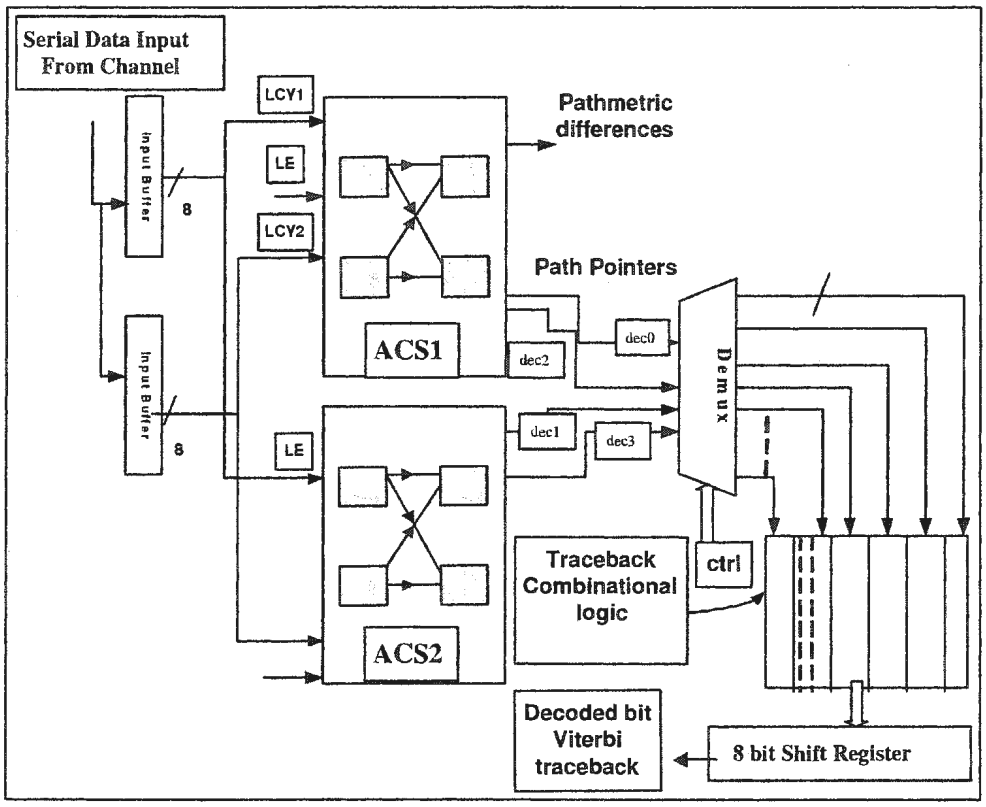


Figure 4.5: Viterbi algorithm architecture

The Branch Metric Unit

For each state/input combination, the branch metric bm can be computed from the *received symbols* Lcy_1, Lcy_2 and the *estimated* symbols u, x by means of exclusive-or gates (for hard decision decoding) or by means of fixed point multipliers and adders for soft decision decoding. In the SOVA decoder the modified VA metric is used for

the branch metric computation as explained in Chapter 3.

The Add Compare Select unit

Add refers to the addition of branch metrics from the two incoming branches to the path metrics of their origin states. *Compare* refers to the comparison of the results of the previous operation, i.e., the comparison of the two resulting path metrics for each state and *Select* is used to indicate the selection of the best path. The best path metric and state history information along with the competing state histories are outputted by this unit for each state in a stage. Only the path metric differences need to be stored instead of the absolute path metrics, thus reducing storage. The trellis which is used for the decoding of the overall received frame of symbols is composed of many such ACS units. Since each stage of the trellis has a butterfly structure, it can be broken into identical elements containing a pair of origin and destination states and four interconnecting branches as shown in Figure 4.6.

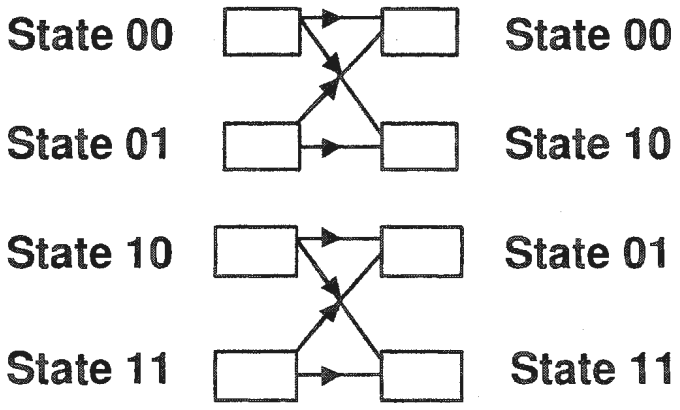


Figure 4.6: Butterfly structure of trellis

Since the entire trellis consists of multiple instantiations of the same simple element, we can design a single circuit that adds the branch metric to the previous state value for both paths entering a node and then compares the two paths, selecting the maximum path. Figure 4.7 shows the structure of the ACS unit used in our implementation. The parameter $pm0$ refers to path metric of state 0 and $bm0$ refers to branch metric (SOVA metric distance from received code symbols) of the branch leading to next state. The parameter pm at the output is the maximum of $((pm0 + bm0), (pm1 + bm1))$.

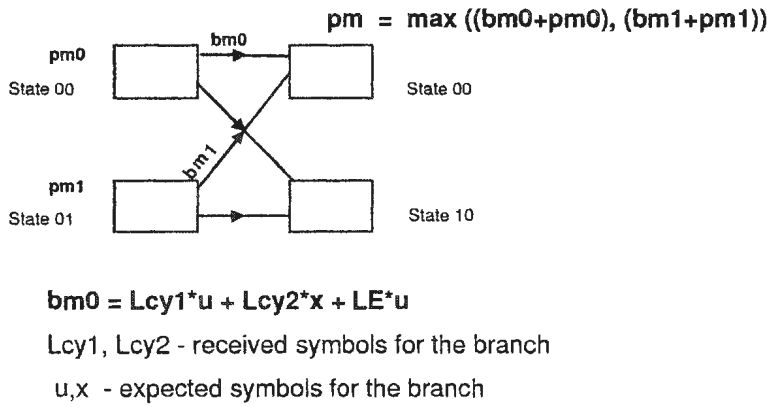


Figure 4.7: ACS unit structure showing the branch metrics (bm) and the path metric (pm)

In our design, we combined the branch metric computation and Add Compare Select unit into one set of operations. For a 4-state encoder one stage of trellis can be represented as two butterfly structures and corresponds to a 2-bit encoder, each state has two incoming paths as shown in Figure 4.6.

Each butterfly element outputs the path metrics for two states. So two butterfly

elements were needed for the implementation. There are two possible implementations of the butterfly element. One approach is to use a single butterfly module and call it twice in each stage. Alternately we could dedicate two butterfly elements for each stage. This approach uses more area for the computing elements but has a smaller control circuit and, overall a lower power requirement. The latter approach was used and the symbolic view is shown in Figure 4.8. The soft values (*LCY1,LCY2 and LE*)[1] represent that inputs to the ACS unit. The branch metrics (*BMstraight, BMcross*), path metric (*BMstraight, BMcross*) and the state history (*dec0, dec2*) are all computed at intervals whenever the signal to start computation (*compute*) goes high.

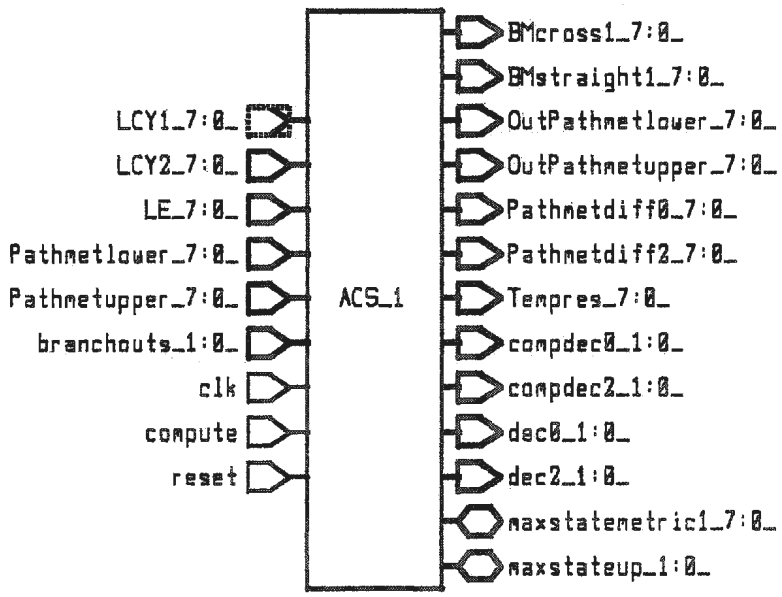


Figure 4.8: ACS unit symbolic view

[1]The italicised name within brackets correspond to pin names in the symbolic view

The Path Management and Storage Unit

The ACS operations are done recursively for each stage until a window is completed. In the simulation a sliding window technique was adopted to reduce hardware. Figure 4.9 shows the sliding window process. Here the modified SOVA sliding window with reduced SOVA tracebacks compared to VA traceback is shown although the implementation is done with same length for the VA and SOVA tracebacks due to time constraints. The hardware implementation was targeted for a smaller frame

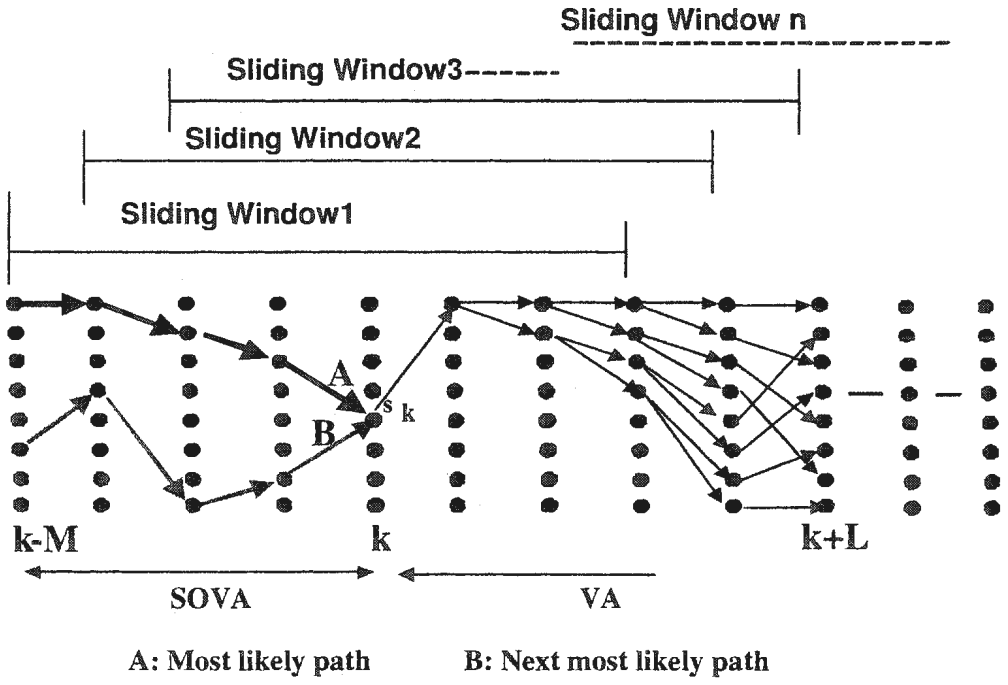


Figure 4.9: Sliding window architecture

size of 34 bits and hence a smaller window length is sufficient. A window length of 8 for the Viterbi and SOVA traceback is used in the implementation in order

to satisfy the criteria that the traceback length is $\approx 3 * K$. This means that at the end of the window length, the VA and SOVA tracebacks are done and the first decoded bit along with its reliability value are released. Then the window slides one bit position to consider the next 8 stages of the trellis. This process goes on until the last window is reached. In the last window a shrinking window approach is used until the last bit is decoded. In the shrinking window a reduced traceback path is used. The outputs of the ACS unit are managed in the path management and storage unit. Two approaches can be used to record survivor branches: register-exchange and traceback [4].

The register-exchange approach assigns a register to each state. The register records the decoded output sequence along the path, starting from the initial state to the final state. The register for a given node at a given time contains the information bits associated with the surviving partial path that terminates at that node. This approach eliminates the need to traceback, since the register of the final state contains the decoded output sequence. The number of bits that a register is capable of storing is a function of the decoding depth G. The decoder may be designed such that the information bits associated with a surviving branch at time t can be released when the decoder begins operation on the branches at time $t+G$. Since the registers must be capable of sending and receiving strings of bits to and from two other registers and also (in a fast decoder) all of the exchanging must take place simultaneously, it leads to a hardware intensive implementation. Hence, the approach may offer a

high-speed operation, but it is not power efficient due to the need to copy all the registers in a stage to the next stage. The register exchange approach is shown in Figure 4.10.

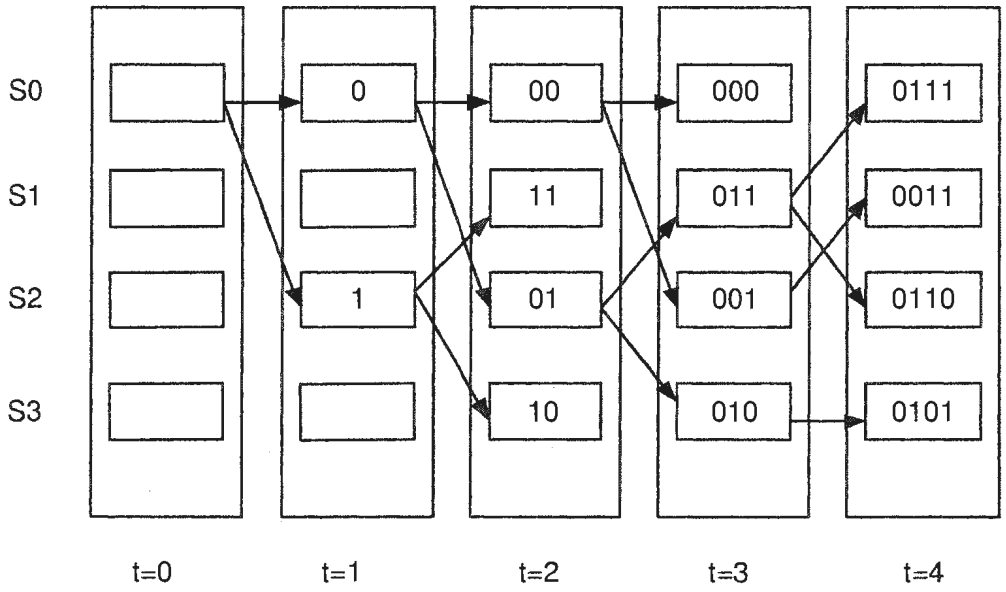


Figure 4.10: Register exchange method

The other approach, called traceback, records the survivor branch of each state. It is possible to traceback the survivor path provided the survivor branch of each state is known. A flip-flop is assigned to each state to store the survivor branch and the flip-flop records 1 (0) if the survivor branch is the upper (lower) path. Normally at the end of a window of trellis building, the state with the largest path metric is selected as the survivor path start state and traceback done through the stored survivor branches to get the decoded output. Concatenation of decoded output bits in reverse order of time forms the maximum likelihood estimate of the transmitted

sequence. The traceback approach is shown in Figure 4.11.

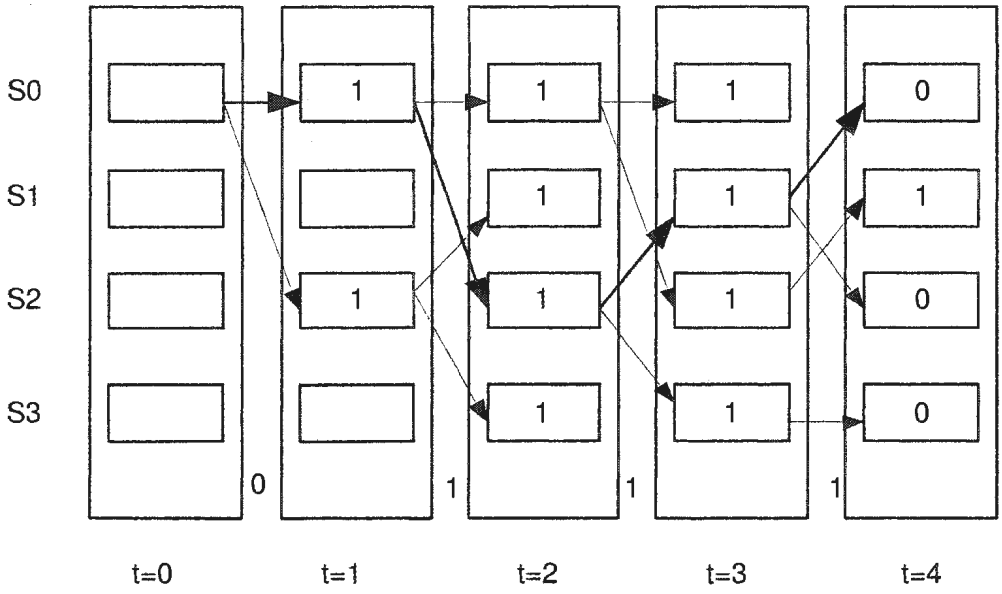


Figure 4.11: Traceback scheme

The power dissipation depends mainly on the switching of signal values and on the number of iterations of decoding. The traceback technique, which requires less switching of signal values, was used, instead of the register-exchange method for the path management unit in the VA and SOVA block implementation. The structure of the implemented survivor path storage module is shown in Figure 4.5. Since there are only 4 states, 2 bits are needed to represent each state. An 8-bit register is thus sufficient to store the state history information at every stage. The 5-bit counter keeps track of the current stage. The survivor path information of the 4 states, which is generated by the butterfly block, represented as *ACS1* and *ACS2* is passed to the register of the stage through the demultiplexer.

The symbolic view of the Viterbi traceback unit is shown in Figure 4.12. As the

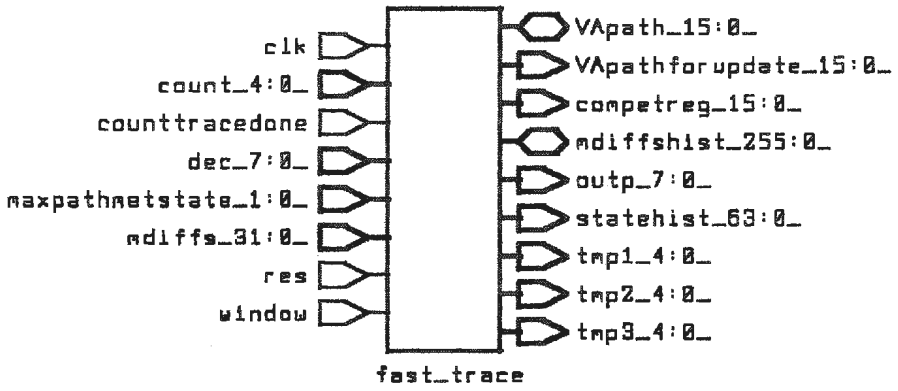


Figure 4.12: Traceback unit symbolic view

count *(count)* decrements from 7 to 0, the state history information (previous state pointers) *(dec)* are stored in registers 8 to 1 respectively. Then starting from the state with the highest metric value *(maxpathmetstate)* the traceback is done through the trellis. In case of SOVA Decoder1 the start state in the last window is the zero state and intermediate windows have maximum path metric states as the start state. This is achieved by keeping track of the window number *(window)* and appropriate multiplexing circuit.

4.6.2 Path Metric Difference Generation

The *ACS1* and *ACS2* units, in addition to generating path metrics and selecting the best path state, also output the difference between the survivor and competing path at every state. This is called the path metric difference. Since we use 8 bits to

represent the path metric differences, we have 32 bits at every stage corresponding to the 4 states. These are stored in 8 column shift registers of 32 bits. In order to calculate the reliability values, a competing traceback is done from every stage of the window through the competing branch of the ML state. The path metric differences of the ML state are also needed for the same operation. The scheme is shown in Figure 4.13. The symbolic view of the interface unit is shown in Figure 4.14.

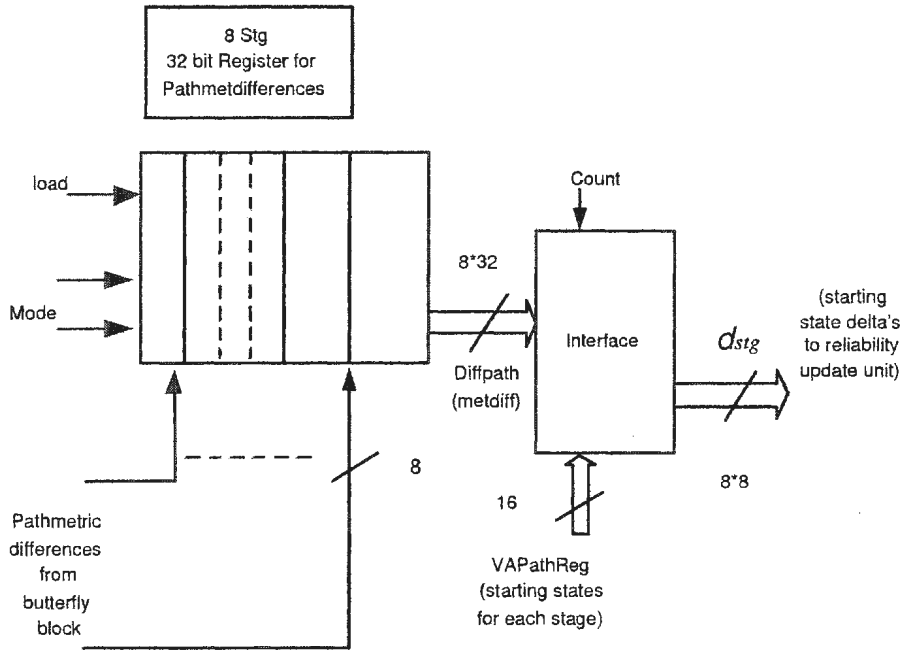


Figure 4.13: Generation of path metric differences for ML states

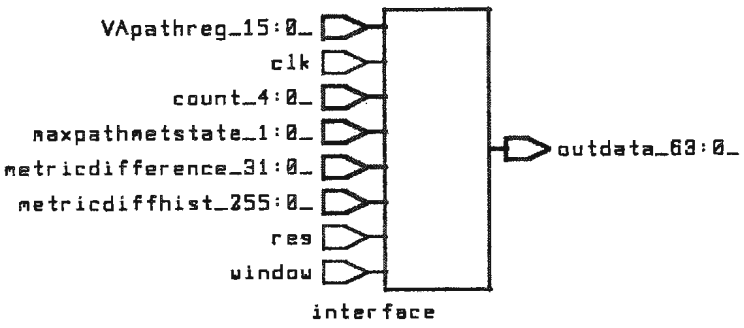


Figure 4.14: Symbolic view of the interface unit

4.6.3 The SOVA Multiple Traceback Unit

Figure 4.15 shows the multiple tracebacks through competing paths for a window size of eight. Here the goal is to release the estimated bit arising due to tracing back through a competing path to the VA path at every stage (if the ML approaches a state with a '1' input, the competitor traces back the '0' path). The estimated bits for the window that results from this operation are stored in a register and passed on to the Soft Value update unit. The register values (SOVA TB bit in Figure 4.15) are compared to the VA bit (Viterbi decoded bit) and at the stages where the competing traceback resulted in a different decision from the VA estimate, a comparison is done between the path metric differences and an initial high reliability value (InitRel). The smallest of these comparison results is taken as the updated soft value for the VA estimate bit.

4.6.4 Soft Value Update Unit

The Soft Value update unit is a part of the multiple trace back unit and consists of eight 8-bit signed magnitude comparators as shown in Figure 4.15. The inputs to each comparator are the pathmetric differences and the initial high reliability value. The SOVA controller unit generates the relevant activation signals to select the comparators during the SOVA traceback. The Viterbi decoded hard decision bit is the input to the Soft Value calculation unit.

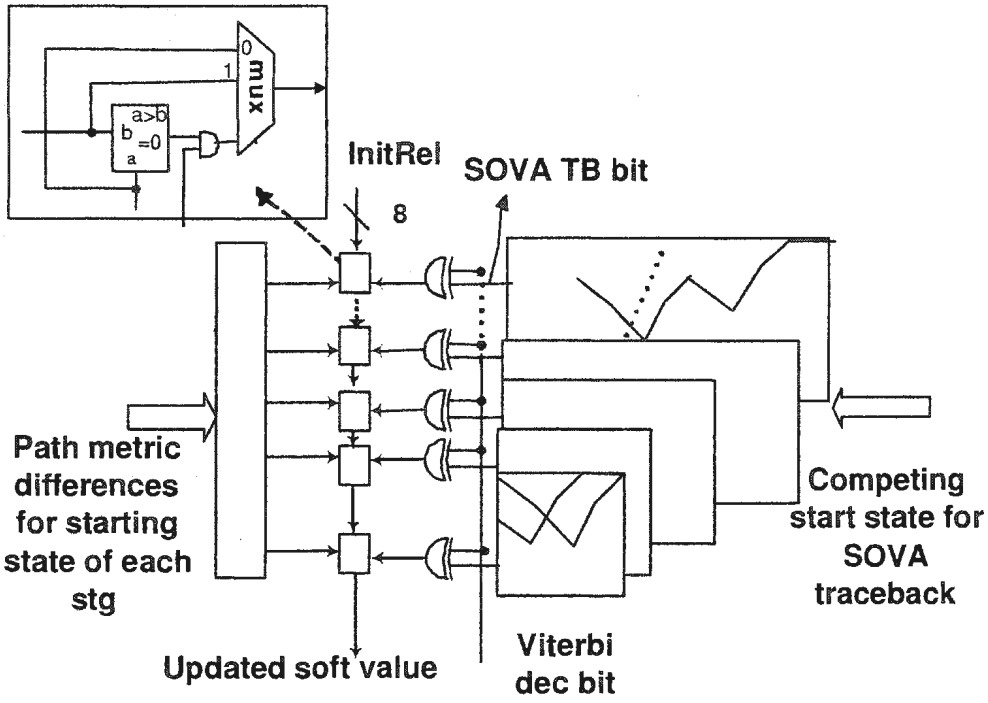


Figure 4.15: Soft value update

4.6.5 The Complete Picture

The complete picture of the SOVA block is shown in Figure 4.16. The final output of the SOVA block is the updated reliability values. The input buffers and the ACS units are not shown in the figure. The updated reliability values are fed into a subtractor unit which has the effect of decorrelating the input as explained before. The output is then the extrinsic values and they are interleaved before being provided as inputs to Decoder2.

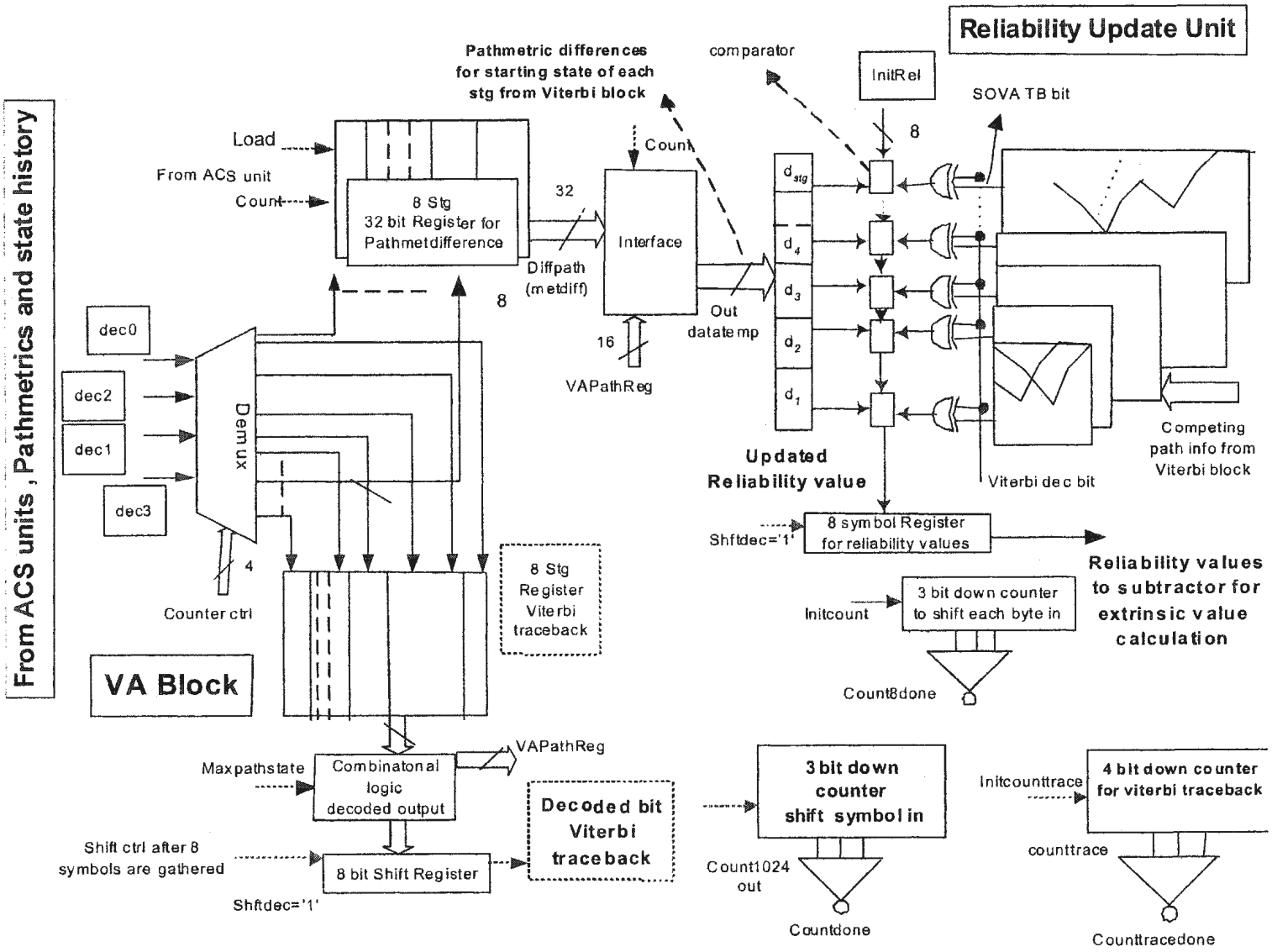


Figure 4.16: SOVA unit data path

4.6.6 Interleaver and Deinterleaver Unit

Since the turbo decoding is done using blocks or frames of data, the size of the interleaver determines the performance gain. The larger the interleaver the higher the gain. Usually random interleavers of 512 or 1024 symbols (A symbol is represented by 8 bits) are used. In hardware, large amounts of memory or a long shift register would be needed to store the interleaved and deinterleaved values. In total, 2 interleavers and 2 deinterleavers are required. Both the interleavers are symbol interleavers while the deinterleaver at the output of the second decoder that gives the final estimate of transmitted sequence is a bit deinterleaver. A behavioral model of a 34 symbol (272 bit) random interleaver, deinterleaver and 34 bit final estimate deinterleaver has been implemented for the decoder.

4.7 Decoder Control Unit

The decoder is controlled by means of three units, namely, the top level controller, the input data collection controller and the SOVA unit controller.

4.7.1 Top Level Controller

The top level turbo decoder controller produces the control signals that activate and deactivate the serial iterative decoders, Decoder1 and Decoder2. Figure 4.17 shows the state diagram and the timing diagram. The controller makes sure that in the first

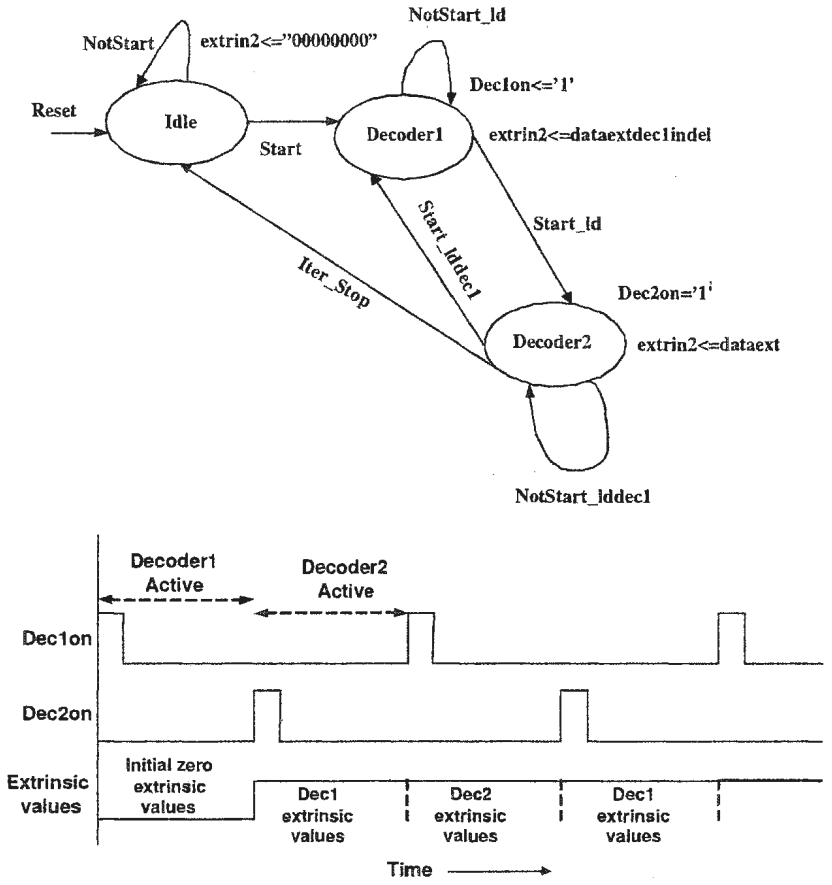


Figure 4.17: Top level controller for turbo decoder

iteration the extrinsic values for Decoder1 are initialized to zero and in subsequent iterations the decoders get the appropriate feedback extrinsic values. This step along with the timing of start signals *Dec1on, Dec2on* for the two decoders, is shown in the timing diagram of Figure 4.17. The start signals for interleaving and deinterleaving are also generated by this controller. At the end of the required number of iterations, the turbo decoder is brought into the idle state.

4.7.2 Input Buffer Controller

The buffer controller state diagram is shown in Figure 4.18. The controller generates the shift, hold and load signals for the input circular shift registers and other shift registers. During the shifting states (*Shifting, Shifting2 and Shifting3*) the received data from the channel are shifted into the input buffers. Once all the symbols are shifted in, the controller moves to the *Idle* state.

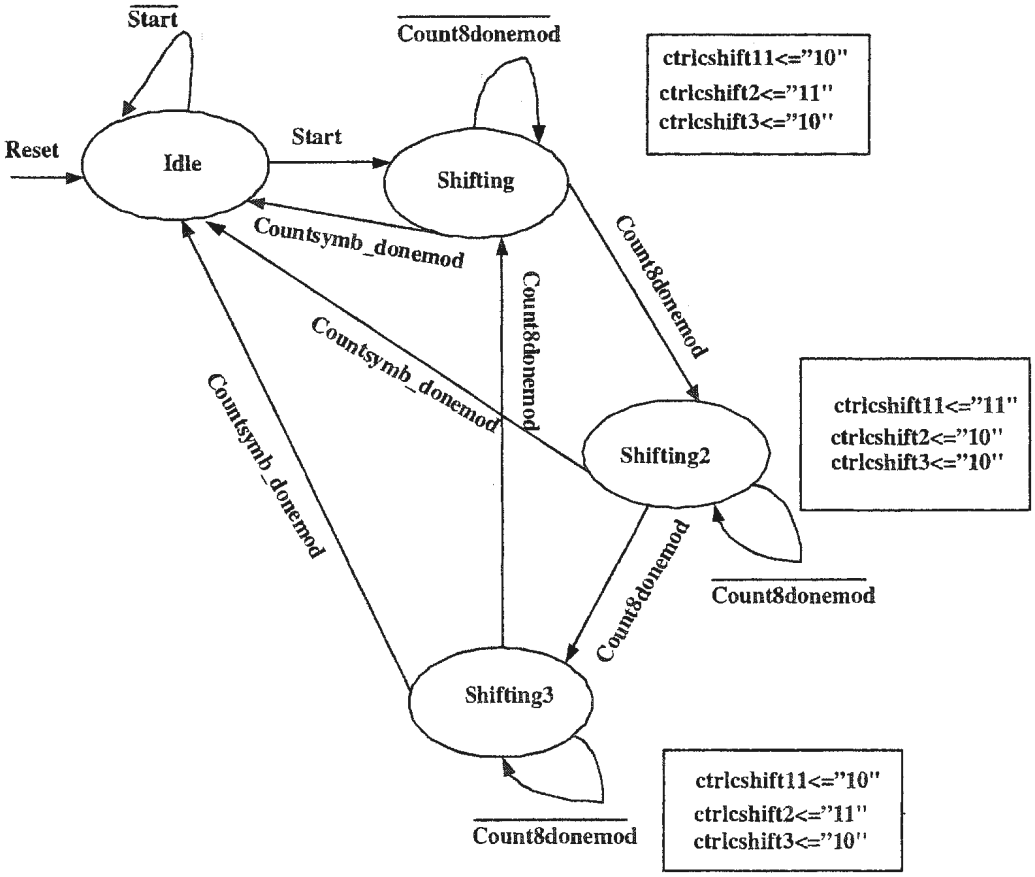


Figure 4.18: Input buffer controller for turbo decoder

4.7.3 SOVA Unit Controller

The main functions of this controller are to produce the *compute* signals for the ACS computation, to determine the start and end of VA, SOVA traceback and to release the VA and SOVA bits by generating the appropriate control signals. In the *wait*

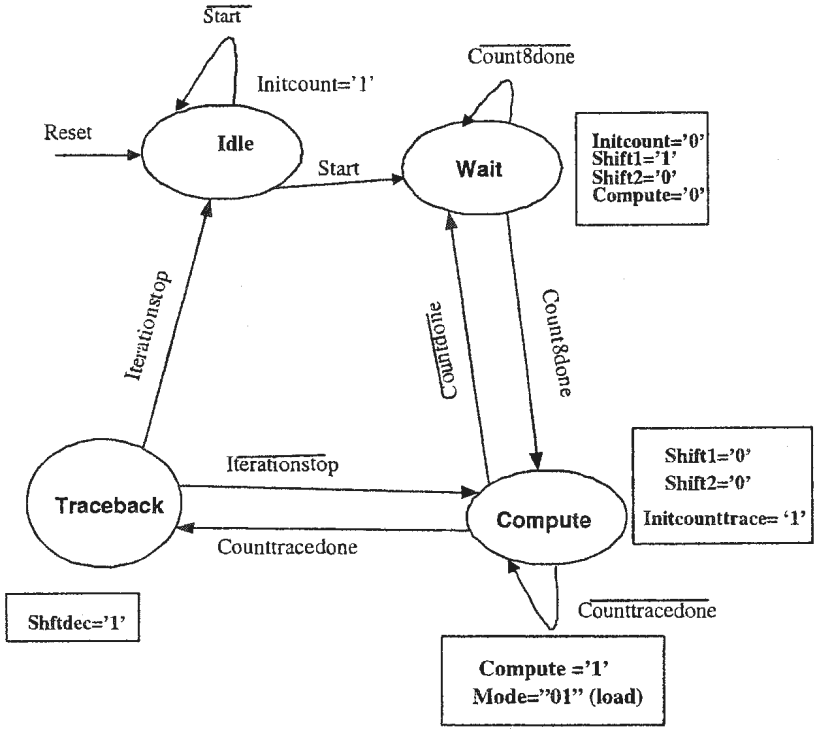


Figure 4.19: Controller for SOVA unit

state, the SOVA unit waits for two symbols to be shifted in for ACS computation. As soon as these symbols are available the state changes to the *compute* state, during which time the ACS computation is done. The traceback counter is initialized and the state changes to the *traceback* state. In this state the *shiftdec* signal activates the VA bit release and triggers the reliability update unit.

4.8 Low Power Modifications

4.8.1 Gating the Clock

The ACS unit outputs the path metric differences and state history information. The state history information (8 bits) corresponding to the 4 states are stored in 8-bit registers. As the symbols come in, the ACS unit takes two symbols at a time and along with extrinsic values outputs a set of state history information to the k^{th} register depending upon the count value. Only one register is accessed at any time and the clocks to the other registers can be disabled. This will reduce power consumption [41]. Figure 4.20 shows the circuit.

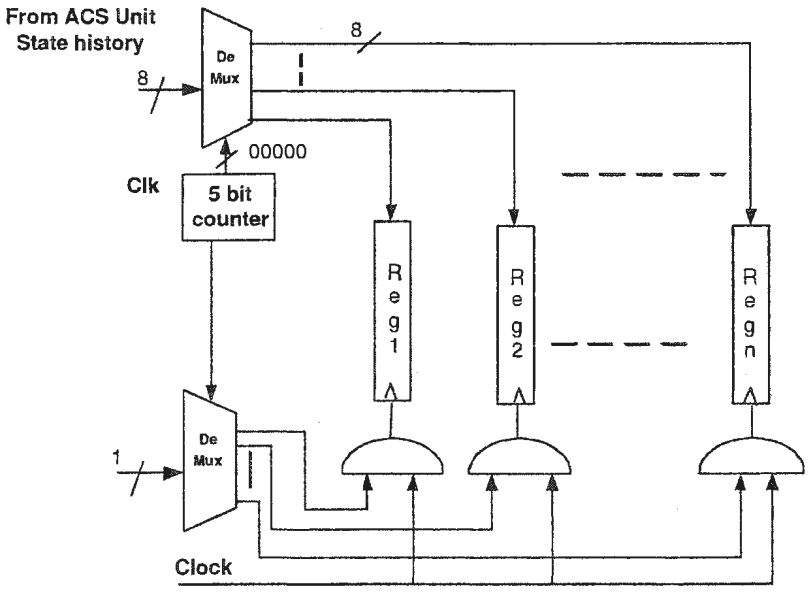


Figure 4.20: Clock gating for low power

4.8.2 Disabling Unused Registers

Once the registers are filled with state history values for a window then the values of these registers are transferred to the traceback block. Then this block traces to the starting state according to the state history commencing from the most likely state. So the register values are useful to this block only at the end of a window and, in the interim, need not be transferred to the traceback block. In other words, for the 8 stage window, the register value transfers to the traceback block are enabled once in 8 clock cycles. This will reduce unnecessary switching of register values and reduce power dissipation.

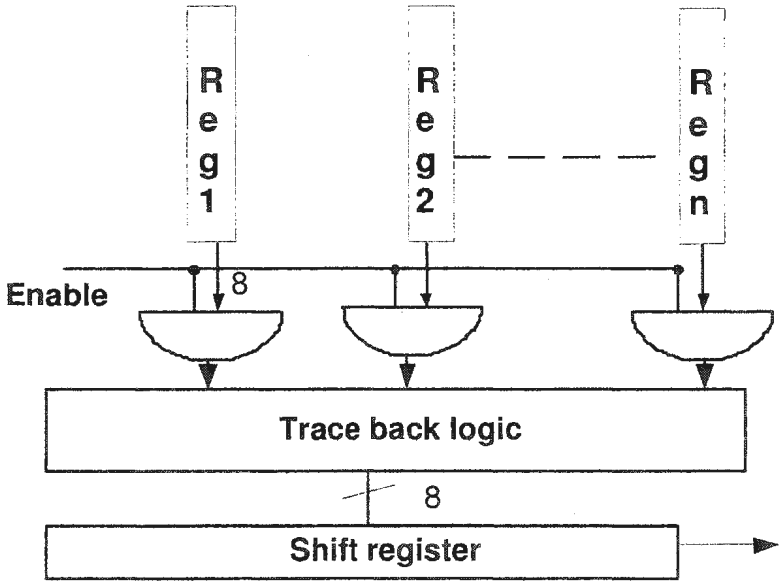


Figure 4.21: Disabling unused registers

4.8.3 Implementation of a Stopping Criterion

Quite often, the received frame is decoded correctly within a few iterations. Stopping criterion determine the point at which the decoding can be stopped. This results in increased throughput and reduced power consumption. A recently proposed [1] technique called SDR (explained in Section 2.5) has been implemented. It looks at the number of sign changes between the extrinsic values of the two component decoders in a turbo decoder and terminates the decoding if this number is negligible as determined by Equation 4.1.

$$D = \begin{cases} \geq q_d \times N & \text{continue iteration} \\ \leq q_d \times N & \text{stop iteration} \end{cases} \tag{4.1}$$

where D is the limit to stop or continue the iterations, N is the frame size and q_d is the sign difference ratio and should be in between 0.001 and 0.01 for non degradable performance. The hardware implementation requires an N bit comparator and a $log_2 N/100$ (for the case of $q_d = 0.01$) counter. In our short frame decoder the SDR implementation was tested with $q_d = 0.1$ and showed some degradation. However the SDR criterion implementation for short frame decoders will have substantial power reduction due to early stopping of the iterative decoding. The implementation scheme is shown in Figure 4.22.

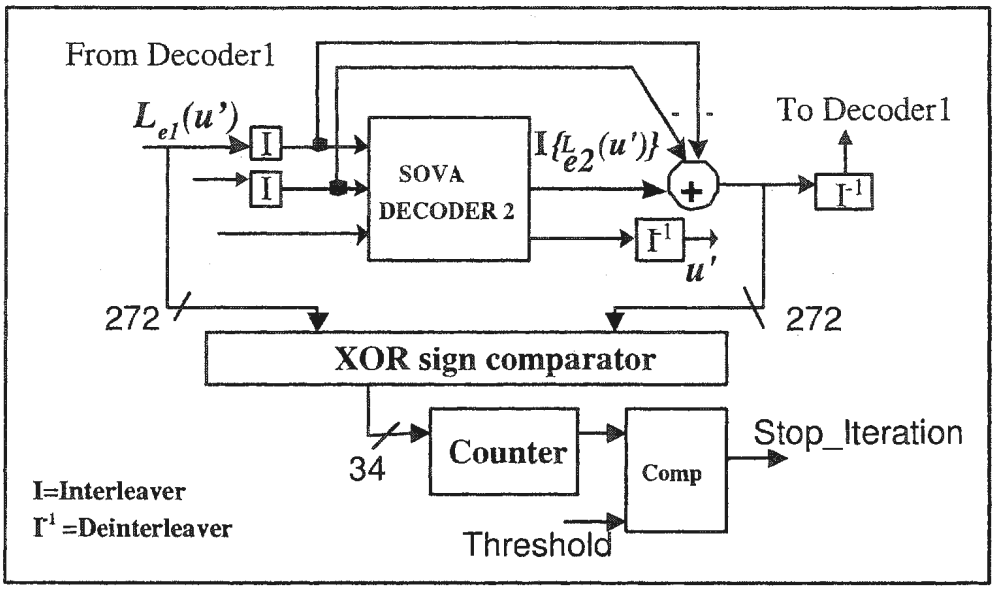


Figure 4.22: Early termination of iterations

4.9 Testing of the Design

The Turbo decoder implementation was tested at different levels of the design hierarchy. The VHDLAN and VHDLDBX tools of Synopsys were used for the compilation and simulation of the individual components and the whole system. The DESIGN_ANALYZER tool of Synopys was used for the synthesis. Simulation waveforms before synthesis are shown in Appendix A. The simulation results indicate the correct functionality of the turbo decoder system. The simulated turbo decoder hardware outputs were tested using the software outputs as the reference. The test inputs to the decoder were read from a file and several frames were tested individually and found to match with the software results. The BER calculation was however not done in the hardware. One sample of the simulation waveforms before

the system was synthesized is presented in Figure 4.23. The parameters *Data1input* and *Data2input* along with *LE* correspond to the input values to Decoder1. The path metric values of the different stages are shown along with Viterbi decoded bits *Qout1*. The waveforms also show the start states MAXSTATETRACE for traceback and the survivor path OUTP for each window.

4.10 Performance and Synthesis Summary

For the simulation a clock frequency of 125 MHz was used. It is seen from the waveforms of Figure 4.23 that the Viterbi decoding has a latency of 1528 ns and the Viterbi decoded bit is released every 128 ns thereafter or the bit rate is approximately 8 Mbits/s. The updated reliability values are output in 16 clock cycles after the release of the Viterbi bit. For a 34 bit frame, iteration1 decoded outputs have a latency of 11968 ns. For 5 iterations the total time taken is around 60000 ns. Therefore the frame decoding rate is approximately 16 KHz and corresponds to a turbo decoding rate of .544 Mbits/s.

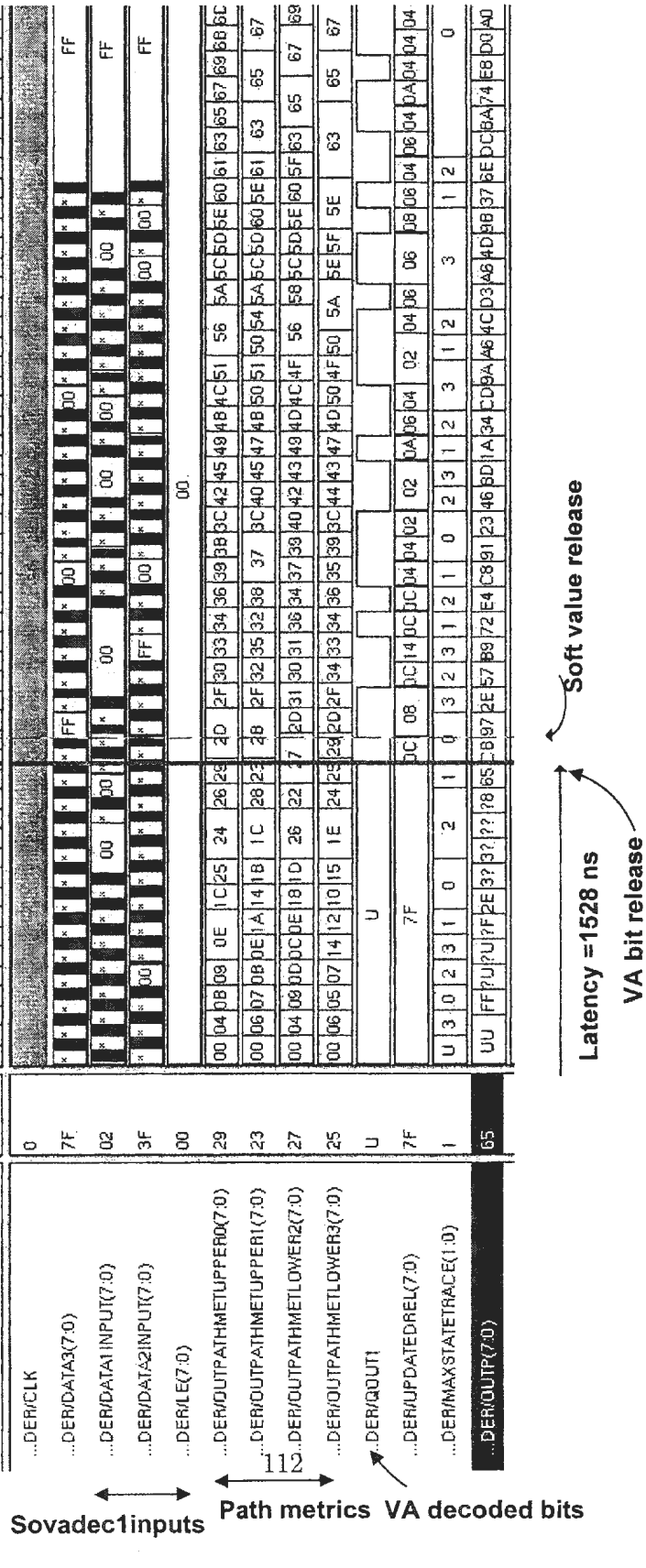


Figure 4.23: SOVA unit functionality

Component	CellArea	Timing(ns)		Power Consumption(mwatt)
unireg	748.06	7.43	MET	.187
mux8	195.2	.31	MET	.314
acs_1	18762.71	37.79	MET	.893
acs_2	18762.71	37.79	MET	.893
compare	1341.64	.46	MET	2.43
mux	48.78	.30	MET	.0798
muxsign	780.6	.30	MET	1.25
uniregsign	2992.256	.35	MET	3.10
dcounter34	914.75	.38	MET	1.22
dcounter8	304.91	.38	MET	.430
dcountertrace	556.98	.38	MET	.793
VAtrace	45258.14	35.74	MET	1.92
interface	27759.82	48.70	MET	1.13
shiftreg	95.58	49.50	MET	.004
flipflop	85.37	49.49	MET	.004
sova_trace	16388.43	38.32	MET	.515
Relupdate	4651.03	20.93	MET	.191
Stage1_ctrl	1646.56	0.48	MET	.025
vitstg_1	**233812.43**	**0.48**	**MET**	**3.93**
Interleaver	34281.25	0.37	MET	37.37
Deinterleaversym	34281.25	0.37	MET	37.35
Deinterleaverbit1	19767.05	0.37	MET	24.82
Circular Shiftreg	1512.40	0.48	MET	2.97
Controls	5168.56	0.48	MET	3.93
Subtractor	1536	0.33	MET	3.00

Table 4.1: Summary of area, timing and power report for the components of the turbo decoder

In Table 4.1, we summarize the synthesis report of the components of the turbo decoder block. Three important parameters are included in the table: the area, timing and power consumption. The timing data indicates whether the slack (difference between required data arrival time between input and output and the actual delay between input and output)for each component has been met and is indicated in table

as "MET". It is seen that the SOVA unit *vitstg_1* has a power dissipation of 3.93 milliwatts. This is the static power dissipation of the unit and involves only the clock switching factor. To determine dynamic power dissipation that will evaluate the effect of switching of signals with different input patterns, a post synthesis simulation has to be carried out and the results of switching factors of the different nodes during this process stored in a file. This file is then to be utilised by the *Power compiler* tool from Synopsys to give the power estimate. The synthesis of the SOVA block along with the synthesis of the interleaver, deinterleaver and one complete section of the turbo decoder was carried out. Since the synthesis of the overall turbo decoder file required changes in the constructs of certain sections of the VHDL code and was time consuming, the post synthesis simulation was not undertaken. The encoder functionality was simulated but not synthesized as the encoder hardware was quite clear and non complex.

The gate counts of the various subunits of the SOVA block are presented in Table 4.2. It is seen that the soft values management unit has the maximum gate count. This is as expected because of the parallel multiple traceback implementation.
The gate counts of the various subunits of the decoder block are presented in Table 4.3. The gate count for a 34 symbol decoder is around 45585 gates as shown in the table, the gate count for a 500 symbol commercial decoder can be estimated to be around 15 times this, and will be around 700000 gates. The interleaver and deinterleaver indeed are the biggest individual blocks in terms of gate count.

Unit	no.of gates
ACS and Control	3066
Hardvalues mgmt.	3734
Softvalues mgmt.	9571
Delay	868
Total no. of gates	17239

Table 4.2: Gate count for the SOVA block

Unit	no.of gates
SOVA blocks	34478
Interleavers.	5624
Deinterleavers.	4434
Controls	424
Input buffers and subtractors	625
Total no. of gates	45585

Table 4.3: Gate count estimate for the turbo decoder block

4.11 Conclusions

This chapter presented the hardware implementation of the encoder and decoder for a turbo coding/decoding system. The design flow recommended by CMC was adopted. The various units of the decoder were analyzed in detail and implemented with the objective of coming up with a low power architecture for the decoder. Three techniques namely clock gating, disabling unused registers and early stopping criterion have been implemented. The SOVA traceback unit was built using a hardware-intensive parallel realization in order to study the speed achievable in this complex step. A 34 symbol decoder was implemented and a window length of eight

was considered for both the VA and SOVA traceback. The implementation was done using an internal data path length of 8 bits and soft input values of 5 bits. Decoding is done bit-by-bit and a new window is considered after each bit. The performance of the hardware is presented in the form of area, timing and power reports. The power savings achieved by the modifications presented earlier remain to be determined. The Viterbi decoding speed attained was 8 Mbits/s and the turbo decoding speed attained was .544 Mbits/s. The gate counts of the main units are also presented.

Chapter 5

Conclusions and Future Work

This chapter presents the contributions of this thesis and the areas of future work related to this research.

5.1 Conclusions

Turbo codes were invented during the last decade and their performance evaluation under different channel conditions and under the influence of varying the code parameters is still being researched. Moreover with their inclusion as part of the third generation cellular standard, efficient and low power hardware architectures are in high demand. The following points describe the contributions of this research.

- **Performance analysis of the turbo encoding/decoding system**

  The simulation of the encoder/decoder was done in a AWGN channel and the decoding performance studied by varying the iteration number, frame size and

interleaver type under different signal to noise ratios. As the interleaver plays an important role in decoding performance, two kinds of interleavers - random and block, were used, and it was found that the random interleaver gives the better performance. The influence of fixed point representation on the decoder performance was also simulated. This formed the basis of a test system to compare the hardware results. A path metric normalization was performed to take care of path metric value overflow.

- **Employing a low power Viterbi decoding architecture as the basic block**

  In keeping with our goal of low power architecture, we investigated the components of the turbo decoder from algorithmic and implementation point of view. This resulted in the employment of the following techniques:

  - Low power consuming traceback technique for the Viterbi Algorithm.
  - Clock gating of the path storage registers.
  - Disabling controls to unused registers.
  - Implementation of the SDR iteration stopping criterion.

- **Employing a parallel traceback scheme for SOVA**

  The traceback method, in comparison to the register exchange method is slower although it has lower power requirements. The single traceback unit could

be used multiple times from the hardware and power savings point of view. However the overall system speed would be affected. In order to improve the speed of the whole decoder, a hardware intensive parallel traceback scheme for the SOVA tracebacks was employed.

- **Implementation of an encoder and a sliding window turbo decoder**

  An encoder was implemented in hardware. The encoder is much smaller in area and has lower power requirements compared to the decoder. A sliding window finite length decoder is comparable in performance to the unlimited length decoding as in the case of convolutional decoding. The sliding window technique is also efficient for hardware implementation and offers lower storage needs leading to lower area and power requirements.

5.2 Future Work

This section outlines some possible improvements on turbo decoder implementation that merit future investigation.

5.2.1 Multiple Bit Release for SOVA

In the present research an attempt was made to release multiple bits at the same time instead of bit by bit decoding. This was successful with the Viterbi algorithm but did not yield good results with multiple soft value release. This was due to

inadequate experimentation and lack of extensive literature search at that time. The advantage of multiple bit release is the increased speed of decoding. Also an attempt was made to simplify the turbo decoding hardware in two ways: eliminating the systematic inputs to the second decoder and eliminating the systematic inputs to the first SOVA decoder. Disabling the systematic inputs to Decoder2 yielded better results compared to disabling systematic inputs to the Decoder1 although it had much lower performance compared to the original configuration. These ideas need to be adequately tested in the future.

5.2.2 Improvements in the Present Implementation

The present implementation is a 34 symbol decoder and was implemented in order to keep the system simple and testable. In order to take advantage of the interleaving gain and for commercial purposes, at least a 500 symbol interleaver and decoder has to be built. Also, methods to improve the decoding speed up to 2 Mbits/s have to be investigated. Increasing the clock to operate at 500 MHz could possibly be one way of achieving this. This needs to be checked out. Another approach is to pipeline the SOVA decoders. This will need additional input buffers for more than one frame. It is then possible to have multiple SOVA decoders that will work on two frames simultaneously or exchange partially decoded frames between them. The Register exchange method for the path management could also be implemented and compared for speed improvement.

5.2.3 Implementation of the Improved SOVA

The SOVA based turbo decoder's error correction performance is inferior by .4 dB in comparison to the Max-log-MAP. Duanyi [28], showed that Battail's version of the SOVA algorithm is comparable to the Max-log-MAP algorithm. Since the Battail's version is much lower in complexity to the Max-log-MAP, this fact can lead to considerable area, power and cost savings if an efficient implementation of Battail's version of SOVA is done.

5.2.4 Power Consumption Analysis

Using the *Power compiler* tool (part of Synopsys tool set) the switching factors at the nodes can be determined by feeding different input patterns and then a better estimate of the power consumption can be made than the one obtained by the power reports from the *VHDLAN*. The present SOVA parallel multiple traceback unit implementation has traded power for speed. This unit can be investigated to see if alternate designs which offer high speed and which will consume lesser power exist. Interleavers/deinterleavers could be realized as wires, thus eliminating gate requirements.

5.2.5 Simulation Under the Influence of Fading

In this thesis the turbo decoding performance was simulated with an AWGN channel. In the practical world there is also fading due to multi path scattering of the signals and this has an important influence on the signal reception of the mobile phone receivers. This fading phenomenon can be modelled employing the Rayleigh and Rician channel models. The turbo decoding performance can then be studied and countermeasures to improve the degradation can be built into the hardware implementation.

References

[1] W. Yufei, *Implementation of Parallel and Serial Concatenated Convolutional Codes.* PhD thesis, Virginia Polytechnic Institute and State University, 2000.

[2] R. Johannesson and K. Zigangirov, *Fundamentals of Convolutional Coding.* IEEE Press, 1999.

[3] C. Shannon, *A mathematical thory of communication.* Bell Sys. Tech. Journal.27:379-423 (Part I), 623-656 (Part II), 1948.

[4] S. Wicker, *Error-Control Systems for Digital Communication and Storage.* MIT Press, 1995.

[5] M. Golay, *Notes on digital coding.* Proc.I.R.E., 1949.

[6] A. Viterbi, "Error bounds for convolutional codes and an asymptotically optimum decoding algorithm.," vol. 13, pp. 260–269, IEEE Transactions on Information Theory, 1967.

[7] D. Divsalar and P. Fabrizio, "Turbo Trellis Coded Modulation with Iterative

Decoding for Mobile Communications, TDA progress report," tech. rep., JPL, NASA, 2001.

[8] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near Shannon limit error-correcting coding and decoding: Turbo-codes," pp. 1064–1070, IEEE Int. Conf. on Commun., May 1993.

[9] S. Benedetto, G. Montorsi, D. Divsalar, and F. Pollara, "Serial Concatenation of Interleaved Codes: Performance Analysis, Design,and Iterative Decoding," vol. 44, pp. 909–926, IEEE Trans. Info. Theory, May 1998.

[10] D. G. Forney, *Concatenated Codes.* MIT Press, Cambridge, MA, 1966.

[11] J. Hagenauer and P.Hoeher, "A Viterbi algorithm with soft-decision outputs and its applications," pp. 1680–1686, Proceedings of the IEEE, GLOBECOM, Oct 1989.

[12] J. Woodard and L. Hanzo, "Comparative Study of Turbo Decoding Techniques: An Overview," vol. 49, pp. 1473–1478, IEEE Transactions on Vehicular Technology, November 2000.

[13] G. Battail, "Ponderation des symboles decodes par l'algorithm de Viterbi," p. 3138, Ann. Telecommun., Jan 1987.

[14] F. hua Huang, "Evaluation of Soft Output Decoding for Turbo Codes," Master's thesis, Virginia Polytechnic Institute and State University, 1997.

[15] J.Hagenauer, E.Offer, and L.Papke, "Iterative Decoding of Binary Block and Convolutional Codes," vol. 130, pp. 429–445, IEEE Transactions on Information Theory, March 1996.

[16] J. Hagenauer, "Source-Controlled Channel Decoding," vol. 43, pp. 2449–2457, IEEE Transactions on Communications, Sept 1995.

[17] J. Hagenauer and P.Robertson, "Iterative Turbo Decoding of Systematic Convolutional Codes with the MAP and SOVA Algorithms," vol. 130, pp. 21–29, ITG-Fachberichte, Sept 1995.

[18] P. Robertson, "Improving Decoder and Code Structure of Parallel Concatenated Recursive Systematic(Turbo) Codes," in *Third Annual International Conference on Universal Personal Communication*, pp. 183–187, 1994.

[19] P. Jung and M. Nasshan, "Performance Evaluation of Turbo Codes for Short Frame Transmission Systems," in *Electronics Letters, 30(2)*, pp. 111–113, 1994.

[20] S. A. Barbulescu and S. Pietrobon, "Terminating the Trellis of Turbo Codes in the Same State," in *Electronics Letters, 31(1)*, 1995.

[21] D. Divsalar and P. Fabrizio, "Turbo Codes for Deep-Space Communications, TDA progress report," tech. rep., JPL, NASA, 1995.

[22] J.Hokfelt, O. Edfors, and T. Maseng, "On the Theory and Performance of Trellis

Termination Methods for Turbo Codes," vol. 19, pp. 838 – 847, IEEE journal on Selected areas in communications, May 2001.

[23] B. Vucetic and J. Yuan, *Turbo Codes, Principles and Applications.* Kluwer Academic publishers, Boston, 2000.

[24] D. Divsalar and P. Fabrizio, "Multiple Turbo Codes for Deep-Space Communications, TDA progress report," tech. rep., JPL, NASA, 1995.

[25] Barbeluscu, S. Adrian, and P. S.Silvio, "Interleaver Design for Turbo Codes," tech. rep., ITR, June 1994.

[26] L. Lin and R. Cheng, "Improvements In SOVA-Based Decoding For Turbo Codes," vol. 3, pp. 1473–1478, IEEE Conference on Communications, June 1997.

[27] M. Fossorier, F. Burkert, S.Lin, and J. Hagenauer, "On the equivalence between sova and max-log-map decodings," vol. 2, pp. 137 – 139, IEEE Communication Letters, May 1998.

[28] D. Wang and H. Kobayashi, "High-performance sova decoding for turbo codes over cdma2000 mobile radio," vol. 1, pp. 189 – 193, IEEE Military Communications Conference, May 2000.

[29] A. Bellaouar and M. Elmasry, *Low-Power Digital VLSI Design, Circuits and systems.* Kluwer Academic Publishers, Norwell, MA, 1995.

[30] A. Chandrakasan, S. Sheng, and R. Brodersen, "Low-power CMOS digital design," vol. 27, pp. 473 – 484, IEEE Journal of Solid-State Circuits, April 1992.

[31] S. Z. K. Koora, F. Poegel and A. Finger, "From algorithm testing to ASIC input code of SOVA algorithm forTURBO-Codes," tech. rep., Dresden University of Technology, Communications Laboratory, Germany, 1996.

[32] G. K. Yeap, *Practical Low Power Digital VLSI Design.* Kluwer Academic Publishers, Norwell, MA, 1995.

[33] M. O. Joeressen and H.Meyr, "High-Speed VLSI Architectures for Soft-Output Viterbi Decoding," vol. 43, pp. 373–384, Proc. of ICASAP, Aug 1992.

[34] K. Seki, S. Kubota, M.Mizoguchi, and S. Kato, "Very low power consumption Viterbi decoder LSIC employing the SST (scarce state transiton) scheme for multimedia mobile communications," vol. 30, pp. 637 – 639, IEE Electronics Letters, April 1994.

[35] D. Oh and S. Hwang, "Design of a Viterbi decoder with low power using minimum-transition traceback scheme," vol. 32, pp. 2198 – 2199, IEE Electronics Letters, Oct 1996.

[36] D. Garrett and M. Stan, "Low Power Architecture of the Soft-Output Viterbi Algorithm," tech. rep., University of Virginia, VA, 1999.

[37] J. Proakis, *Matlab Applications.* Professional Engineer Publishing, 2000.

[38] J. Heller and I. M. Jacobs, "Viterbi Decoding for Satellite and Space Communications," vol. 19, pp. 835–848, IEEE Transactions on Communication Technology, October 1971.

[39] "Universal Mobile Telecommunications System, Multiplexing and Channel Coding (FDD), 3G TS 25.212 version 3.3.0, www.etsi.org."

[40] J. Armstrong, *VHDL Design, Representation and Synthesis.* Prentice Hall, 2000.

[41] S. Ranpara and H. Dong, "Low Power Viterbi Decoder," in *International ASIC conference, Washington, D.C.*, 1999.

Appendix A

Drawings and Simulation Waveforms

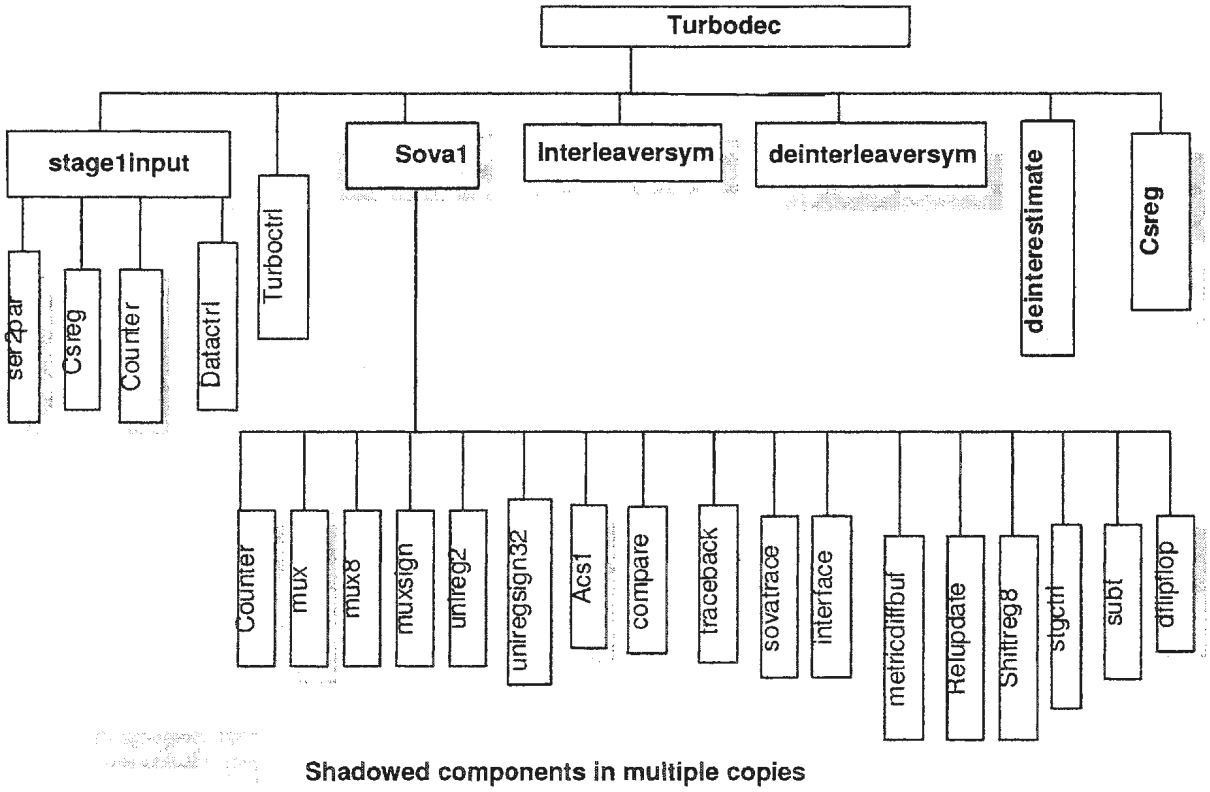
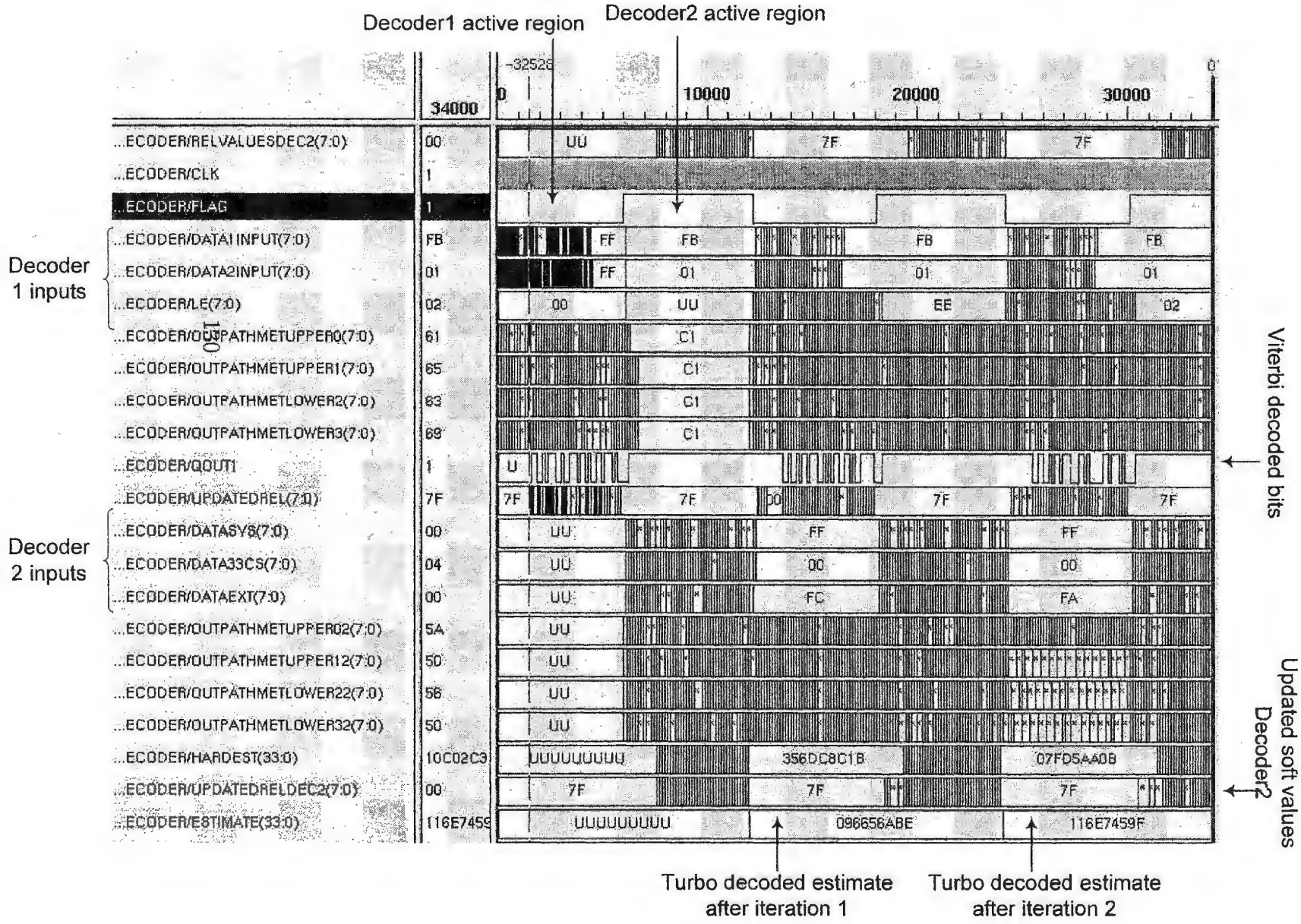


Figure A.1: Module structure


Decoder1 active region
Decoder2 active region
34000
-32528
0
10000
20000
30000
...ECODER/RELVALUESDEC2(7:0)
...ECODER/CLK
...ECODER/FLAG
...ECODER/DATA1INPUT(7:0)
...ECODER/DATA2INPUT(7:0)
...ECODER/LE(7:0)
...ECODER/OUTPATHMETUPPER0(7:0)
...ECODER/OUTPATHMETUPPER1(7:0)
...ECODER/OUTPATHMETLOWER2(7:0)
...ECODER/OUTPATHMETLOWER3(7:0)
...ECODER/QOUT1
...ECODER/UPDATEDREL(7:0)
...ECODER/DATASYS(7:0)
...ECODER/DATA33CS(7:0)
...ECODER/DATAEXT(7:0)
...ECODER/OUTPATHMETUPPER02(7:0)
...ECODER/OUTPATHMETUPPER12(7:0)
...ECODER/OUTPATHMETLOWER22(7:0)
...ECODER/OUTPATHMETLOWER32(7:0)
...ECODER/HARDEST(33:0)
...ECODER/UPDATEDRELDEC2(7:0)
...ECODER/ESTIMATE(33:0)
Decoder 1 inputs
Decoder 2 inputs
Viterbi decoded bits
Updated soft values Decoder2
Turbo decoded estimate after iteration 1
Turbo decoded estimate after iteration 2

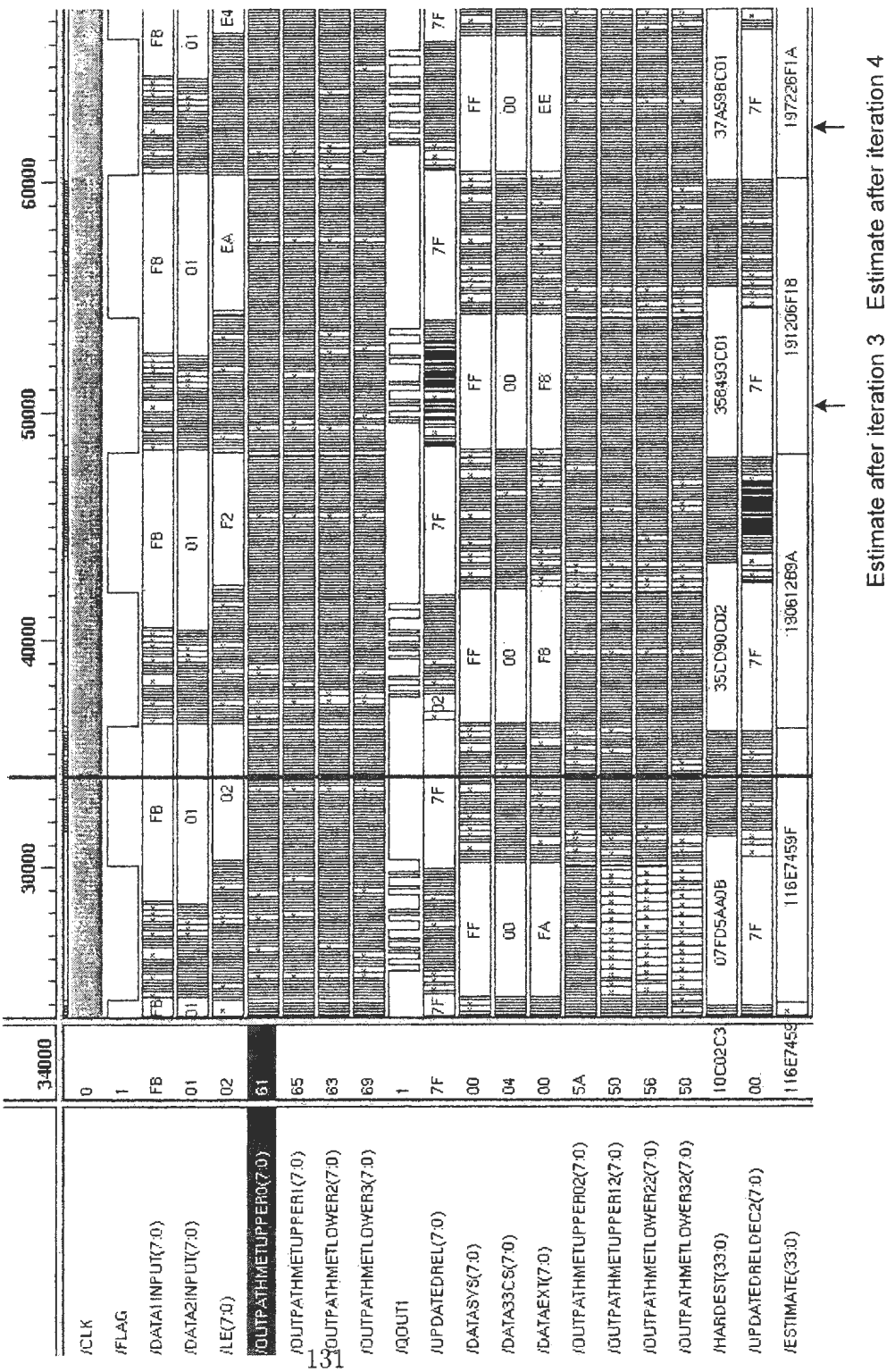


Figure A.2: Timing diagram for 2 iterations

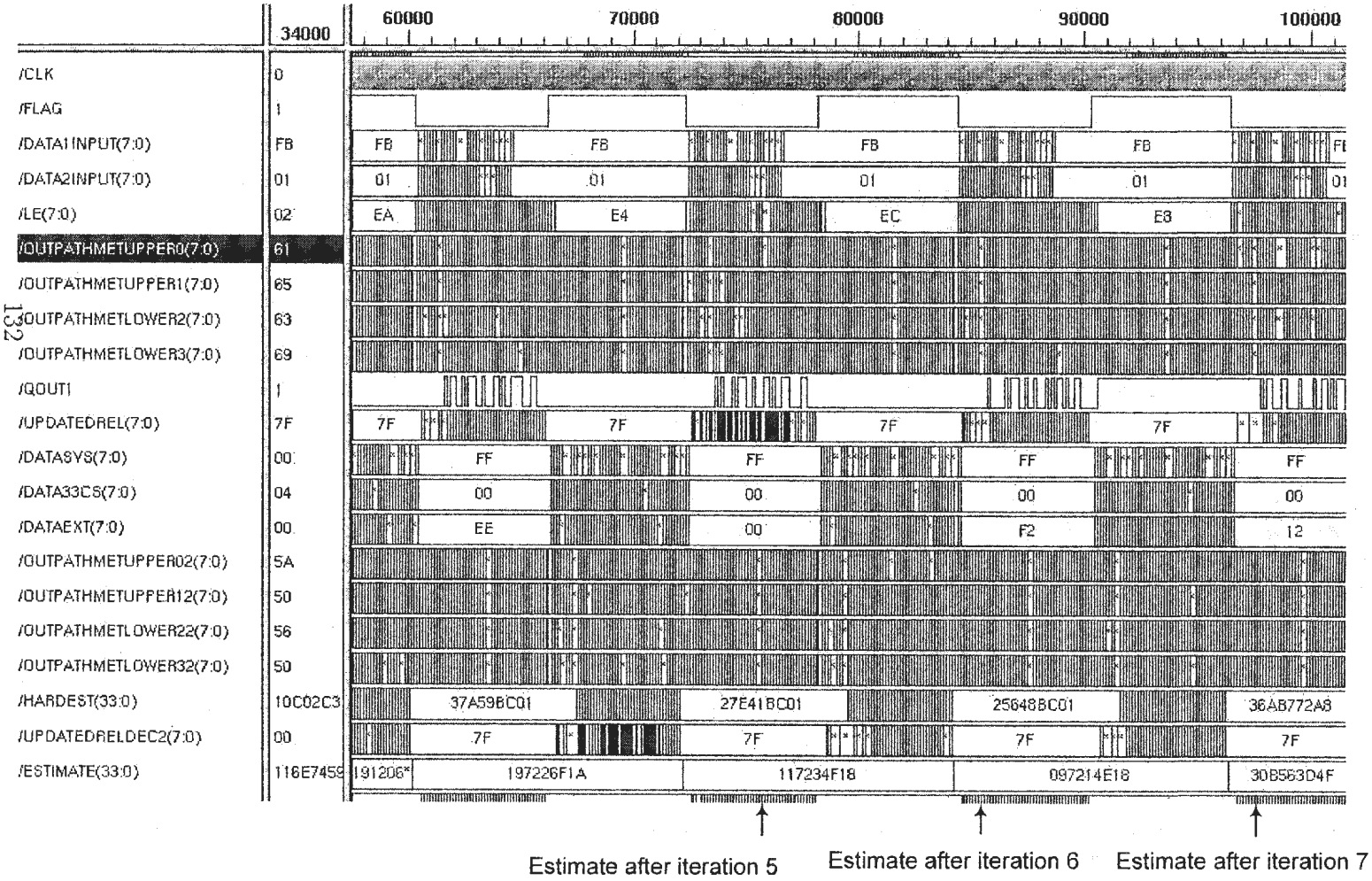


Figure A.3: Timing diagram for 7 iterations

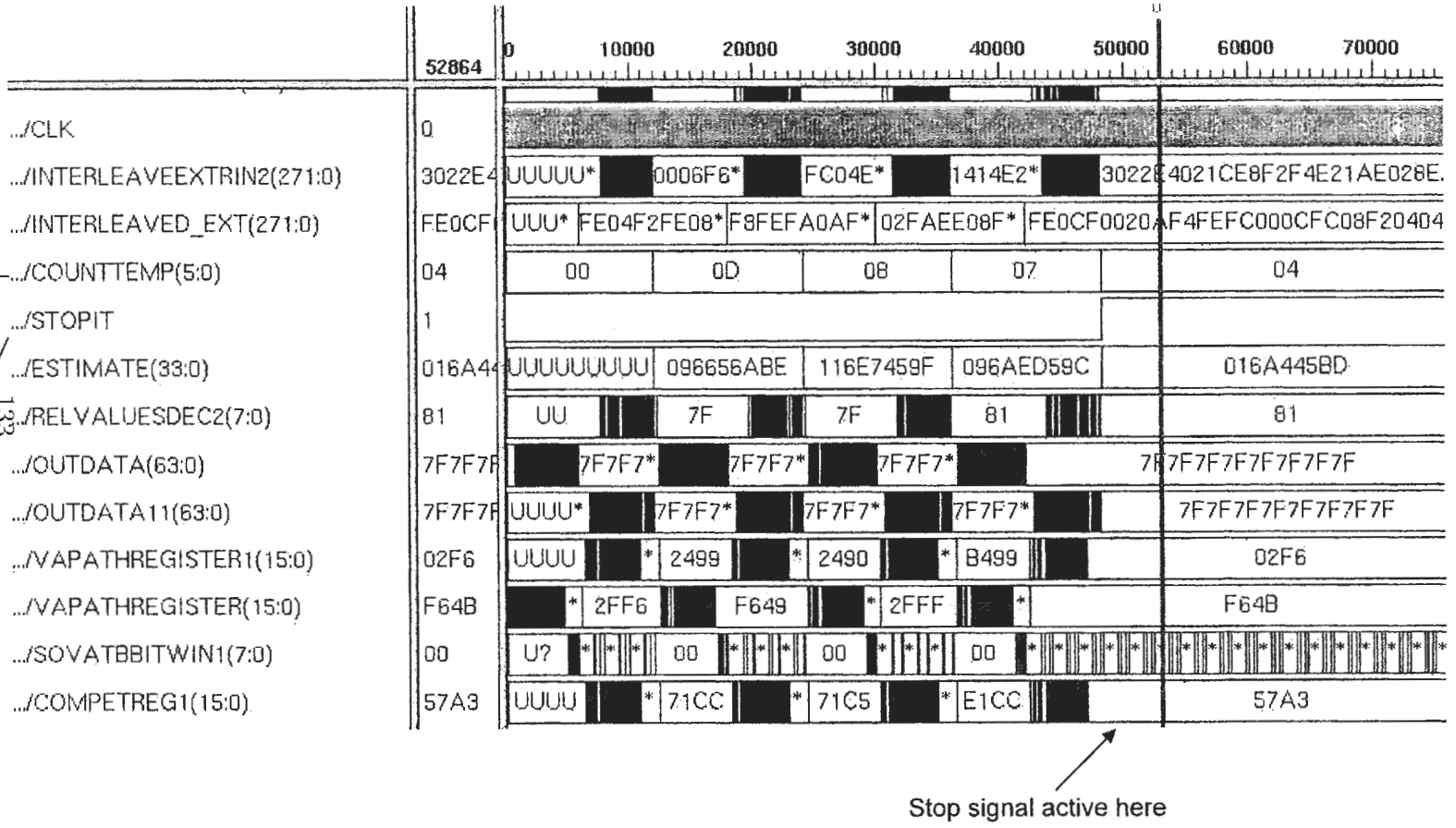


Figure A.4: SDR stopping scheme