INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

ProQuest Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600

Hardware Implementation of Non-binary Turbo Code for DVB/RCS

Yimin Du

A Thesis

in

The Department

of

Electrical and Computer Engineering

Presented in Partial Fulfillment of the Requirements

for the Degree of Master of Applied Science at

Concordia University

Montréal, Québec, Canada

February 2003

ABSTRACT

Hardware Implementation of Non-binary Turbo Code for DVB/RCS

Yimin Du

Double binary convolutional turbo codes, using Circular Recursive Systematic Convolutional (CRSC) codes as component codes, have been shown to outperform binary turbo codes. These codes are adopted in the Digital Video Broadcasting – Return Channel via Satellite (DVB-RCS) standard. The outstanding coding performance of these codes intrigues the investigation of hardware implementation issues. In this thesis, first a simplified Max_Log_MAP algorithm is derived for the Non-binary convolutional turbo code, and then different aspects of the implementation issues of the decoder with VLSI are explored. In addition, a complete decoder VLSI design of non-binary convolutional turbo code for DVB/RCS will be presented. After discussing several quantization and normalization schemes, a new optimal renormalization approach will be proposed. With this new approach, the decoder can be speeded up considerably. In order to save area, a practical simplification method of branch metric calculation is introduced, which makes the whole design much more efficient. From an architectural point of view, an optimal full pipelined structure is designed with the forward path metric and backward path metric recursive circuits being optimized for speed and other functions including concise interleaver generation, data input, branch metric calculation being optimized for area. In the last part of this thesis, another pipelined area saving method is proposed. The design is modeled in Very high speed integrated circuit Hardware Description Language (VHDL) and synthesized on a single chip FPGA (Xilinx Virtex-E). According to the RTL level and gate level simulation results and the in-chip test result, the decoder can work up to 7 Mbits/s data rate at 6 iterations with VirtexE FPGA.

Acknowledgements

I would like to express my gratitude to my supervisor Dr. M. R. Soleymani for giving me the opportunity to research and work in both coding theory and VLSI fields. His encouragement, stimulative suggesting and valuable supports help me in all the time of my research and writing of this thesis. I also feel deeply indebted to him for creating an convenient research environment and providing all advanced development tools and devices for my research works.

I want to take this opportunity to thank all specialist, administrators and technical staff, especially Mr. Ted for their continuous supports in using computer system and design tools. Finally, I would like to express my thanks to all researchers and students in our lab. I benefit a lot from their friendly and kindly helps.

Table of Contents

LIST OF FIGURES vii
LIST OF TABLES viii
LIST OF ACRONYMS ix

CHAPTER 1 1

INTRODUCTION 1

1.1 DEVELOPMENT OF CHANNEL CODING 2
1.2 IMPLEMENTATION OF CHANNEL CODES WITH VLSI 5
1.3 OVERVIEW OF THIS THESIS 7

CHAPTER 2 9

DVB/RCS TURBO CODE AND MAP DECODING ALGORITHM 9

2.1 TURBO CODE 9
2.1.1 Encoder of the component code *10*
2.1.2 Interleaver Design *11*
2.1.3 Trellis Termination and Code Puncturing *12*
2.1.4 Decoding of the Turbo Codes *13*
2.2 NON BINARY TURBO CODE FOR DVB/RCS 15
2.2.1 Encoder of non binary turbo code for DVB/RCS *16*
2.2.2 Circular Recursive Systematic Converlutional (CRSC) code *21*
2.3 MAP DECODING ALGORITHM 22
2.3.1 MAP decoding for binary turbo code *23*
2.3.2 MAP decoding for non binary turbo code *25*
2.3.3 Log-MAP decoding algorithm *28*
2.3.4 Max-Log-MAP decoding algorithm *30*

CHAPTER 3 34

IMPLEMENTATION ISSUES OF DVB/RCS CODE 34

3.1 ALGORITHM SIMPLIFICATION 35
3.2 QUANTIZATION 35
3.2.1 Input data quantization *36*
3.2.2 Inner data Quantization *38*
3.3 NORMALIZATION 39
3.3.1 Rescaling *40*
3.3.2 Modulo Normalization *41*
3.3.3 New optimized normalization for DVB/RCS code *44*
3.4 SIMPLIFICATION OF BRANCH METRICS CALCULATION 45

3.4.1 Gamma calculation of DVB/RCS turbo code *46*
3.4.2 Proposed reduction approach for Gamma calculation *48*
3.5 EFFECT OF CORRECTION COEFFICIENT 51

CHAPTER 4 ..... **53**

HARDWARE ARCHITECTURE OF THE DECODER **53**

4.1 GENERAL DESCRIPTION AND DESIGN PARTITION 54
4.2 MAX-LOG-MAP DECODER 56
4.2.1 Full Pipeline architecture *60*
4.2.2 Alpha and beta recursion update structure *65*
4.2.3 Extrinsic value calculation structure *67*
4.2.4 Further area reduction with resource sharing *68*
4.3 READ DATA BLOCK DESIGN 71
4.4 INTERLEAVER GENERATION BLOCK DESIGN 72
4.5 OUT DATA BLOCK DESIGN 73
4.6 INTERFACE OF DVB/RCS DECODER 74

CHAPTER 5 ..... **79**

SIMULATION AND SYNTHESIS RESULTS **79**

CHAPTER 6 ..... **83**

CONCLUSION AND FUTURE WORK **83**

Future work *84*

BIBLIOGRAPHY **85**

List of Figures

Figure 2-1 Generalized turbo encoder 10
Figure 2-2 RSC encoder with memory 2 for rate ½ 11
Figure 2-3 Soft-in/Soft-out decoder 14
Figure 2-4 Iterative decoding procedure for two elementary codes 15
Figure 2-5 DVB/RCS double binary turbo code encoder 16
Figure 2-6 Double binary turbo code encoder 17
Figure 3-1 The Frame Error Rate for ATM blocks, 8 iterations 37
Figure 3-2 Bit Error Rate for ATM blocks, 8 iterations 38
Figure 3-3 Diagram of A*B + C*D 48
Figure 3-4 Diagram of part of gamma function 49
Figure 3-5 Simplified diagram of part of gamma function 49
Figure 3-6 Simplified gamma Function 50
Figure 4-1 Model structure of the decoder design 54
Figure 4-2 Decoder structure 55
Figure 4-3 Timing diagram of DVB/RCS decoder 55
Figure 4-4 Data dependency graph of the decoder architecture 59
Figure 4-5 Data path of the MAP decoder 61
Figure 4-6 Timing diagram of sequentially executed MAP decoder 62
Figure 4-7 Timing diagram of pipelined architecture 63
Figure 4-8 Data path of the optimized pipelined architecture 65
Figure 4-9 The basic circuit of the alpha update 65
Figure 4-10 Alpha update circuit with the proposed normalization 67
Figure 4-11 The sigma implementation circuit 68
Figure 4-12 Operation schedule of the MAP decoder 69
Figure 4-13 Operation schedule with pipelined resources 70
Figure 4-14 State diagram of the read data block 72
Figure 4-15 Input timing of the decoder 75
Figure 4-16 Output timing of the decoder 76
Figure 5-1 General design procedure 80
Figure 5-2 Block diagram of DVB/RCS simulation 81
Figure 5-3 Synthesis result with Xilinx Virtex XCV1000E-8 FPGA 82

List of Tables

Table 2-1 Turbo code permutation parameters 18
Table 2-2 Circulation state correspondence table 19
Table 2-3 Puncturing patterns of DVB/RCS double binary turbo code 19
Table 2-4 The length of the encoded block 20
Table 3-1 Bit width of inner data for DVB/RCS code 42
Table 3-2 State transfer and encoder output 47
Table 4-1 Decoder signal pin-out 74
Table 4-2 Code rate setting 77
Table 4-3 Frame Size Settings 77

List of Acronyms

8PSK	8-ary Phase Shift Keying
ALAP	As Late As Possible Algorithm
APP	A Posteriori Probability
ASAP	As Soon As Possible Algorithm
AWGN	Additive White Gaussian Noise
ASIC	Application Specific Integrated Circuit
BCH	Bose-Chaudhuri-Hocquenghem code
BER	Bit Error Rate
Bps	Bit per second
BPSK	Binary Phase Shift Keying
CRSC	Circular Recursive Systematic Convolutional
CSA	Canadian Space Agency
DSP	Digital Signal Process
DVB-RCS	Digital Video Broadcasting-Return Channel via Satellite
FEC	Forward Error Correction
FER	Frame Error Rate
FPGA	Field Programmable Gate Array
LLR	Log-Likelihood Ratio
MAP	Maximum a posteriori Probability
MPGA	Mask Programmable Gate Array
PCCC	Parallel Concatenated Convolutional Code
PSK	Phase Shift Keying
QAM	Quadrature Amplitude Modulation
QPSK	Quadrature Phase Shift Keying
SNR	Signal to Noise Ratio
RS	Reed-Solomon code
RSC	Recursive Systematic Convolutional Code
SISO	Soft-Input Soft-Output
SOVA	Soft-Output Viterbi Algorithm
TCC	Turbo Convolutional Code
TPC	Turbo Product Code

Chapter 1

Introduction

Digital communication is playing more and more important role in our daily life with increasing demand for large scale, high speed and reliable data transmission in the military, government, industry, education and private spheres. The rapid development of computer technology and VLSI technology promote the development of digital communication and enable the modern communication systems and devices providing us powerful and convenient means to store and exchange information. In digital communication engineering, a major concern of the designer is the control of errors so that a high quality data transmission can be achieved. Thus, error control coding, channel coding, becomes a fundamental element of any digital communication system. Until now a great deal of efforts have been made on channel coding theory and methods to obtain high reliability and high speed data transmission or data storage [1]. Many efficient and practical encoding and decoding schemes have been invented and used in every application domain, especially for deep space and satellite communication systems. In this thesis, we concentrate on the hardware implementation issues of channel coding for satellite communication.

1.1 Development of channel coding

The subject of channel coding, motivated by practical problems, covers several disciplines, in particular mathematics, electronic engineering and computer science. In the very beginning of the error-control coding history, Claude Shannon in 1948 proved the existence of error-control codes. By using these codes, under suitable conditions and at rates less than channel capacity, error-free information could be transmitted for all practical applications [2]. Unfortunately, Shannon's theory relied on random coding argument and gave no indication of the structure of the codes that would have a performance near channel capacity. Since this time, researchers and engineers in error-control coding field have been attempting to find more effective error-control codes that will provide near capacity performance. Thus far, there are two major techniques for building codes, block codes and convolutional codes.

During the 1950s, coding research was emphasized on block code with algebraic approaches. The first practical binary linear block code, Hamming code, was introduced by Richard Hamming [3]. And in 1957, a kind of more powerful codes, cyclic codes, were first studied by Prange [4]. These codes are attractive for two reasons: first, they are easy to implement by using shift registers; and second, it is possible to find various practical methods for decoding them. In principle, the decoding method devised by Meggitt [5] applies to any cyclic code, but refinements were necessary for practical implementation; this technique is known as error-trapping decoding. The practical error-trapping decoding was realized independently by Kasami [6], Mitchell [7][8], and Rudolph [9]. It is most effective for decoding single-error-correcting codes, some double-error-correcting codes and burst-error-correcting codes rather than for long and high rate codes with large error-correcting capability. The (23, 12) Golay code discovered by Golay in 1949 [10] is the only known multiple-error-correcting binary perfect code which

is capable of correcting any combination of three or fewer random errors and it can be easily decoded by Kasami's error-trapping technique.

Hocquenghem In 1959 [11], Bose and Chaudhuri in 1960 [12] independently found a large class of multiple-error-correcting cyclic codes, called the BCH codes. Generalization of the binary BCH codes to both binary and non-binary BCH codes obtained by Gorenstein and Zierler in 1961 [15]. In addition, Reed and Solomon independently found an extremely important and practical class of non-binary BCH codes [13]. The Reed Solomon (RS) codes came into widespread use in many communication and computer storage systems. The first decoding algorithm for binary BCH codes was devised by Peterson in 1960 [14], refined by Gorenstein and Zierler [15], extended by Berlekamp [16], Massey [17], Chien [18], Forney [19], and others. Among all these algorithms, Berlekamp's iterative algorithm and Chien's search algorithm are the most efficient ones.

In general, codes for correcting random errors are not effective for burst errors. The codes designed to correct burst errors is called burst-error-correcting codes. Cyclic codes for single-burst-error correction were first studied by Abramson [20]. In the effort to generalize Abramson's results, Fire discovered a large class of burst-error-correcting cyclic codes [21], and these codes can be decoded with a very simple circuit.

The majority-logic decoder is another effective device for decoding certain classes of cyclic block codes. In 1963 Massey was the first to present a unified treatment for majority-logic decoding algorithms [22]. The maximum-length codes and the difference-set codes are two small subclasses of one-step majority decodable cyclic codes. Finite-geometry codes (Euclidean geometry codes and projective geometry codes) have their own structure and rules of orthogonalization for the multiple-step majority decodable codes. Rudolph investigated finite geometry codes which were extended and generalized by many coding researchers [23].

Another remarkable achievement in the development of error correcting coding theory was the discovery of convolutional codes. Convolutional codes were first

introduced by Elias [24] in 1955 as an alternative to block codes. Shortly thereafter, Wozencraft proposed sequential decoding as an efficient decoding scheme for convolutional codes 25] and Massey proposed a less effective but simpler-implement decoding scheme called threshold decoding [26]. Since then, convolutional codes have been used to digital communication over wire and radio channels. In 1967, Viterbi proposed the Viterbi algorithm for maximum-likelihood decoding [27]. Later, Omura and Forney proved that the Viterbi algorithm is the best maximum-likelihood decoding algorithm for convolutional codes with shorter constraint lengths [28][29].

Another remarkable contribution of foney is that he proposed serial concatenated codes which achieved a good trade-off between coding gain and complexity [30]. In practical applications of this approach in space communication, the inner code uses convolutional code and the outer code uses low redundancy RS code. As the inner code decoder generates burst errors, an interleaver is typically incorporated between inner code and outer code to decorrelate the received symbols due to burst errors.

Turbo Codes were developed with a similar idea of concatenated codes, connecting two codes and separating them with an interleaver. But, in Turbo codes two component codes that are usually identical are connected in parallel. The first Turbo code was brought by Berrou et al. in 1993 [31] to the coding community. In his paper, impressive simulation results were presented, achieving an Eb/No of 0.7 dB with a ½ code rate. Since his astonishing work, a new generation of coding techniques and schemes has begun to be researched and come into applications gradually. The MAP algorithm, first presented by Bahl, Cocke, Jelinik and Raviv in 1974 [32], is normally used as decoding algorithm of Turbo codes. The component codes of turbo codes can be either convolutional codes which are called Turbo convolution codes (TCC) or block codes which are called turbo block codes. Several decoding algorithms for both binary TCCs and turbo block codes have been developed [33] including soft-output Viterbi algorithm (SOVA), MAP algorithm, and Log-MAP algorithm.

To improve coding performance TCCs are also extended into non-binary forms, for example, double binary Turbo convolution code. Since these codes are very flexible and easily adaptable to a wide range of data block sizes and coding rates, they are adopted in the DVB for Return Channel via Satellite (DVB-RCS), which places satellite in a favorable position, especially as a necessary complement in countries where ADSL and cable modem cannot economically cover more than 75% of the population [34].

There are two recent techniques adopted in this double binary convolutional turbo coding so that the coding performance is improved compared to binary turbo codes:

- Parallel concatenation of circular recursive systematic convolutional (CRSC) codes [35] makes convolutional turbo codes efficient for coding of blocks of data;
- Double-binary elementary codes provide better error-correcting performance than binary codes for equivalent implementation complexity [36].

1.2 Implementation of channel codes with VLSI

The advancement in VLSI technology has dramatically impacted the development of communication systems. Especially in the coding field, it provides us with cheaper means to implement complicated coding algorithms. VLSI (Very Large Scale Integration) refers to a technology through which it is possible to implement large circuits in silicon. Normally, CMOS is the technology of choice to implement digital logic and memory in VLSI circuit. Higher speed, lower power dissipation, and higher circuit density are the critical consideration in choosing an implementation technology.

There are several design approaches used to generate physical representations of circuits. They belong to two general classes: The full-custom design and the semi-custom design. Current semi-custom design styles include gate-array design style, standard-cell design style, Macro-cell design style, PLA (programmable logic array), and FPGA (field programmable gate-array) [37]. Full-custom layout refers to manual layout design which

is only suitable to small design because it is a time consuming and difficult task. Gate arrays, also called MPGAs (mask programmable gate array), consist of a large number of transistors which have been prefabricated on a wafer in the form of a regular two-dimensional array. Therefore, it takes a short time to get the chip fabricated, but it has limitation on layout due to the limited amount of wiring space. Standard-call and Macro-cell design styles are based on standard cell library which provides various kinds of logic blocks. It offers more flexibility than MPGA, but it is more difficult to fabricate. PLA provides a convenient way to implement two-level sum-of-products logic expressions. PLAs are commonly used to implement the control path of a digital circuit or a combinational circuit, rather than the data path of a circuit. Similar to MPGA, a FPGA also consists of a two-dimentional array of logic blocks. The advantage of FPGA is that both the logic blocks and the interconnects are field programmable, but their speed of operations and the density become worse.

An ASIC (Application Specific Integrated Circuit), unlike a microprocessor, is a circuit which performs a specific function in a particular application. It is often optimized in the area and the performance for a special application. To reduce the time to market, the technologies adopted by an ASIC are among Gate-array, standard-cell, and PLA.

When implementing a design, we often choose technology among ASICs, FPGAs and DSPs. All of them can be used in implementation of error-control codes. A DSP (Digital Signal Processor) is essentially a microcomputer having a specialized hardware and instruction set that is designed to do real time processing functions. DSPs are highly flexible because they can be easily programmed to perform any algorithm with assembly language or a high-level language like C. They allow the fastest design cycles. The new model DSP chip of TI, TMS32064, operating up to 4800 MIPS, has several coprocessors for telecommunication application including Viterbi decoder and TCC decoder [38][73]. Implementing signal processing in a sequential manner reduces the data rate of the DSPs. To obtain higher performance, FPGA or ASIC has to be used. Nowadays with a hardware description language, VHDL or Verilog, a designer can easily design an application.

Using a synthesis tools, the design can be synthesized and configured into an FPGA. It can realize high degree of parallelism compared to DSP, faster time to market compared to ASIC, so FPGAs are most ideally suited for prototyping application. ASICs lead to the fastest speed, more power efficiency and optimal resource saving, however, they require a longer design cycle and are the most costly. Thus, they are not suited for prototype products and products with small volume of production.

Recently many researchers are focusing on the issues of hardware implementation of Turbo codes. Several papers have been published on the hardware implemetation issues and effective architectures of binary TCC [39][40][41][42][43][44][45] [46][47][48][49][50][51]. However, the papers on design architecture of non-binary turbo code have not been published. The Turboconcept company has announced its products of non-binary turbo code, but has not published its design in detail [52][53].

1.3 Overview of this thesis

This thesis is dedicated to hardware implementation issues of non-binary turbo codes covering algorithm analysis, algorithm simplification, optimal architecture design and RTL lever synthesis. It presents the VLSI design of a complete non-binary turbo decoder for DVB/RCS application from high level to low level.

In Chapter 2, the background of turbo codes is presented. First, the binary turbo codes are explained, followed by description of non-binary Turbo code for DVB/RCS application. The DVB/RCS Turbo code standard is described in detail. The CRSC (Circular Recursive System Convolutional) code, the component code of DVB/RCS Turbo code, is also discussed. Then, the decoding algorithms for both binary Turbo codes and non-binary Turbo codes are derived. These decoding algorithms include MAP algorithm, Log-MAP algorithm and MAX-Log-MAP algorithm.

In Chapter 3, this thesis concentrates on several implementation issues, the system level design including quantization, renormalization and metrics calculation simplification. For the quantization issues, both input data quantization and inner data quantization are discussed. After that, two renormalization schemes are addressed and a new renormalization scheme is proposed. In this chapter, a new metrics calculation reduction approach is also given, which bring a tremendous saving of chip area.

In Chapter 4, the architecture of DVB/RCS code decoder is designed, and the optimal pipelined architecture has been achieved. The critical MAP algorithm component is optimized in speed, while other components such as data reading, interleaver etc. are optimized in area. All the optimized architecture of the design will be illustrated component by component.

In Chapter 5, all levels simulation results and synthesis results for the decoder are given. The results prove that this design for DVB/RCS non-binary turbo code decoder obtains a great success. The performance of the decoder reaches up to 7 Mbits/s with 6 iterations, better than existing commercial products with the same technology family.

In Chapter 6, conclusion is drawn and suggestions for future work on this topic are given.

Chapter 2

DVB/RCS turbo code and MAP decoding algorithm

In this chapter, first the background knowledge of turbo codes is introduced, then followed by the specification of non-binary convolutional code for DVB/RCS. Moreover, various MAP decoding algorithms for both binary case and non-binary case are derived in detail. Especially, Max-Log-MAP decoding algorithm, which is proved to be a good trade off between decoding complexity and decoding performance, is explored and chosen as decoding algorithm in this decoder design,.

2.1 Turbo code

Turbo coding is accomplished by the parallel concatenation of two or more systematic codes. A generalized turbo encoder is shown in Figure 2.1, where Im represents an interleaver. Each interleaver scrambles the information frames in a pseudo-random fashion and feeds its output into a component encoder. For most applications,

only two such encoders are used. In general, the turbo coding issues include component code design, interleaver design, trellis termination method and the choice of decoding algorithm. In the following sections, all these design issues will be discussed in detail.

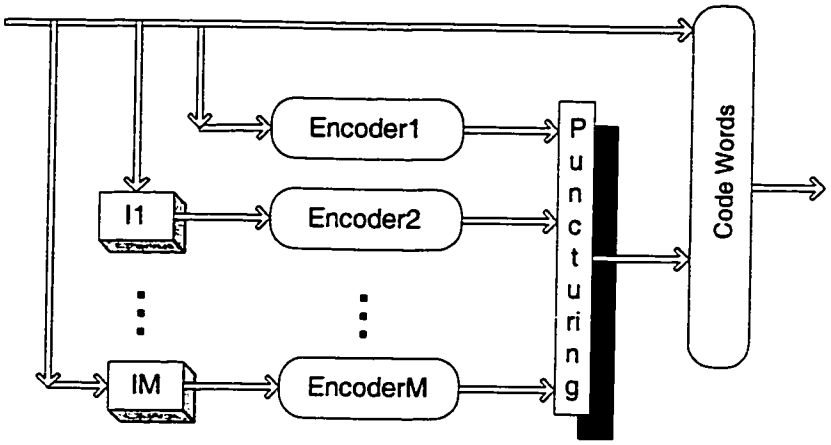

Figure 2-1 Generalized turbo encoder

2.1.1 Encoder of the component code

Basically, any systematic code, either a block code or a convolutional code, can be used as a component code of turbo codes, and if a block code is used as a component code, this turbo code is called turbo block code (TBC); if a convolutional code is used as a component code, the turbo code is called turbo convolutional code (TCC). In most cases, a recursive systematic convolutional code (RSC) is chosen as the component code, because the use of a convolutional code makes it possible for the decoder to utilize a modified version of the Viterbi algorithm and the recursive code has better performance than the non-recursive code [31].

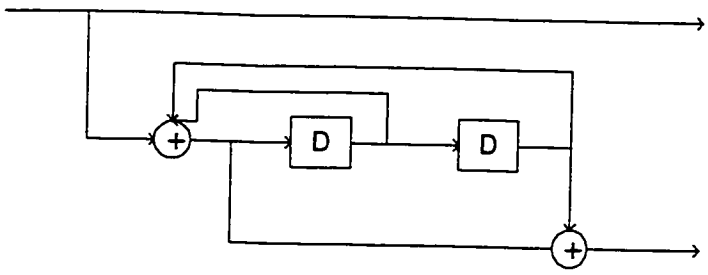

Figure 2-2 RSC encoder with memory 2 for rate ½

An RSC code can be constructed with a feedback register. Figure 2.1 shows an RSC code for rate ½ with memory 2. The generator matrix of this code can be represented as

$$G(D) = \left(1 \quad \frac{g_1(D)}{g_0(D)} \right) \tag{2-1}$$

where $g_0(D)$ and $g_1(D)$ are feedback and feed-forward polynomials, respectively.

$g_0(D) = 1 + D + D^2$

$g_1(D) = 1 + D^2$

To achieve a better performance of turbo codes, we should design an RSC component code with maximum effective free distance which is a key factor determining the decoding performance of the turbo code, especially when working in the region of high SNR's. [54] [55]

2.1.2 Interleaver Design

Interleaving, a process of rearranging the ordering of an information sequence, is used before the information data is encoded by the second component encoder. It plays a fundamental role in turbo coding scheme, affecting the performance of turbo codes considerably.

In turbo coding, interleaver has several functions. First, it is used to construct a long block code from small memory convolutional codes as long codes can achieve a better coding performance. Secondly, it spreads out burst errors, changing uncorrectable error pattern to correctable error pattern. Thirdly, interleave provides scrambled information data to the second component encoder and decorrelates the inputs to the two component decoders, as a result an iterative sub-optimum decoding algorithm based on uncorrelated information data between the two component decoders can be applied. Finally, interleave can effectively break low weight input sequences, thus increase the code free distance or reduce the number of code words which have small distance in the code distance spectrum.

An odd-even interleaver is a particular type of interleaver which maps even position elements to even position and odd position elements to odd positions. In interleaver design, choosing an odd-even interleaver is often necessary because this kind of interleave can guarantee that error protection is uniformly distributed across the information sequence.

Since interleaver design is essential to achieving high performance coding, it intrigues many turbo code researchers. Till now, many interleaving strategies have been proposed such as block interleavers, convolutional interleavers, cyclic shift interleavers, S-random interleavers, code matched interleavers, etc.

2.1.3 Trellis Termination and Code Puncturing

Trellis termination means driving the encoder to all-zero state. By driving the encoder to an all-zero state at the end of trellis, we can obtain a better distance property, therefore, getting a better error correcting performance. Actually, there are two ways to implement trellis terminate. One is to force the encoder to return the all-zero state; another approach is to initialize the encoder at the same state as it will end up with.

Obviously, the advantage of the second approach is that this approach does not require transmission of the tail bits.

Puncturing is a process of removing certain symbols from the code words, therefore, reducing the code word length and increasing the overall code rate. For example, by puncturing the rate 1/3 turbo encoder various code rates such as 1/2, 2/3, 3/4, 5/6, 6/7 and so on can be acquired. When carrying out puncturing, we delete some output bits of the encoder according to a chosen pattern defined by a puncturing matrix P. For instance, a rate ½ turbo code can be obtained by puncturing a rate 1/3 turbo code. The commonly used puncturing matrix is given by

$$P = \begin{bmatrix} 1 & 1 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}$$

where the puncturing period is 2. According to the puncturing matrix, the parity check digits are alternatively deleted.

2.1.4 Decoding of the Turbo Codes

Turbo codes can be decoded by MAP or ML decoding methods based on overall code trellis, but these decoders could be too complex for large interleaver size. In practice, iterative decoding methods are used instead, which are also an important feature of turbo codes.

In the first paper on turbo code [31], Berrou et. al. proposed an iterative decoding scheme based on soft output Viterbi algorithm (SOVA). The SOVA differs from the Viterbi algorithm in the sense that it generates soft output and attempts to minimize the error rate by estimating the a posteriori probabilities (APP) instead of by obtaining the ML estimate of the transmitted code word. Nowadays, MAP algorithm or its modified versions are often used. Because of the requirement of the iterative decoding scheme, the

decoding algorithm of the component code of turbo codes have to be in soft input soft output version (SISO). Figure 2.3 shows an elementary SISO decoder.

For a general case, the turbo decoder consists of M elementary decoders. Each elementary decoder generates a soft value for each received bit. After each iteration of the decoding process, every elementary decoder feeds its soft output to next elementary decoder. To get a satisfactory performance, 6 to 10 iterations are generally needed.

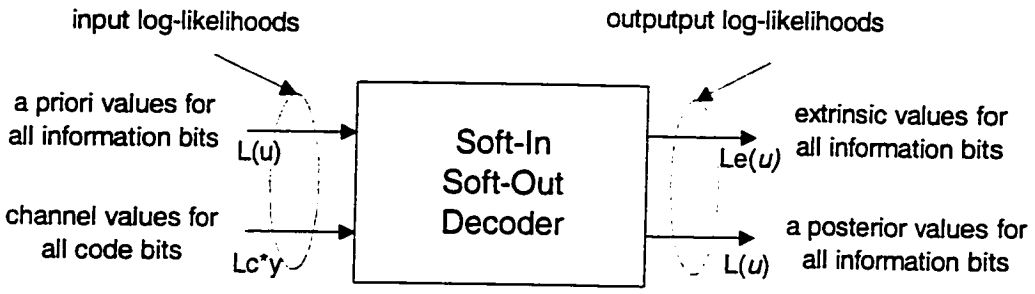

Figure 2-3 Soft-in/Soft-out decoder

For the two elementary decoder case, the first decoder takes the received channel values Lc*y and the a priori value L(u) (for the first iteration it is set to zero; after the first iteration, it comes from the extrinsic value of the second decoder) as inputs. The decoder then generates a soft output L(u) on all information bits and an extrinsic information Le(u) which contains the soft output information from all the other coded bits. The two inputs to the second decoder are the channel values with the interleaved information bits and the interleaved extrinsic information of the first decoder. The block diagram is shown in Figure 2.4. The soft output of the information bits for any decoder can be represented as [33]:

$$L_c(\hat{u}) = L_c * y + L(u) + L_e(\hat{u}) \qquad (2\text{-}2)$$

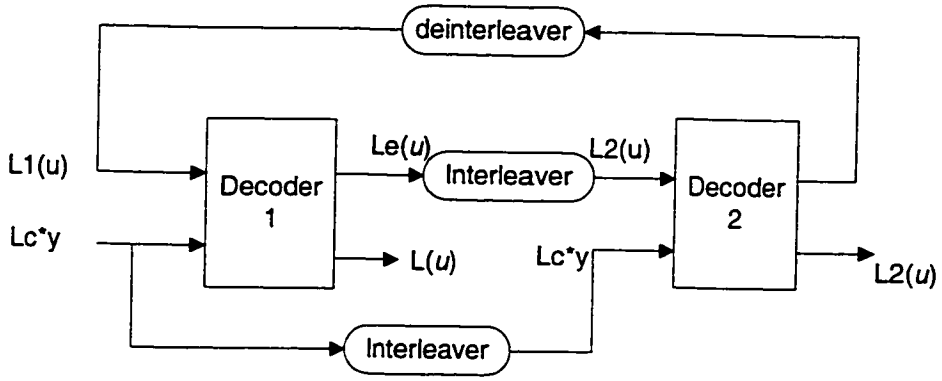

Figure 2-4 Iterative decoding procedure for two elementary codes

2.2 Non binary turbo code for DVB/RCS

DVB/RCS standard (EN 301 790 in ETSI) is a standard approved by the DVB Committee for Return Channel via Satellite to provide two-way, full-IP, asymmetric communications via satellite [56]. Since the convolutional turbo codes are very flexible, easily adaptable to a large number of data block sizes and coding rates, they have been adopted in the DVB/RCS standard. According to the standard requirement, the bit rate of the transmission of data has to reach up to 2 Mbps. The component code of the turbo code adopted by DVB/RCS standard is double binary circular recursive systematic convolutional code (CRSC), which has better performance than ordinary binary turbo code. This will be discussed later.

2.2.1 Encoder of non binary turbo code for DVB/RCS

The encoder of double binary turbo code for the DVB/RCS is depicted in Figure 2.5. First, the data sequence is encoded, and then the permutated sequence is fed into the component encoder.

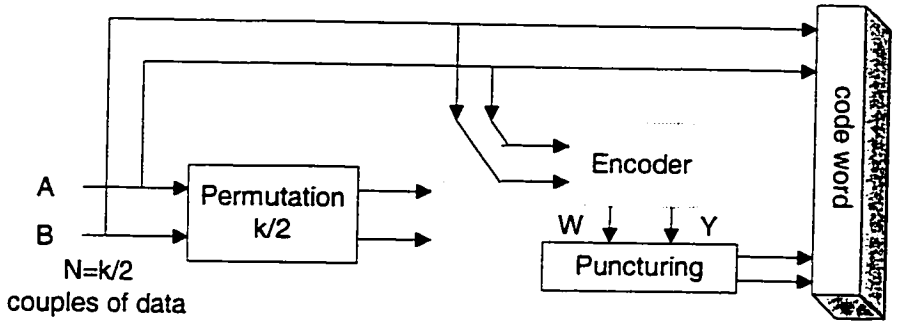

Figure 2-5 DVB/RCS double binary turbo code encoder

The component code of the DVB/RCS standard is a double binary circular recursive systematic convolutional code. The encoder, shown in Figure 2.6, is fed by blocks of k bits (k = 2*N bits). N is a multiple of 4. The polynomials defining the connections are presented in octal and symbolic notations as follows:

- for the feedback branch: 15, $1+D+D^3$;
- for the Y parity bits: 13, $1+D^2+D^3$;
- for the W parity bits: 11, $1+D^3$;

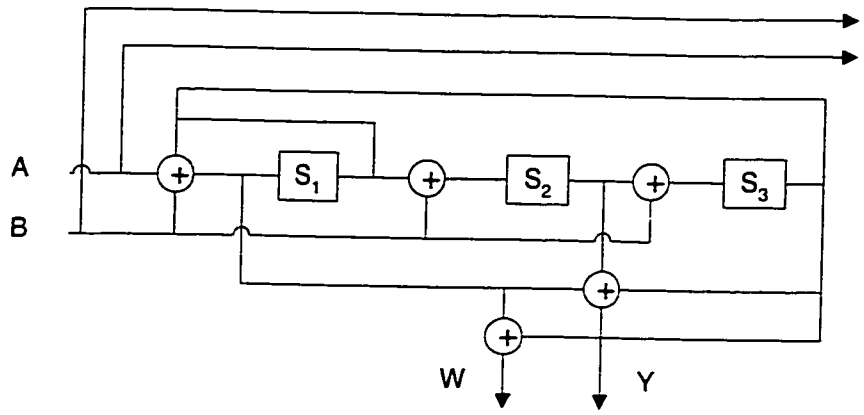

Figure 2-6 Double binary turbo code encoder

The permutation is done on two levels, level 1 is inside the couples, level 2 is between couples:

Set the permutation parameters P_0, P_1, P_2, and P_3

$j = 0, \ldots N-1$

for level 1

If j mod. 2 = 0, let (A,B) = (B,A) (invert the couple)

For level 2

$I = P_0 * j + P + 1 \bmod N$

- if j mod 4 = 0, then P = 0;
- if j mod 4 = 1, then $P = N/2 + P_1$;
- if j mod 4 = 2, then $P = P_2$;
- if j mod 4 = 3, then $P = N/2 + P_3$;

Table 2-1 Turbo code permutation parameters

Frame size in couples	P_0	$\{P_1, P_2, P_3\}$
N = 48 (12 bytes)	11	{24, 0, 24}
N = 64 (16 bytes)	7	{34, 32, 2}
N = 212 {53 bytes}	13	{106, 108, 2}
N = 220 (55 bytes)	23	{112, 4, 116}
N = 228 (57 bytes)	17	{116, 72, 188}
N = 424 {106 bytes}	11	{6, 8, 2}
N = 432 (108 bytes)	13	{0, 4, 8}
N = 440 (110 bytes)	13	{10, 4, 2}
N = 848 {212 bytes}	19	{2, 16, 6}
N = 856 (214 bytes)	19	{428, 224, 652}
N = 864 (216 bytes)	19	{2, 16, 6}
N = 752 {188 bytes}	19	{376, 224, 600}

Table 2.1 gives the combinations of the default parameters to be used. The interleaving relations obey the odd/even rule. This allows the puncturing patterns to be identical for both encoders.

The encoders are initialed by the circulation state S_{c1} or S_{c2}. The circulation states S_{c1} or S_{c2} are calculated by the following steps.

- Initialize the encoder with state 0, and then encode the sequence in the natural order for the determination of S_{c1} or encoded it in the interleaved order for the determination of S_{c2}. In both cases, the final state of the encoder is denoted S_n^0;
- According to the length N of the sequence, use the following correspondence Table 2.2 to find S_{c1} or S_{c2}.

Table 2-2 Circulation state correspondence table

S^0_{n-1} N mod. 7	0	1	2	3	4	5	6	7
1	S_c=0	S_c=6	S_c=4	S_c=2	S_c=7	S_c=1	S_c=3	S_c=5
2	S_c=0	S_c=3	S_c=7	S_c=4	S_c=5	S_c=6	S_c=2	S_c=1
3	S_c=0	S_c=5	S_c=3	S_c=6	S_c=2	S_c=7	S_c=1	S_c=4
4	S_c=0	S_c=4	S_c=1	S_c=5	S_c=6	S_c=2	S_c=7	S_c=3
5	S_c=0	S_c=2	S_c=5	S_c=7	S_c=1	S_c=3	S_c=4	S_c=6
6	S_c=0	S_c=7	S_c=6	S_c=1	S_c=3	S_c=4	S_c=5	S_c=2

Seven code rates are defined for the DVB/RCS turbo code: R = 1/3, 2/5, 1/2, 2/3, 3/4, 4/5, 6/7. This is obtained through puncturing (selectively deleting the parity bits). The puncturing patterns of Table 2.3 are used.

Table 2-3 Puncturing patterns of DVB/RCS double binary turbo code

$$1/3\begin{matrix}Y\\W\end{bmatrix}\begin{matrix}1&1&1&1&1&1&1&1\\1&1&1&1&1&1&1&1\end{matrix}\quad 2/5\begin{matrix}Y\\W\end{bmatrix}\begin{matrix}1&1&1&1&1&1&1&1\\1&0&1&0&1&0&1&0\end{matrix}$$

$$1/2\begin{matrix}Y\\W\end{bmatrix}\begin{matrix}1&1&1&1&1&1&1&1\\0&0&0&0&0&0&0&0\end{matrix}\quad 2/3\begin{matrix}Y\\W\end{bmatrix}\begin{matrix}1&0&1&0&1&0&1&0\\0&0&0&0&0&0&0&0\end{matrix}$$

$$3/4\begin{matrix}Y\\W\end{bmatrix}\begin{matrix}1&0&0&1&0&0&1&0\\0&0&0&0&0&0&0&0\end{matrix}\quad 4/5\begin{matrix}Y\\W\end{bmatrix}\begin{matrix}1&0&0&0&1&0&1&0\\0&0&0&0&0&0&0&0\end{matrix}$$

$$6/7 \begin{matrix} Y \\ W \end{matrix} \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Rate 1/3, 2/5, 1/2, 2/3 and 4/5 are exact ones, independently of the block size. Rate 3/4 and 6/7 are exact ones only if N is a multiple of 3. In other cases, the actual rates are very slightly lower than the nominal ones. Depending on the code rate, the length of the encoded block is presented in Table 2.4

Table 2-4 The length of the encoded block

Length of encoded block	= 2N + M	= N + M
Code rate	for R<1/2	for R >= 1/2
R = 1/3	M = N	
R = 2/5	M = N/2	
R = 1/2		M = N
R = 2/3		M = N/2
R = ¾		M = N/3 if N mod.3 = 0 M = (N-4)/3 + 2 if N mod.3 = 1 M = (N-4)/3 + 2 if N mod.3 = 2
R = 4/5		M = N/4
R = 6/7		M = N/6 if N mod.3 = 0 M = (N-4)/6 + 1 if N mod.3 = 1 M = (N-4)/6 + 2 if N mod.3 = 2

As for the transmission of the turbo code for DVB/RCS, two orders of transmission are applied:

- In the natural order, all couple (A, B) are transmitted first, followed by all couples (Y1, Y2), and then all couples (w1, w2).
- In the reverse order, the couples (Y1, Y2) are transmitted first, in their natural order, followed by the couples (W1, W2, and then finally by the couples (A, B).

Each couple is mapped to one QPSK constellation point.

2.2.2 Circular Recursive Systematic Converlutional (CRSC) code

Convolutional codes are not originally suited for encoding information transmitted in block form, because they have to be truncated at some point, which will degrade the coding performance. Forcing the encoder state is one way to solve this problem, but this scheme could degrade the spectral effeciency of the transmission. Adopting circular coding [35] is a scheme that not only provides another solution for above mentioned problem, but it also avoid the drawback of the scheme of forcing the encoder state. With circular convolutional codes, the encoder retrieves the initial state at the end of the encoding operation. The decoding trellis can therefore be seen as a circle and decoding may be started everywhere on the circle.

For a recursive convolutional code, at time instant i the encoder state S_i is a function of the previous state S_{i-1} and the input vector X_i. Let G be the generator matrix, the following recursive relation hold:

$$S_i = G\, S_{i-1} + X_i. \qquad (2\text{-}3)$$

If k is the input sequence length, S_k may also be expressed as a function of initial state S_0

$$S_k = G^k * S_0 + \sum_{p=1}^{k} G^{k-p} * X_p \qquad (2\text{-}4)$$

It is possible to find a state S_c such that $S_c = S_k = S_0$

$$S_c = (1+G^k)^{-1}\sum_{p=1}^{k} G^{k-p} * X_p \qquad (2\text{-}5)$$

To determine S_c, we get S_k^0 by pre-encoding the data sequence from all-zero state.

$$S_k^0 = \sum_{p=1}^{k} G^{k-p} * X_p \qquad (2\text{-}6)$$

From (2-5), the value of S_c can be linked to S_k^0 as follows:

$$S_c = (1+G^k)^{-1} S_k^0 \qquad (2\text{-}7)$$

After obtaining S_c, data sequence can be encoded starting from the circulation state.

When the circularly recursive codes are used for turbo code, there are two ways to apply the circular code. One is calculating the circulation state S_c for the whole sequence of length 2k; another way is separating the two circular codes and calculating the circulation state separately. Depending on the case, data encoding may be represented by one or two circular trellises. When the MAP algorithm is used as the decoding algorithm, decoding the sequence consists of going round the circular trellis anti-clockwise for the backward process, and clockwise for the forward process. For both processes, probabilities computed at the end of a turn are used as initial values for the next turn. In practice, a prologue decoding step is exploited, performed on a part of the circle for a few time units, to guide the process (both forward and backward) towards an initial state which is a good estimate of the circulation state.

2.3 MAP decoding algorithm

Any turbo code including turbo block codes and turbo convolutional codes can be presented by trellis. In general, there are two kinds of trellis based strategies for decoding of linear codes: the maximum likelihood (ML) algorithm like Vitrbi algorithm and maximum a posteriori probability (MAP) algorithm. When it comes to turbo codes, the soft output Viterbi algorithm, improved version of Viterbi algorithm, is used instead of

Viterbi algorithm. The soft output Viterbi algorithm is a sub-optimal decoding algorithm, producing a sub-optimal reliability measure for each received symbol, while the MAP algorithm is an optimal decoding algorithm, producing an optimal estimate of the received symbol [57].

2.3.1 MAP decoding for binary turbo code

From the trellis of a binary linear code, the log-likelihood ratio of the estimate of bit u_k is presented as [33]:

$$L(\hat{u}_k) = Log\frac{P(u_k = +1/y)}{P(u_k = -1/y)}$$

$$= Log\frac{\sum_{u_k=+1}^{(s',s)} p(s',s,y)}{\sum_{u_k=-1}^{(s',s)} p(s',s,y)} \qquad (2\text{-}8)$$

the sum of p(s',s,y) in the numerator or in the denominator is taken over all existing transitions from state s' to state s labeled with the information bit $u_k = +1$ or with $u_k = -1$, respectively. Over a memoryless transmission channel, the joint probability p(s',s,y) can be written as:

$$p(s',s,y) = p(s',y_{j<k}) \cdot p(s,y_k/s') \cdot p(y_{j>k}/s)$$

$$= \underbrace{p(s',y_{j<k})} \cdot \underbrace{p(s'/s) \cdot p(y_k/s,s')} \cdot \underbrace{p(y_{j>k}/s)}$$

$$= \alpha_{k-1}(s') \cdot \gamma_k(s',s) \cdot \beta_k(s) \qquad (2\text{-}9)$$

The forward and backward recursions are derived as

$$\alpha_k(s) = \sum_{s'} \gamma_k(s',s) \cdot \alpha_{k-1}(s') \qquad (2\text{-}10)$$

$$\beta_{k-1}(s') = \sum_{s} \gamma_k(s',s) \cdot \beta_k(s) \qquad (2\text{-}11)$$

The forward and backward recursion are initialized with $\alpha_0 = 1$ and $\beta_N = 1$. We express the coded bits as $x_{k,v}$ v=2,...n, then whenever a transition between s' and s exists, the branch transition probabilities are given by

$$\gamma_k(s',s) = p(y_k / u_k) \cdot p(u_k)$$

where

$$p(u_k = \pm 1) = \frac{e^{\pm L(u_k)}}{1+e^{\pm L(u_k)}}$$

$$= (\frac{e^{L(u_k)/2}}{1+e^{L(u_k)}}) e^{L(u_k)u_k/2}$$

$$= A_k e^{L(u_k)u_k/2} \qquad (2\text{-}12)$$

$$p(y_k / u_k) = B_k \exp(\frac{1}{2} L_c y_{k,1} u_k + \frac{1}{2}\sum_{v=2}^{n} L_c y_{k,v} x_{k,v}) \qquad (2\text{-}13)$$

Because A_k and B_k are equal for all thansition, γ_k can be expressed as

$$\gamma_k(s',s) = \exp[(\frac{1}{2}(L(u_k) + L_c y_{k,1})u_k + \frac{1}{2}\sum_{v=2}^{n} L_c y_{k,v} x_{k,v}] \qquad (2\text{-}14)$$

$$\gamma_k(s',s) = \exp[(\frac{1}{2}(L(u_k) + L_c y_{k,1})u_k] \cdot \gamma_k^{(e)} \qquad (2\text{-}15)$$

with

$$\gamma_k^{(e)}(s',s) = \exp[\frac{1}{2}\sum_{v=2}^{n} L_c y_{k,v} x_{k,v}]$$

As a result, we obtain

$$\hat{L(u_k)} = Log \frac{\sum_{u_k=+1}^{(s',s)} \gamma_k(s',s) \cdot \alpha_{k-1}(s') \cdot \beta_k(s)}{\sum_{u_k=-1}^{(s',s)} \gamma_k(s',s) \cdot \alpha_{k-1}(s') \cdot \beta_k(s)}$$

$$= L_c y_{k,1} + L(u_k) + Log \frac{\sum_{u_k=+1}^{(s',s)} \gamma_k^{(e)}(s',s) \cdot \alpha_{k-1}(s') \cdot \beta_k(s)}{\sum_{u_k=-1}^{(s',s)} \gamma_k^{(e)}(s',s) \cdot \alpha_{k-1}(s') \cdot \beta_k(s)}$$

$$= L_c y_{k,1} + L(u_k) + L_e(\hat{u_k}) \tag{2-16}$$

The $L_e(\hat{u}_k)$ is called the extrinsic value

$$L_e(\hat{u}_k) = Log \frac{\sum_{u_k=+1}^{(s',s)} \gamma_k^{(e)}(s',s) \cdot \alpha_{k-1}(s') \cdot \beta_k(s)}{\sum_{u_k=-1}^{(s',s)} \gamma_k^{(e)}(s',s) \cdot \alpha_{k-1}(s') \cdot \beta_k(s)} \tag{2-17}$$

or

$$L_e(\hat{u}_k) = L(\hat{u}) - L(u_k) - L_c y_{k,1} \tag{2-18}$$

2.3.2 MAP decoding for non binary turbo code

The turbo decoding algorithm for codes based on binary trellis requires a single Log-Likelihood Ratio(LLR). But, when it comes to non-binary codes, a multidimensional LLR (MLLR) must be defined instead indicated as $L(u_k)$ and $L^e(u_k)$, which include the three following scalar LLRs:

$$L_A(\hat{u}_k) = Log \frac{P(u_k = (0,0)/y)}{P(u_k = (0,1)/y)};$$

$$L_B(\hat{u}_k) = Log \frac{P(u_k = (0,0)/y)}{P(u_k = (1,0)/y)};$$

$$L_C(\hat{u}_k) = Log \frac{P(u_k = (0,0)/y)}{P(u_k = (1,1)/y)}; \tag{2-19}$$

$$L(\hat{u}) = \left[L_A(\hat{u}_k) \quad L_B(\hat{u}_k) \quad L_C(\hat{u}_k) \right] \tag{2-20}$$

Similar to binary case, scalar LLRs can be written as

$$L_A(\hat{u}_k) = Log \frac{\sum_{u_k=(0.0)}^{(s',s)} p(s',s,y)}{\sum_{u_k=(0.1)}^{(s',s)} p(s',s,y)} \tag{2-21}$$

$$L_B(\hat{u}_k) = Log \frac{\sum_{u_k=(0.0)}^{(s',s)} p(s',s,y)}{\sum_{u_k=(1.0)}^{(s',s)} p(s',s,y)} \tag{2-22}$$

$$L_C(\hat{u}_k) = Log \frac{\sum_{u_k=(0.0)}^{(s',s)} p(s',s,y)}{\sum_{u_k=(1.1)}^{(s',s)} p(s',s,y)} \tag{2-23}$$

where

$$p(s',s,y) = p(s', y_{j<k}) \cdot p(s, y_k / s') \cdot p(y_{j>k} / s)$$

$$= \underbrace{p(s', y_{j<k})} \cdot \underbrace{p(s'/s) \cdot p(y_k / s, s')} \cdot \underbrace{p(y_{j>k} / s)}$$

$$= \alpha_{k-1}(s') \cdot \gamma_k(s',s) \cdot \beta_k(s) \tag{2-24}$$

The forward and backward recursions are over 4 states where the transition exist:

$$\alpha_k(s) = \sum_{s'} \gamma_k(s',s) \cdot \alpha_{k-1}(s') \tag{2-25}$$

$$\beta_{k-1}(s') = \sum_{s} \gamma_k(s',s) \cdot \beta_k(s) \tag{2-26}$$

The forward and backward path metrics are initialized with equal probability for all states.

$$\gamma_k^i(s',s) = \exp[(\frac{1}{2} L_c y_{k,1}) u_k + \frac{1}{2} \sum_{v=2}^{n} L_c y_{k,v} x_{k,v}] \cdot P(u_k) \tag{2-27}$$

$$\gamma_k^{(e)}(s',s) = \exp[\frac{1}{2} \sum_{v=2}^{n} L_c y_{k,v} x_{k,v}] \tag{2-28}$$

where i, u_k, $y_{k,v}$, $x_k \in$ (00, 01, 10,11).

By putting (2-4) in (2-21) – (2-23), we acquired

$$L_A(\hat{u}_k) = Log \frac{\sum_{u_k=(0.0)}^{(s',s)} \gamma_k(s',s) \cdot \alpha_{k-1}(s') \cdot \beta_k(s)}{\sum_{u_k=(0.1)}^{(s',s)} \gamma_k(s',s) \cdot \alpha_{k-1}(s') \cdot \beta_k(s)} \tag{2-29}$$

$$L_B(\hat{u}_k) = Log \frac{\sum_{u_k=(0.0)}^{(s',s)} \gamma_k(s',s) \cdot \alpha_{k-1}(s') \cdot \beta_k(s)}{\sum_{u_k=(1.0)}^{(s',s)} \gamma_k(s',s) \cdot \alpha_{k-1}(s') \cdot \beta_k(s)} \tag{2-30}$$

$$L_C(\hat{u}_k) = Log \frac{\sum_{u_k=(0.0)}^{(s',s)} \gamma_k(s',s) \cdot \alpha_{k-1}(s') \cdot \beta_k(s)}{\sum_{u_k=(1.1)}^{(s',s)} \gamma_k(s',s) \cdot \alpha_{k-1}(s') \cdot \beta_k(s)} \tag{2-31}$$

and the extrinsic values are,

$$L_A^{(e)}(\hat{u}_k) = Log \frac{\sum_{u_k=(0.0)}^{(s',s)} \gamma_k^{(e)}(s',s) \cdot \alpha_{k-1}(s') \cdot \beta_k(s)}{\sum_{u_k=(0.1)}^{(s',s)} \gamma_k^{(e)}(s',s) \cdot \alpha_{k-1}(s') \cdot \beta_k(s)} \tag{2-32}$$

$$L_B^{(e)}(\hat{u}_k) = Log \frac{\sum_{u_k=(0.0)}^{(s',s)} \gamma_k^{(e)}(s',s) \cdot \alpha_{k-1}(s') \cdot \beta_k(s)}{\sum_{u_k=(1.0)}^{(s',s)} \gamma_k^{(e)}(s',s) \cdot \alpha_{k-1}(s') \cdot \beta_k(s)} \tag{2-33}$$

$$L_C^{(e)}(\hat{u}_k) = Log \frac{\sum_{u_k=(0.0)}^{(s',s)} \gamma_k^{(e)}(s',s) \cdot \alpha_{k-1}(s') \cdot \beta_k(s)}{\sum_{u_k=(1.1)}^{(s',s)} \gamma_k^{(e)}(s',s) \cdot \alpha_{k-1}(s') \cdot \beta_k(s)} \tag{2-34}$$

2.3.3 Log-MAP decoding algorithm

To simplify the calculation, we can transform the MAP algorithm to Log domain, then we obtain the Log-MAP algorithm. In Log-MAP algorithm, all the multiplications are converted into additions, and also the Jacobian algorithm could be used to simplify the calculations. The decoding performance of the Log-MAP algorithm should be the same as the MAP algorithm.

$$Log(e^{\delta_1} + e^{\delta_2}) = \max(\delta_1, \delta_2) + Log(1 + e^{-|\delta_2 - \delta_1|})$$
$$= \max(\delta_1, \delta_2) + f_c(|\delta_1 - \delta_2|) \quad (2\text{-}35)$$

where $f_c(\cdot)$ is a correction function, which can be implement using a look-up table.

In the binary case, all expressions for Log-MAP algorithm can be written as:

$$\gamma_k^L(s', s) = Log[\gamma_k(s', s)]$$
$$= (\frac{1}{2}(L(u_k) + L_c y_{k,1})u_k + \frac{1}{2}\sum_{v=2}^{n} L_c y_{k,v} x_{k,v} \quad (2\text{-}36)$$

$$\gamma_k^{(e)L}(s', s) = Log[\gamma_k^{(e)}(s', s)]$$
$$= \frac{1}{2}\sum_{v=2}^{n} L_c y_{k,v} x_{k,v} \quad (2\text{-}37)$$

$$\alpha_k^L(s) = Log[\alpha_k(s)]$$
$$= Log\sum_{s'} e^{\gamma_k^L(s',s) + \alpha_{k-1}^L(s')} \quad (2\text{-}38)$$

$$\beta_{k-1}^L(s') = Log\beta_{k-1}(s')$$
$$= Log\sum_{s} e^{\gamma_k^L(s',s) + \beta_k^L(s)} \quad (2\text{-}39)$$

$$\hat{L(u_k)} = Log \frac{\sum_{u_k=+1}^{(s',s)} e^{\gamma_k^L(s',s)+\alpha_{k-1}^L(s')+\beta_k^L(s)}}{\sum_{u_k=-1}^{(s',s)} e^{\gamma_k^L(s',s)+\alpha_{k-1}^L(s')+\beta_k^L(s)}}$$

$$= Log \sum_{u_k=+1}^{(s',s)} e^{\gamma_k^L(s',s)+\alpha_{k-1}^L(s')+\beta_k^L(s)} - Log \sum_{u_k=-1}^{(s',s)} e^{\gamma_k^L(s',s)+\alpha_{k-1}^L(s')+\beta_k^L(s)} \qquad (2\text{-}40)$$

$$\hat{L_e(u_k)} = Log \frac{\sum_{u_k=+1}^{(s',s)} e^{\gamma_k^{(e)L}(s',s)+\alpha_{k-1}^L(s')+\beta_k^L(s)}}{\sum_{u_k=-1}^{(s',s)} e^{\gamma_k^{(e)L}(s',s)+\alpha_{k-1}^L(s')+\beta_k^L(s)}}$$

$$= Log \sum_{u_k=+1}^{(s',s)} e^{\gamma_k^{(e)L}(s',s)+\alpha_{k-1}^L(s')+\beta_k^L(s)} - Log \sum_{u_k=-1}^{(s',s)} e^{\gamma_k^{(e)L}(s',s)+\alpha_{k-1}^L(s')+\beta_k^L(s)} \qquad (2\text{-}41)$$

In the non-binary case, all expressions for Log-MAP algorithm can be written as:

$$\gamma_k^{iL}(s',s) = Log[\gamma_k^i(s',s)]$$

$$= \frac{1}{2} L_c y_{k,1} u_k + \frac{1}{2} \sum_{v=2}^{n} L_c y_{k,v} x_{k,v} + LogP(u_k) \qquad (2\text{-}42)$$

$$\gamma_k^{i(e)L}(s',s) = Log[\gamma_k^{i(e)}(s',s)]$$

$$= \frac{1}{2} \sum_{v=2}^{n} L_c y_{k,v} x_{k,v} \qquad (2\text{-}43)$$

$$\alpha_k^L(s) = Log[\alpha_k(s)] = Log \sum_{s'} e^{\gamma_k^i(s',s)+\alpha_{k-1}^L(s')} \qquad (2\text{-}44)$$

$$\beta_{k-1}^L(s') = Log\beta_{k-1}(s') = Log \sum_{s} e^{\gamma_k^i(s',s)+\beta_k^L(s)} \qquad (2\text{-}45)$$

$$L_A(\hat{u}_k) = Log \frac{\sum_{u_k=00}^{(s',s)} e^{\gamma_k^L(s',s)+\alpha_{k-1}^L(s')+\beta_k^L(s)}}{\sum_{u_k=01}^{(s',s)} e^{\gamma_k^L(s',s)+\alpha_{k-1}^L(s')+\beta_k^L(s)}}$$

$$= Log \sum_{u_k=00}^{(s',s)} e^{\gamma_k^L(s',s)+\alpha_{k-1}^L(s')+\beta_k^L(s)} - Log \sum_{u_k=01}^{(s',s)} e^{\gamma_k^L(s',s)+\alpha_{k-1}^L(s')+\beta_k^L(s)} \quad (2\text{-}46)$$

$$L_B(\hat{u}_k) = Log \sum_{u_k=00}^{(s',s)} e^{\gamma_k^L(s',s)+\alpha_{k-1}^L(s')+\beta_k^L(s)} - Log \sum_{u_k=10}^{(s',s)} e^{\gamma_k^L(s',s)+\alpha_{k-1}^L(s')+\beta_k^L(s)} \quad (2\text{-}47)$$

$$L_C(\hat{u}_k) = Log \sum_{u_k=00}^{(s',s)} e^{\gamma_k^L(s',s)+\alpha_{k-1}^L(s')+\beta_k^L(s)} - Log \sum_{u_k=11}^{(s',s)} e^{\gamma_k^L(s',s)+\alpha_{k-1}^L(s')+\beta_k^L(s)} \quad (2\text{-}48)$$

$$L_A^e(\hat{u}_k) = Log \frac{\sum_{u_k=00}^{(s',s)} e^{\gamma_k^{(e)L}(s',s)+\alpha_{k-1}^L(s')+\beta_k^L(s)}}{\sum_{u_k=01}^{(s',s)} e^{\gamma_k^{(e)L}(s',s)+\alpha_{k-1}^L(s')+\beta_k^L(s)}}$$

$$= Log \sum_{u_k=00}^{(s',s)} e^{\gamma_k^{(e)L}(s',s)+\alpha_{k-1}^L(s')+\beta_k^L(s)} - Log \sum_{u_k=01}^{(s',s)} e^{\gamma_k^{(e)L}(s',s)+\alpha_{k-1}^L(s')+\beta_k^L(s)} \quad (2\text{-}49)$$

$$L_B^e(\hat{u}_k) = Log \sum_{u_k=00}^{(s',s)} e^{\gamma_k^{(e)L}(s',s)+\alpha_{k-1}^L(s')+\beta_k^L(s)} - Log \sum_{u_k=10}^{(s',s)} e^{\gamma_k^{(e)L}(s',s)+\alpha_{k-1}^L(s')+\beta_k^L(s)} \quad (2\text{-}50)$$

$$L_C^e(\hat{u}_k) = Log \sum_{u_k=00}^{(s',s)} e^{\gamma_k^{(e)L}(s',s)+\alpha_{k-1}^L(s')+\beta_k^L(s)} - Log \sum_{u_k=11}^{(s',s)} e^{\gamma_k^{(e)L}(s',s)+\alpha_{k-1}^L(s')+\beta_k^L(s)} \quad (2\text{-}51)$$

2.3.4 Max-Log-MAP decoding algorithm

Omitting the Jacobian correction function in the Log_MAP algorithm, we attain the Max-Log-MAP algorithm. The Max-Log-MAP algorithm has a degraded performance, so it is a sub-optimal algorithm, but the implementation complexity is greatly reduced. A quantized version of the Jacobian function can be added to compensate the loss. The resulting algorithm is the Max*_Log_MAP algorithm.

In the binary case, the Max-Log-MAP algorithm can be presented as:

$$\gamma_k^{ML}(s',s) = (\frac{1}{2}(L(u_k) + L_c y_{k,1})u_k + \frac{1}{2}\sum_{v=2}^{n} L_c y_{k,v} x_{k,v} \quad (2\text{-}52)$$

$$\gamma_k^{(e)ML}(s',s) = \frac{1}{2}\sum_{v=2}^{n} L_c y_{k,v} x_{k,v} \quad (2\text{-}53)$$

$$\alpha_k^{ML}(s) = \underset{s'}{Max}[\gamma_k^{ML}(s',s) + \alpha_{k-1}^{ML}(s')] \quad (2\text{-}54)$$

$$\beta_{k-1}^{ML}(s') = \underset{s'}{Max}[\gamma_k^{ML}(s',s) + \beta_k^{ML}(s)] \quad (2\text{-}55)$$

$$\hat{L(u_k)} = \underset{(s',s)u_k=+1}{Max}[\gamma_k^{ML}(s',s) + \alpha_{k-1}^{ML}(s') + \beta_k^{ML}(s)] - \underset{(s',s)u_k=-1}{Max}[\gamma_k^{ML}(s',s) + \alpha_{k-1}^{ML}(s') + \beta_k^{ML}(s)] \quad (2\text{-}56)$$

$$\hat{L_e(u_k)} = \underset{(s',s)u_k=+1}{Max}[\gamma_k^{(e)ML}(s',s) + \alpha_{k-1}^{ML}(s') + \beta_k^{ML}(s)] - \underset{(s',s)u_k=-1}{Max}[\gamma_k^{(e)ML}(s',s) + \alpha_{k-1}^{ML}(s') + \beta_k^{ML}(s)] \quad (2\text{-}57)$$

In the non-binary case, the Max-Log-MAP algorithm can be depicted as:

$$\gamma_k^{iML}(s',s) = \frac{1}{2} L_c y_{k,1} u_k + \frac{1}{2}\sum_{v=2}^{n} L_c y_{k,v} x_{k,v} + LogP(u_k) \quad (2\text{-}58)$$

$$\gamma_k^{i(e)ML}(s',s) = \frac{1}{2}\sum_{v=2}^{n} L_c y_{k,v} x_{k,v} \quad (2\text{-}59)$$

$$\alpha_k^{ML}(s) = \underset{s'}{Max}[\gamma_k^{ML}(s',s) + \alpha_{k-1}^{ML}(s')] \quad (2\text{-}60)$$

$$\beta_{k-1}^{ML}(s') = \underset{s'}{Max}[\gamma_k^{ML}(s',s) + \beta_k^{ML}(s)] \quad (2\text{-}61)$$

$$\hat{L_A(u_k)} = \underset{(s',s)u_k=00}{Max}[\gamma_k^{ML}(s',s) + \alpha_{k-1}^{ML}(s') + \beta_k^{ML}(s)] - \underset{(s',s)u_k=01}{Max}[\gamma_k^{ML}(s',s) + \alpha_{k-1}^{ML}(s') + \beta_k^{ML}(s)] \quad (2\text{-}62)$$

$$L_B(\hat{u}_k) = \underset{(s',s)u_k=00}{Max}[\gamma_k^{ML}(s',s) + \alpha_{k-1}^{ML}(s') + \beta_k^{ML}(s)]$$

$$- \underset{(s',s)u_k=10}{Max}[\gamma_k^{ML}(s',s) + \alpha_{k-1}^{ML}(s') + \beta_k^{ML}(s)] \quad (2\text{-}63)$$

$$L_C(\hat{u}_k) = \underset{(s',s)u_k=00}{Max}[\gamma_k^{ML}(s',s) + \alpha_{k-1}^{ML}(s') + \beta_k^{ML}(s)]$$

$$- \underset{(s',s)u_k=11}{Max}[\gamma_k^{ML}(s',s) + \alpha_{k-1}^{ML}(s') + \beta_k^{ML}(s)] \quad (2\text{-}64)$$

$$L_A^e(\hat{u}_k) = \underset{(s',s)u_k=00}{Max}[\gamma_k^{(e)ML}(s',s) + \alpha_{k-1}^{ML}(s') + \beta_k^{ML}(s)]$$

$$- \underset{(s',s)u_k=01}{Max}[\gamma_k^{(e)ML}(s',s) + \alpha_{k-1}^{ML}(s') + \beta_k^{ML}(s)] \quad (2\text{-}65)$$

$$L_B^e(\hat{u}_k) = \underset{(s',s)u_k=00}{Max}[\gamma_k^{(e)ML}(s',s) + \alpha_{k-1}^{ML}(s') + \beta_k^{ML}(s)]$$

$$- \underset{(s',s)u_k=10}{Max}[\gamma_k^{(e)ML}(s',s) + \alpha_{k-1}^{ML}(s') + \beta_k^{ML}(s)] \quad (2\text{-}66)$$

$$L_C^e(\hat{u}_k) = \underset{(s',s)u_k=00}{Max}[\gamma_k^{(e)ML}(s',s) + \alpha_{k-1}^{ML}(s') + \beta_k^{ML}(s)]$$

$$- \underset{(s',s)u_k=11}{Max}[\gamma_k^{(e)ML}(s',s) + \alpha_{k-1}^{ML}(s') + \beta_k^{ML}(s)] \quad (2\text{-}67)$$

From following equations

$$L_A^e = LogP_0 - LogP_1$$

$$L_B^e = LogP_0 - LogP_2$$

$$L_C^e = LogP_0 - LogP_3 \quad (2\text{-}68)$$

where

$P_{0k} = P(u_k = 00)$, $P_{1k} = P(u_k = 01)$, $P_{2k} = P(u_k = 10)$, $P_{3k} = P(u_k = 11)$.

It is easy to get the probabilities approximately

$$LogP_0 = Min(L_A^e, Le_B^e, L_C^e) \quad (2\text{-}69)$$

$$LogP_1 = Min(-L_A^e, -L_A^e + Le_B^e, -L_A^e + L_C^e) \quad (2\text{-}70)$$

$$LogP_2 = Min(-L_B^e, -L_B^e + Le_A^e, -L_B^e + L_C^e) \quad (2\text{-}71)$$

$$LogP_3 = Min(-L_C^e, -L_C^e + Le_A^e, -L_C^e + L_B^e) \qquad (2\text{-}72)$$

Chapter 3

Implementation issues of DVB/RCS code

In general, the most efficient hardware implementation of the DVB/RCS turbo code means to reach the best performance in terms of speed, area, and low power consumption without loss of error correction capability. But it seems impossible to get both the best implementation performance and the best error correction capability an the same time. Thus, the most efficient implementation is always the tradeoff between hardware complexity and decoding ability. In order to achieve the expected performance, simplifications must be attempted at different abstraction levels (e.g. system, architecture, RTL, gate, transistor level). The simplifications of the two lowest levels are often made with synthesis tools provided by specialized hardware device design companies, while the simplifications related to the higher levels are implemented by the designers who have deep knowledge of coding theory. In other words, Application knowledge can be exploited to significantly simplify high-level design towards lower implementation complexity. So in this thesis, we focus on the high-level issues. In this chapter, the top-

most level (system level) issues are mainly discussed, while architecture level and RTL level issues will be presented in Chapter 4.

3.1 Algorithm simplification

In hardware design, The MAP algorithm is too difficult to implement although it has the best error correction performance. Basically the difficulties of implementation with MAP algorithm include the numerical representation of probability, non-linear functions and mixed multiplication and additions of these values. Converting the MAP into the logarithmic domain and substituting the logarithms by Jacobian logarithms yields the so-called Log-MAP algorithm. The Log-MAP avoids the numerical problems of the MAP and the performance of the Log-MAP is equivalent to MAP [58]. So, from the implementation point of view, the MAP algorithm should always be implemented in the logarithmic domain. By omitting the correction term of the Jacobian logarithms, the Max-Log-MAP algorithm is obtained. According to simulation result, the Max-Log-MAP is similar to Log-MAP in decoding performance for DVB/RCS code [59]. This is the reason why Max-Log-MAP is adopted in this report.

3.2 Quantization

All decoding algorithms including Max-Log-MAP are usually specified in the floating-point domain. To get an efficient implementation, fixed-point number representation has to be used, which implies transformation from floating-point to fixed-point is mandatory for the decoder design. The primary goal of this transformation or quantization design for hardware implementation is to find a fixed-point model that has all bits-widths as small as possible under the condition of an acceptable degradation of the coding performance. In hardware implementation, the reduction of data-path bit-widths, control complexity and memory size leads to a reduction of area and power

consumption and an increase of speed. The input data quantization and inner data quantization have major influence on the control complexity and directly determine the bit-widths of data-path and size of memory. The smaller the bit-widths of quantization, the better the performance of the decoder in terms of speed, area and the energy consumption. Obviously quantization also affects the decoding performance. Thus, the optimized quantization is a key factor for the implementation cost.

3.2.1 Input data quantization

As mentioned before, quantization of the input data is very important for all kinds of turbo decoders. What we are interested in is how various quantization schemes affect the performance of the decoder. The simplest way to find the best quantization scheme is simulation approach, in which the best choice is decided according to the simulation result.

The uniform quantation is often adopted in decoder design. Other fancy quantization methods will not be helpful to improve the quantization results for decoders. Normally, q : f denotes a quantization scheme, in which q presents the total bits and f presents the fractional part of the total bits. For binary turbo code the schemes such as 3:1, 3:2, 4:2, 4:1, 4:3, 5:2 and 5:3 for Max-Log-MAP algorithm have been investigated [60][61]. From the simulation result it can be found that the difference between the 4:2 case and the 5:2 case is negligible within a wide range of BERs. However, the difference between the 4:2 and the 3:1 quantization schemes is quite significant. Also the 4:1 scheme performs worse than the 4:2 scheme because the performance degradation due to its lower precision is more significant than the performance improvement by its larger dynamic range. Thus, the 4:2 scheme is the best scheme, which has the best tradeoff between implementation complexity and code performance.

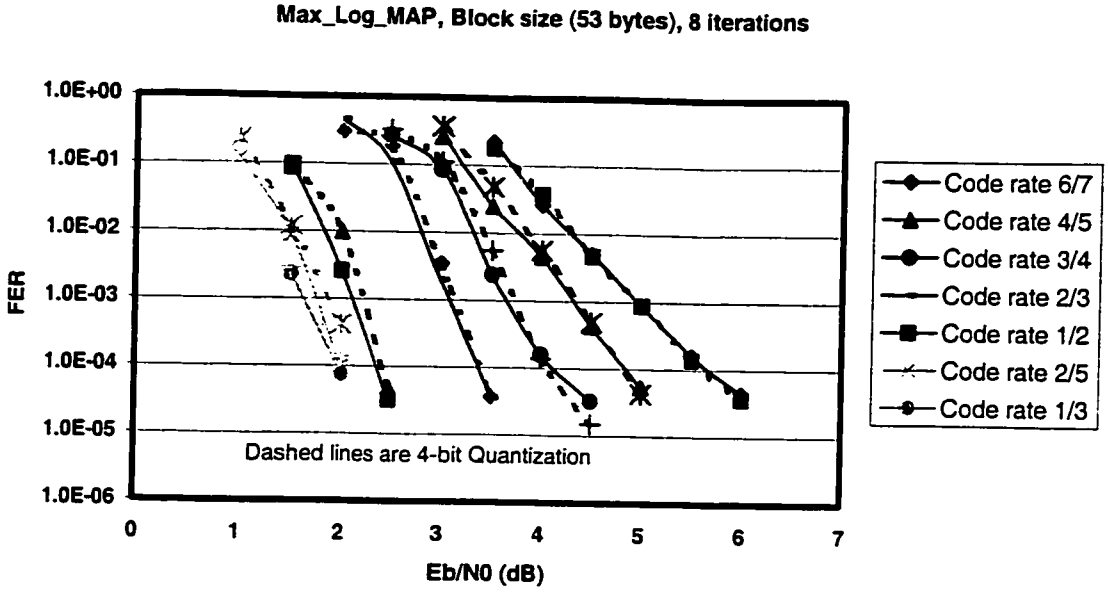

Figure 3-1 The Frame Error Rate for ATM blocks, 8 iterations

As for DVB/RCS non-binary turbo code the simulation results are similar to binary turbo code. Figure 3-1 and Figure 3-2 [59] show the simulation results of DVB/RCS turbo code with the 4:2 case and no quantization case for Max-Log-MAP decoder. It can be seen that the degradation of the code performance for the 4:2 scheme compared to the one for no quantization case is very small. So, the 4:2 quantization scheme is used in this report.

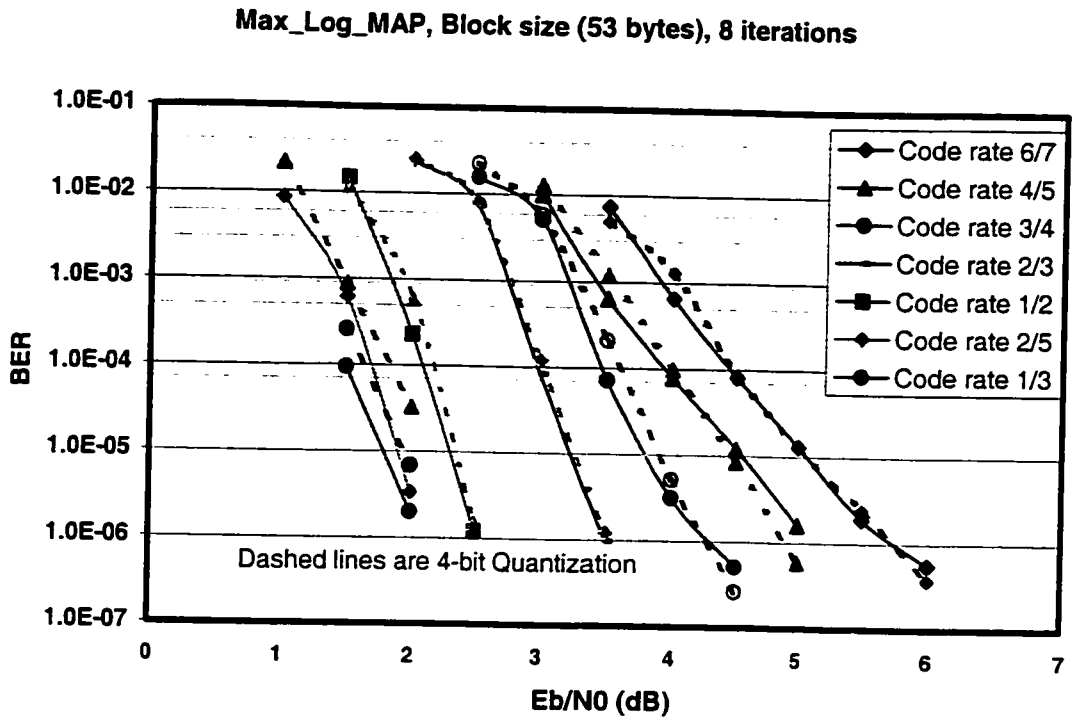

Figure 3-2 Bit Error Rate for ATM blocks, 8 iterations

3.2.2 Inner data Quantization

Inner data quantization refers to quantization of extrinsic information (L^e), the forward path metrics (α), the backward path metrics (β) and branch metrics (γ). In order to simplify the implementation of the decoders, in most cases the same bit width is chosen for all the inner data if there is no big performance difference.

L^e values are the a priori probability for the next iteration. In Log domain the absolute value of L^e could be very large, which can be observed in the simulation. This means a big bit width has to be chosen. Fortunately, when L^e is very large, the probability will be very close to 1.0. So, the effects on the next iteration could be expected to be very small. Therefore, the bit width for L^e turns out not to be so large that it can be a hurdle for

hardware implementation. Various quantization schemes for L^e such as 5:2, 5:1, 6:2, and 6:1 have been examined [60]. The 6:2 scheme is proved to be the optimal choice.

The bit widths of α and β are the most significant factors influencing the speed, the area and the power consumption of the decoder implementation. Both an analytical approach and a simulation approach can be used to explore the optimal quantization scheme. As discussed before, α and β are accumulated and are recursively computed. To prevent arithmetic overflow, the normalization schemes must be employed, which will be discussed in detail in next section. For the Max-Log-MAP algorithm actually only the difference values of all alphas and betas are needed so the normalization is made possible. Till now many efforts have been made to obtain the best scheme [62][63]. For the Log-MAP algorithm the single sized 9:4 scheme is the best choice and for the Max-Log-MAP algorithm the single sized 8:3 scheme performs the best. Even both of these two schemes are better than the corresponding infinite precision model slightly for five and above five decoding iterations in decoding performance under the assumption of a AWGN channel and a Rayleigh-fading channel.

In this thesis, for all the inner data we choose a double side 8:2 quantization scheme, setting the maximum extrinsic information value to 2^{-5} and for the input data we choose the 4:2 quantization scheme. Roughly speaking according to above discussion this practical quantization model will not degrade the decoding performance.

3.3 Normalization

The forward path metrics and the backward path metrics require periodic re normalization to prevent arithmetic overflow because they can be accumulated without bounds during Max-Log-MAP algorithm recursion. Basically there are two approaches that can be used for renormalization: rescaling approach and modulo approach. In this section, these two approaches will be introduced first, and then a new optimal

normalization approach for DVB/RCS non binary code will be proposed, which combines all advantages of the rescaling approach and modulo approach and overcomes all the disadvantage of these two approaches.

3.3.1 Rescaling

For the Max-Log-MAP decoder the rescaling renormalization can be achieved by subtracting the minimum path metric in each time step [61]. Since the useful information is contained within the differences of the path metrics, the normalization does not cause any change of the Max-Log-MAP algorithm. The advantages of this approach is as follows:

- Rescaling can be easily combined with saturating the path metrics in order to minimize bit widths. As stated above, the 8:2 quantization scheme can be employed not causing the degradation of coding performance.
- From reliability point of view, it is easy to implement because saturation provide more reliability.

The disadvantages of the rescaling approach include:

- Renormalization by subtracting the minimum metric causes a prolonged critical path for a hardware implementation and therefore a degraded maximum decoding speed. We cannot solve this drawback by pipelining the design because data hazards exist. As we know, since a loop exists in path metric calculation, if we do not open the loop, we cannot avoid these data hazards.
- Additional subtraction increases the area of the hardware implementation.

3.3.2 Modulo Normalization

An alternative renormalization technique is modulo scheme, which uses tow's complement arithmetic [64][65]. The key idea of modulo renormalization is not to invest how to avoid overflow, but instead to accommodate overflow in such a way that it does not affect the correctness of the result. In modulo renormalization the two properties of the Max-Log-MAP algorithm are exploited:

- The useful information is contained within the differences of the path metrics.
- The difference between metrics is bounded [64].

In two's complement arithmetic the bit width c refers to the additive group over

$$\{-2^{c-1}, 1-2^{c-1}, \ldots, 2^{c-1}-1\}$$

and in modulo renormalization the modulo operators are used instead of the normal operators. It is obvious that the bit width c has to be enough to allow the absolute value of the biggest metric Δ_{max} difference and satisfies

$$2^{c-1} \geq |\Delta_{max}| \tag{3-1}$$

Let B be an upper bound for the absolute values of the signed branch metrics:

$$|\gamma| \leq B \tag{3-2}$$

For DVB/RCS turbo code m = k – 1 equals 3 being the memory order of a code with constraint length k. The difference between any pair of forward path metrics (α) or any pair of backward path metrics (β) is then bounded [28]:

$$|\alpha_k(s_1) - \alpha_k(s_2)| \leq 6B \tag{3-3}$$

$$|\beta_k(s_1) - \beta_k(s_2)| \leq 6B, \quad s_1, s_2 \in S. \tag{3-4}$$

Similarly the candidate path metrics are upper bounded as follows:

$$|\alpha_k(s_1) + \gamma_1(s_1,s_1') - (\alpha_k(s_2) + \gamma_1(s_2,s_2'))| \leq 8B \tag{3-5}$$

$$s_1, s_2, s_1', s_2' \in S.$$

Thus, the bit widths for the forward path metrics and the candidate path metrics can be obtained:

$$c_{pm} = \log_2 6B + 1 \quad (3\text{-}6)$$

$$c_{cm} = \log_2 8B + 1 \quad (3\text{-}7)$$

These upper bounds are not very tight for hardware implementation because we want the bit width as small as possible. To get the smallest value for DVB/RCS code, simulation has been done [59] and the results are presented in Table 3-1. It can be seen the real bit width for this code should be 11, which is too big for hardware design.

Table 3-1 Bit width of inner data for DVB/RCS code

4 bit quantization

Iteration Num	Min γ	Min α	Min β	Max Lu
1 Iteration		-348	-348	626
2 Iteration		-576	-576	1024
3 Iteration		-688	-688	1344
4 Iteration		-864	-864	1664
5 Iteration		-992	-992	1920
6 Iteration		-1152	-1152	2288
7 Iteration		-1280	-1280	2544
8 Iteration		-1392	-1392	2768

3 bit quantization

Iteration Num	Min γ	Min α	Min β	Max Lu
1 Iteration	-64	-96	-96	160
2 Iteration	-112	-128	-128	240
3 Iteration	-160	-176	-176	352
4 Iteration	-192	-208	-208	416
5 Iteration	-224	-240	-240	480
6 Iteration	-240	-272	-272	544
7 Iteration	-288	-304	-304	608
8 Iteration	-366	-336	-366	664

From the above discussion the characteristics of modulo renormalization can be concluded:

- Modulo renormalization avoids the major drawback of rescaling scheme, renormalization by subtraction, because the metrics stay implicitly normalized.
- Without subtraction the hardware circuits can work at a higher speed, and the area for subtraction is saved.
- The disadvantage of this approach is that the bit width needed is larger than rescaling scheme. Normally, for high-speed MAP decoder one additional bit results in approximately 25% decrease in speed and increase in area.

3.3.3 New optimized normalization for DVB/RCS code

In order to get a better decoding performance a new renormalization approach, based on the rescaling scheme, is proposed, which overcomes the disadvantages of both the rescaling and the modulo renormalization.

If the bit width we choose is c, the biggest absolute value of the path metrics should be 2^{c-1}. Let us assume the path metrics $\alpha_{k-1}(s) < 2^{c-2}$, then the candidate path metrics for next time unit

$$\left|\alpha_{k-1}(s_{k-1,i}) + \gamma_i(y_k, s_{k-1,i}, s_k)\right| < 2^{c-2} + B \qquad (3\text{-}8)$$

If we limit the extrinsic information within a proper value, the following inequality can hold easily.

$$B \le 2^{c-2} \qquad (3\text{-}9)$$

Therefore,

$$\left|\alpha_{k-1}(s_{k-1,i}) + \gamma_i(y_k, s_{k-1,i}, s_k)\right| < 2^{c-1} \qquad (3\text{-}10)$$

If we further limit $B \le 2^{c-3}$, under the condition of the previous path metric $\alpha_{k-2} \le 2^{c-2}$, the candidate path metrics for next time unit $< 2^{c-1}$.

As long as the candidate path metrics are less than 2^{c-1} under the condition of the path metrics of the previous time unit less than 2^{c-2}, the approach described by the following steps can be used for renormalization instead of rescaling scheme:

- At each time instant, all path metrics are checked to see if any of the metrics is larger than 2^{c-2}. If it is larger than 2^{c-2}, 2^{c-2} is subtracted from all path metrics for all states at this time unit, otherwise, all path metrics are kept as they are. In hardware implementation a very simple combinational logic can be used to fulfill it. Using this approach, we simplified the renormalization greatly compared to the rescaling scheme renormalization; the speed of the decoder can be faster; the area of the decoder can be reduced. But, compared to the modulo renormalization, this approach

still has to spend some time to implement the comparison of the path metrics with 2^{c-2} and the subtraction of the path metrics with 2^{c-2}. So, it is not the optimized normalization.

- To get rid of the compare part from the critical path in terms of time, the previous path metrics is used instead of the present time unit path metrics. Of course, this substitution could cause one unit time delay for the normalization. But according to (3-8), (3-9), (3-10) this delay could not produce any error. And then, in hardware implementation the compare circuit can be paralleled with the addition circuit of ACS. Now in the timing graph the comparison disappears.
- The last hurdle left is the subtraction. To eliminate the subtraction from the critical path we replace all path metrics with the corresponding candidate path metrics and parallel the subtraction logic circuit with the compare circuit of the ACS. As the subtraction logic is very simple, the execution time of the subtraction must be less than the execution time of the compare circuit of the ACS.

Now the optimized renormalization has been achieved, which has the same bit width as rescsaling scheme and the same length of datapath as modulo renormalization. With this new optimized renormalization approach the decoder performance in terms of speed can be improved by 30% to 40% compared to either scheme.

3.4 Simplification of branch metrics Calculation

Unlike the alpha and the beta calculation, the branch metric gamma calculations for the Max-Log-MAP algorithm do not affect the speed of the decoder in hardware implementation. But, it does affect the area of the decoder and the power consumption as well. Therefore, the calculation simplification of gamma is necessary for hardware implementation.

3.4.1 Gamma calculation of DVB/RCS turbo code

For Max-Log-MAP algorithm the branch metric $\gamma_i(s_{k-1}, s_k)$ depends on the received symbols and the extrinsic information, and the extrinsic branch metric $\gamma_i^{(e)}(s_{k-1}, s_k)$ only depends on the coded information:

$$\gamma_i(s_{k-1}, s_k) = L_e + \frac{1}{2}\sum_{v=1}^{n} L_c * y_{k1,v} * x_{k,v} \tag{3-11}$$

$$\gamma_i^{(e)}(s_{k-1}, s_k) = \frac{1}{2}\sum_{v=3}^{n} L_c * y_{k1,v} * x_{k,v} \tag{3-12}$$

For DVB/RCS non binary code,

$i \in \{00, 01, 10, 11\}$

$s_k \in \{000, 001, 010, 011, 100, 101, 110, 111\}$

The state vecter S_i, input vecter X_i and matrix G are given by

$$S_i = \begin{bmatrix} s_{1,i} \\ s_{2,i} \\ s_{3,i} \end{bmatrix};$$

$$X_i = \begin{bmatrix} A_i + B_i \\ B_i \\ B_i \end{bmatrix};$$

$$G = \begin{bmatrix} 1 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix};$$

the state transfer function can be inferred as:

$$S_i = GS_{i-1} + X_i \tag{3-13}$$

Further more, if the output vector Yi and metric G1, G2 are expressed as,

$$Y_i = \begin{bmatrix} y \\ w \end{bmatrix};$$

$$G_1 = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix};$$

$$G_2 = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix};$$

Table 3-2 State transfer and encoder output

Current State	Input bits	Next State	Output Y W		Current State	Input bits	Next State	Output Y W
000	00	000	00		100	00	110	11
000	01	111	11		100	01	001	00
000	10	100	11		100	10	010	00
000	11	011	00		100	11	101	11
001	00	100	00		101	00	010	11
001	01	011	11		101	01	101	00
001	10	000	11		101	10	110	00
001	11	111	00		101	11	001	11
010	00	001	10		110	00	111	01
010	01	110	01		110	01	000	10
010	10	101	01		110	10	011	10
010	11	010	10		110	11	100	01
011	00	101	10		111	00	011	01
011	01	010	01		111	01	100	10
011	10	001	01		111	10	111	10
011	11	110	10		111	11	000	01

the output of the encoder is:

$$Y_i = G_1 X_i + G_2 S_i \qquad (3\text{-}14)$$

From (1-13) (1-14), state transfer and encoder output table can be obtained in Table 3-2. This table is very useful for Gamma calculation simplification, which will be discussed in the next section.

3.4.2 Proposed reduction approach for Gamma calculation

Even though the Max-Log-MAP algorithm is used, a lot of calculations are still involved to obtain the gamma values. In order to reduce the number of calculations, the author proposes a simple but effective approach for calculation simplification, which is much like the method to reduce Boolean functions with ROBDD (reduced ordered binary decision diagram)[66][67].

- First, we present the calculations of (3-11) (3-12) graphically. For example, formula $Y = X_1 * X_2 + X_3 * X_4$ can be presented in Figure 3-3.
- If we check the gamma calculation presented in this kind of graph, many identical subtrees can be found. Figure 3-4 is an example.

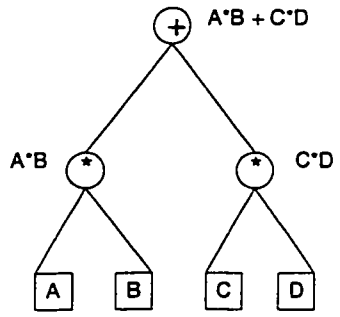

Figure 3-3 Diagram of A*B + C*D

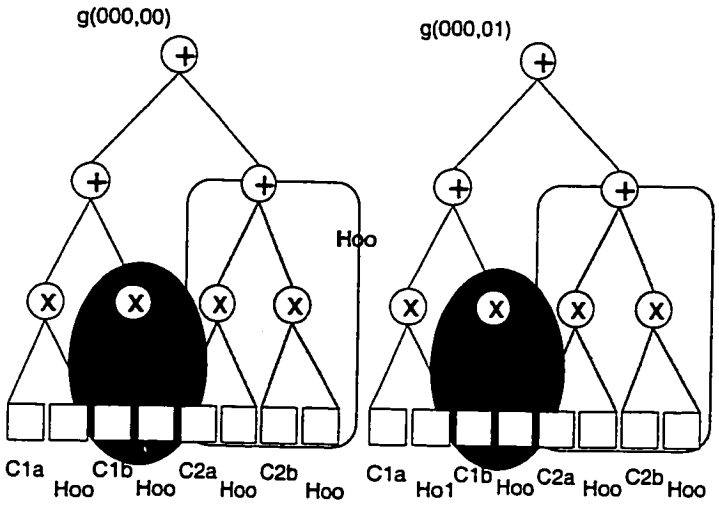

Figure 3-4 Diagram of part of gamma function

- If these redundant parts are eliminated, i.e, identical trees are represented only once like Figure 3-5, the reduced gamma calculation results. Figure 3-6 shows the reduced gamma calculation graph for DVB/RCS code.

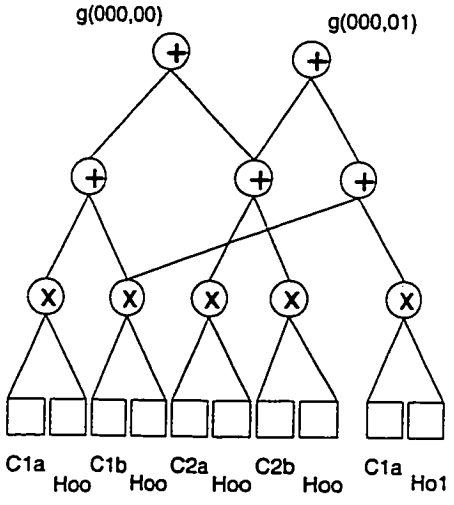

Figure 3-5 Simplified diagram of part of gamma function

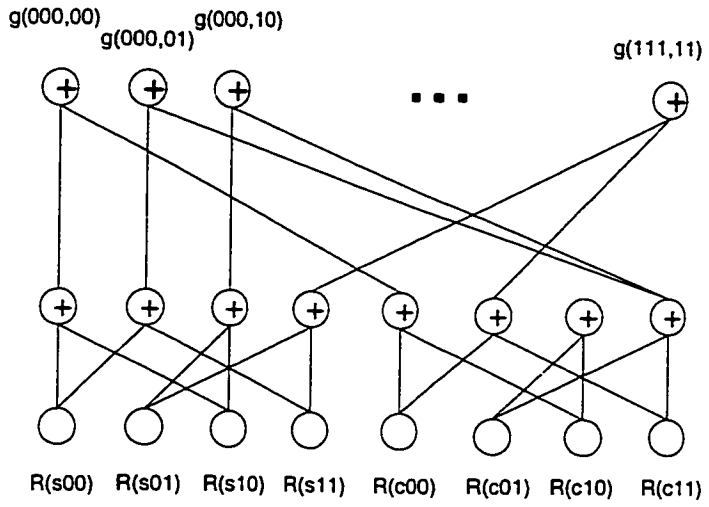

Figure 3-6 Simplified gamma Function

From the reduced gamma calculation graph it can be seen that all the gamma values come from only 8 values:

$$
\begin{aligned}
C_{00} &= R(y_{c0} \mid c_0 0) + R(y_{c1} \mid c_1 0) \\
C_{01} &= R(y_{c0} \mid c_0 0) + R(y_{c1} \mid c_1 1) \\
C_{10} &= R(y_{c0} \mid c_0 1) + R(y_{c1} \mid c_1 0) \\
C_{01} &= R(y_{c0} \mid c_0 1) + R(y_{c1} \mid c_1 1) \\
S_{00} &= R(y_{i0} \mid i_0 0) + R(y_{i1} \mid i_1 0) \\
S_{01} &= R(y_{i0} \mid i_0 0) + R(y_{i1} \mid i_1 1) \\
S_{10} &= R(y_{i0} \mid i_0 1) + R(y_{i1} \mid i_1 0) \\
S_{11} &= R(y_{i0} \mid i_0 1) + R(y_{i1} \mid i_1 1)
\end{aligned}
\qquad (3\text{-}15)
$$

where $R(y \mid i) = \frac{1}{2}\, y * i$, and y_i refers to the received value of information bit; y_c refers to the received value of parity bit; i is a hypothetical value; $c_0 0$ means the hypothetical value of c_0 is 0.

$R(y \mid i)$ is implemented with either multiplication circuits or lookup tables. An alternative is obtaining C and S directly with lookup tables, but the lookup tables for C and S are 8 bits input and 8 bits output, while those for $R(y \mid i)$ are 4 bits input and 7 bits output. Thus, in terms of area calculating $R(y \mid i)$ should be better.

It is noticed that the C and S are independent of the decoding iteration so these values can be performed only once prior to other calculations and can be reused in each MAP iteration. Also using this method the total memory size does not increase as the pre-calculated intermediate terms just replace the received symbols.

A dedicated circuit is implemented for the gamma values of the DVB/RCS turbo decoder. With this proposed calculation reduction approach, the total area for gamma calculation can be reduced by about 60% and the power consumption can also be reduced greatly.

3.5 Effect of correction coefficient

Max_Log_MAP algorithm degrades the coding performance due to omitting the Jacobin correction function. In the case of binary codes, the correction function is

$$Log(1 + e^{-|x-y|}) \qquad (3\text{-}16)$$

the degradation of the performance is about 0.4 dB [34]. In the case of non-binary codes, the correction function is

$$Log(1 + e^{-|a|} + e^{-|b|} + e^{-|c|}) \qquad (3\text{-}17)$$

where $-|a|$, $-|b|$ and $-|c|$ are the three values among x-max(x,y,z,w), y-max(x,y,z,w), z-max(x,y,z,w), w-max(x,y,z,w). The degradation of the performance is less than 0.1 dB, less significent than in the case of binary codes [34].

In hardware implementation, a simplified correction approach, the Max*_Log_MAP algorithm, is some time used. This approach chooses a one-bit approximation for the correction term. With a look-ahead implementation circuit, adding this kind of correction function will not degrade the performance in terms of speed and will only increase the area of the decoder by 5%-10%.

For double binary CRSC code, the simulation results show that the one-bit Max_Log_MAP algorithm has very similar coding performance to the Log_MAP algorithm [59]. In hardware implementation, if we keep the same speed performance of the decoder, the area use will increase by 5%-10%, while if we keep the same area use, the speed performance will degrade by about 5%.

Chapter 4

Hardware architecture of the decoder

In this thesis, the DVB/RCS non-binary turbo decoder is implemented according to DVB/RCS standard [56], and the decoding algorithm is Max-log-MAP algorithm. This decoder can support various block sizes (from 96 bits to 1728 bits) and coding rates (7 coding rates) to suit DVB/RCS applications as this kind of applications need the coding scheme to be flexible. This implementation outperforms almost all other commercially realizable codes, so it can also be used for many other communication applications.

By using a top-down approach, the author makes extensive use of hierarchy, regularity, modularity and locality strategies to make the design more effective and to reduce the design complexity. The DVB/RCS decoder is a complex system, so the adoption of highly structured design methodologies is definitely necessary. In structure the decoder is designed to be composed of many models and sub-models; the lowest level models directly employ the Xilinx Cores such as block memory and arithmetic, which have the best performance for Xilinx technology. Figure 4-1 shows the model structure of this design.

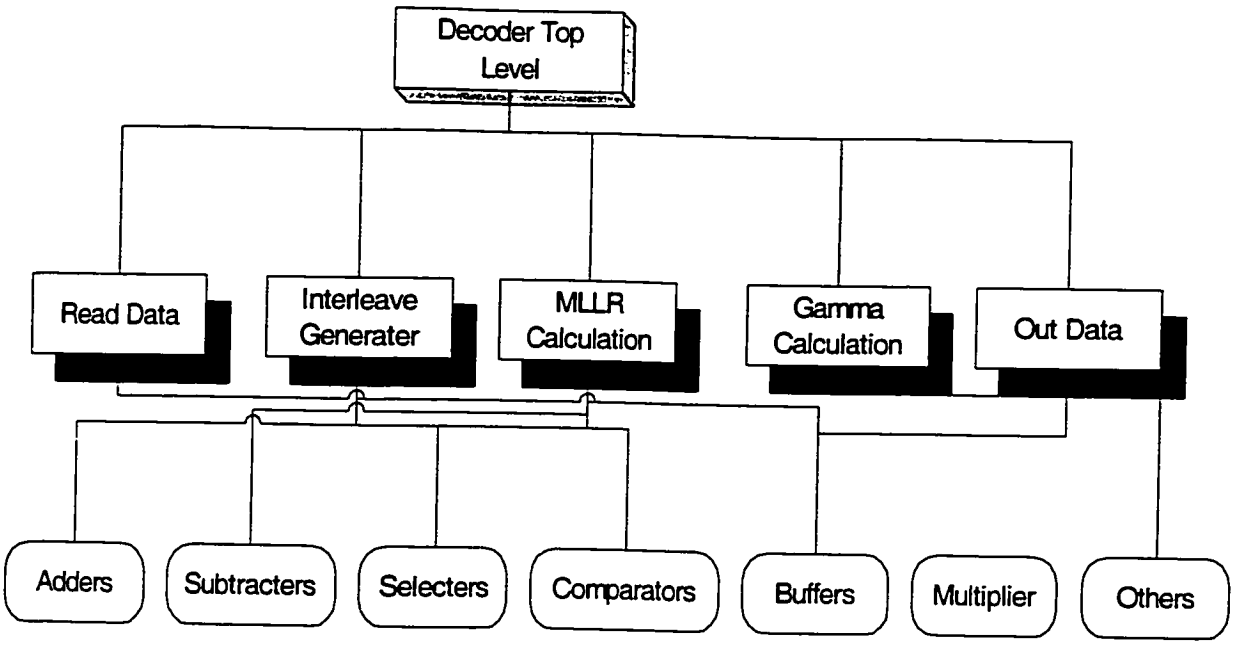

Figure 4-1 Model structure of the decoder design

4.1 General description and design partition

The first step in the system design phase involves hierarchically partitioning the system into a set of cooperating units, which is a crucial design step that will have a major influence on both the system cost and performance. Generally, it is favorable to partition the design in such a way that interoperation between units is minimized, since a data transfer between two units requires control logics to synchronize the processes. According to this principle the decoder is mainly partitioned into 4 parts, which are Input control (Read data), Output control, MAP algorithm, and Interleaver because they are comparatively independent of each other. Figure 4-2 provides a basic architecture block diagram of the decoder. All these 4 units are working parallel to keep the decoder having a high decoding speed. Because the execution time of the Read data part and the MAP

part is longer than the execution time of the Interleaver part or the Out-data part, and because the 4 units are independent, the decoder can be easily scheduled to achieve optimal area at the same time. Figure 4-3 gives the general timing diagram of the DVB/RCS decoder.

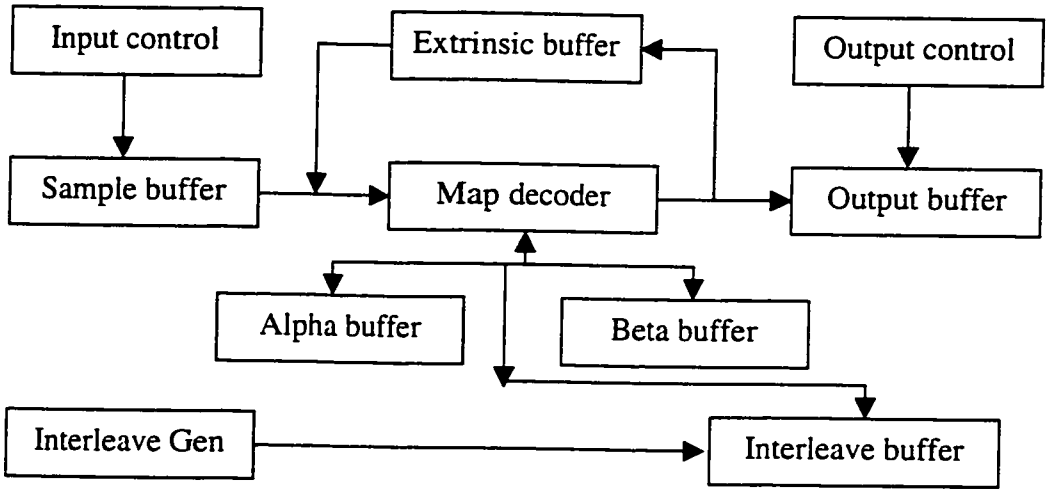

Figure 4-2 Decoder structure

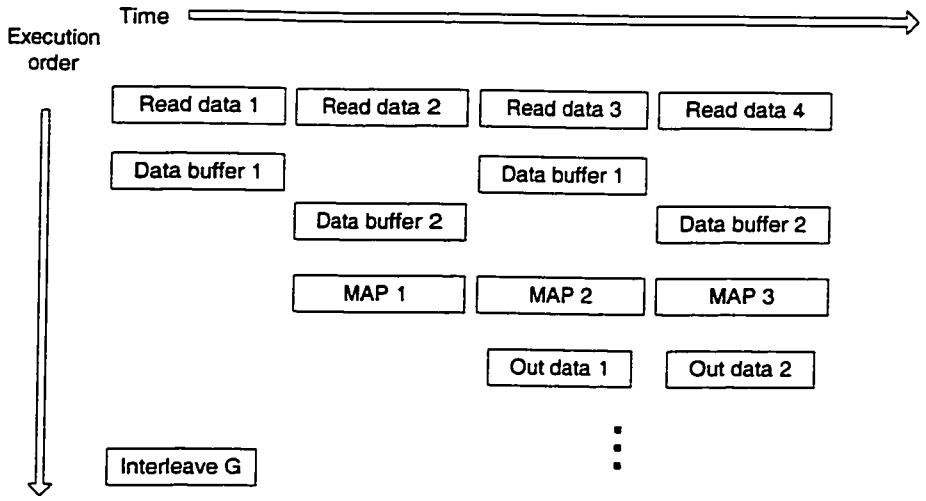

Figure 4-3 Timing diagram of DVB/RCS decoder

In normal operation, sampled data is first input into a sample buffer. The sample buffer performs double buffering, allowing the next frame to be input while the decoder processes the current frame. The decoded result is stored in out-data buffer and the out-data circuit controls the output procedure. The interleaver between the component convolutional codes is precalculated based on algebraic laws according to the frame size used, avoiding the use of a lot of look-up tables for every frame sizes. MAP decoder is the most complex part of the design. It is optimized in both speed and area by using pipeline technique and algorithm optimization.

4.2 Max-Log-MAP decoder

The DVB/RCS turbo decoder is based on the Max-Log-MAP algorithm. A successful design of implementing this algorithm is the key to achieve an optimal decoding performance for the decoder. Although the Max-Log-MAP has been greatly simplified compared to MAP algorithm, it still is a memory intensive and computationally intensive algorithm. This is primarily because there are a lot of computations of the gamma function, the alpha and beta functions and the LLR functions. The case for DVB/RCS code is even worse than for binary turbo code since there are more computations for DVB/RCS code. The problems of the gamma function have been discussed in the previous section. The alpha and beta functions, as we know, are recursive operations, which begin at opposite ends of the received sequence. In hardware design, loop calculations are normally the critical paths that limit the performance of the design in terms of speed. For the DVB/RCS decoder the alpha and beta calculations are the only crucial issue for us to improve the decoder speed. To solve this problem we have two options which are either simplifying the data path of the alpha and beta calculation as much as possible or opening the calculation loop. The slide windows algorithm can be used to open the loop [68], but this algorithm could degrade the decoding performance of the DVB/RCS code although it works very well for the binary turbo code. As for how

worse it could be, there has not yet been anything published in the open literature. If we want to speedup the decoder further, maybe the second option should be considered. Since the MLLR functions are dependent on both the alpha and the beta, at least one of the alpha and beta values has to be stored in memory before the sigma functions are being calculated. The memory requirement cannot be reduced by modifying scheduling or resource sharing since the alpha and beta functions are recursive and begin at opposite ends of the trellis.

The design of the sigma functions should be focused on saving the area as it does not affect the speed of the decoder significantly in a pipeline structure, which can be seen in the following scheduling discussion. Depending on the architecture for the hardware implementation, we can choose either to store both the alpha and the beta or to store the MLLR in memory. But, the MLLR is preferred because it needs less memory.

In the Max-Log-MAP algorithm for the DVB/RCS code, the gamma function is defined in (3-11) (3-12); the alpha and the beta functions are defined by the recursion

$$\alpha_k(s) = \underset{s' \in S}{Max}(\alpha_{k-1}(s') + \gamma_k(s', s)) \tag{4-1}$$

$$\beta_{k-1}(s) = \underset{s' \in S}{Max}(\beta_k(s) + \gamma_k(s', s)) \tag{4-2}$$

where the summation is over all 4 states where the transition exist.

For the non-binary turbo codes, a multidimensional LLR (MLLR) must be used indicated as $L(u_k)$ and $L^e(u_k)$. The vector $L(u_k)$ include the three following scalar LLRs:

$$L_A = \underset{s',s:u_k=00}{Max}(\alpha_{k-1}(s') + \gamma_k(s', s)) + \beta_k(s)) - \underset{s',s:u_k=01}{Max}(\alpha_{k-1}(s') + \gamma_k(s', s)) + \beta_k(s)) \tag{4-3}$$

$$L_B = \underset{s',s:u_k=00}{Max}(\alpha_{k-1}(s') + \gamma_k(s', s)) + \beta_k(s)) - \underset{s',s:u_k=10}{Max}(\alpha_{k-1}(s') + \gamma_k(s', s)) + \beta_k(s)) \tag{4-4}$$

$$L_C = \underset{s',s:u_k=00}{Max}(\alpha_{k-1}(s') + \gamma_k(s',s)) + \beta_k(s)) - \underset{s',s:u_k=11}{Max}(\alpha_{k-1}(s') + \gamma_k(s',s)) + \beta_k(s)) \quad (4\text{-}5)$$

and for the vector $L^e(u_k)$ we have,

$$L_A^e = \underset{s',s:u_k=00}{Max}(\alpha_{k-1}(s') + \gamma_k^e(s',s)) + \beta_k(s)) - \underset{s',s:u_k=01}{Max}(\alpha_{k-1}(s') + \gamma_k^e(s',s)) + \beta_k(s)) \quad (4\text{-}6)$$

$$L_B^e = \underset{s',s:u_k=00}{Max}(\alpha_{k-1}(s') + \gamma_k^e(s',s)) + \beta_k(s)) - \underset{s',s:u_k=10}{Max}(\alpha_{k-1}(s') + \gamma_k^e(s',s)) + \beta_k(s)) \quad (4\text{-}7)$$

$$L_C^e = \underset{s',s:u_k=00}{Max}(\alpha_{k-1}(s') + \gamma_k^e(s',s)) + \beta_k(s)) - \underset{s',s:u_k=11}{Max}(\alpha_{k-1}(s') + \gamma_k^e(s',s)) + \beta_k(s)) \quad (4\text{-}8)$$

Hence, the MLLR is defined as:

$$L(u_k) = [L_A(u_k)\ \ L_B(u_k)\ \ L_C(u_k)\ ]$$

$$LnP_0 = Min(L_A^e, Le_B^e, L_C^e) \quad (4\text{-}9)$$

$$LnP_1 = Min(-L_A^e, -L_A^e + Le_B^e, -L_A^e + L_C^e) \quad (4\text{-}10)$$

$$LnP_2 = Min(-L_B^e, -L_B^e + Le_A^e, -L_B^e + L_C^e) \quad (4\text{-}11)$$

$$LnP_3 = Min(-L_C^e, -L_C^e + Le_A^e, -L_C^e + L_B^e) \quad (4\text{-}12)$$

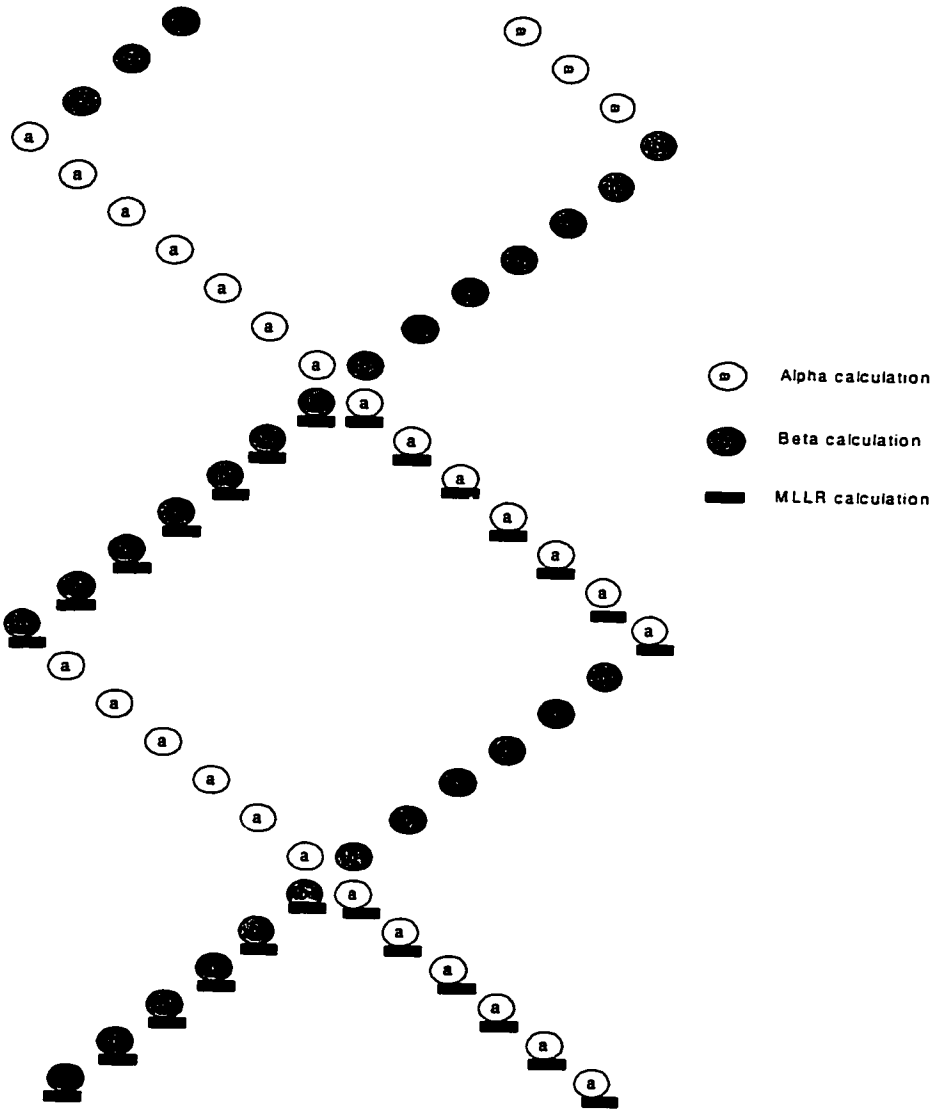

Figure 4-4 Data dependency graph of the decoder architecture

With the above calculation formulas in mind, now we move to the questions of the scheduling. The purpose of the scheduling is to distribute all the process in time so that the specified performance is met and the amount of resources required is minimized. Minimizing the amount of hardware will also minimize the power consumption. As the partitioning has been done, it is possible to perform a static scheduling, which is easier to achieve an optimal architecture. In terms of resource, we manage to treat the scheduling

of the design as a resource adequate scheduling problem because resource limited scheduling is much more complex than resource adequate scheduling.

Figure 4-4 presents the architecture diagram of the MAP decoder. Since the DVB/RCS non-binary code is a circular code, in the case Max-Log-MAP algorithm is applied, decoding the sequence consists of going round the circular trellis anti-clockwise for the backward process, and clockwise for the forward process [35], during which data are decoded and the extrinsic information is built. To guide the process towards an initial state the decoding process is preceded by a prologue decoding step, performed on a part of the circle. By doing this the initial state can be set to a good estimate of the circulation state. The computations of the alpha and beta recursions are carried out in parallel so that the decoder can speedup effectively. In the middle of the trellis, the MLLR functions can be computed as both alpha and the beta are ready. Therefore, one decoding sub-iteration can be implemented in the trellis length time unites.

4.2.1 Full Pipeline architecture

According to the Max-Log-MAP algorithm, a simple top level functional structure of the data flow of the decoder is shown in Figure 4-5. The blocks within the decoder in the data path are the gamma function, the alpha and beta function, the interleave function and the MLLR function. If all the input data are executed sequentially as shown in Figure 4-6, the basic MAP decoder structure is built. While the advantages of this structure are the minimum use of area and the simplicity of the design of the control unit, the decoding speed of the decoder is very low and cannot meet the requirement of typical applications. By putting the data path in parallel, the speed of the decoder can be improved greatly, but this is not an efficient way because the parallel structure will cost a lot of area. Therefore, in this thesis we design a full pipelined structure which is optimal in both area and speed.

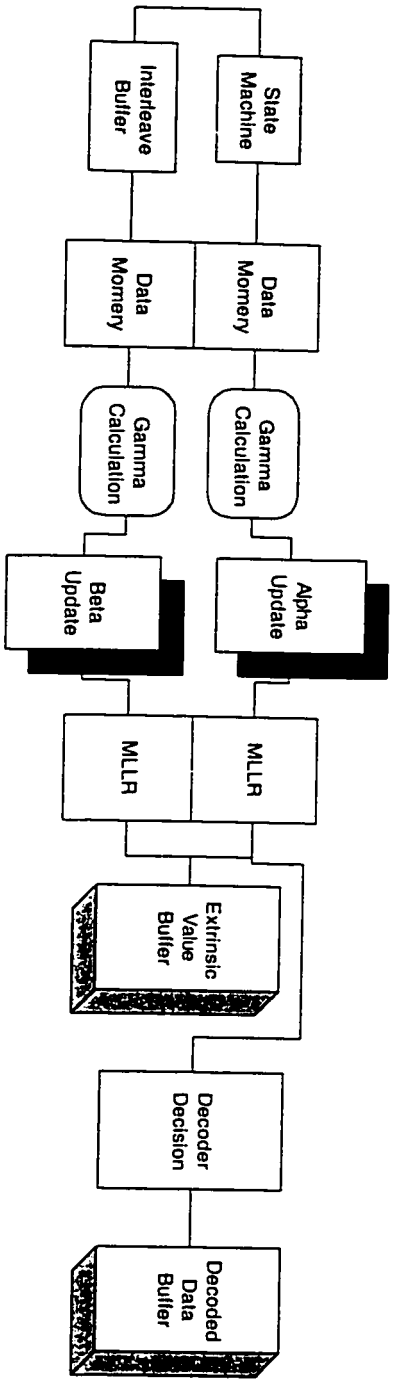

Figure 4-5 Data path of the MAP decoder

Pipelining is an implementation technique which is an efficient method of increasing the throughput of a sequential algorithm [69][70]. A pipeline with n stages allows n computations to be processed concurrently and attains a speedup of n over sequential processing. Under ideal condition, the time between output data in the pipelined structure is equal to

Time interval (pipelined) = Time interval (nonpipelined) / Number (pipe stages)

As throughput is the inverse of the longest critical path, ideally we should break the critical path into paths of equal length to achieve the best pipeline result. For non-recursive algorithm, in principle, the throughput (the maximum input data rate) could be "infinite", if only the critical path is divided into infinitesimally small pieces. Thus, the decoder speed will be infinite. However, the alpha and beta functions of the Max-Log-MAP algorithm are recursive, so there is no way to obtain an "infinitive" performance. Since the pipeline technique is only applicable to the computations which are not inside a recursive loop, the alpha and beta functions cannot be further pipelined

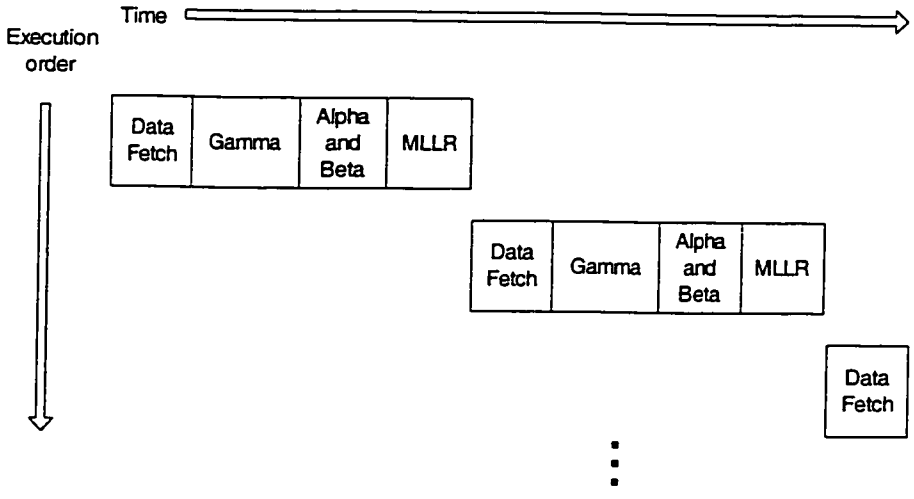

Figure 4-6 Timing diagram of sequentially executed MAP decoder

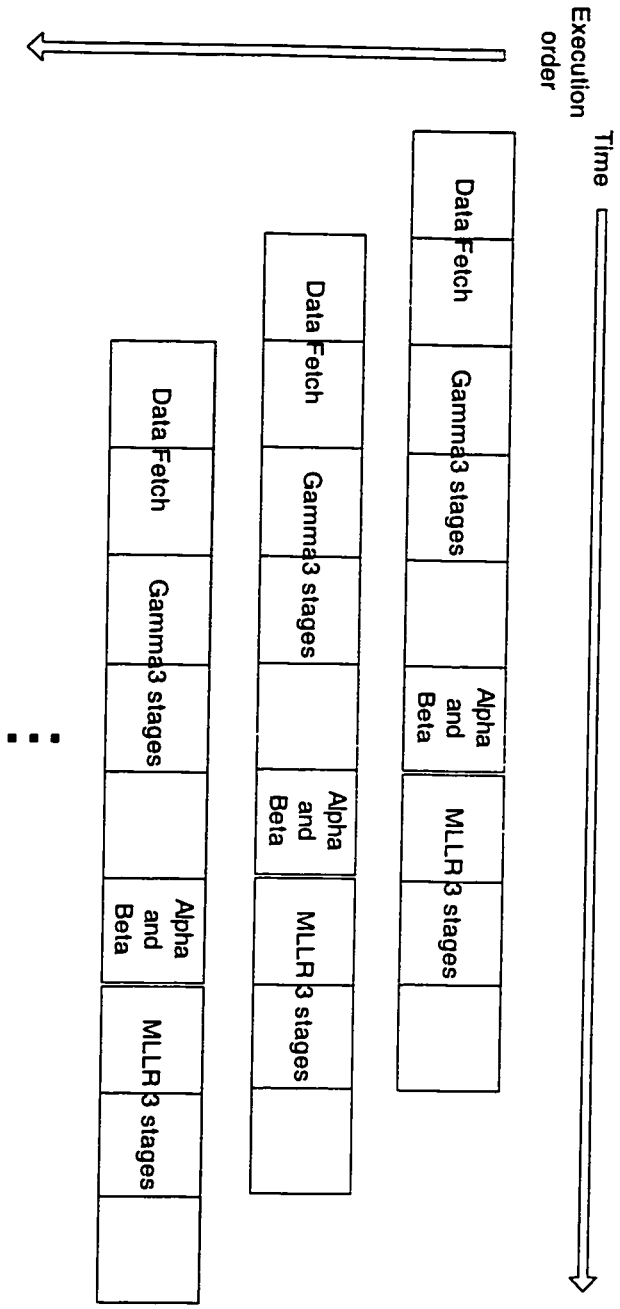

Figure 4-7 Timing diagram of pipelined architecture

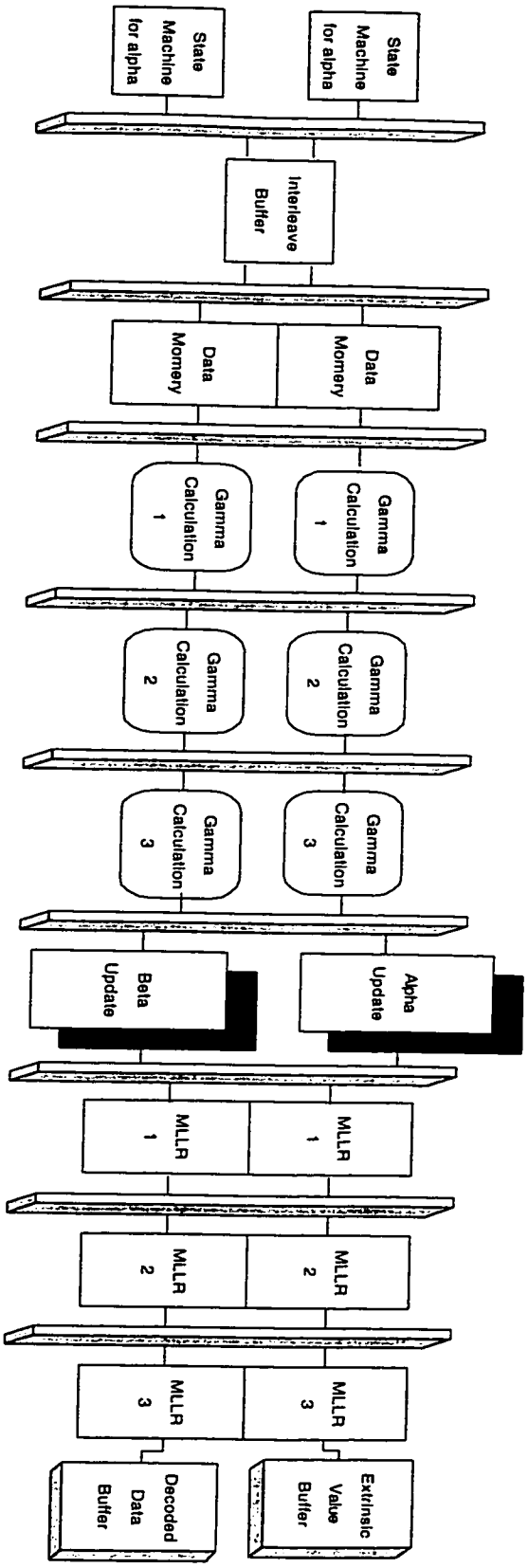

Figure 4-8 Data path of the optimized pipelined architecture

Figure 4-7 is an optimized pipelined structure for Max-Log-MAP decoder and Figure 4-8 is the data path of the optimized pipelined structure for Max-Log-MAP decoder of the DVB/RCS code. The pipelined structure overcomes the drawbacks of both sequential structure and the parallel structure; it almost has the same speed as the parallel structure and the same area as the sequential structure. But correspondingly the design complexity of the control units is much higher than the other structures. Pipeling also increases the latency and one should be careful that latency does not exceed the requirements of the given application.

For this design, from Figure 4-8 we can see that the data path is divided into 9 stages and the gamma function and the sigma function are divided into 3 stages respectively. According to the above discussion, the performance in terms of the speed is improved 9 times compared to the sequential structure.

4.2.2 Alpha and beta recursion update structure

The basic limitation of the MAP decoder in terms of the speed of decoding is the alpha and beta recursion because this circuit cannot be pipelined as we discussed above.

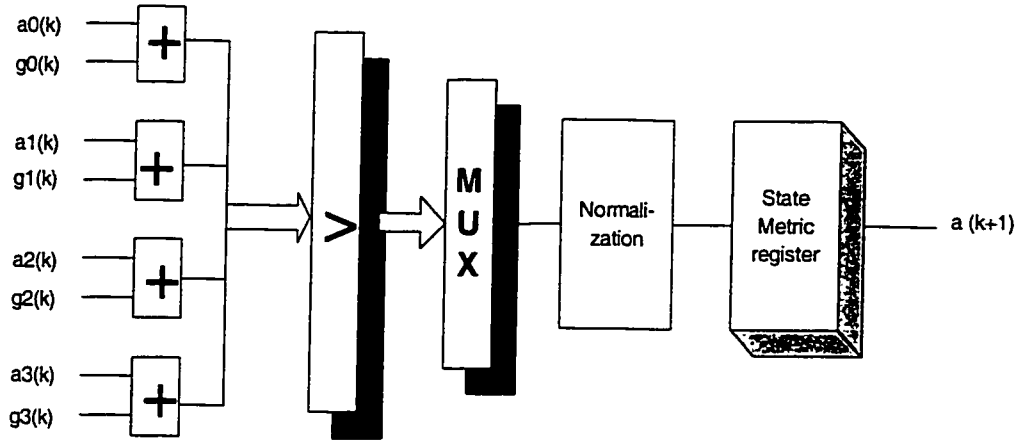

Figure 4-9 The basic circuit of the alpha update

In general, all the computations for all states in the trellis must be completed before the next recursion starts. To improve the performance, first, we must try to simplify the computation logic of the alpha and beta function as much as possible. Secondly we use the dedicated resource binding strategy in this part of this circuit. Actually we sacrifice the area here for a better performance. In addition, as the alpha calculation and the beta calculation are independent, the computation of the alpha and the beta are done in parallel. This requires a separate circuit for alpha and beta recursions. The parallel computation of the alpha and beta functions doubles the decoding speed of the decoder.

Figure 4-9 presents the basic alpha recursion update circuit. The beta circuit is the same as the alpha circuit. The gamma values are the only inputs for the alpha or the beta update circuit and the update function for the alpha or beta recursion are ACSs (Addition, Compare and Selection). Thus, a lot of adders, comparators and selectors are needed. From the discussion in previous section we know that a normalization circuit is a must for the implementation of the alpha and beta functions. The effective new renormalization approach proposed in Section 1.3.3 is realized in Figure 4-10.

Since the transitions of the alpha and beta in the trellis are different, there is a slight difference between the optimized alpha circuit and the optimized beta circuit. In order to further save area with resource sharing technique, we prefer to make a common circuit which can be used for both alpha function and beta function. This will be discussed in detail in the following section.

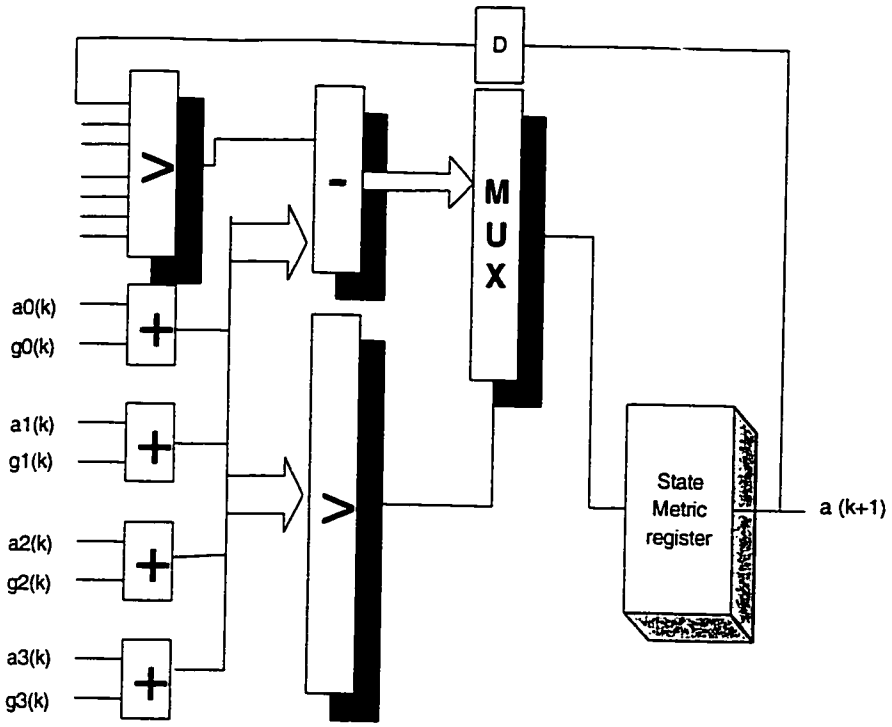

Figure 4-10 Alpha update circuit with the proposed normalization

4.2.3 Extrinsic value calculation structure

As stated before the extrinsic values are computed with the alpha values, the beta values and the gamma(e) values, using Equation (4-3) – (4-12). The inputs of the sigma calculation unit are the alpha, the beta and the gamma(e) and the outputs are extrinsic values or the decoded values depending on the iteration; if the iteration is the last one, the outputs are the decoded value and the gamma input should be used.

Since the alpha and the beta recursions begin at opposite ends of the trellis, the necessary information will not be available to compute the MLLR values until each recursion is half finished. From this point on, two MLLR values can be computed at the same time so the MLLR calculation can finish right after the alpha and the beta

calculations finish, which means that roughly every subiteration needs only the same time units as the length of the trellis.

The MLLR calculation is not recursive, therefore pipeline technique can be used to increase the throughput. In our design 3 pipeline stages are used, resulting in the same throughput as the alpha and beta circuits. Figure 4-11 shows the MLLR implementation circuit.

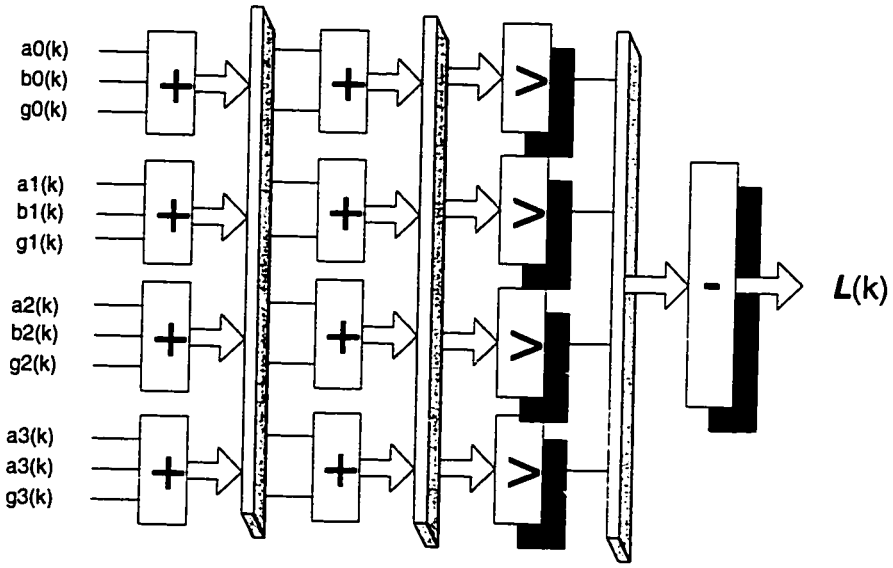

Figure 4-11 The MLLR implementation circuit

4.2.4 Further area reduction with resource sharing

Area, the resources needed, is another important performance measure for the decoder, affecting the cost of the implementation. The amount of the resources can be minimized by letting the computations that do not overlap in time share resources. The number and type of resources are determined from the operation schedule. Usually, the largest number of concurrent computations determines the number of resources required. Especially, the number of resources required in the pipelined implementations depends

on the clock frequency; the higher the frequency the larger is the number of operations executing concurrently, therefore, larger is number of required resources.

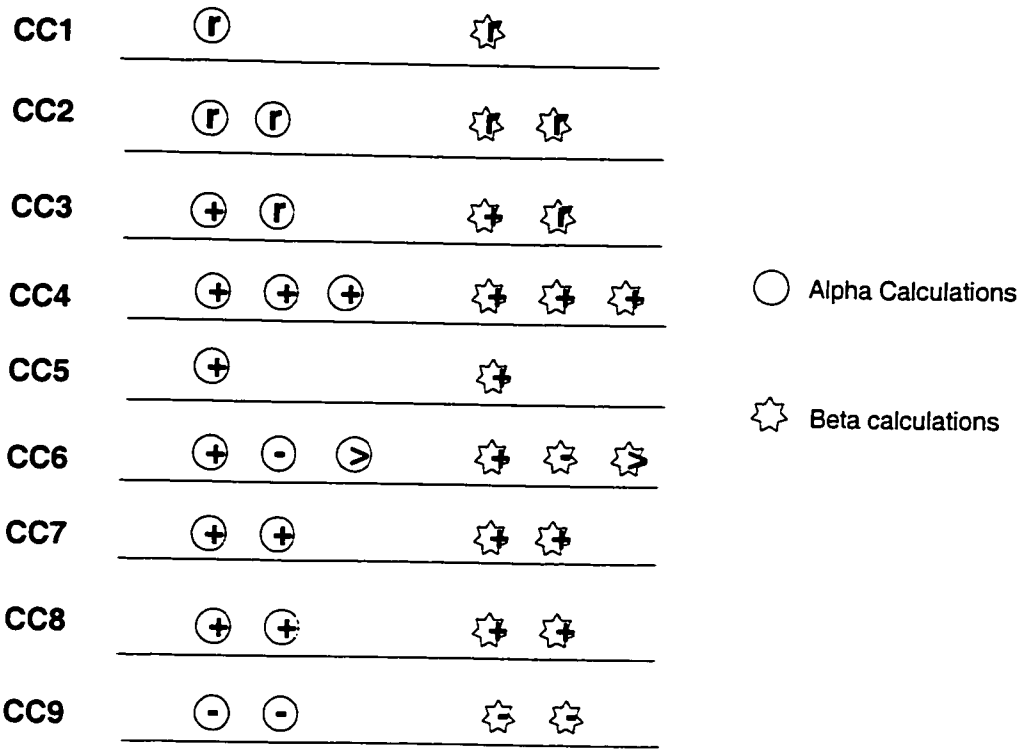

Figure 4-12 Operation schedule of the MAP decoder

Resource reduction problems can be solved by using clique partitioning, which provides an effective way of determining which computations can share resources [71]. In addition, there are many scheduling algorithms to improve a design such as the ASAP (As Soon As Possible), ALAP (As Late As Possible), Critical path list scheduling, Force-directed scheduling, etc [71]. Normally, ASAP and ALAP scheduling are used to determine the time range in which the operations can be scheduled, and then force-

directed scheduling or other algorithms can be used to obtain the optimized scheduling to achieve the minimized resources.

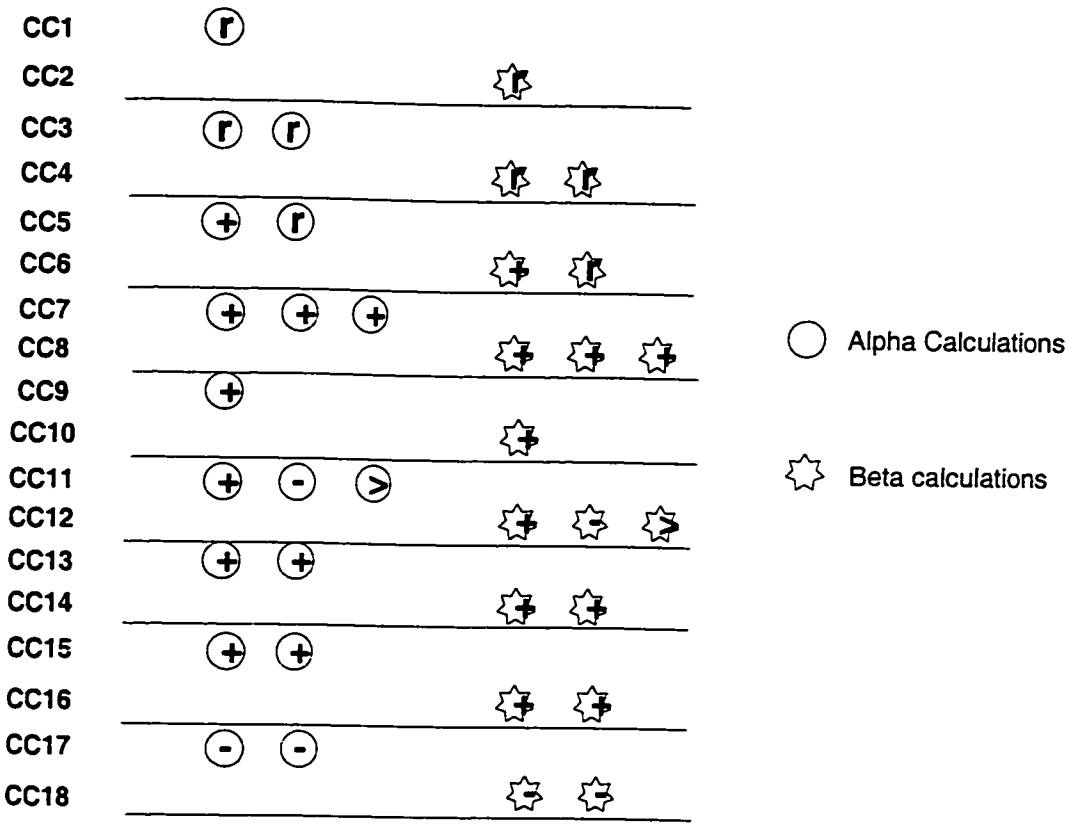

Figure 4-13 Operation schedule with pipelined resources

For pipelined structure, pipelined resources scheme is also an excellent approach to save the area of the design further and the list scheduling algorithm can be extended to handle pipelined resources by allowing the scheduling of overlapping computations with different start times and no data hazards. In the design of the DVB/RCS non-binary code decoder, this approach can be used in all computation units like gamma, alpha, beta and MLLR. The normal scheduling for alpha and beta functions presented before is shown in Figure 4-12. After using pipelined resource scheme, pipelining the alpha and beta circuit

further, we gain the new scheduling presented in Figure 4-13. From the scheduling diagram, it can be easily found that the alpha function and the beta function can share the same resources, saving almost half of the original resources. This approach is also applicable to the gamma and the sigma circuits. In general, with this area reduction scheme proposed for the DVB/RCS decoder, the area of the whole design should be saved by over 40% for FPGA implementation.

4.3 Read data block design

The functions of the read data block include Synchronizing the input signals, transforming the input data with different frame size and different code rate into a uniform data structure, dividing the input data into system part, code 1 part and code2 part and storing them into corresponding memories. This block works in parallel with other blocks, exchanging data only through memory.

When the Set signal is valid, the read data block takes the code rate and the frame size signal as the decoder parameters, keeping and using them until next valid Set signal. With different code rates, the input data have been punctured according to the puncturing map of the DVB/RCS code, thus the read data block has to recover the data after receiving this data and then stores them into the memory.

In the DVB/RCS standard, the order of transmission is in the order of systematic bits, Y1 and Y2 bits and W1 and W2 bits. Following this order, the read data block is designed being composed of 3 stages: systematic read stage, Y read stage and W read stage. The stage diagram is shown in Figure 4-14. Having read and stored the whole frame of data, the read data block will read the next frame immediately and store the data into the second memory. Meanwhile, the MAP decoder will work with the memory that the read data block is not using. Whenever it finishes a frame of data, the read data block will also give a finish signal that could start the MAP decoder to work.

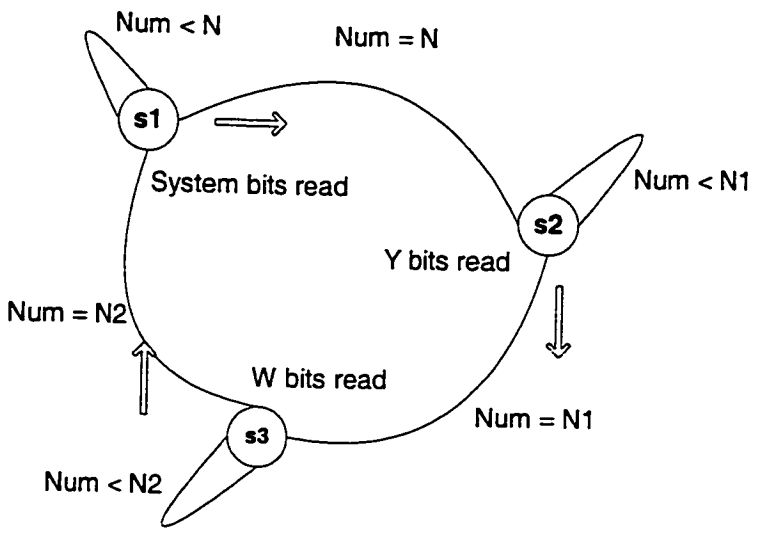

Figure 4-14 State diagram of the read data block

As for the timing and the format of reading the input data, please refer to the decoder interface section, Section 4.6.

4.4 Interleaver generation block design

The interleaver of the DVB/RCS code is done in two levels. The first level is inside the couples, invert the couple every other couple; the second one is between the couples:

$$I = P0*j + P + 1 \bmod N \tag{4-13}$$

- if j mod 4 = 0, then P = 0;
- if j mod 4 = 1, then P = N/2 + P1;
- if j mod 4 = 2, then P = P2;
- if j mod 4 = 3, then P = N/2 + P3;

The combinations of the default parameters to be used are provided in Table 2-1.

For the interleaver design, we focus on area saving under a certain speed constraint given by the read data circuit. To acquire a concise design, we always have to change a process into a recursive form. This is applicable to both software design and hardware design. Equation (4-13) can be modified into

$$i_{n+1} = i_n + P_0 \mod N \qquad (4\text{-}14)$$

$$i_0 = P + 1$$

It can be seen that not only does the multiplication disappear, but the addition computations of Equation (4-14) are also simpler than Equation (4-13).

The interleaver generation function will be conducted right after the frame size signal is available and should finish before the MAP decoder starts to work. The generated interleaved data is stored in a memory, called interleaver table, which can be used anytime needed. In this design, only the interleave table is needed since we successfully avoid using de-interleave table in the MAP decoder by managing the schedule of reading and writing the inner data.

4.5 Out data block design

All the decoded data is placed in the data output buffer by the MAP decoder circuit. Out data block is responsible for transforming the decoded data into a form the user required, outputting them to its user and synchronizing the output interface of the decoder with other circuits. This block works independently, but it cannot start before the time that the decoder done signal of the MAP decoder and cannot finish later than the time next decode data become available.

The detailed design of this part of circuit should be according to the customer's requirements. A temporary design has been implemented. The data format and the output timing refer to the interface section.

4.6 Interface of DVB/RCS decoder

Decoder signal pin-out

Table 4-1 shows the pin-out of the decoder. These are normal input and output signals used for the decoder.

Table 4-1 Decoder signal pin-out

Signal name	I/O Pin	Description
Reset	Input	Reset of the decoder
Enable	Input	Enable decoder operation
Code_rate	Input	Selects the code rate
Clock	Input	Decoder clock
Frame_size	Input	Select the frame size
Num_its	Input	Number of decoder sub-iterations
Set	Input	Set signal for decoder parameter
Frame_start	Input	Indicates the start of the turbo frame
Datain	Input	I & Q data input
Datain_v	Input	Valid signal for corresponding I & Q data inputs
Dataout_req	Input	Request for data output
Dataout	Output	Output data
dataout_v	Output	Indicates valid data is on the dataout pin

Data input format and timing

The decoder receives QPSK in-phase and quadrature samples. Both I and Q sample inputs are 4 bits in width (signed 2's complement) with 2 bits as fractional part. Figure 4-15 shows the input timing.

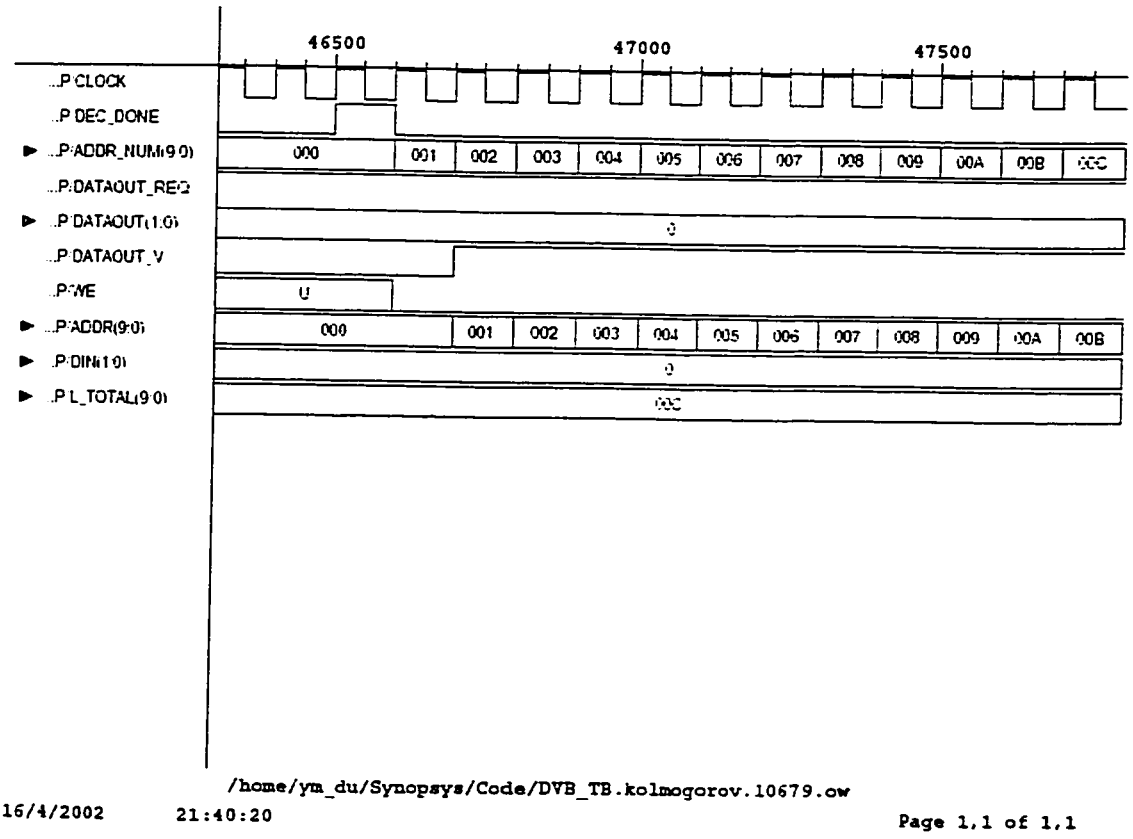

Figure 4-15 Input timing of the decoder

Data output format and timing

The data output circuit is driven by Dec-done signal from MAP decoder component and Dataout_req signal. If a frame is ready for output and if an output request is made, data is output in I&Q format like input data. The data output timing diagram is provided in Figure 4-16. If other format is needed, such as in bytes, we can change the decoder output to other formats.

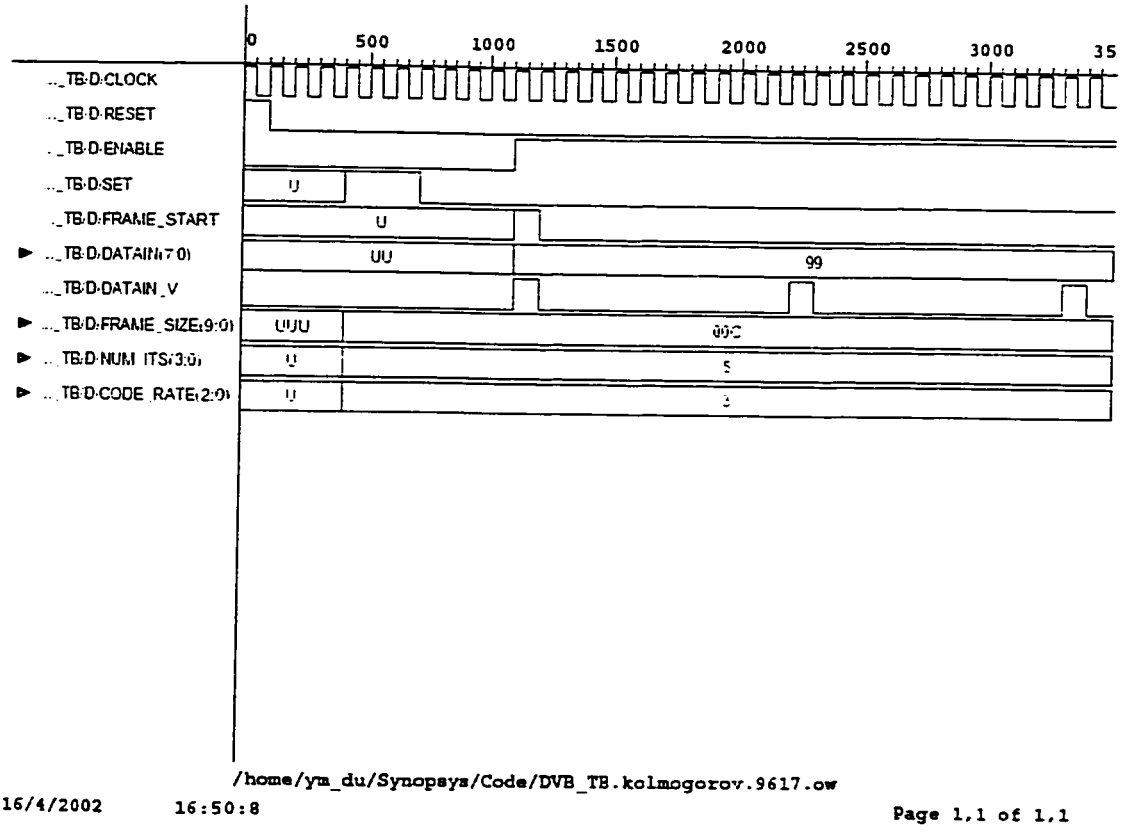

Figure 4-16 Output timing of the decoder

Decoder Parameters

This decoder implements the full DVB-RCS standard including 12 frame sizes (from 96 to 1728), 7 code rates (1/3, 2/5, 1/2, 2/3, 3/4, 4/5, 6/7) and any choice of iteration number. All these parameters can be altered at run time.

Table 4-2 Code rate setting

Parameter	000	001	010	011	100	101	110
Code rate	1/3	2/5	1/2	2/3	3/4	4/5	6/7

Table 4-3 Frame Size Settings

Parameter	0000	0001	0010	0011	0100	010i	0110
Frame Size byte	12	16	53	55	57	106	108
Frame Size bit	96	128	424	440	456	848	864

Parameter	0111	1000	1001	1010	1011
Frame Size byte	110	188	212	214	216
Frame Size bit	880	1504	1696	1712	1728

Latency

The MAP decoder will start to process once a complete frame has been received. The delay of processing depends on the frame size and the number of iterations.

Approximately the processing delay (in number of cycles) can be calculated using the following formula:

Delay = (Framesize / 2 + 9) * (number of subiterations)

Maximum throughput (data rate) ≈ Clock frequency / number of iteration

For Virtex-E device (43MHz), if iteration number is set to 4, the data rate can reach about 10Mbits/second.

Whatever the case is the MAP decoder should finish before the next frame input ends. Also, the data output should complete before the last subiteration of the next MAP calculation begins.

Chapter 5

Simulation and synthesis results

The simulation and synthesis of the design follow the general procedure shown in Figure 5-1. First, the design is modeled with VHDL in RTL level and then the RTL level simulation is performed. This simulation concentrates on checking the functionality of the design. The block diagram of the DVB/RCS code simulation is presented in Figure 5-2, where the functions in the dashed block are implemented in Matlab. A test bench for this design has been designed. During simulation, this test bench is used to exercise the functionality of the design of the decoder until the design's output signals match the expected waveform. For the RTL simulation, we use Synopsys's simulation tool, VHDLsim. For simulation in other levels we also use the same tool and the same test bench.

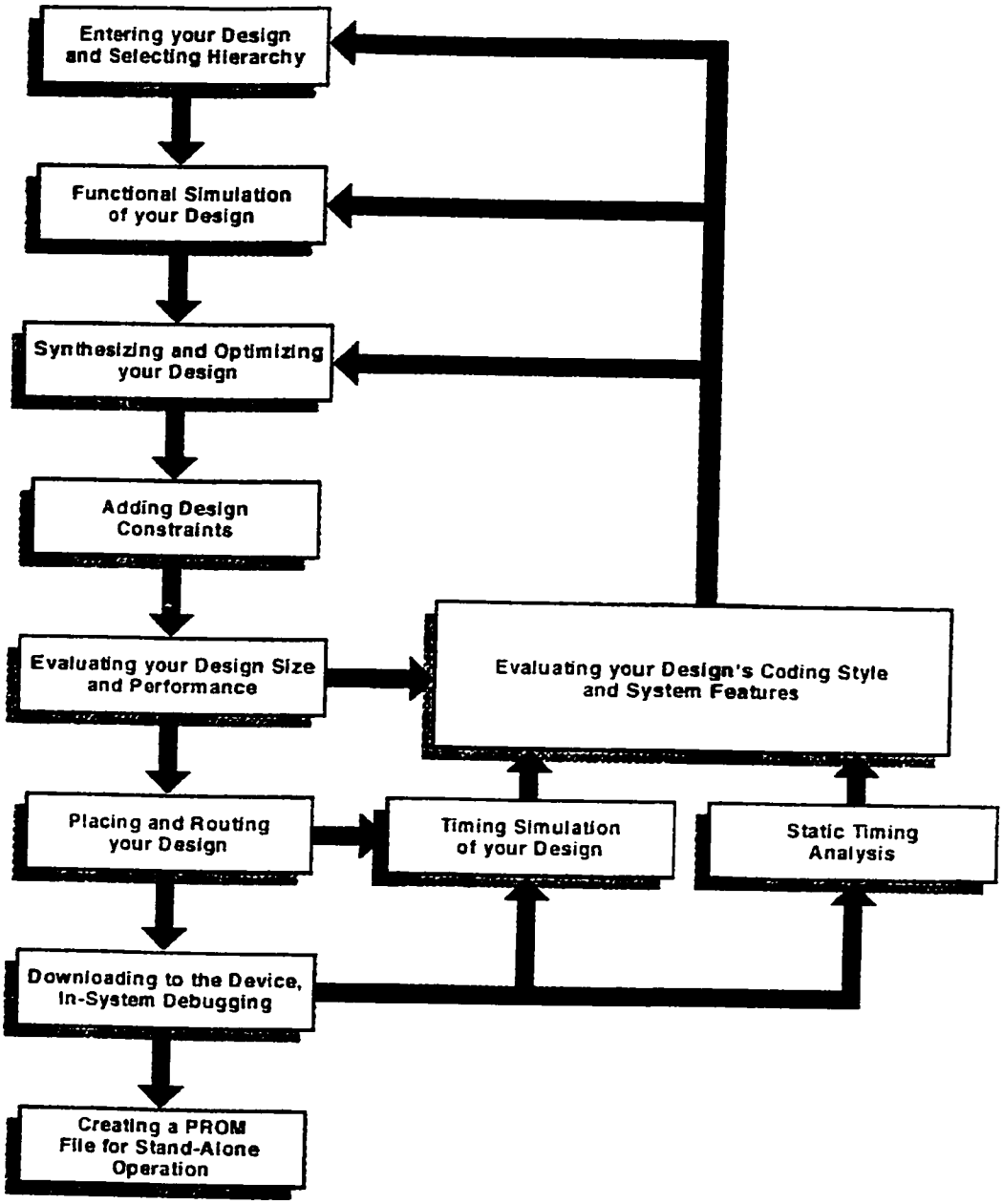

Figure 5-1 General design procedure

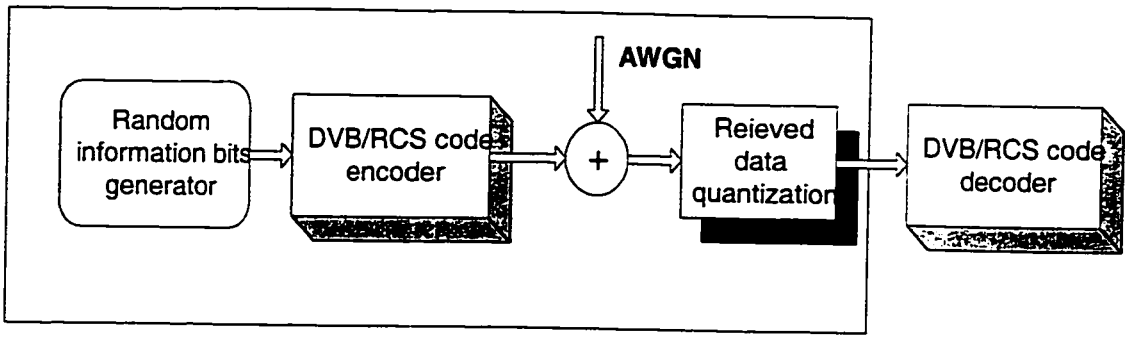

Figure 5-2 Block diagram of DVB/RCS simulation

After satisfying the functionality of the design, we synthesize the design with Design Compiler of Synopsys tools. When all constraints are met, the synthesis result, the netlist file, is passed to the place and route tools. For place and route, we use the Design Manager of Xilinx tools. This place and route result has to be verified again with gate level simulation. Unlike RTL simulation, the gate level simulation will check both functionality and timing. The synthesis result with Xilinx Virtex XCV1000E-8 FPGA is shown in Figure 5-3. It can be seen that the clock frequency can reach as high as 43MHz, corresponding to about 7 Mbits/second information bit rate with 6 iterations, much better than TurboConcept's TC1000 product, 4Mbits [52].

To perform the in-chip test for the design, a Virtex-E development kit (ADS-XLX-VE-DEV) made by AVNET is used [72]. A new test bench is designed because it has to be written into the FPGA chip along with the whole decoder and the input data needed for decoding. In the end, the in-chip test of the decoder obtains the same result as the gate level simulation.

```
Release v3.3.08i - Par D.27
Tue Jul 30 04:58:44 2002
Constraints file: dvb_topfk.pcf
Loading design for application par from file
/var/tmp/xil_AAA0pNWnH."Synopsys_edif" is an NCD, version 2.35, device
xcv1000e, package fg1156, Finished resolving physical constraints.
Device utilization summary:

   Number of External IOBs            33 out of 660     5%
   Number of BLOCKRAMs                64 out of 96     66%
   Number of SLICEs                 7517 out of 12288  61%

Overall effort level (-ol):   5 (set by user)
Placer effort level (-pl):    5 (set by user)
Placer cost table entry (-t): 7
Router effort level (-rl):    5 (set by user)

Generating PAR statistics.
   The Delay Summary Report
   The Score for this design is: 8417
The Number of signals not completely routed for this design is: 0
   The Average Connection Delay for this design is:        1.966 ns
   The Maximum Pin Delay is:                              11.825 ns
   The Average Connection Delay on the 10 Worst Nets is:  10.594 ns

   Asterisk (*) preceding a constraint indicates it was not met.
------------------------------------------------------------------------
  Constraint                    | Requested  | Actual     | Logic Levels
------------------------------------------------------------------------
TS_clock  TIMEGRP"clock"23 nS   | 23.000ns   | 23.123ns   | 23
HIGH 50.000 %                   |            |            |
------------------------------------------------------------------------
TS_mlgp FROM TIMEGRP "ucon" TO  | 46.000ns   | 37.543ns   | 26
   TIMEGRP "FFS" TS_clock * 2.000|           |            |             |
------------------------------------------------------------------------
1 constraint not met.
Dumping design to file ../p1.dir/5_5_7.ncd.
All signals are completely routed.
Total REAL time to PAR completion: 1 hrs 33 mins 41 secs
Total CPU time to PAR completion: 54 mins 23 secs
Placement: Completed - No errors found.
Routing: Completed - No errors found.
Timing: Completed - 8 errors found.

PAR done.
```

Figure 5-3 Synthesis result with Xilinx Virtex XCV1000E-8 FPGA

Chapter 6

Conclusion and future work

This thesis presents a VLSI implementation design of DVB/RCS non-binary code decoder and describes several related issues of the hardware implementation. The optimal pipelined architecture of the DVB/RCS decoder with Max-Log-MAP algorithm proposed in this thesis is proved to be very efficient, making the decoder achieve a very good performance. Several quantization schemes for the received bits and the extrinsic information are exploited and the best choice considering the tradeoff between the hardware complexity and the performance are discussed. After discussing a number of existing renormalization schemes, a new optimal renormalization approach is proposed and used in this design, which improves the decoder performance in terms of speed greatly. A practical simplification method of the gamma function computation design in this thesis contributes a lot to the reduction of the decoder area and power consumption. In this design, the alpha and beta recursion circuit, the performance limiting part, is optimized in speed to allow the hardware implementation with a high data rate; a concise interleave generation circuit is realized and optimized in area. At last, a further area

reduction method for pipelined structure is proposed, which could tremendously save the area of the DVB/RCS decoder by 30%-40%.

The whole design has been implemented and verified by both simulation and in-chip test with Xilin Virtex FPGA. By slight modification, this design can be easily mapped into other more advanced technologies. As a result, the performance of the DVB/RCS decoder will be improved considerably.

Future work

Although this design has been proved to be successful, there is still room for improvement. First, there are many details in this design that can be refined to improve the performance, especially the area. Secondly, currently this design is implemented in Virtex-E technology, thus to improve performance mapping this design into an advanced ASIC technology is necessary. Thirdly, till now the whole design has not considered and optimized the power consumption aspect. For some applications, power consumption is a main concern so we should further improve this design to save power consumption, for example, by using iteration early stop criterion. Finally, in some cases where higher speed is needed, sliding window algorithm should be exploited because with this algorithm the decoding speed could be increased greatly [68].

Bibliography

[1] Shu Lin and D.J.Costello, Error Control Coding: Fundamentals and Applications, Englewood Cliffs, N.J.: Prentice Hall Inc., 1993.

[2] C. E. Shannon, A Mathematical Theory of Communication, Bell Syst. Tech. J. 27, pp. 379-423 (Part I), 623-656 (Part II), July 1948.

[3] R.W.Hamming, Error Detecting and Error Correcting Codes, Bell syst. Tech. J. 29, pp. 147-160, April 1950.

[4] E. Prange, Cyclic Error-Correcting Codes in Two Symbols, AFCRC-TN-57, 103, Air Force Cambridge Research Center, Cambridge, Mass., September 1957.

[5] J. E. Meggitt, Error Corecting Codes and Their Implementation, IRE Trans. Inf. Theory, IT-7, pp. 232-244, October 1961.

[6] T. Kassami, A Decoding Method for Multiple-Error-Correcting Cyclic Codes by Using Threshold Logics, Conv. Rec. Info Process. Soc. Jap, Tokyo, November 1961.

[7] M. E. Mitchell et al., Coding and Decoding Operation Research, G.E. Advanced Electronics Final Report on Contract AF 19 (604)-6183, Air Force Cambridge Research Center, Cambridge, Mass., 1961.

[8] M. E. Mitchell, Error-Trap Decoding of Cyclic Codes, G.E. Report No. 62MCD3, General Electric Corporation, Oklahoma City, Okla., December 1962.

[9] L. Ruodolph, Easily Implemented Error-Correction Encoding-Decoding, G.E. Report No. 62MCD2, General Electric Corporation, Oklahoma City, Okla, December 1962.

[10] M.J.E.Golay, Notes on Digital Coding, Proc. IRE, 37, pp. 657-663, June 1949.

[11] A. Hocquenghem, Error-Correction Codes, Chiffres, 2, pp. 147-156, 1959.

[12] R. C. Bose and D. K. Ray-Chaudhuri, On a Class of Error Correcting Binary Group Codes, Inf. Control, 3, pp. 68-79, March 1960.

[13] I. S. Reed and G. Solomon, Polynomial Codes over Certain Finite Field, J. Soc. Ind. Appl. Math., 8, pp. 300-304, June 1960.

[14] W. W. Peterson, Encoding and Error-Correction Procedures for the Bose-Chaudhuri Codes, IRE Trans. Inf. Theory, IT-6, pp. 459-470, September 1960.

[15] D. Gorenstein and N. Zierler, A class of Cyclic Linear Error-Correcting Codes in P^m Symbles, J. Soc. Ind. Appl. Math., 9, pp. 107-214, June 1961.

[16] E. R. Berlekamp, On Decoding Binary Bose-Chaudhuri-Hocquenghem Codes, IEEE Trans. Inf. Theory, IT-11, pp. 577-580, October 1965.

[17] J. L. Massey, Step-by-Step Decpoding of the Bose-Chaudhuri-Hocquenghem Codes, IEEE Trans. Inf. Theory, IT-11, pp. 580-585, October 1965.

[18] R. T. Chien, Cyclic Decoding Procedure for the Bose-Chaudhuri-Hocquenghem Codes, IEEE Trans. Inf. Theory, IT-10, pp. 357-363, October 1964.

[19] G. D. Forney, On Decoding BCH Codes, IEEE Trans. Inf. Theory, IT-11, pp. 549-557, October 1965.

[20] N. Abramson, A class of Systematic Codes for Non-independent Errors, IRE Trans. Inf. Theory, IT-5, pp. 150-157, December 1959.

[21] P. Fire, A Class of Multiple-Error-Correcting Binary Codes for non-independent Errors, Sylvania Report No. REL-E-2, Sylvania Electronic Defense Laboratory, Reconnaissance Systems Division, Mountain View, Calif., March 1959.

[22] I. S. Reed, A Class of Multiple Error-correcting Codes and Decoding Scheme, IRE Tran., IT-4, pp. 38-49, September 1954.

[23] L. D. Rudolph, a class of Majority Logic Decodable Codes, IEEE Trans. Inf. Theory, IT-13, pp. 305-307, April 1967.

[24] P. Elias, Coding for Noisy Channel, IRE Conv. Rec., Part 4, pp. 37-47, 1955.

[25] J. M. Wozencraft and B. Reiffen, Sequential Decoding, MIT Press, Cambridge, Mass., 1961.

[26] J. L. Massey, Threshold Decoding, MIT Press, Cambridge, Mass., 1963.

[27] A. J. Viterbi, Error Bounds for Convolution Codes and an Asymptotically Optimum Decoding Algorithm, IEEE Trans. Inf. Theory, IT-13, pp.260-269, April 1967.

[28] J. K. Omura, On the Viterbi Decoding Algorithm, IEEE Trans., Inf. Theory, IT-13, pp. 177-179, Jan. 1969.

[29] G. D. Forney, Convolutional Codes II: Maximum Likelihood Decoding, Inf. Control, 25, pp. 222-266, July 1974.

[30] G. D. Forney, Convolutional Codes, Cambridge, MA: MIT Press, 1966.

[31] C. Berrou, A. Glavieux and P. Thitimajshima, Near Shannon Limit Error-correcting Coding and Decoding: Turbo-codes(1), IEEE Proc., Int. Conf. on Communications, pp. 1064-1070, Geneva, Switzerland, May 1993.

[32] L. Bahl, J. Cocke, F. Jelinek, and J. Raviv, Optimal Decoding of Linear Codes for Minimizing Symbol Error rate, IEEE Trans. Inf. Theory, IT-20, pp. 284-287, March 1974.

[33] J. Hagenauer, Iterative Decoding of Binary Block and Convolutional Codes, IEEE Trans. Inf. Theory, Vol. 42, NO. 2, pp. 429-445, March 1996.

[34] C. Douillard, M. Jezequel, C. Berrou, N. Brengarth, J. Tousch and N. Pham, The Turbo Code Standard for DVB-RCS, IEEE 2nd Inter. Symposium on Turbo Codes & Related Topics, Brest France, September 2000.

[35] C. Berrou, C. Douillard and M. Jezequel, Multiple Parallel Concatenation of Circular Recursive Convolutional (CRSC) Codes, Annals of Telecommunications, Vol. 54, No. 3-4, March-April 1999.

[36] C. Berrou and M. Jezequel, Non-binary Convolutional Codes for Turbo Coding, Electronics Letters, Vol. 35, No. 1, pp. 39-40, January 1999.

[37] N. H. Weste, Principles of CMOS VLSI Design, Addison Wesly, 1994.

[38] Viterbi Decoder Coprocessor User's Guide, Texas Instruments, 30 April 2001.

[39] G. Masera, G. Piccinini, M. R. Roch, and M. Zamboni, VLSI Architectures for Turbo Codes, IEEE Trans. on VLSI sys., Vol. 7, No. 3, September 1999.

[40] A. Worm, H. Lamm, and N. When, VLSI Architectures for High-speed MAP Decoders, IEEE Proc. pp. 446-453, 2000.

[41] F. Viglione, G. Masera, R. R. Roch, and M. Zamboni, A 50M Bit/s Turbo decoder, Proc. Automation and Test in Europe Conference and Exihibition 2000.

[42] A. Worm, H. Michel, F. Gilbert, G. Kreiselmair, M. Thul, and N. Welmi, Advanced Implementation Issues of Turbo Decoders, IEEE 2nd Inter. Symposium on Turbo Codes & Related Topic, pp.351-354, Brest France, September 2000.

[43] Y. Z. Cheng, and Y. T. Su, A New Real-time MAP Decoding Algorithm, IEEE 2nd Inter. Symposium on Turbo Codes & Related Topic, pp.508-510, Brest France, September 2000.

[44] N. V. Stralen and S. Hladik, Design of Turbo Decoder ASIC, IEEE 2nd Inter. Symposium on Turbo Codes & Related Topic, pp. 275-278, Brest France, September 2000.

[45] F. Berens, A. Worm, H. Michel, and N. When, Implementation Aspects of Turbo Decoders for Future Radio Applications, IEEE Inter. Conference on VTC, pp. 2601-2605, 1999.

[46] A. Worm, H. Lamm, and N. When, A High-speed MAP Architecture with Optimized Memory Size and Power Consumption, IEEE Workshop on Signal Processing Systems, pp. 265 –274, 2000.

[47] F. Berens, T. Bing, H. Michel, A. Worm, and P. W. Baier, Performance of low Complexity Turbo-codes in the UTRA-TDD-mode, IEEE VTS 50th, Vol. 5, pp. 2621-2625, 1999.

[48] F. Gilbert, A Worm, and N. When, Low Power Implementation of a Turbo Decoder on Programmable Architectures, Proc. of the ASP-DAC Asia and South Pacific, pp. 400-403, 2001.

[49] H. Michel, A, Worm, M. Munch, and N. When, Hardware/software Trade-offs for Advanced 3G Channel Coding, Proc. of Design, Automation and Test in Europe Conference and Exhibition, pp 396-401, 2002.

[50] A, Worm, H. Lamm, and N. When, Design of Low-power High-speed Maximum a Priori Decoder Architectures, Proc. of Design, Automation and Test in Europe Conference and Exhibition, pp. 258-265, 2001.

[51] A. Worm, P. Hoeher, and N. When, Turbo-decoding Without SNR Estimation, IEEE Communications Letters, Volume: 4 Issue: 6, pp. 193-195, June 2000.

[52] C. Douillard, M. Jezequel, C. Berrou, N. Brengarth, J. Tousch, and N. Pham, The Turbo Code for DVB-RCS. IEEE 2nd Inter. Symposium on Turbo codes, Brest France, September 2000.

[53] N. Brengarth, R. Novello, N.Pham, V. Piloni, and J. Tousch, DVB-RCS Turbo Code on a Commercial OPB Satellite Payload: SKYPLEX, IEEE 2nd Inter. Symposium on Turbo Codes, Brest France, September 2000.

[54] S. Benedetto and G. Montorsi, Unveiling Turbo-code: Some Results on Parallel Concatenated Coding Schemes, IEEE Trans. Inf. Theory, Vol. 42, No. 2, pp. 409-428, March 1996.

[55] S. Benedetto and G. Montorsi, Design of Parallel Concatenated Convolutional Codes, IEEE Trans. Commun., Vol. 44, No. 5, pp. 591-600, May 1996.

[56] European Telecommunication Standard Institute "Digital Video Broadcasting : Interactive Channel for Satellite Distribution Channel" (DVB/RCS) EN 301 790.

[57] P. Robertson, P. Hoeher, Optimal and Sub-Optimal Maximum a Posteriori Algorithms Suitable for Turbo Decoding, IEEE Trans, Comm. Theory, Vol. 8, No.2, March-April 1997.

[58] P. Robertson, E. Villebrum, and P. Hoeher: A Comparison of Optimal and Sub-optimal MAP Decoding Algorithms Operating in The Log Domain, Proc. ICC'95, pp. 1009-1013, June 1995.

[59] M. R. Soleymani, Y. Z. Gao, and U. Vilaipornsawai, Turbo Coding for Satellite and Wireless Communications, Kluwer Academic Publishers, Norwell, Mass. USA, 2002.

[60] Z. F. Wang, H. Suzuki, and K.K.Parhi, VLSI Implementation Issuses of Turbo Decoder Design for Wireless Applications, IEEE Proc. SiPC'99, pp. 503-512, October 1999.

[61] H. Michel, and N. When, Turbo Decoder Quantization for UMTS, IEEE Comm. Letter, Vol 5, No.2, pp. 55-57, February 2001.

[62] J. Yi, S. Hong and W. E. Stark, VLSI Design and Implementation of Low-complexity Adaptive Turbo-code Encoder and Decoder for Wireless Mobile Communication Applications, IEEE Workshop on Signal Processing Systems, Design and Implementation, 1998.

[63] H. Michel, A. Worm, and N. When, Influence of Quantization on the Bit-error Performance of Turbo Decoders, IEEE Inter. Conference on VTC, pp. 581-585, 2000.

[64] A. Hekstra, An Alternative to Metric Rescaling in Viterbi Decoders, IEEE Tran. Symposium on Comm., pp. 1220-1222, November 1989.

[65] C. B. Shung, VLSI Architectures for Metric Normalization in Viterbi Algorithm, IEEE Proc. ICC'90, pp. 1723-1728, 1990.

[66] T. Kropf, Introduction to Formal Hardware Verification, Spring Verlag, 1999.

[67] C. Meinel, T.theobald, Algorithms and Date Structures in VLSI Design, Springer, 1998.

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

[69] M. J. Smith, Application Specific Integrated Circuits, Addison Wesly, 1997.

[70] D.A. Patterson and J. L. Hennessy, Computer Orhanization and Design: The Hardware/Software Interface, Morgan Kaufmanu, 1997.

[71] G. D. Michelli, Synthesis and Optimization of Digital Circuits, McGraw-Hill, 1994.

[72] Xilinx Virtex-E FPGA Development Kit User's Guide, Avnet, 2001.

[73] Turbo Decoder Coprocessor User's Guide, Texas Instruments, 30 Apr. 2001.