

國立交通大學
電子工程學系
碩士論文

高速渦輪解碼器晶片設計及其在CCSDS系統上
的應用

High Throughput Turbo Decoder Chip Implementation for
CCSDS System Applications

研究生：莊翔琮

指導教授：方偉騏 博士

中華民國九十八年六月

高速渦輪解碼器晶片設計及其在 CCSDS 系統上的應用

High Throughput Turbo Decoder Chip Implementation
for CCSDS System Applications

研究生：莊翔琮

Student : Hsiang-Tsung Chuang

指導教授：方偉騏

Advisor : Wai-Chi Fang

國立交通大學 電子工程學系

電子研究所 碩士班

碩士論文

A Thesis

Submitted to Department of Electronics Engineering & Institute of Electronics

College of Electrical and Computer Engineering

National Chiao Tung University

In Partial Fulfillment of the Requirements

for the Degree of

Master of Science

in

Electronics Engineering

June 2009

Hsinchu, Taiwan, Republic of China

中華民國九十八年六月

高速渦輪解碼器晶片設計及其在 CCSDS 系統上的應用

研究生：莊翔琮

指導教授：方偉騏 博士

國立交通大學電子工程學系(研究所)碩士班

中文摘要

由於渦輪碼有著優異的錯誤更正能力，所以在近十年來已經被廣泛的運用在通訊系統上。然而由於渦輪碼複雜的結構使得其速度無法有效提升，本論文將改善解碼器的架構使渦輪解碼器速度有效提升。

由於渦輪碼的時脈是被遞迴結構所限制的，我們利用偏移加法-比較-選擇器和一級 CSA 的架構，來減少主要路徑延遲；除此之外，我們更進一步提出了 hybrid 4-inputs addition/subtraction 基數-4 的遞迴結構使得此架構的吞吐量和傳統的遞迴結構相比有近 80% 的提升。另一方面，傳統渦輪解碼必須跑到固定次數的迭代以確保事前資訊已經收斂，但如此一來造成速度慢，高延遲和功率浪費。事實上，當通道狀況好的時候，渦輪解碼會提早收斂，因此，藉由分析，我們選用 HDA2 提早停止方法來降低迭代次數來達到高吞吐量的目的。

根據實驗分析，此渦輪解碼器在 UMC90 nm 製程下最高能達到的時脈頻率為 357.14MHz，以及在單塊 MAP 解碼器之下，渦輪解碼器能達到 77.62MS/s 的傳輸速度，晶片面積為 1.59mm²。另外，由於平行化的渦輪解碼會發生記憶體碰撞的問題，我們可以利用修正過的退火演算法將這問題解決，並且在十四塊 MAP 解碼器之下，渦輪解碼器能達到 884.91MS/s 的傳輸速度，晶片面積為 17.64mm²。

High Throughput Turbo Decoder Chip Implementation for CCSDS System Applications

Student: Hsiang-Tsung Chuang Advisor: Dr. Wai-Chi Fang

Department of Electronics Engineering Institute of Electronics

National Chiao Tung University

Abstract

Turbo codes have been applied widely in communication systems over the last decade due to its excellent error correction ability. However, because of complex structure, the data rate of turbo decoder could not improve more efficiently. Therefore, the thesis presents improved architectures to increase its data rate.

The operating frequency of turbo decoder is greatly limited by the recursion unit. In order to decrease the critical path delay, the OACS and one stage CSA structure is employed. Furthermore, the hybrid 4-inputs addition/subtraction radix-4 recursion architecture is presented for CCSDS turbo decoder and finally the relative throughput of proposed recursion unit is faster than traditional one around 80%. On the other hand, the decoding process has to run a certain number of iterations to ensure the extrinsic have converged. In fact, turbo decoder may converge earlier when the channel condition is good. Hence, an early stopping criterion could be employed to reduce the number of iterations.

After chip implementation in 90nm process, the maximum clock rate 357.14MHz can be achieved, and the 1.59mm² core area can support the maximum data rate 77.62MS/s of turbo decoder with single MAP decoder. Besides, if the parallel MAP decoders are considered, the memory collision could be happened. We can introduce the modified annealing algorithm to solve the collision problems. The 17.64mm² core area can support the maximum data rate 884.91MS/s of turbo decoder with fourteen MAP decoders.

誌謝

這些年的碩士生涯，讓我收穫良多，在交大的這段日子，讓我學到了許多做學問的方法，另外感謝指導教授方偉騏老師這兩年的指導及幫助，讓我能夠在這麼好的環境中做研究，另外也要感謝王盛弘學長和 Si2 實驗室的陳志龍學長幫忙架設實驗室的工作站讓我們在跑模擬的時候能夠無後顧之憂。

再來要感謝實驗室的學長、同學及學弟們，在這一些日子來互相幫忙一起建立起實驗室和一起打球。另外感謝同學凱信，在每次遇到瓶頸的時候，總是能夠互相打氣，在研究上也能夠互相討論。最後感謝我的父母親讓我在求學的生涯能夠不必擔心到生活上的問題，能讓我在實驗室裡面專心的做研究，最後我要再次向每個幫助過我的人說聲謝謝。



CONTENTS

口試委員會審定書	#
中文摘要	i
Abstract	ii
誌謝	iii
CONTENTS	iv
LIST OF FIGURES	vii
LIST OF TABLES	x
Chapter 1 Introduction	1
1.1 Background of Turbo Codes	1
1.2 Motivation and Objective	1
1.3 Thesis Organization	2
Chapter 2 Overview of Turbo Codes System	3
2.1 The Structure of Turbo Code	3
2.1.1 Encoder of Turbo Code	3
2.1.2 CCSDS Encoder	4
2.1.3 Decoder of Turbo Code	8
2.2 The Turbo Decoder Algorithm	10
2.2.1 The MAP Algorithm	10
2.2.2 The Log-MAP Algorithm	16
2.2.3 The Maximum Log (ML) MAP Algorithm	17
2.3 Sliding Window Method for Turbo Decoding	20
Chapter 3 Turbo Decoder Design Consideration	24

3.1	The Proposed Structure of Parallel Turbo Decoder.....	24
3.2	The Parallel Turbo Decoder.....	25
3.2.1	Sliding Window Timing Diagram.....	25
3.2.2	Parallel Sliding Window Decoding.....	26
3.2.3	The Interleaver of Parallel Turbo Decoders.....	27
3.3	Early Stopping Criteria.....	33
Chapter 4	Soft-In-Soft-Out (SISO) Decoder Design Consideration.....	37
4.1	SISO Decoder Architecture.....	37
4.2	Radix-4 Log-MAP Algorithm.....	38
4.3	The Architecture of Recursion State Metric.....	40
4.3.1	OASC Structure.....	41
4.3.2	Proposed Radix-4 Log-MAP Recursion State Metric.....	43
4.3.3	The State Metric Normalization.....	47
4.4	The Structure of Branch Metric.....	49
4.5	The Structure of Log-Likelihood Ratios (LLR).....	51
4.5.1	Traditional LLR Computation Unit (LCU) Based on Radix-2 Log-MAP Algorithm.....	51
4.5.2	LLR Computation Unit (LCU) Based on Radix-4 Log-MAP Algorithm.....	51
Chapter 5	System Simulation and Performance Analysis.....	53
5.1	The Bit-Width Estimation of Soft-Input Information.....	53
5.2	The Bit-Width Estimation of Lex.....	54
Chapter 6	Turbo Decoder Implementation in FPGA and ASIC.....	58
6.1	The FPGA Implementation Results.....	58
6.2	The ASIC Implementation Results.....	62

6.3 Comparison.....	66
Chapter 7 Conclusions.....	68
REFERENCE.....	69



LIST OF FIGURES

Figure 2.1 Turbo encoder with puncture	4
Figure 2.2 Interpretation of Permutation	7
Figure 2.3 Turbo Encoder Block Diagram	7
Figure 2.4 The basic Structure of Turbo Decoder	9
Figure 2.5 Performance comparison under different iteration numbers in CCSDS interleaver (N=1784, code rate = 1/3, state = 16, MAP algorithm, BPSK)....	9
Figure 2.6 Trellis diagram for 4 states RSC encoder.....	11
Figure 2.7 MAP decoding flow chart	16
Figure 2.8 The correct function	19
Figure 2.9 The BER performance of ML-MAP algorithm compare with Log-MAP algorithm (N=1784, code rate = 1/3, state = 16, MAP algorithm, BPSK) ...	19
Figure 2.10 The turbo decoding trellis diagram including the forward and backward direction	20
Figure 2.11 Timing diagram for sliding window (refer to [8]).....	23
Figure 2.12 Performance comparison under different sizes of sliding window in CCSDS interleaver (N=1784, code rate = 1/3, state = 16, MAP algorithm, BPSK).....	23
Figure 3.1 The proposed turbo decoder structure.....	25
Figure 3.2 (a) Space and time relationship for memory-bank management (b) Space and time relationship for memory-bank management.....	26
Figure 3.3 Schedule of the parallel sliding window technique.....	27
Figure 3.4 (a) conventional turbo decoding without collision (b) parallel turbo decoding with collision problem.....	28

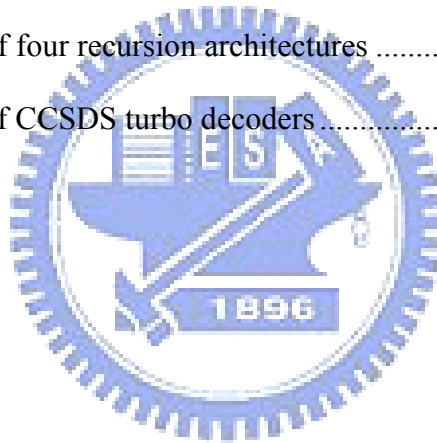
Figure 3.5 (a) The column (b) The tiling of the mapping matrix in this example.....	31
Figure 3.6 The BER performance comparisons of early stopping criteria.....	35
Figure 3.7 The average number of iteration of six early stopping criteria	35
Figure 4.1 Block diagram for sliding window log-MAP decoder (SISO decoder).....	38
Figure 4.2 Trellis diagram (a) Radix-2 trellis (b) Radix-4 trellis	39
Figure 4.3 A traditional recursion architecture (with normalization)	40
Figure 4.4 Three different locations of the register in the data flow of the recursive algorithm result in three kinds of ACSO recursion architecture (refer to [12])	42
Figure 4.5 Architecture of a recursion OACS unit.....	42
Figure 4.6 A radix-4 recursion unit.....	43
Figure 4.7 Improved radix-4 recursion OACS architecture	44
Figure 4.8 Hybrid 4-inputs subtraction	45
Figure 4.9 Structure of GLUT used in improved OACS architecture.....	46
Figure 4.10 Integer ranges at forward and backward recursion arch.	48
Figure 4.11 The structure of radix-4 ACS unit within normalization	49
Figure 4.12 The branch metric unit (BMU) for radix-4 log-MAP algorithm	50
Figure 4.13 Traditional LLR Computation Unit (LCU) Based on Radix-2 Log-MAP Algorithm.....	51
Figure 4.14 Trellis diagram (a) Radix-2 trellis (b) Radix-4 trellis	52
Figure 4.15 LLR Computation Unit (LCU) Based on Radix-4 Log-MAP Algorithm....	52
Figure 5.1 The comparison of BER performance for various soft inputs	54
Figure 5.2 The comparison of BER performance for various soft inputs and extrinsic information	55
Figure 5.3 The comparison of BER performance for various soft inputs and extrinsic	

information	56
Figure 5.4 The comparison of various channel reliability.....	57
Figure 6.1 Development and design flow of the process	59
Figure 6.2 The flow graph of turbo decoder.....	60
Figure 6.3 Turbo decoder I/O diagram under FPGA verification	62
Figure 6.4 ASIC verification flow	63
Figure 6.5 Chip layout of turbo decoder with single SISO decoder	64
Figure 6.6 Chip layout of parallel turbo decoder by SoC Encounter	65
Figure 6.7 Performance comparisons among those three architectures	66



LIST OF TABLES

Table 2.1 Specified Information Block Lengths.....	5
Table 2.2 Codeblock Lengths for Supported Code Rates (Measured in Bits).....	5
Table 2.3 Parameters k_1 and k_2 for Specified Information Block Lengths	6
Table 4.1 ELUT function block approximation.....	46
Table 5.1 Proposed Turbo Decoder Specification	57
Table 6.1 I/O ports definition	61
Table 6.2 Area report for each component of SISO decoder.....	63
Table 6.3 Comparison of four recursion architectures	66
Table 6.4 Comparison of CCSDS turbo decoders	67



Chapter 1 Introduction

1.1 Background of Turbo Codes

Turbo codes [1] were invented in 1993 by C. Berrou, A. Glavieux and P. Thitimajshima. Turbo codes have outstanding error correction performance and their performance near the Shannon capacity limit by 0.7 dB [2]. Therefore, there are many researches on the realizations of turbo codes, and turbo codes have been applied widely for various communication standards, i.e., WiMax (Worldwide Interoperability for Microwave Access) [3], 3GPP (3rd Generation Partnership Project) [4], and CCSDS (Consultative Committee for Space Data Systems) [5].

1.2 Motivation and Objective

Turbo codes have become one of the necessary specifications for the state-of-the-art communication systems. How to efficiently realize the turbo decoder in the integrated circuit always causes much research attention.

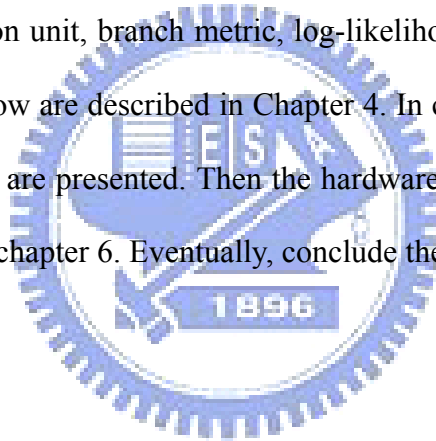
For traditional turbo decoder, it suffers high latency due to the iterative decoding process. However, it requires high throughput and low latency of turbo decoding to apply high throughput communication system. To solve the high latency problem, the parallel SISO decoder architecture could be introduced to minimize the latency. The other draw for traditional turbo decoder is the memory usage, the sliding window (SW) can use to split the recursion of MAP algorithm into sub-recursions to decrease the memory usage heavy.

Our work is to implement a high throughput rate and low latency turbo decoder where the area and the power are enhanced a little. In this thesis, we aim at the turbo

decoder implementation of CCSDS on Field-Programmable Gate Arrays (FPGAs) and automatically place and route (APR).

1.3 Thesis Organization

This thesis consists of six chapters. Chapter 1 introduces the background and motivation of turbo codes. In chapter 2, the basic structures of turbo codes for CCSDS are presented. Moreover, turbo decoding algorithm (BCJR algorithm [6]) also introduces. In chapter 3, we discuss the proposed structure and methods of the turbo decoder, including the mapping interleaving law, and early stopping criteria. The improved radix-4 recursion unit, branch metric, log-likelihood ratio (LLR) unit and the data flow of sliding window are described in Chapter 4. In chapter 5, system simulation and performance analysis are presented. Then the hardware implementation results and comparison are shown in chapter 6. Eventually, conclude the thesis in chapter 7.



Chapter 2 Overview of Turbo Codes System

Turbo codes

Turbo code [1] was invented in 1993 by Berrou, Glavieux and Thitimajshima, has outstanding error correction performance. Special features of turbo code are as follows: (1) Turbo encoder is composed of two parallel-concatenated recursive systematic convolutional code (RSC) with a large block size. (2) A pseudo random interleaver is used to re-permute the input sequence for the second RSC encoder. (3) Turbo decoder uses the maximum a posterior probability (MAP) algorithm. (4) The iterative technology is used. Those features make turbo decoder great ability for error correcting and almost near the Shannon capacity limit.

2.1 The Structure of Turbo Code

2.1.1 Encoder of Turbo Code

Turbo encoder is constructed by two parallel concatenated recursive systematic convolutional (RSC) encoders, each with a small number of states, and an interleaver to separate the RSC encoders (Figure 2.1). Puncturing is an option to increase bit error rate (BER) or speed. After encoding a frame that includes N input bits, we need to make sure the initial state is all-zero state for the next block. Hence, the tail bits need to drive the encoder to all-zero state. The number of tail bits is equal to the number of delay elements of RSC encoder.

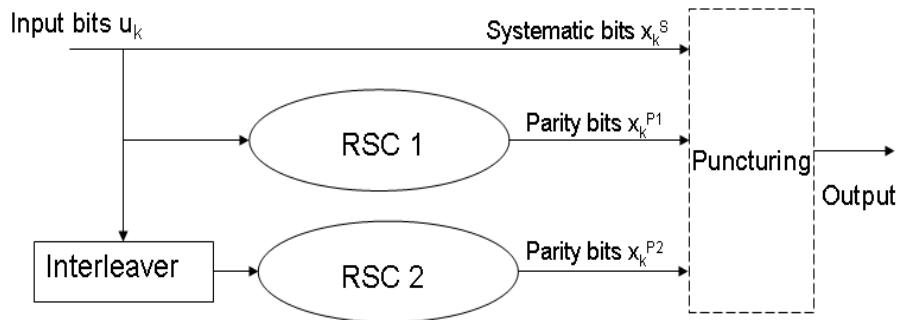


Figure 2.1 Turbo encoder with puncture

2.1.2 CCSDS Encoder

The recommended turbo code is a systematic code with the following specification:

- (a) **Code type:** Systematic parallel concatenated turbo code
- (b) **Number of component codes:** 2
- (c) **Type of component codes:** Recursive convolutional codes
- (d) **Number of states of each**

Convolutional component code: 16

- (e) **Nominal Code Rates:** $r = 1/2, 1/3, 1/4, \text{ or } 1/6$ (selectable)

- (f) The specified message block lengths N are shown in Table 2.1. They are chosen for compatibility with the corresponding Reed-Solomon interleaving depths, also shown in Table 2.1. After encoding a block includes N input messages, we add four bits as tail bits. The corresponding code block lengths in bits: $n=(k+4)/r$, for the specified code rates are shown in Table 2.2

Table 2.1 Specified Information Block Lengths

Information block length N, bits	Corresponding Reed-Solomon interleaver depth I	Notes
1784 (=223*1 octets)	1	For very low data rates or latency ↓ For highest coding gain
3568 (=223*2 octets)	2	
7136 (=223*4 octets)	4	
8920 (=223*5 octets)	5	
16384	Not Applicable	

Table 2.2 Codeblock Lengths for Supported Code Rates (Measured in Bits)

Information block length N	Codeblock length n			
	Rate 1/2	Rate 1/3	Rate 1/4	Rate 1/6
1784	3576	5364	7152	10728
3568	7144	10716	14288	21432
7136	14280	21420	28560	42840
8920	17848	26772	35696	53544
16384	32776	49164	65552	98328

(g) Turbo Code Permutation:

The interleaver for turbo codes is a fixed bit-by-bit permutation of the entire block of data. Unlike the symbol-by-symbol rectangular interleaver used with Reed-Solomon codes, the turbo code permutation scrambles individual bits and resembles a randomly selected permutation in its lack of apparent orderliness.

The recommended permutation for each specified block length k is given by a particular reordering of the integers 1, 2, ..., k as generated by the following algorithm:

- 1st step: Express k as k_1k_2 . The parameters k_1 and k_2 for the specified block

sizes are given in Table 2.3

- 2nd step: Do the following operation for s=1 to s=k to obtain permutation numbers $\pi(s)$, p_q denotes one of the following eight prime integers:

$$p_1 = 31; p_2 = 37; p_3 = 43; p_4 = 47; p_5 = 53; p_6 = 59; p_7 = 61; p_8 = 67$$

Table 2.3 Parameters k_1 and k_2 for Specified Information Block Lengths

Information block length	k_1	k_2
1784	8	223
3568	8	223 x 2
7136	8	223 x 4
8920	8	223 x 5
16384	(note)	(note)
Note – these parameters are currently under study and will be incorporated in a later version		

$$m = (s - 1) \bmod 2$$

$$i = \left\lfloor \frac{s-1}{2k_2} \right\rfloor$$

$$j = \left\lfloor \frac{s-1}{2} \right\rfloor - ik_2$$

$$t = (19i + 1) \bmod \frac{k_1}{2}$$

$$q = t \bmod 8 + 1$$

$$c = (p_q j + 21m) \bmod k_2$$

$$\pi(s) = 2\left(t + c \frac{k_1}{2} + 1\right) - m$$

The interpretation of the permutation numbers is such that the s-th bit read out on line “in b” in Figure 2.3 is the $\pi(s)$ th bit of the input information block, as

shown in Figure 2.2.

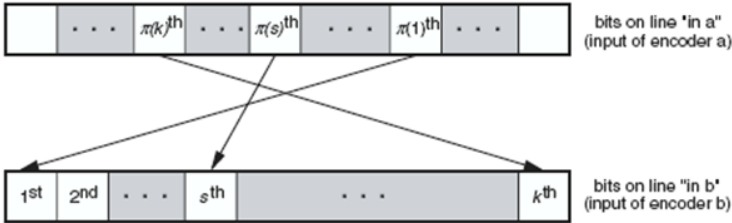


Figure 2.2 Interpretation of Permutation

(h) Turbo Encoder Block Diagram:

All connection vectors (Backward and Forward) for both component codes: $G_0 = (10011)$, $G_1 = (11011)$, $G_2 = (10101)$, $G_3 = (11111)$. In Figure 2.3, each input frame of N information bits is held in a frame buffer, and the bits in the buffer are read out in two different orders for the two RSC encoders. The first component encoder (a) operates on the bits in unpermuted order (“in a”), while the second component encoder (b) receives the same bits permuted by the interleaver (“in b”)

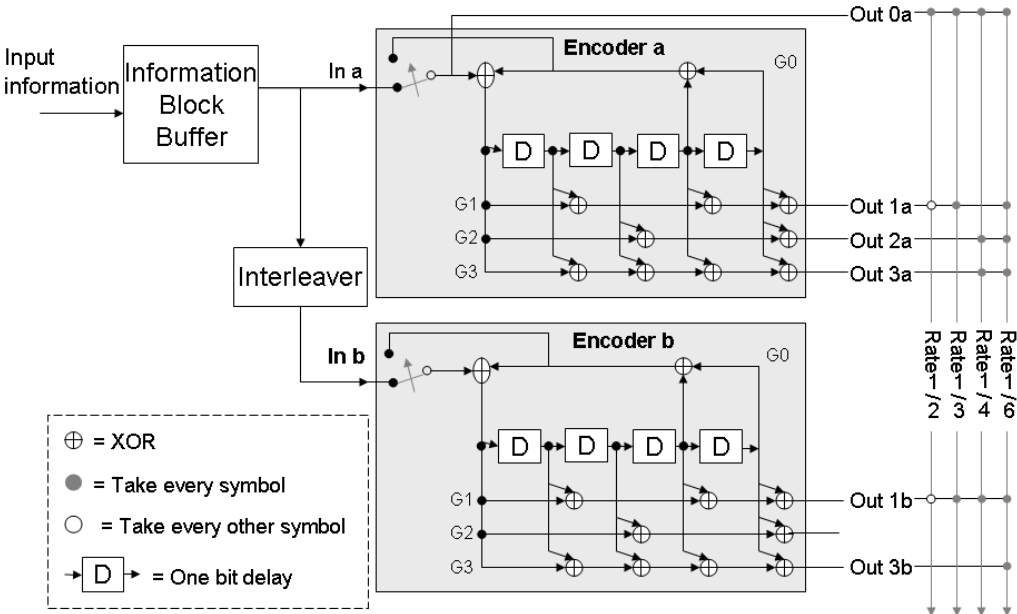


Figure 2.3 Turbo Encoder Block Diagram

2.1.3 Decoder of Turbo Code

The basic structure of turbo decoder is illustrated in Figure 2.4, the main components are two soft-in soft-out (SISO) decoders, interleaver and de-interleaver. Those three input sequence are received from channel, where y_s is the received systematic sequences, y_{p1} and y_{p2} are the received parity information sequences generated from the first and the second RSC encoder, separately.

The general turbo decoder consists of two SISO decoders, which serially concatenated via an interleaver or a de-interleaver. The SISO decoder is implemented according to maximum a posterior probability (MAP) algorithm [6] or soft-output Viterbi algorithm (SOVA) [7]. At first, the SISO decoder1 take y_s and y_{p1} as input to produce two kinds of the soft outputs: log-likelihood ratio ($L_{lr,1}$) and extrinsic information (L_{ex1}). After producing L_{ex1} , the L_{ex1} re-permute via interleaver and used as the a priori probabilities (L_{in2}) of the input sequence for the SISO decoder2. Besides, interleaved y_s sequence and take it and y_{p2} as input to produce the log-likelihood ratio ($L_{lr,2}$) and extrinsic information (L_{ex2}). Similarly, the L_{ex2} re-permute via de-interleaver and used as the a priori probabilities (L_{in1}) of the input sequence for the SISO decoder1. Above all procedure, we defined it “one time turbo decoding iteration”. The more iteration procedures, the more decoder performance could be improved. However, there is no evident improvement after a certain number of iterations. This reason is the a priori probabilities (L_{in}) are saturation. After the last iteration, the $L_{lr,2}$ sequences make a hard decision after de-interleaver. Performance comparison under different iteration numbers in CCSDS interleaver is shown in Figure 2.5.

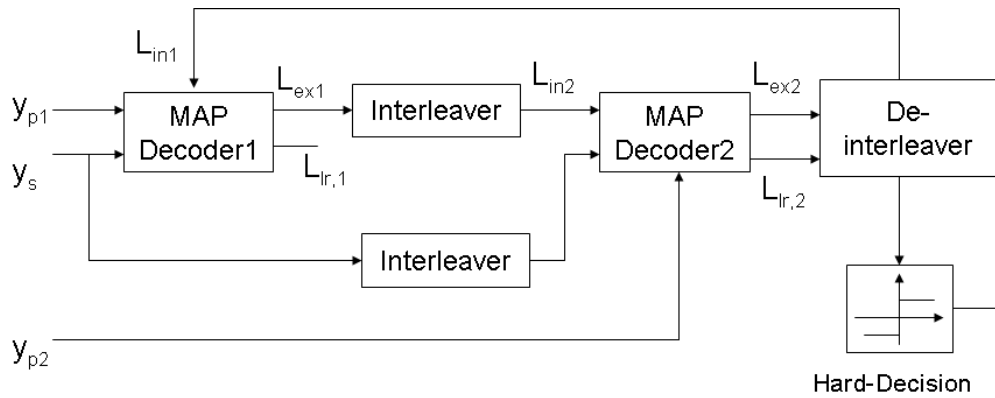


Figure 2.4 The basic Structure of Turbo Decoder

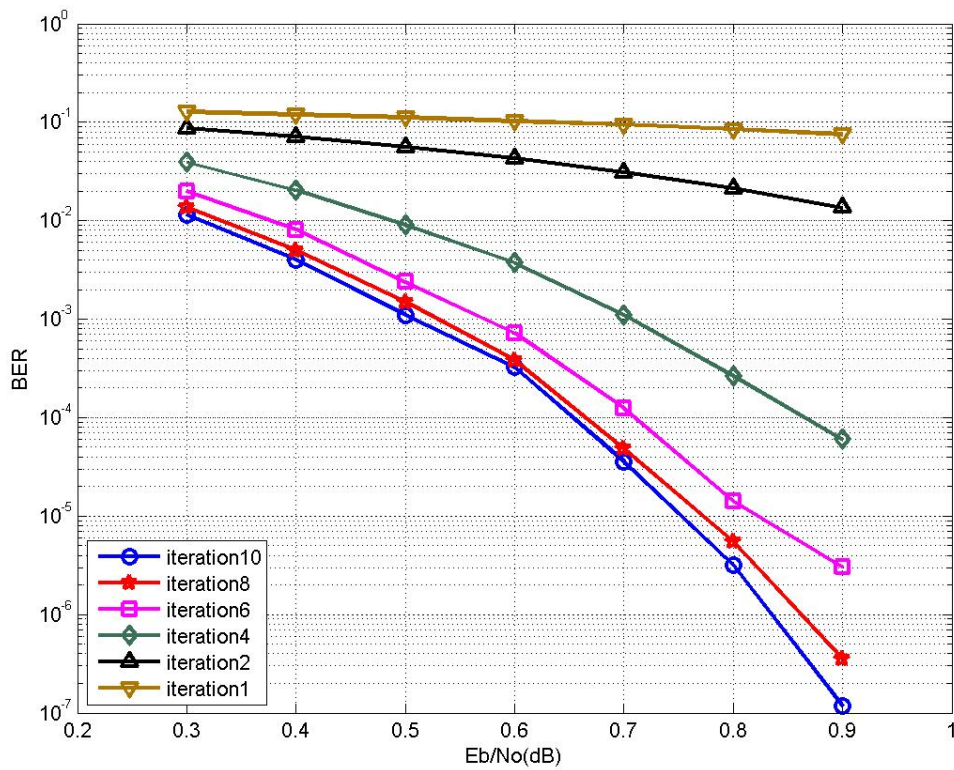


Figure 2.5 Performance comparison under different iteration numbers in CCSDS interleaver (N=1784, code rate = 1/3, state = 16, MAP algorithm, BPSK)

2.2 The Turbo Decoder Algorithm

2.2.1 The MAP Algorithm

The MAP algorithm (so-called BCJR algorithm) [6] was first introduced in 1974 by Bahl, Cocke, Jelinik and Raviv. The MAP algorithm is optimal for estimating the states and the outputs of a Markov process. Due to minimizing the bit (or symbol) error rate (BER), the MAP algorithm generates the soft output (likelihood ratios) defined as $P(u_k | y)$, based on received code sequence y , to estimate the hard value for the transmitted information bit u_k at time k . In order to decision more easier, the logarithm of likelihood ratios (LLR) is used. The LLR of the k^{th} input bit of the input sequence U is defined as:

$$L(u_k) = L(u_k | y) = \ln \frac{P(u_k = +1 | y)}{P(u_k = -1 | y)} \quad (2.1)$$

For $1 \leq k \leq N$, where N is the frame size and the decision rule is defined as:

$$u_k = \begin{cases} +1 & \text{if } L(u_k) \geq 0 \\ -1 & \text{otherwise} \end{cases} \quad (2.2)$$

Figure 2.6 shows a trellis diagram for four states RSC encoder as an example. If the last states $S_{k-1} = s'$ and the present states $S_k = s$, then the input bit u_k can be estimated. Note that the dashed lines express the transitions from S_{k-1} to S_k caused by the input information bit u_k of +1, and the solid lines express the transitions from S_{k-1} to S_k caused by the input information bit u_k of -1. Then the equation (2.1) can be rewritten as:

$$L(u_k) \triangleq \ln \frac{P\{u_k = +1 | y\}}{P\{u_k = -1 | y\}} = \ln \frac{\sum_{u_k=+1} P\{S_{k-1}, S_k | y\}}{\sum_{u_k=-1} P\{S_{k-1}, S_k | y\}} = \ln \frac{\sum_{u_k=+1} P\{S_{k-1}, S_k, y\}}{\sum_{u_k=-1} P\{S_{k-1}, S_k, y\}} \quad (2.3)$$

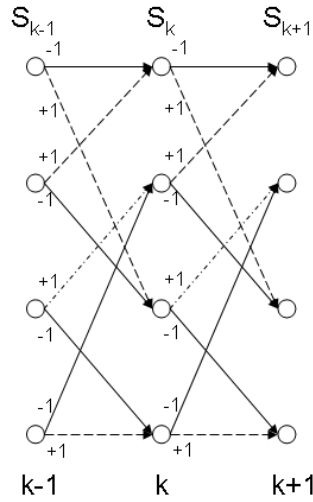


Figure 2.6 Trellis diagram for 4 states RSC encoder

Assume the channel is memoryless and using the Bayes' rule, we can the individual probabilities $P\{S_{k-1}, S_k, y\}$ from the numerator or denominator as:

$$\begin{aligned}
 P\{S_{k-1}, S_k, y\} &= P\{S_{k-1}, S_k, y_{j < k}, y_k, y_{j > k}\} \\
 &= P\{y_{j > k} | S_{k-1}, S_k, y_{j < k}, y_k\} \times P\{S_k, y_k | S_{k-1}, y_{j < k}\} \times P\{S_{k-1}, y_{j < k}\} \quad (2.4) \\
 &\stackrel{\text{Markov}}{=} P\{y_{j > k} | S_k\} \times P\{S_k, y_k | S_{k-1}\} \times P\{S_{k-1}, y_{j < k}\}
 \end{aligned}$$

Now, we defined following metrics:

- The forward recursion metric A :

$$A_{k-1}(s') = P\{S_{k-1}, y_{j < k}\} \quad (2.5)$$

- The backward recursion metric B :

$$B_k(s) = P\{y_{j > k} | S_k\} \quad (2.6)$$

- The branch transition metric Γ :

$$\Gamma_k(s', s) = P\{S_k, y_k | S_{k-1}\} \quad (2.7)$$

and the assumption that the channel is memoryless and using Bayes' rule, $A_k(s)$ can be derived from (2.8)

$$\begin{aligned}
A_k(s) &= P\{S_k, y_{j \leq k}\} \\
&= \sum_{\text{all } S_{k-1}} P\{S_{k-1}, S_k, y_{j \leq k}\} \\
&= \sum_{\text{all } S_{k-1}} P\{S_k, y_k | S_{k-1}, y_{j < k}\} P\{S_{k-1}, y_{j < k}\} \\
&\stackrel{\text{Markov}}{=} \sum_{\text{all } S_{k-1}} P\{S_k, y_k | S_{k-1}\} P\{S_{k-1}, y_{j < k}\} \\
&= \sum_{\text{all } S_{k-1}} \Gamma_k(s', s) \cdot A_{k-1}(s')
\end{aligned} \tag{2.8}$$

Note that since the registers are empty at the beginning in the turbo encoder, so we assume the trellis has the initial state $S_0 = 0$, the initial conditions for $A_0(s)$ are:

$$\begin{cases} A_0(S_0 = 0) = 1 \\ A_0(S_0 = s) = 0 \quad \text{for all } s \neq 0 \end{cases} \tag{2.9}$$

Similar to the derivation of $A_k(s)$, $B_k(s)$ can be written as:

$$\begin{aligned}
B_k(s) &= P\{y_{j > k} | S_k\} \\
&= \sum_{\text{all } S_{k+1}} P\{S_{k+1}, y_{j > k} | S_k\} \\
&= \sum_{\text{all } S_{k+1}} P\{S_{k+1}, y_{k+1}, y_{j > k+1}, S_k\} / P\{S_k\} \\
&= \sum_{\text{all } S_{k+1}} P\{y_{j > k+1} | S_{k+1}, y_{k+1}, S_k\} P\{S_{k+1}, y_{k+1} | S_k\} \\
&= \sum_{\text{all } S_{k+1}} P\{y_{j > k+1} | S_{k+1}\} P\{S_{k+1}, y_{k+1} | S_k\} \\
&= \sum_{\text{all } S_{k+1}} B_{k+1}(s'') \cdot \Gamma_{k+1}(s, s'')
\end{aligned} \tag{2.10}$$

Note that since the registers are empty at the ending in the turbo encoder, so we assume the trellis has the initial state $S_N = 0$, the initial conditions for $B_N(s)$ are:

$$\begin{cases} B_N(S_N = 0) = 1 \\ B_N(S_N = s) = 0 \quad \text{for all } s \neq 0 \end{cases} \tag{2.11}$$

Now, we know to calculate the forward recursion metric and the backward recursion metric that needs to acquire the branch transition metric first. The derivation of the branch transition metric $\Gamma_k(s', s)$ as below:

$$\begin{aligned}
\Gamma_k(s',s) &= P\{S_k, y_k | S_{k-1}\} \\
&= \frac{P\{S_k, y_k, S_{k-1}\}}{P\{S_{k-1}\}} \\
&= \frac{P\{S_k, S_{k-1}\}}{P\{S_{k-1}\}} \times \frac{P\{S_k, S_{k-1}, y_k\}}{P\{S_k, S_{k-1}\}} \\
&= P\{S_k | S_{k-1}\} \times P\{y_k | S_k, S_{k-1}\} \\
&= P(u_k)P(y_k | x_k)
\end{aligned} \tag{2.12}$$

Where u_k is the input bit which would cause the transition from state $S_{k-1} = s'$ to state $S_k = s$ as illustrated in Figure 2.6, x_k and y_k are the corresponding transition codeword and the received symbol from channel, separately. Note that $P(u_k)$ is the a-priori probability of the input bit u_k . According to the definition of the a-priori log-likelihood ratio:

$$L_a(u_k) \triangleq \ln \frac{P\{u_k = +1\}}{P\{u_k = -1\}} \tag{2.13}$$

The a-priori probability can be expressed as:

$$\begin{aligned}
P\{u_k = \pm 1\} &= \frac{e^{\pm L_a(u_k)}}{1 + e^{\pm L_a(u_k)}} \\
&= \left[\frac{e^{-L_a(u_k)/2}}{1 + e^{-L_a(u_k)}} \right] \cdot e^{u_k \cdot L_a(u_k)/2} \\
&= A_k \cdot e^{u_k \cdot L_a(u_k)/2}
\end{aligned} \tag{2.14}$$

For a given $L_a(u_k)$, the parameter A_k is independent of the actual value of $u_k = +1$ or -1 .

For an addition white Gaussian noise (AWGN) channel, the term $P(y_k | x_k)$ in (2.12) can be written as:

$$\begin{aligned}
P(y_k | x_k) &= \left(\frac{1}{\sqrt{2\pi\sigma^2}} \right)^n e^{-\frac{\sum_{i=1}^n (y_k^{(i)} - x_k^{(i)})^2}{2\sigma^2}} \\
&= \left\{ \left(\frac{1}{\sqrt{2\pi\sigma^2}} \right)^n e^{-\frac{\sum_{i=1}^n [(y_k^{(i)})^2 + (x_k^{(i)})^2]}{2\sigma^2}} \right\} \cdot e^{-\frac{\sum_{i=1}^n y_k^{(i)} \cdot x_k^{(i)}}{\sigma^2}} \\
&= B_k e^{\sum_{i=1}^n L_c y_k^{(i)} \cdot x_k^{(i)} / 2}
\end{aligned} \tag{2.15}$$

Here, $L_c = 4E_s / N_0$ is called channel reliability. The terms A_k and B_k in (2.14) and (2.15) are equal for all transitions at the same time index, and hence will omit those terms in the following. Therefore, the branch transition metric can be rewritten to the equation (2.16):

$$\begin{aligned}
\Gamma_k(s', s) &= P(u_k)P(y_k | x_k) \\
&= A_k \cdot e^{u_k \cdot L_c(u_k)/2} \cdot B_k e^{\sum_{i=1}^n L_c y_k^{(i)} \cdot x_k^{(i)} / 2} \\
&\stackrel{\text{drop } A_k \& B_k}{=} e^{u_k \cdot L_c(u_k)/2} \cdot e^{\sum_{i=1}^n L_c y_k^{(i)} \cdot x_k^{(i)} / 2}
\end{aligned} \tag{2.16}$$

Final, substituting (2.8), (2.10) and (2.16) into (2.3), the LLR value can be further expressed as:

$$\begin{aligned}
L(u_k) &\triangleq \ln \frac{P\{u_k = +1 | y\}}{P\{u_k = -1 | y\}} \\
&= \ln \frac{\sum_{u_k = +1} P\{S_{k-1}, S_k | y\}}{\sum_{u_k = -1} P\{S_{k-1}, S_k | y\}} \\
&= \ln \frac{\sum_{u_k = +1} P\{S_{k-1}, S_k, y\}}{\sum_{u_k = -1} P\{S_{k-1}, S_k, y\}} \\
&= \ln \frac{\sum_{(s', s) \in C_k^{+1}} A_{k-1}(s') \Gamma_k(s', s) B_k(s)}{\sum_{(s', s) \in C_k^{-1}} A_{k-1}(s') \Gamma_k(s', s) B_k(s)}
\end{aligned} \tag{2.17}$$

By the way, the LLR value can be also expressed as:

$$\begin{aligned}
L(u_k) &= \ln \frac{\sum_{(s',s) \in C_k^{+1}} A_{k-1}(s') \Gamma_k(s',s) B_k(s)}{\sum_{(s',s) \in C_k^{-1}} A_{k-1}(s') \Gamma_k(s',s) B_k(s)} \\
&= \ln \frac{\sum_{(s',s) \in C_k^{+1}} A_{k-1}(s') [P(u_k = +1) P(y_k | x_k)] B_k(s)}{\sum_{(s',s) \in C_k^{-1}} A_{k-1}(s') [P(u_k = -1) P(y_k | x_k)] B_k(s)} \\
&= L_a(u_k) + \ln \frac{\sum_{(s',s) \in C_k^{+1}} A_{k-1}(s') \left[\exp\left(-\frac{\sum_{i=1}^n (y_k^{(i)} - x_k^{(i)})^2}{2 \cdot \sigma^2}\right) \right] B_k(s)}{\sum_{(s',s) \in C_k^{-1}} A_{k-1}(s') \left[\exp\left(-\frac{\sum_{i=1}^n (y_k^{(i)} - x_k^{(i)})^2}{2 \cdot \sigma^2}\right) \right] B_k(s)} \\
&= L_a(u_k) + \ln \frac{\sum_{(s',s) \in C_k^{+1}} A_{k-1}(s') \left[\exp\left(-\frac{(y_k^{(1)} - x_k^{(1)})^2 + \sum_{i=2}^n (y_k^{(i)} - x_k^{(i)})^2}{2 \cdot \sigma^2}\right) \right] B_k(s)}{\sum_{(s',s) \in C_k^{-1}} A_{k-1}(s') \left[\exp\left(-\frac{(y_k^{(1)} - x_k^{(1)})^2 + \sum_{i=2}^n (y_k^{(i)} - x_k^{(i)})^2}{2 \cdot \sigma^2}\right) \right] B_k(s)} \\
&= L_a(u_k) + \ln \frac{\left[\exp\left(-\frac{(y_k^{(1)} - 1)^2}{2 \cdot \sigma^2}\right) \cdot \sum_{(s',s) \in C_k^{+1}} A_{k-1}(s') \right] \left[\exp\left(-\frac{\sum_{i=2}^n (y_k^{(i)} - x_k^{(i)})^2}{2 \cdot \sigma^2}\right) \right] B_k(s)}{\left[\exp\left(-\frac{(y_k^{(1)} + 1)^2}{2 \cdot \sigma^2}\right) \cdot \sum_{(s',s) \in C_k^{-1}} A_{k-1}(s') \right] \left[\exp\left(-\frac{\sum_{i=2}^n (y_k^{(i)} - x_k^{(i)})^2}{2 \cdot \sigma^2}\right) \right] B_k(s)} \\
&= L_a(u_k) + \frac{4 \cdot y_k^{(1)}}{2 \cdot \sigma^2} + \ln \frac{\sum_{(s',s) \in C_k^{+1}} A_{k-1}(s') \left[\exp\left(-\frac{\sum_{i=2}^n (y_k^{(i)} - x_k^{(i)})^2}{2 \cdot \sigma^2}\right) \right] B_k(s)}{\sum_{(s',s) \in C_k^{-1}} A_{k-1}(s') \left[\exp\left(-\frac{\sum_{i=2}^n (y_k^{(i)} - x_k^{(i)})^2}{2 \cdot \sigma^2}\right) \right] B_k(s)} \\
&= L_a(u_k) + L_C \cdot y_k^{(1)} + L_{ex}(u_k)
\end{aligned} \tag{2.18}$$

The term $L_{ex}(u_k)$ is called extrinsic information. Due to the extrinsic information is a redundant information that introduces by the RSC encoder, it is independent on systematic input and a-priori value $L_a(u_k)$ from LLR. The term $L_{ex}(u_k)$ is passed to the input of the next decoder as the a-priori value $L_a(u_k)$ after (de-)interleaving. The overall MAP decoding flow is illustrated in Figure 2.7.

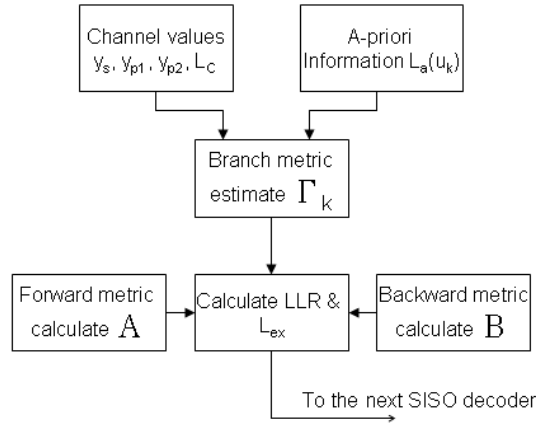


Figure 2.7 MAP decoding flow chart

2.2.2 The Log-MAP Algorithm

Although BCJR (MAP) algorithm will be fine for BER performance, that is very difficult and wasteful to implementation in hardware point of view. Therefore, the following algorithm will be from the hardware point of view to talk about the questions and solutions from papers.

The log-MAP algorithm is a transformation of MAP algorithm and without any performance loss in practical implementation. It operates in logarithm domain, and multiplication is converted to addition. Before introducing this algorithm, the Jacobian function is defined as:

$$\begin{aligned}
 \ln(e^{x_1} + e^{x_2}) &\triangleq \max^*(x_1, x_2) \\
 &= \max(x_1, x_2) + \ln(1 + e^{-|x_1 - x_2|}) \\
 &= \max(x_1, x_2) + lut(x_1, x_2) \\
 &= \max(x_1, x_2) + f_c(|x_1 - x_2|)
 \end{aligned} \tag{2.19}$$

Where $f_c(\cdot)$ is a correction term, it can be implemented using a simple look-up table (LUT). The Jacobian function can be further expressed as:

$$\begin{aligned}
 \ln(e^{x_1} + e^{x_2} + \dots + e^{x_N}) &\triangleq \max^*(x_1, x_2, \dots, x_N) \\
 &= \max^*(\dots \max^*(\max^*(x_1, x_2), x_3), \dots, x_N)
 \end{aligned} \tag{2.20}$$

Now, we can re-express the forward recursion metric $\alpha_k(s)$ from (2.8):

$$\begin{aligned}
\alpha_k(s) &= \ln A_k(s) \\
&= \ln \sum_{\text{all } S_{k-1}} \Gamma_k(s', s) \cdot A_{k-1}(s') \\
&= \ln \sum_{\text{all } S_{k-1}} e^{\gamma_k(s', s)} \cdot e^{\alpha_{k-1}(s')} \\
&= \ln \sum_{\text{all } S_{k-1}} e^{\gamma_k(s', s) + \alpha_{k-1}(s')} \\
&= \max_{s'}^* (\gamma_k(s', s) + \alpha_{k-1}(s'))
\end{aligned} \tag{2.21}$$

Where the branch metric can be expressed as:

$$\begin{aligned}
\gamma_k(s', s) &\triangleq \ln \Gamma_k(s', s) \\
&= \ln \left(e^{u_k \cdot L_a(u_k)/2} \cdot e^{\sum_{i=1}^n L_c \gamma_k^{(i)} \cdot x_k^{(i)}/2} \right) \\
&= \frac{1}{2} \cdot \left(u_k \cdot L_a(u_k) + \sum_{i=1}^n L_c \cdot \gamma_k^{(i)} \cdot x_k^{(i)} \right)
\end{aligned} \tag{2.22}$$

We can also derive the backward recursion metric $\beta_k(s)$ in logarithm domain as:

$$\begin{aligned}
\beta_k(s) &= \ln B_k(s) \\
&= \ln \sum_{\text{all } S_{k+1}} \Gamma_{k+1}(s, s'') \cdot B_{k+1}(s'') \\
&= \ln \sum_{\text{all } S_{k+1}} e^{\gamma_{k+1}(s, s'')} \cdot e^{\beta_{k+1}(s'')} \\
&= \ln \sum_{\text{all } S_{k+1}} e^{\gamma_{k+1}(s, s'') + \beta_{k+1}(s'')} \\
&= \max_{s''}^* (\gamma_{k+1}(s, s'') + \beta_{k+1}(s''))
\end{aligned} \tag{2.23}$$

Finally, from (2.17) can be expressed as:

$$\begin{aligned}
L(u_k) &= \ln \frac{\sum_{(s', s) \in C_k^{+1}} A_{k-1}(s') \Gamma_k(s', s) B_k(s)}{\sum_{(s', s) \in C_k^{-1}} A_{k-1}(s') \Gamma_k(s', s) B_k(s)} \\
&= \max_{(s', s) \in C_k^{+1}}^* [\alpha_{k-1}(s') + \gamma_k(s', s) + \beta_k(s)] \\
&\quad - \max_{(s', s) \in C_k^{-1}}^* [\alpha_{k-1}(s') + \gamma_k(s', s) + \beta_k(s)]
\end{aligned} \tag{2.24}$$

2.2.3 The Maximum Log (ML) MAP Algorithm

In hardware point of view, in spite of the log-MAP algorithm had reduced the

hardware cost, it is still too complex for some embedded applications. Hence, the ML-MAP algorithm is proposed with less complicated arithmetic, while a little performance loss compare with log-MAP algorithm. According to (2.21), (2.23), (2.24), we express the forward state metric, the backward state metrics and the LLR value for ML-MAP algorithm as (2.25), (2.26), and (2.27):

$$\begin{aligned}\alpha_k(s) &= \max_{s'}^* (\gamma_k(s', s) + \alpha_{k-1}(s')) \\ &\cong \max_{s'} (\gamma_k(s', s) + \alpha_{k-1}(s'))\end{aligned}\quad (2.25)$$

$$\begin{aligned}\beta_k(s) &= \max_{s''}^* (\gamma_{k+1}(s, s'') + \beta_{k+1}(s'')) \\ &= \max_{s''} (\gamma_{k+1}(s, s'') + \beta_{k+1}(s''))\end{aligned}\quad (2.26)$$

$$\begin{aligned}L(u_k) &= \max_{(s', s) \in C_k^{*+}} [\alpha_{k-1}(s') + \gamma_k(s', s) + \beta_k(s)] \\ &\quad - \max_{(s', s) \in C_k^{-}} [\alpha_{k-1}(s') + \gamma_k(s', s) + \beta_k(s)] \\ &\cong \max_{(s', s) \in C_k^{*+}} [\alpha_{k-1}(s') + \gamma_k(s', s) + \beta_k(s)] \\ &\quad - \max_{(s', s) \in C_k^{-}} [\alpha_{k-1}(s') + \gamma_k(s', s) + \beta_k(s)]\end{aligned}\quad (2.27)$$

Compare with log-MAP algorithm, the difference between the two algorithms is the correct function $\ln(1 + e^{-|x_1 - x_2|})$ that can be implemented with a look-up-table (LUT). The maximum output value of the correct function is about 0.7 when $x_1 = x_2$, and the output value of the correct function can be omitted when the absolute value greater than 2. The correct function is illustrated in Figure 2.8.

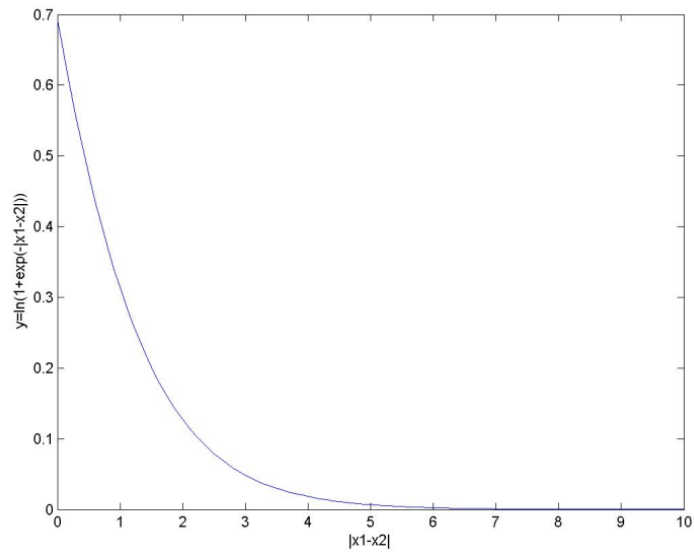


Figure 2.8 The correct function

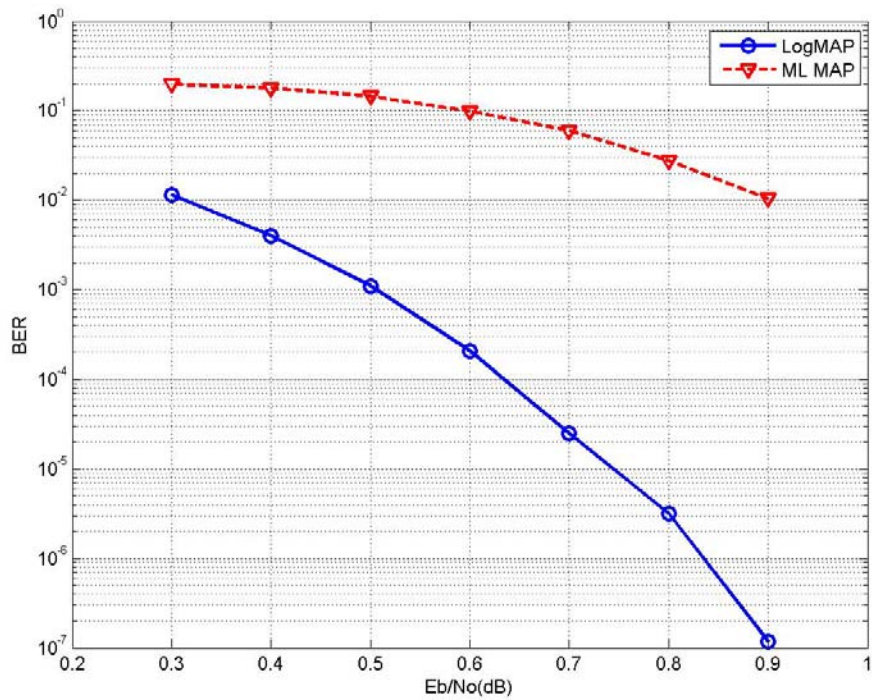


Figure 2.9 The BER performance of ML-MAP algorithm compare with Log-MAP algorithm (N=1784, code rate = 1/3, state = 16, MAP algorithm, BPSK)

2.3 Sliding Window Method for Turbo Decoding

In the traditional turbo decoding algorithm included MAP algorithm, log-MAP algorithm and Max-log-MAP algorithm. No matter what algorithm is used, the decision is based on forward and backward recursion metrics. We have to store every branch metric (γ) and forward state metric α (or backward state metric β) at every stage unit the backward state metric β (or forward state metric α) has been calculated out as shown in Figure 2.10, so as to calculate LLR in (2.24).

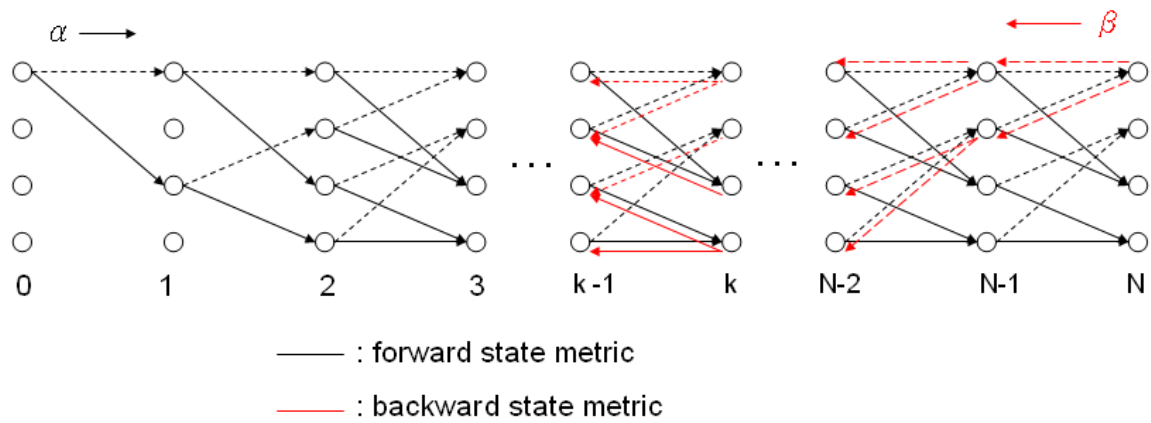


Figure 2.10 The turbo decoding trellis diagram including the forward and backward direction

Now in hardware point of view the drawback of the log-MAP algorithm (or MAP algorithm) are the excessive memory required and a long decoding latency. As describe in the above, the entire state metric history must be stored, out to the end of trellis, at which point the backward state metric begins and decisions can be output starting with the last branch without the need to store any but the last set of state metrics computed backward state metric. This storage requirement is apparently excessive. Taking CCSDS specification as an example, according to encoder structure, we have 16 states

in trellis diagram, if we express every state by 9 bits, it would need $9 \times 16 = 144$ bits of storage per stage, and if the frame size is 1784 bits, the turbo decoder must at least have 144×1784 bits to store for traditional MAP decoding algorithm.

Due to a lot of memory requirement and decoding latency for traditional MAP decoding, sliding window [8] method was proposed by Viterbi in 1998. We now briefly describe this method which reduces the memory requirement to just a few thousand bits, independent of the frame size N . Figure 2.11 indicates the bit processing times for one forward processor α and two backward processors β operating in synchronism with received branch symbols. L means the sliding window size (typically 5~10 times constraint length). The label for each “node” below means the branch time instance. The main thought for sliding window is that we would estimate the set of backward state metrics via applying learning period (L). The basis for this approach is the fact that the backward processor can start cold in any state at any time; initially, the backward state metrics produced are almost worthless, but a few constraint lengths, the set of state metrics are as reliable as if the process had been started at the initial (or final) node. This applies equally to backward state metric as well as forward state metric. In Figure 2.11, dashed line means that the un-reliable backward metric calculation (learning period). This backward processor is so-called dummy- β processor. After learning period computation, we get a reliable value for backward state metrics to take the initial value.

Now we take the first decoder output (LLR) as an example to explain how those processors work. Let the received branch symbols be delayed by $2L$ trellis times. Then, from time $2L$ to $3L$, we calculate all forward state metric start from the initial node 0 to L and storing these in memory, at the same time ($2L \sim 3L$), the first backward processor starts to learning the initial backward state by the received symbol from node $2L$ to L (note that the direction is reverse with forward processor). During time $2L$ to $3L$ (or

learning period), we do not store any backward metrics until time goes to $3L$. At this time ($3L$), due to forward processor had already computed forward state metric from node 0 to L , so we can get the valid decoder output (node L to 0) from forward and backward state metric at time $3L$ to $4L$

Also, the procedure of the second backward processor will be same as the first backward processor. While the first backward processor decode output from node L to 0 at time $3L$ to $4L$. From time $3L$ to $4L$, we calculate all forward state metric start from the node L to $2L$ and storing these in memory, at the same time ($3L\sim 4L$), the second backward processor starts to learning the initial backward state by the received symbol from node $3L$ to $2L$. After learning period, we get the valid decoder output (node $2L$ to L) at time $4L$ to $5L$. The two backward processors will take turn to decode out as the timing shows in Figure 2.11. We now also take CCSDS turbo code as an example, the forward algorithm only needs to store $2L$ sets of forward state metrics, since after its first $2L$ computations (performed by time $4L$), its first set of metrics will be discarded, then the empty memory can be filled starting with the new state metric for the node $2L+1$ to $4L$. Thus, the storage requirement for a 16 trellis states using 9-bits to express forward state metric is only $2L*16*9=288L$ bits in all. If we assume $L = 32$, the storage requirement is approximately equal to 9K bits. That is the way for sliding window method saving huge memory and latency. After above mention, we simulated five different sizes of siding window as shown in Figure 2.12, and the sliding window size 32 is more suitable in order best performance.

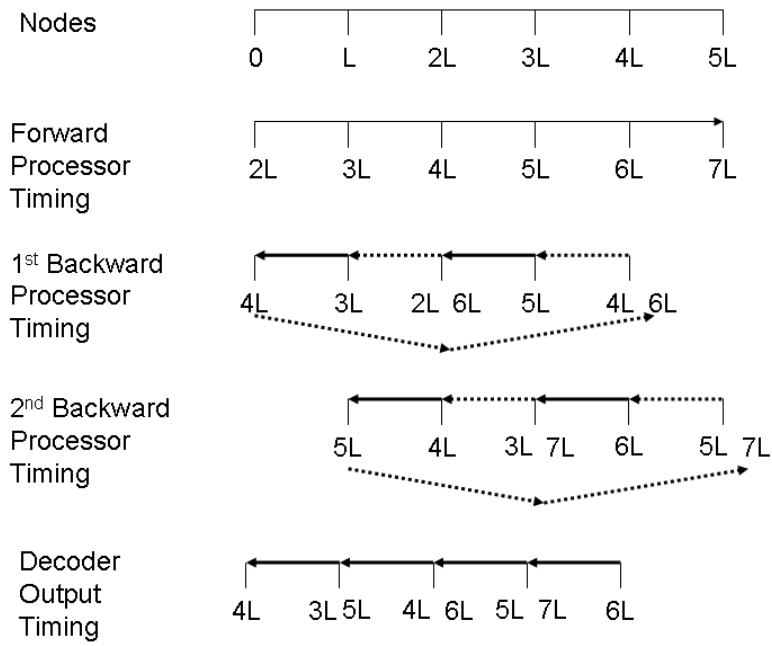


Figure 2.11 Timing diagram for sliding window (refer to [8])

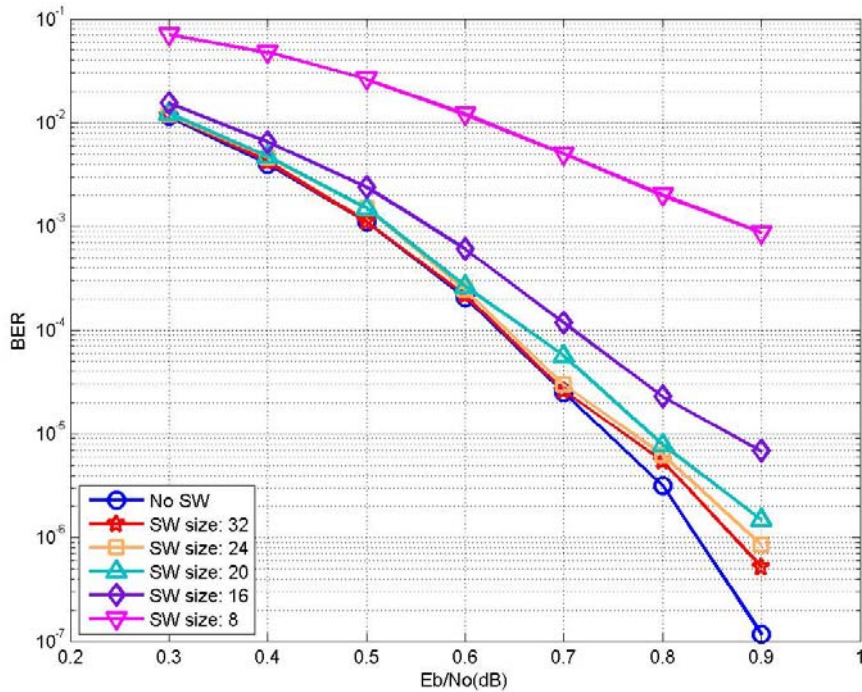
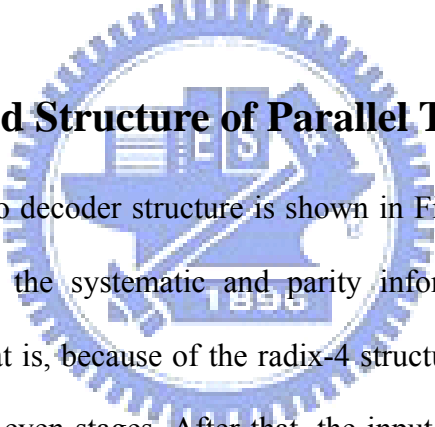


Figure 2.12 Performance comparison under different sizes of sliding window in CCSDS interleaver ($N=1784$, code rate = 1/3, state = 16, MAP algorithm, BPSK)

Chapter 3 Turbo Decoder Design Consideration

To analyze why the throughput of traditional turbo decoder is not fast enough, the most important reasons are limited by the operating frequency, a certain number of iterations, latency of sliding window and number of SISO decoders. In this Chapter, our proposed structure of turbo decoder is briefly presented to improve the disadvantage in the section 4.1, and then the methods to improve the throughput of turbo decoder are introduced, i.e. parallelism and early stopping criteria. On the other hand, due to the parallel decoding process, we have to solve the collision problem without any buffer.

3.1 The Proposed Structure of Parallel Turbo Decoder



The proposed turbo decoder structure is shown in Figure 3.1. At the beginning, the input sequences (i.e. the systematic and parity information) are stored in the “IN_BUF” memories. That is, because of the radix-4 structure, the input sequences are divided from the odd and even stages. After that, the input sequences are accessed for the ‘SISO Decoder’ block decoding. As the log-likelihood ratios (LLR) and the extrinsic sequences (Lex) are produced, the Lex sequences are re-permuted to be the a-priori information (La) for the next iteration. Based on the high-radix or parallelism architecture, the mapping interleaving rule [21] with contention free is employed. Furthermore, in order to increase the data rate of turbo decoder, the HDA2 technique had been introduced and the hardware overhead should be negligible.

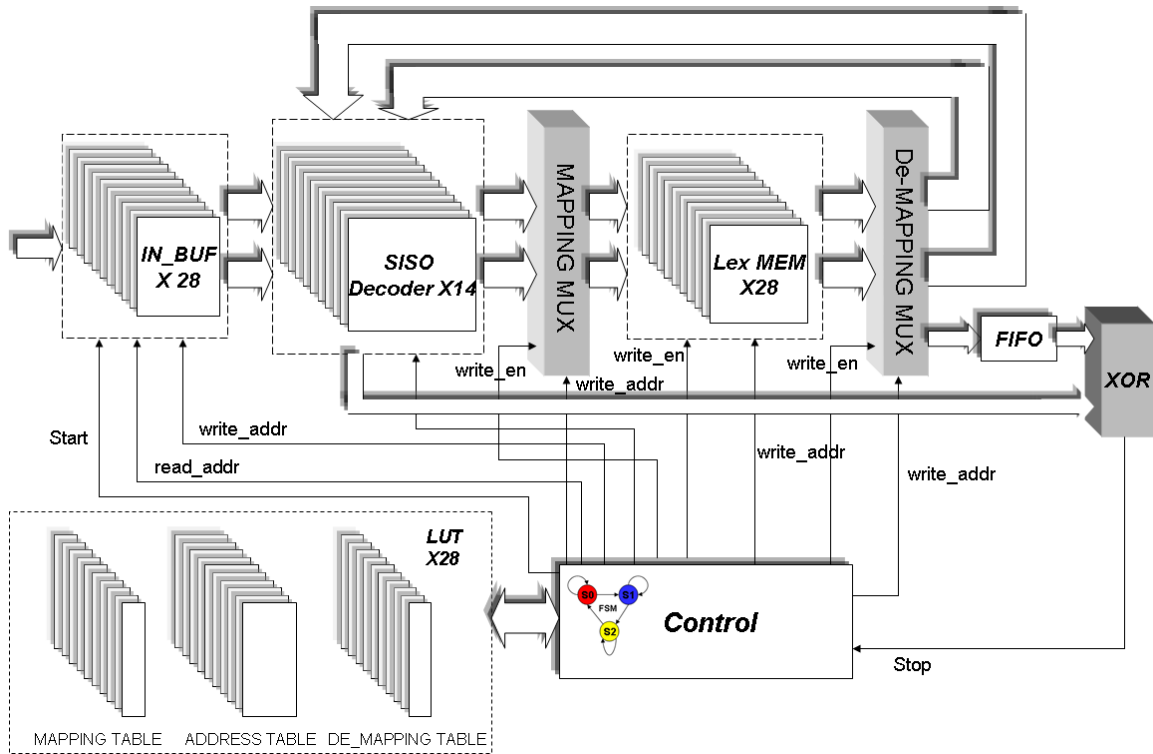


Figure 3.1 The proposed turbo decoder structure

3.2 The Parallel Turbo Decoder

3.2.1 Sliding Window Timing Diagram

In a traditional sliding window, four dual port memory banks are needed. Besides, the latency of the traditional sliding window is at least $4L$, where L is the window size. In our architecture, the function for each memory-bank is illustrated in Figure 3.2 (a). The black-slash block represents a store of the soft-input symbols to the memory bank, and the dotted block represents a read of the soft-input symbols to compute the forward state metrics α ; the slash block represents first a read of the previous soft-input symbols to compute the backward state metrics β followed by an immediate store of the next input received symbols. A detailed dynamic description is also illustrated in Figure 3.2 (b). The gray solid arrow represents calculation of the backward state metrics β . Furthermore, the dummy- β is calculated directly from the input symbols without the use

of any memory-banks. Based on the above reason, two memory banks are enough for our sliding window. Once we have the forward and backward state metrics, the soft output calculator is employed to decode the LLR out. Therefore, the latency of our proposed SW method is only about $2L$.

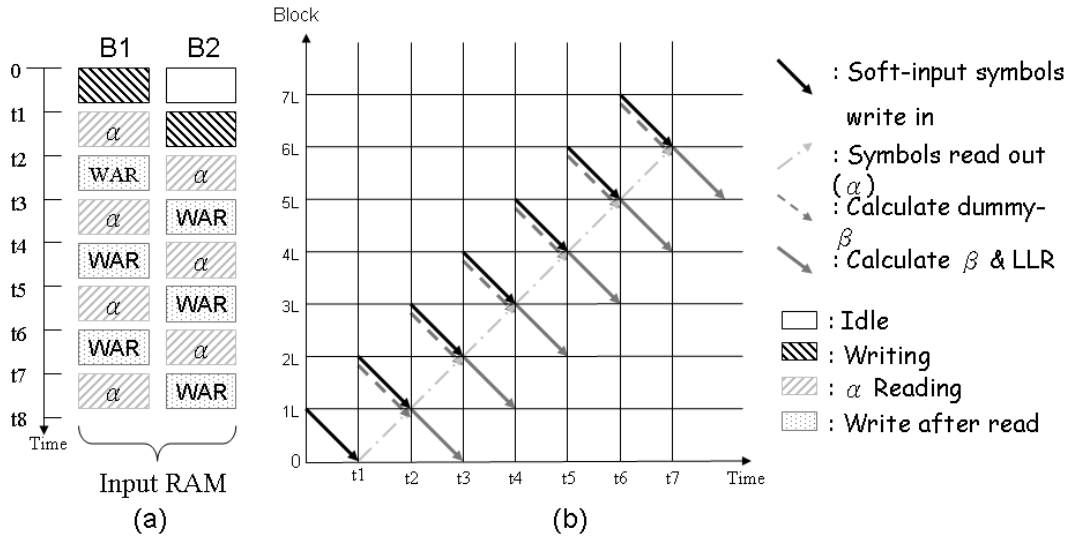


Figure 3.2 (a) Space and time relationship for memory-bank management (b) Space and time relationship for memory-bank management

3.2.2 Parallel Sliding Window Decoding

Due to the parallel SISO decoding process, the flow diagram of sliding window could be a little altered as shown in Figure 3.3 that called ‘Parallel Sliding Window’. In Figure 3.3 giving ‘N’ SISO decoders, the PSW method divides the block K in N ‘super windows’, and performs sliding window decoding in each self SISO decoder. The ‘super windows’ and the regular windows inside them can both be initialized by the method that intruding in section 3.2.1. Furthermore, in order to obtain reliable values of the forward state metrics, we use the forward state metrics of the previous iteration to ensure the initial value as reliable as traditional log-MAP algorithm. Hence, additional

registers would be needed, and the number of registers are N . The size of each register is: $(\text{number of states}) \cdot (\text{bits for forward metrics representations})$

Here, we would like to note several properties of the suggested PSW technique:

- Owing to the memories and the processing hardware cost, the area grows linearly with the number of parallel SISO decoders N , and the decoding latency drops linearly with N , making this PSW method very suitable for a parallel architecture.
- For the same decoding latency as parallel log-MAP decoding and almost the same amount of processing hardware much less intermediate memory is used.
- The decoding performance can be very closely estimated using the results obtained for sliding window decoding.

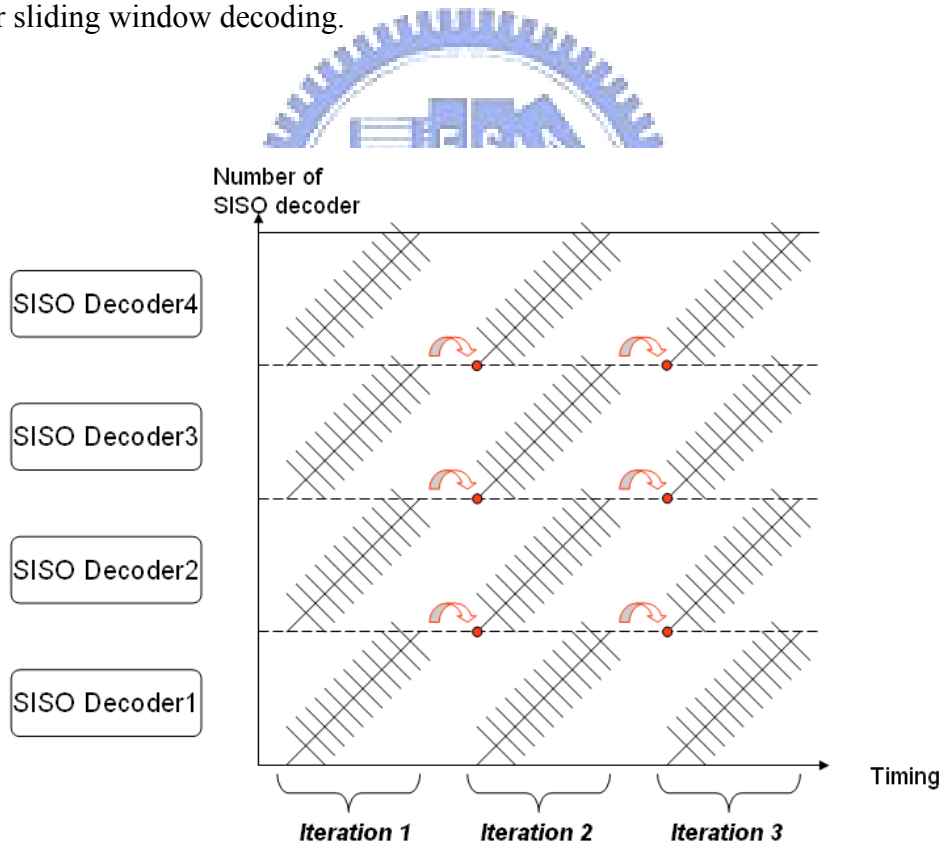


Figure 3.3 Schedule of the parallel sliding window technique

3.2.3 The Interleaver of Parallel Turbo Decoders

In this section we discuss the collision problem [20] in the parallel turbo decoding

process. The memory conflict problem is that the different SISO decoders work in parallel, it is necessary to access the extrinsic information by each SISO decoders in different RAM memories. In fact, depending on the specific permutation rule, it may happen that different SISO decoders try to access the same memory bank at the same time instant. We describe the problem in Figure 3.4, for a conventional turbo decoder in Figure 3.3 (a), it would not happen the collision problem as the only one SISO decoder stores or reads extrinsic information; while taking an example as 4-parallel SISO decoders in Figure 3.3(b), we permute the four extrinsic information in order and write the four extrinsic information according to interleaving order, then we find that SISO2 and SISO4 decoders simultaneously access the same memory bank. In the next cycle, we also find that SISO1 and SISO3 decoders access simultaneously the same memory bank.

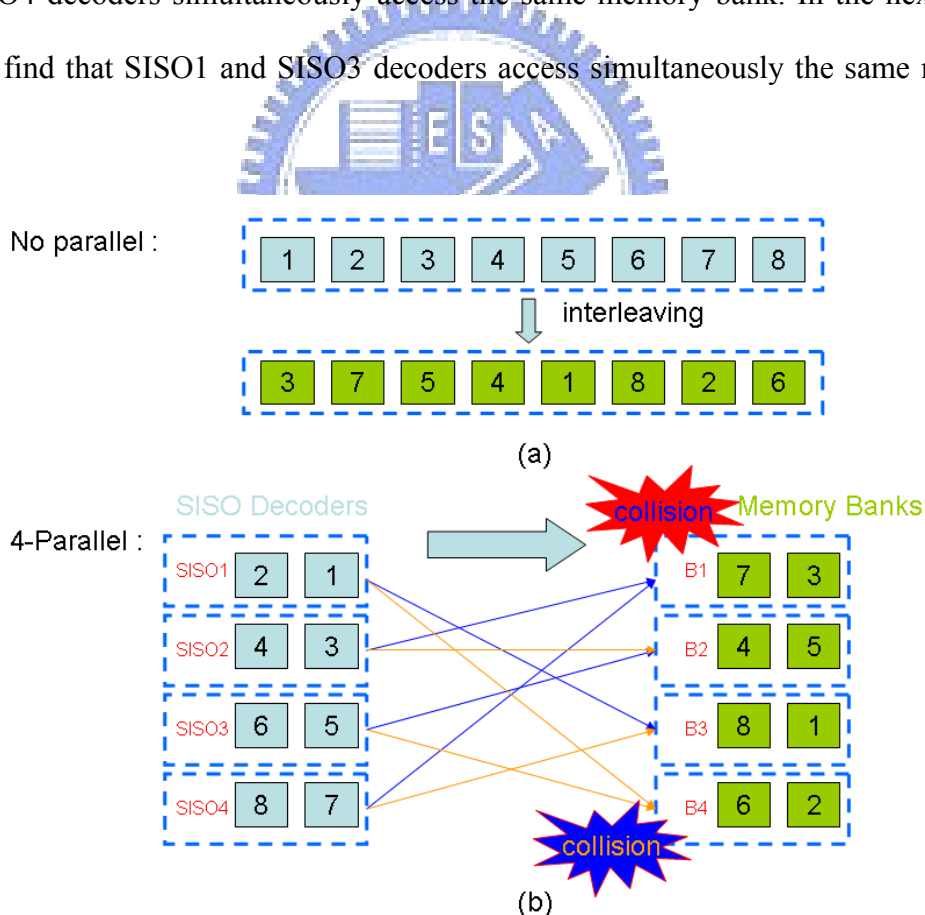


Figure 3.4 (a) conventional turbo decoding without collision (b) parallel turbo decoding with collision problem

To solve the problem, [21] had proposed a feasible method that can be used for any interleaver rules. We explain this method as follows. Given N_w banks of memory, each SISO decoder works on a sub-block with length $w = L/N_w$. If we number all extrinsic information from 1 to L, the j -th SISO exports those values from $(j-1)w+1$ to jw . We assume that all SISO decoders (i.e. $SISO_1$ to $SISO_N_w$) export their extrinsic information at time instant i are those in position $i, w+i, \dots, (N_w-1)w+i$, and those relative position after write in (interleaving) or read out (de-interleaving) the memory banks are $\pi(i), \pi(w+i), \dots, \pi((N_w-1)w+i)$.

To formalize the problem of mapping from decoders to memory banks, we can define a pair of functions $(M, S): \{1, \dots, L\} \rightarrow \{1, \dots, N_w\} \times \{1, \dots, w\}$, with the following meaning: For each decoder, the i -th output is written in the memory bank indexed by $M(i)$, in position $S(i)$. The condition of lack of collisions translates then into the following constraints on $M, \forall k, k' = 1, \dots, L, k \neq k'$, where \equiv_w means 'equal modulo w ':

$$k \equiv_w k' \rightarrow M(k) \neq M(k') \quad (3.1)$$

$$k \equiv_w k' \rightarrow M(\pi(k)) \neq M(\pi(k')) \quad (3.2)$$

Notice that the above constraints only depend on π , and that no constraint is imposed on the shift function S .

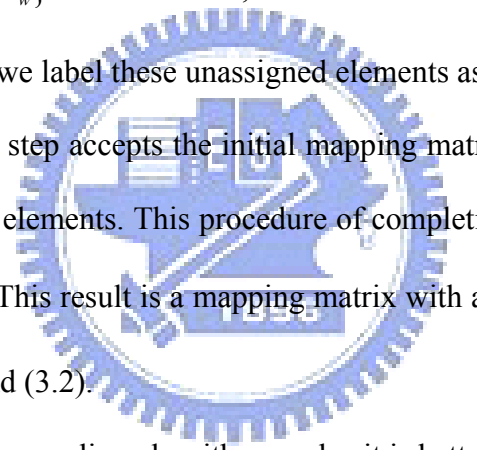
It is useful to represent the mapping function as a $N_w \times w$ rectangular matrix, the mapping matrix, whose (i, j) -th element, $i = 1, \dots, N_w, j = 1, \dots, w$, represents the value of $M((i-1)w + j)$. In this way, constraint (3.1) translates into a constraint on the columns of such a matrix, while constraint (3.2) that depends on permutation π , defines a tiling of the matrix. Now, let us given an index k and defined the following

two sets:

$$C(k) = \{k' : k' = [(i-1)w + k] \bmod w, \quad i = 1, \dots, N_w\} \quad (3.3)$$

$$T(k) = \{\pi(k') : k' = [(i-1)w + k] \bmod w, \quad i = 1, \dots, N_w\} \quad (3.4)$$

Given an interleaver π , the problem is to find a mapping matrix that satisfies (3.1) and (3.2). Here, we present an algorithm that gives the desired mapping matrix for any interleaver π . The algorithm can be described as below:

- First step: Any step that produces an initial mapping matrix with the following properties: every column and tile contains at most one element equal to every symbol in $\{1, \dots, N_w\}$. Nevertheless, there are some elements which are not assigned yet, and we label these unassigned elements as '-'. 
- Second step: This step accepts the initial mapping matrix output in the first step and fills all blank elements. This procedure of completing the mapping matrix is called annealing. This result is a mapping matrix with all elements in $\{1, \dots, N_w\}$, satisfying (3.1) and (3.2).

To understand how the annealing algorithm works, it is better to give an example.

Example 1: Suppose $L = 30$, $N_w = 5$, $w = 6$, and suppose the permutation π , for

instance:

$$\begin{aligned} \pi = & (29, 17, 5, 11, 21, 24, \\ & 7, 2, 30, 28, 15, 10, \\ & 22, 16, 1, 12, 3, 27, \\ & 19, 14, 9, 25, 20, 4, \\ & 13, 26, 18, 6, 8, 23) \end{aligned}$$

Thus, the column and tiling of the mapping matrix in this example can be shown in Figure 3.5(a), (b), respectively.

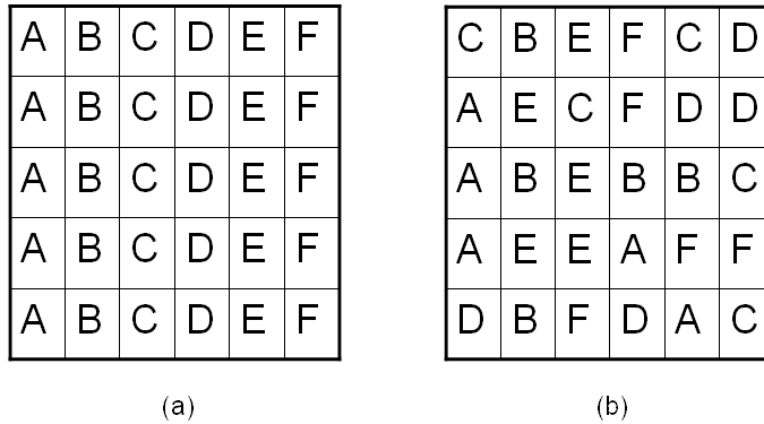


Figure 3.5 (a) The column (b) The tiling of the mapping matrix in this example

Where the two sets are according to (3.3) and (3.4), for example, the indices 28 and 9 of the tiling matrix are:

$$T(28) = \{ \pi(k') : k' = [(2-1)6 + 28] \bmod 6 = 4 \rightarrow D, \quad i = 1, \dots, 5 \}$$

$$T(9) = \{ \pi(k') : k' = [(4-1)6 + 9] \bmod 6 = 3 \rightarrow C, \quad i = 1, \dots, 5 \}$$

Suppose the output of the first step is the following initial mapping matrix:

$$\begin{pmatrix} 4 & 1 & - & 2 & 2 & 2 \\ 1 & 4 & 1 & 3 & 3 & 4 \\ 2 & 2 & 2 & 5 & 4 & 3 \\ 3 & 5 & 3 & 4 & 1 & - \\ 5 & 3 & 4 & 1 & 5 & 5 \end{pmatrix}$$

Where the blanks in its (1,3) and (4,6) elements. The procedure of annealing starts from one of them, and we choose the (1,3) element and fill in the blank with the value that is not represented in its column yet, i.e., the value is 5. However, this change will cause a collision to happen, because (1,3) and (4,2) belong to the same tile E and both have the value 5. Owing to this reason, (4,2) is changed to the value 1 that is not represented in its tile E yet, as:

$$\begin{pmatrix} 4 & 1 & - & 2 & 2 & 2 \\ 1 & 4 & 1 & 3 & 3 & 4 \\ 2 & 2 & 2 & 5 & 4 & 3 \\ 3 & 5 & 3 & 4 & 1 & - \\ 5 & 3 & 4 & 1 & 5 & 5 \end{pmatrix} \rightarrow \begin{pmatrix} 4 & 1 & 5 & 2 & 2 & 2 \\ 1 & 4 & 1 & 3 & 3 & 4 \\ 2 & 2 & 2 & 5 & 4 & 3 \\ 3 & 1 & 3 & 4 & 1 & - \\ 5 & 3 & 4 & 1 & 5 & 5 \end{pmatrix}$$

Now, there is a collision happened in column 2 (the value 2 appears two times), so (1,2) is changed to 5. However, this change will cause a collision due to (1,2) and (3,4) belong to the same tile B and both have the value 5. So (3,4) is changed to the value 1 that is not represented in its tile E yet, as:

$$\begin{pmatrix} 4 & 1 & 5 & 2 & 2 & 2 \\ 1 & 4 & 1 & 3 & 3 & 4 \\ 2 & 2 & 2 & 5 & 4 & 3 \\ 3 & 1 & 3 & 4 & 1 & - \\ 5 & 3 & 4 & 1 & 5 & 5 \end{pmatrix} \rightarrow \begin{pmatrix} 4 & 5 & 5 & 2 & 2 & 2 \\ 1 & 4 & 1 & 3 & 3 & 4 \\ 2 & 2 & 2 & 1 & 4 & 3 \\ 3 & 1 & 3 & 4 & 1 & - \\ 5 & 3 & 4 & 1 & 5 & 5 \end{pmatrix}$$

Repeat the same procedure above mentioned all the while until no iterant number appears to the same tile and column. Hence, the final result is the following mapping matrix and one can verify that constraint (4.1) and (4.2) are all satisfied:

$$\begin{pmatrix} 4 & 5 & 5 & 2 & 2 & 2 \\ 5 & 4 & 1 & 3 & 3 & 4 \\ 2 & 2 & 2 & 1 & 4 & 3 \\ 3 & 1 & 3 & 4 & 5 & 1 \\ 1 & 3 & 4 & 5 & 1 & 5 \end{pmatrix}$$

The annealing procedure can be decomposed into several cycles, each of them starting with a blank element, picked at random, and ending when no collisions are produced. After a cycle is ended, a new one starts if there are still blanks in the mapping matrix, otherwise, the annealing procedure is over. In the previous example, there are 4 cycles.

3.3 Early Stopping Criteria

In traditional turbo decoding, in order to achieve a satisfactory performance, the turbo decoding has to run a certain number of iterations to ensure the extrinsic values have converged. This results in low speed, long decoding latency and large energy consumption as well. In fact, turbo decoder may early converge when the channel condition is good. Hence, an early stopping criterion should be employed to reduce the number of iterations. For the hardware point of view, a good stopping criterion should save as many iterations as possible with no or insignificant performance loss. At the same time, the hardware overhead should be negligible.

Here, we briefly introduce some early stopping criteria [23], then compare their advantages and disadvantages:

- ◆ **HDA (Hard-Decision Aided) Criterion:** This criterion compares the decoded bits of the two continuous iterations. The turbo decoding is stopped working after iteration i , where $i \geq 2$:

$$\mathbb{S}(L^{i,2}(u_k)) = \mathbb{S}(L^{i-1,2}(u_k)), \forall k \in 1, \dots, K \quad (3.5)$$

Where $L^{i,j}(u_k)$ denotes the log-likelihood ratios (LLR) output from the j^{th} decoder in i^{th} iteration, and the K and $\mathbb{S}(x)$ denote the frame size and the sign bit of x .

- ◆ **HDA2 Criterion:** The method of HAD criterion is extended in [24] and we only represent a criterion that due to only this criterion has similar hardware implementation complexity, while the others require double or triple hardware implementation complexity. Therefore, the decoding process is stopped after iteration i for $i \geq 2$, if:

$$\mathbb{S}(L^{i,1}(u_k)) = \mathbb{S}(L^{i,2}(u_k)), \quad \forall k \in 1, \dots, K \quad (3.6)$$

- ◆ **SDR2 (Sign Different Ratio) Criterion:** This criterion was proposed in [25], according to (2.18), since the term $L_C \cdot y_k^S$ is fixed for all iterations, the change in the magnitudes of the LLR is owing to changes in the magnitudes of the extrinsic information. The hard decision based on $L_C \cdot y_k^S + L_a^{i,2}(u_k)$ from the SISO1 decoder shows in Figure 2.4 should agree with the hard decision based on the LLR at the output of the SISO2 decoder, where the term $L_a^{i,2}(u_k)$ is the term $L_{ex}^{i,1}(u_k)$ from the SISO1 decoder after interleaving. Hence, the decoding process is terminated after iteration i for $i \geq 1$, if:

$$\sum_{k=1}^K (\mathbb{S}(L^{i,2}(u_k)) \oplus \mathbb{S}(L_C \cdot y_k^S + L_a^{i,2}(u_k))) = 0 \quad (3.7)$$

- ◆ **Min-LLR Criterion:** the criterion had proposed a method that use the minimum of absolute values of the LLR to decide the turbo decoding is terminating or not. This decoding process is stopped after iteration i for $i \geq 1$, if:

$$\min_{1 \leq k \leq K} |L^{i,2}(u_k)| > \theta \quad (3.8)$$

- ◆ **Decoding Metrics Criterion:** This criterion is decided by three variables: the minimum of the absolute values of the LLR, the minimum of the absolute values of the extrinsic information, and the number of the non-matching bits (NMb). The idea of NMb evaluates the number of sign-bit differences between the LLR and the extrinsic information for the same SISO decoder of the same iteration. The turbo decoding process is stopped after iteration i for $i \geq 1$, if:

$$\left(\min_{1 \leq k \leq K} |L^{i,j}(u_k)| > \theta_1 \right) \& \left(\min_{1 \leq k \leq K} |L_{ex}^{i,j}(u_k)| > \theta_2 \right) \& \left(\sum_{k=1}^K (S(L^{i,j}(u_k)) \oplus S(L_{ex}^{i,j}(u_k))) < \theta_3 \right) \quad (3.9)$$

Where & denotes the ‘AND’ operation.

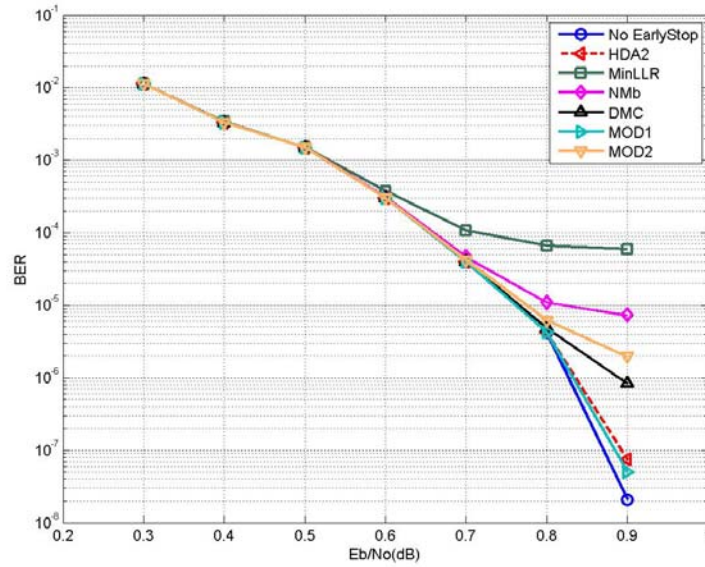


Figure 3.6 The BER performance comparisons of early stopping criteria

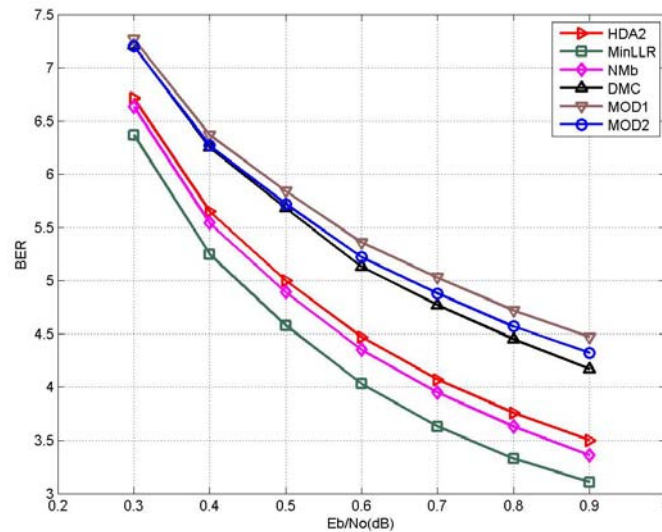


Figure 3.7 The average number of iteration of six early stopping criteria

After MATLAB simulation, the simulation results are presented in Figure 3.6 and Figure 3.7. In Figure 3.6, the HDA2 and MOD1 criteria are the best performance among all curves; and in Figure 3.7, the MinLLR criterion is the least number of iteration among all curves, however, the NMB and HDA2 criteria are also good enough even though the MinLLR criterion has the least iterative number. Finally, we select the HDA2 criterion after a comprehensive survey.



Chapter 4 Soft-In-Soft-Out (SISO) Decoder Design

Consideration

Due to the maximum a posterior probability (MAP) algorithm, turbo codes are one of the most powerful error correcting codes. However, its clock frequency is limited by recursion architecture of SISO decoder. In this chapter, section 4.1 shows the proposed SISO decoder structure. Then we introduce radix-4 log-MAP algorithm and the proposed ACS architecture of radix-4 log-MAP decoder in order to improve the throughput. Finally, we also introduce the architectures of branch metric and LLR.

4.1 SISO Decoder Architecture

The block diagram of the radix-4 MAP decoder is shown in Figure 4.1. During the SISO decoding process, the soft-input symbols are written to the four single-port memories, which work like ping-pong buffers and are read by the ACS α or β block to calculate the branch and state metrics. The state metrics computed from the ACS α block, are stored in “Alpha RAM”, and are later fetched by the LLR unit for LLR calculation when the ACS β block state metrics become available. In order to decrease the latency and memory, the dummy β ACS block fetches the soft-input symbols directly.

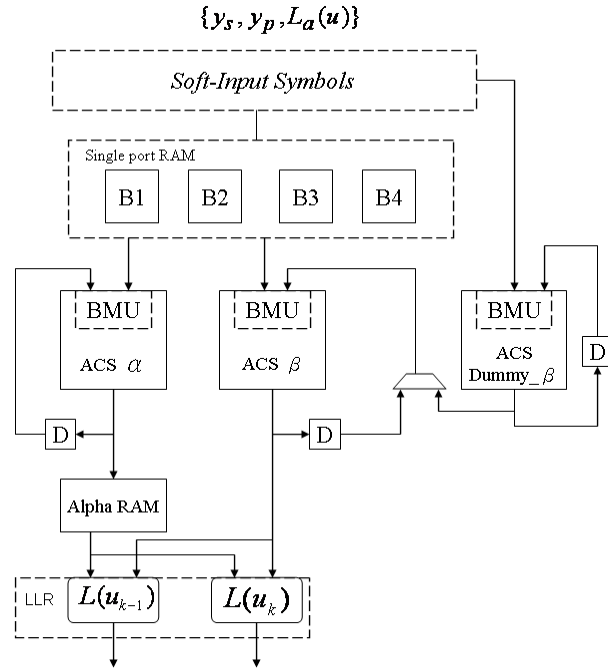


Figure 4.1 Block diagram for sliding window log-MAP decoder (SISO decoder)

4.2 Radix-4 Log-MAP Algorithm

The throughput of turbo decoder is limited by the critical path delay of ACS unit due to the recursion architecture. For the radix-4 decoder, if we directly implement the radix-4 algorithm, the critical path delay will be twice that of the radix-2 recursion unit, therefore, it cannot increase the throughput. Many articles [9-11] on recursion architectures have been presented to solve this problem. In this section we briefly introduce the radix-4 algorithm and next section the radix-4 recursion architecture will be presented.

Radix-4 architecture processes two stages per clock cycle as shown in Figure 4.2 (b), i.e. the decoder computes two bits per clock cycle; whereas radix-2 architecture processes only one trellis stage per clock cycle and its trellis diagram as shown in Figure 4.2 (a). The radix-4 trellis contains only the stage at the even times ($k=0, k=2, \dots$). Each node has four incoming paths (i.e. four candidates to select) and four

outgoing paths. Due to the radix-2 MAP algorithm was introduced in chapter 2, we express the recursion of the state metrics as followed:

- The forward recursion metric $\alpha_k(s)$:

$$\begin{aligned} \alpha_k(s) &= \max_{s'}^* (\gamma_k(s', s) + \alpha_{k-1}(s')) \\ &= \max_{s'}^* \{ \gamma_k(s', s) + \max_{s''}^* [\gamma_{k-1}(s'', s') + \alpha_{k-2}(s'')] \} \\ &= \max_{(s', s'')}^* [\gamma_k(s', s) + \gamma_{k-1}(s'', s') + \alpha_{k-2}(s'')] \end{aligned} \quad (4.1)$$

- The backward recursion metric $\beta_k(s)$:

$$\begin{aligned} \beta_{k-2}(s'') &= \max_{s'}^* (\gamma_{k-1}(s'', s') + \beta_{k-1}(s')) \\ &= \max_{s'}^* \{ \gamma_{k-1}(s'', s') + \max_s^* [\gamma_k(s', s) + \beta_k(s)] \} \\ &= \max_{(s, s')}^* [\gamma_{k-1}(s'', s') + \gamma_k(s', s) + \beta_k(s)] \end{aligned} \quad (4.2)$$

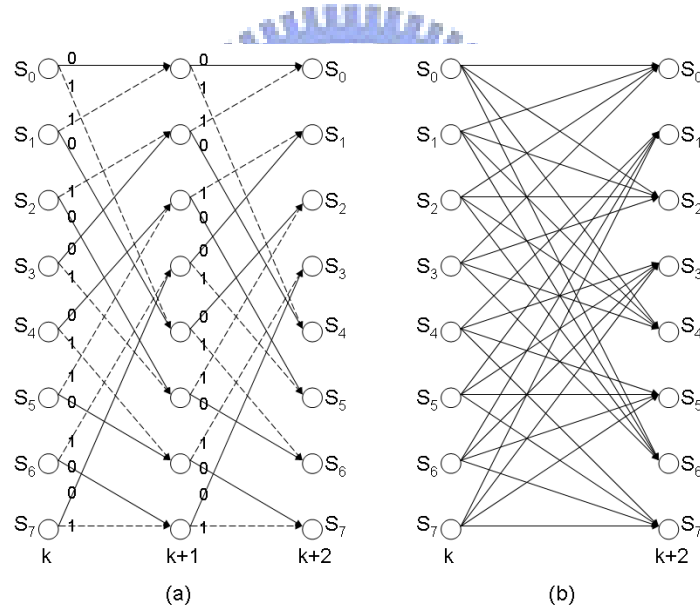


Figure 4.2 Trellis diagram (a) Radix-2 trellis (b) Radix-4 trellis

Finally, the log-likelihood ratios (LLR) can be written as:

$$\begin{aligned}
L(u_k) &= \max_{(s',s) \in C_k^{-1}}^* [\alpha_{k-1}(s') + \gamma_k(s',s) + \beta_k(s)] \\
&\quad - \max_{(s',s) \in C_k^{-1}}^* [\alpha_{k-1}(s') + \gamma_k(s',s) + \beta_k(s)] \\
&= \max_{(s',s) \in C_k^{-1}}^* \left\{ \max_{s''}^* [\gamma_{k-1}(s'',s') + \alpha_{k-2}(s'')] + \gamma_k(s',s) + \beta_k(s) \right\} \\
&\quad - \max_{(s',s) \in C_k^{-1}}^* \left\{ \max_{s''}^* [\gamma_{k-1}(s'',s') + \alpha_{k-2}(s'')] + \gamma_k(s',s) + \beta_k(s) \right\} \\
&= \max_{(s'',s',s) \in C_k^{+1}}^* \{ \alpha_{k-2}(s'') + \gamma_{k-1}(s'',s') + \gamma_k(s',s) + \beta_k(s) \} \\
&\quad - \max_{(s'',s',s) \in C_k^{-1}}^* \{ \alpha_{k-2}(s'') + \gamma_{k-1}(s'',s') + \gamma_k(s',s) + \beta_k(s) \}
\end{aligned} \tag{4.3}$$

$$\begin{aligned}
L(u_{k-1}) &= \max_{(s'',s') \in C_{k-1}^{+1}}^* [\alpha_{k-2}(s'') + \gamma_{k-1}(s'',s') + \beta_{k-1}(s')] \\
&\quad - \max_{(s'',s') \in C_{k-1}^{-1}}^* [\alpha_{k-2}(s'') + \gamma_{k-1}(s'',s') + \beta_{k-1}(s')] \\
&= \max_{(s'',s') \in C_{k-1}^{+1}}^* \left\{ \alpha_{k-2}(s'') + \gamma_{k-1}(s'',s') + \max_s^* [\gamma_k(s',s) + \beta_k(s)] \right\} \\
&\quad - \max_{(s'',s') \in C_{k-1}^{-1}}^* \left\{ \alpha_{k-2}(s'') + \gamma_{k-1}(s'',s') + \max_s^* [\gamma_k(s',s) + \beta_k(s)] \right\} \\
&= \max_{(s'',s',s) \in C_{k-1}^{+1}}^* \{ \alpha_{k-2}(s'') + \gamma_{k-1}(s'',s') + \gamma_k(s',s) + \beta_k(s) \} \\
&\quad - \max_{(s'',s',s) \in C_{k-1}^{-1}}^* \{ \alpha_{k-2}(s'') + \gamma_{k-1}(s'',s') + \gamma_k(s',s) + \beta_k(s) \}
\end{aligned} \tag{4.4}$$

Moreover, if we want to improve our throughput more, we can use higher radix (e.g. radix-8, radix-16) log-MAP algorithm, but it may increase the area significantly.

4.3 The Architecture of Recursion State Metric

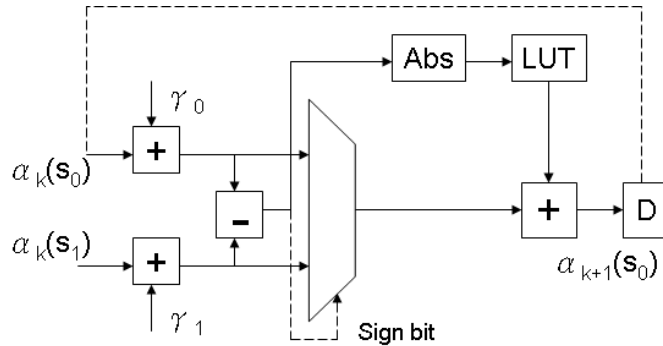


Figure 4.3 A traditional recursion architecture (with normalization)

4.3.1 OASC Structure

For a radix-2 MAP decoder, the traditional recursion architecture and equation are shown in Figure 4.3 and (2.21), respectively, and the recursion architecture is called the add-compare-select-offset (ACSO) unit.

To analyze the recursion architecture, we make the recursion architecture expand to two trellis stages as shown in Figure 3.5 [12]. Pipelining those three different positions of the recursion loop registers. The first zone is type (a) architecture. It results in an ACSO unit. The second zone is type (b) architecture. It leads to a compare-select-offset-add (CSOA) unit. The third zone is type (c) architecture. It leads to an offset-add- compare-select (OACS) unit. We briefly compare the critical path of those three type architectures. In the case of type (a), the critical path is consisted of the propagation carry adder (t_C), the propagation of the one full adder (t_{FA}) for comparison, the time of the LUT block access (t_{LUT}), the multiplier (t_{MUX}), and the time of the propagation carry adder (t_C) again due to adding the LUT value. The total critical is describe as (3.5). In the case of type (c), that is mean the OACS architecture, the critical path is consisted of the propagation carry adder (t_C) only in the first adder. Due to the propagation of carry adder, only one full adder (t_{FA}) for the addition of the branch metric in the critical path as well as the propagation of the one full adder (t_{FA}) for comparison, the time of the LUT block access (t_{LUT}), the multiplier (t_{MUX}). Then the total critical is decrease from (3.5) to (3.6):

$$t_{ACSO} = n_{SM} \cdot t_C + t_{FA} + MAX(t_{LUT}, t_{MUX}) + n_{SM} \cdot t_C \quad (4.5)$$

$$t_{OACS} = n_{SM} \cdot t_C + 2 \cdot t_{FA} + MAX(t_{LUT}, t_{MUX}) \quad (4.6)$$

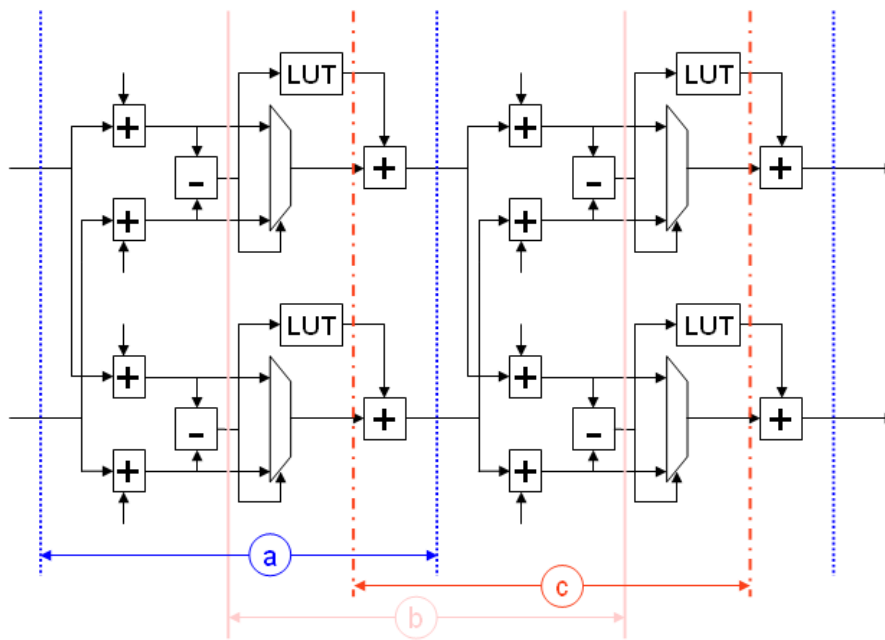


Figure 4.4 Three different locations of the register in the data flow of the recursive algorithm result in three kinds of ACSO recursion architecture (refer to [12])

Because the critical path of CSOA architecture is the same as OACS architecture, we compare OACS unit with CSOA unit in area point of view, OACS unit needs n_{SM} bits and n_{LUT} bits registers, whereas OACS unit needs $3 \cdot n_{SM}$ bits registers. As a result of area, we use OACS-based concept and radix-2 OACS architecture as shown in Figure 4.5.

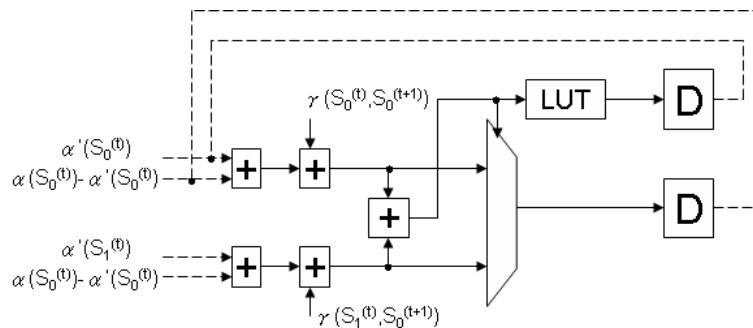


Figure 4.5 Architecture of a recursion OACS unit

4.3.2 Proposed Radix-4 Log-MAP Recursion State Metric

Although the radix-4 architecture reduces the total number of stages by 50% compare with the radix-2 architecture, it is expected the complexity and the branch bits increase in radix-4 architecture design. Therefore, the overall critical path of radix-4 decoder will a little larger than radix-2 decoder. Hence, our design challenge of the radix-4 decoder is to design an ACS recursion unit which its critical path is less than twice of the radix-2 ACS recursion unit.

According to (4.1), radix-4 recursion unit has four candidates to select. We mention the \max^* function again in (4.7), and directly implement in this equation in Figure 4.6. The gray area expresses two-input \max^* ACS unit (i.e. $\max^*(x_1, x_2) = \max(x_1, x_2) + \ln(1 + e^{-|x_1 - x_2|})$). It is clear that the critical path delay in Figure 4.6 is double that of radix-2 ACSO unit. To improve this problem, [9-10] proposed many ideas, but a little performance loss.

$$\begin{aligned} \ln(e^{x_1} + e^{x_2} + e^{x_3} + e^{x_4}) &\triangleq \max^*(x_1, x_2, x_3, x_4) \\ &= \max^*(\max^*(x_1, x_2), \max^*(x_3, x_4)) \end{aligned} \quad (4.7)$$

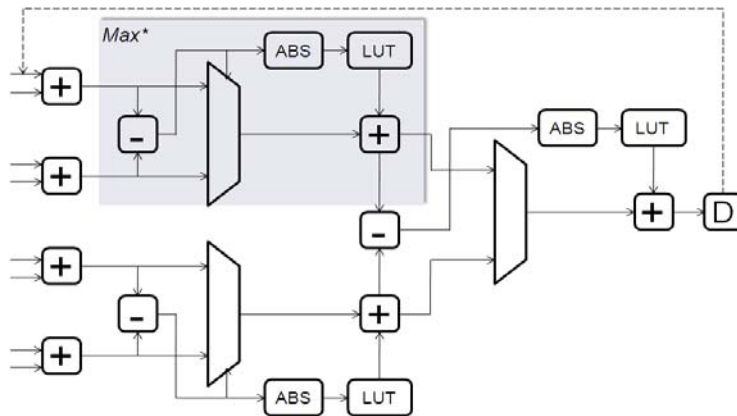


Figure 4.6 A radix-4 recursion unit

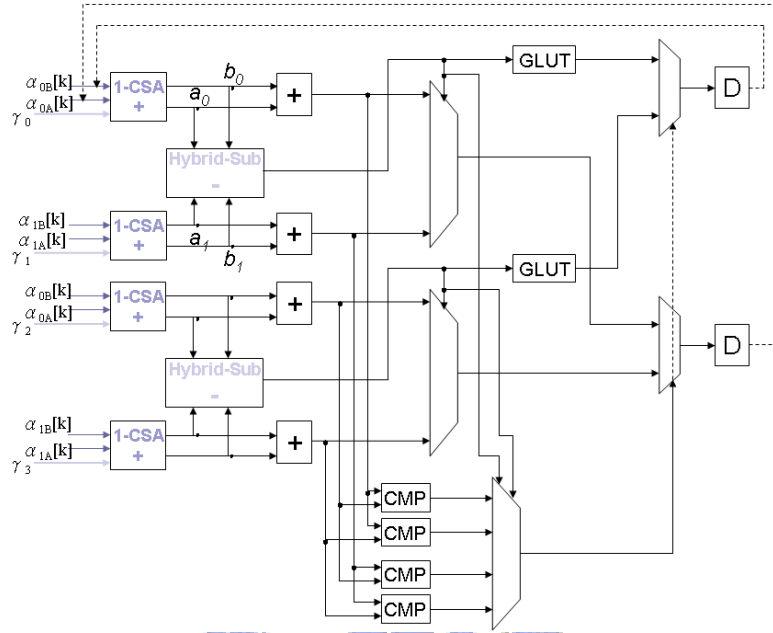


Figure 4.7 Improved radix-4 recursion OACS architecture

In our proposed design, in order to further increase the throughput, we use the OACS [4] architecture as shown in Figure 4.7 and the computation for the max* function can be expressed as:

$$\max^*(w, x, y, z) \approx \max \{ \max^*(w, x), \max^*(w, x) \} \quad (3.8)$$

The outputs of four comparators and the MSB of the difference output of each of two subtractors are fed to an array of multiplexers to select the maximum value of the four inputs, and its associated LUT index. In addition, we employ a one stage carry-save adder described in [10] to reduce a three-number addition to a two-number addition. Moreover, in order to further increase the clock rate, the hybrid 4-inputs addition/subtraction (e.g. $a_0 + b_0 - a_1 - b_1$) is proposed and the structure like signed

binary digit (SBD) addition/subtraction [29]. It is clear that the term $a_0 + b_0 - a_1$ is computed by the plus-plus-minus (PPM) adder of the first stage, and then the sum 's' and carry 'c' are produced from the PPM adder of the first stage and b_1 as the inputs of the PPM adder of the second stage as shown in Figure 4.8. Hence, the difference of each of two inputs could be early derived. Finally, the critical path delay of our design is less than three times the delay of a 10-bit adder.

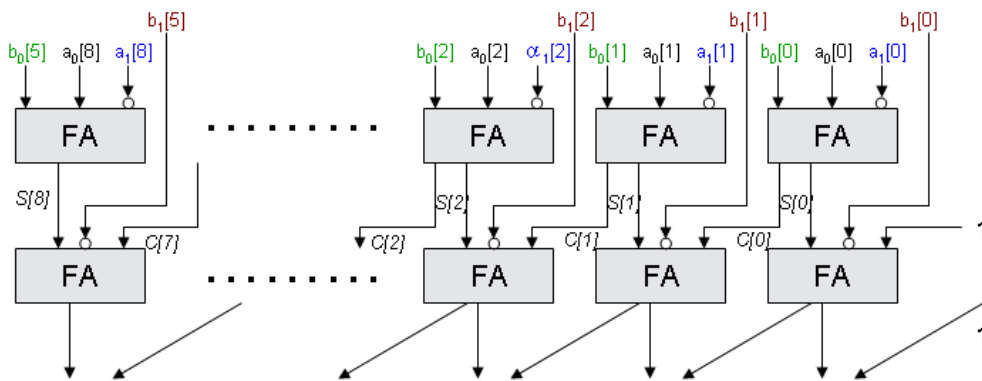


Figure 4.8 Hybrid 4-input subtraction

In addition, the generalized LUT (GLUT) structure is illustrated in Figure 4.9, the advantage of the GLUT structure is it does not need to compute the absolute value from subtraction operation, while estimate the correct term by Ls2 and ELUT block. The Ls2 function block is used to determine if the absolute value of the input is less than two or not, and the ELUT is used a smaller LUT only with 3-bit inputs and obtain 2-bit outputs. The output Z of Ls2 function can be express as $Z = \overline{S}(b_7 + b_6 + \dots + b_3) + S(b_7 \cdot b_6 \cdot \dots \cdot b_3)$. Besides, the inputs of ELUT block include the sign bit to make sure the output value correction, and Table 3.1 shows the LUT approximation value of Figure 4.9. Finally, the outputs (c0, c1) of ELUT block are combined with the output Z of Ls2 function block

by AND gate. That is, if the absolute value of the GLUT input is greater than 2.0, the output from the GLUT is zero. Otherwise, the GLUT output is decided by ELUT block.

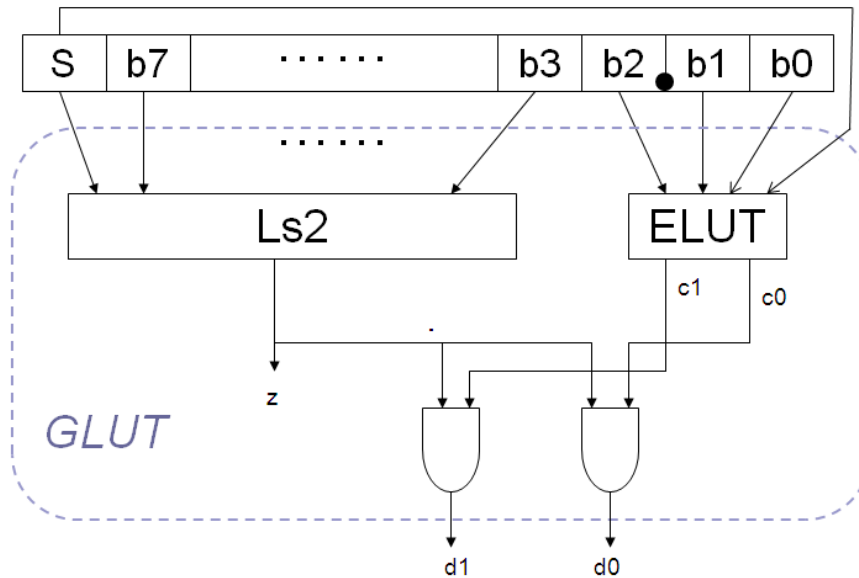


Figure 4.9 Structure of GLUT used in improved OACS architecture

Table 4.1 ELUT function block approximation

(b_2, b_1, b_0)	$ x $	$f(x)$
000	0.00	0.75
001	0.25	0.50
010	0.50	0.50
011	0.75	0.50
100	1.00	0.25
101	1.25	0.25
110	1.50	0.25
111	1.75	0.25

4.3.3 The State Metric Normalization

A significant issue for hardware implementation of turbo decoder, fixed-point implementation is necessary. Due to the finite numerical range representation, the forward and backward state metrics would overflow by using log-MAP recursion algorithm. This problem can be solved by a normalization method [13] or by using modulo arithmetic [14], [15]. In this section, we only address the rescaling method.

According to the proof of [14], [16], the bit-width w has to be large to allow straightforward evaluation of differences Δ :

$$\lceil ld\Delta_{\max} \rceil + 1 \leq w \quad (4.8)$$

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

$$|\gamma_k(s, s')| \leq B, \quad s, s' \in S \quad (4.9)$$

With $m = K - 1$ being the memory order of a RSC code with constraint length K , the difference between any two state metrics of the same trellis stage k is bounded as:

$$|\alpha_k(s_0) - \alpha_k(s_1)| \leq 2mB, \quad s_0, s_1 \in S \quad (4.10)$$

Based on (3.10), the require bit-width w_{sm} for the state metrics after a recursion is written as:

$$w_{sm} = \lceil ld(2mB) \rceil + 1 \quad (4.11)$$

Again, we can derive the candidate state metrics are upper bounded as:

$$|\alpha_k(s_0) + \gamma_{k+1}(s_0, s') - [\alpha_k(s_1) + \gamma_{k+1}(s_1, s'')]| \leq 2mB + 2B = 2(m+1)B \quad (4.12)$$

The require bit-width w_{csm} for the candidate state metrics is written as:

$$w_{csm} = \lceil ld(2(m+1)B) \rceil + 1 \quad (4.13)$$

As analyze above, we know the forward and backward state metrics will be bounded in a range after a few trellis stages computation. Therefore, the proposed

approach is the rescaling of the state metric via condition subtraction of a fixed value. We assume the upper bound of branch is 32 as shown in Figure 4.10, and the constraint length $K=5$ for CCSDS, the upper bound ($2(m+1)B = 320$) of the forward and backward state metrics will be evaluated by (3.13), and the require bit-width $w_{esm} = 9$ bits. Hence, if one of the state metrics is larger than 480, all the state metrics will be subtracted of 128 to guarantee all the state metrics would not overflow. By the way, when one or more state metrics over 480, the minimum value of the state metric (not less than 160) will not less than the maximum value of branch metric due to the upper bound of state metric. This ensures all state metrics are positive values.

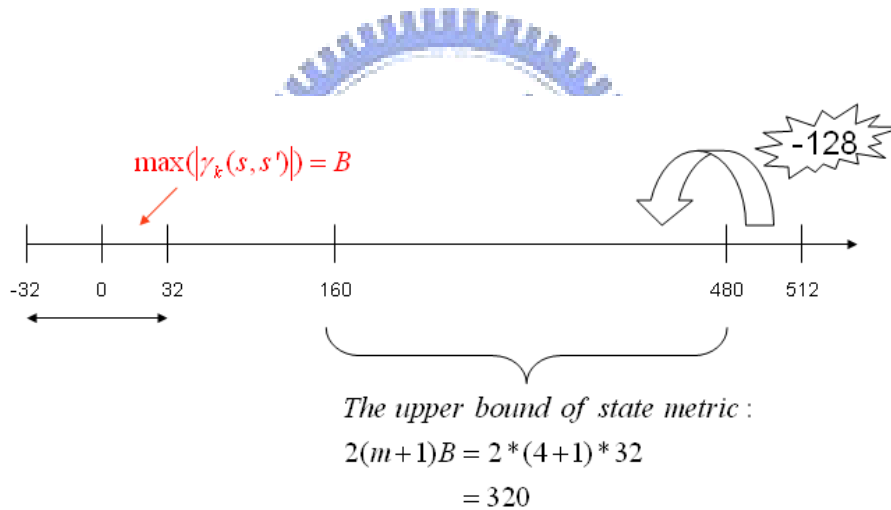


Figure 4.10 Integer ranges at forward and backward recursion arch.

This rescaling approach [13] only leads to a little critical path delay with the recursion unit. The structure within normalization is illustrated in Figure 4.11, take radix-4 ACS unit as an example, and the blue area is the normalization part, this structure detects the four candidates larger than 960 or not, if more than one of the four candidates larger than 960, the first OR gate send a “true 1” signal to the next OR gate, else send a “false 0” signal to the next OR gate. There are 16 input signals (the number

of input signals is according to the number of state metrics) in the second OR gate, if one of them send a “true 1” signal, then all state metrics will be subtracted by 256, otherwise doing nothing for original state metrics.

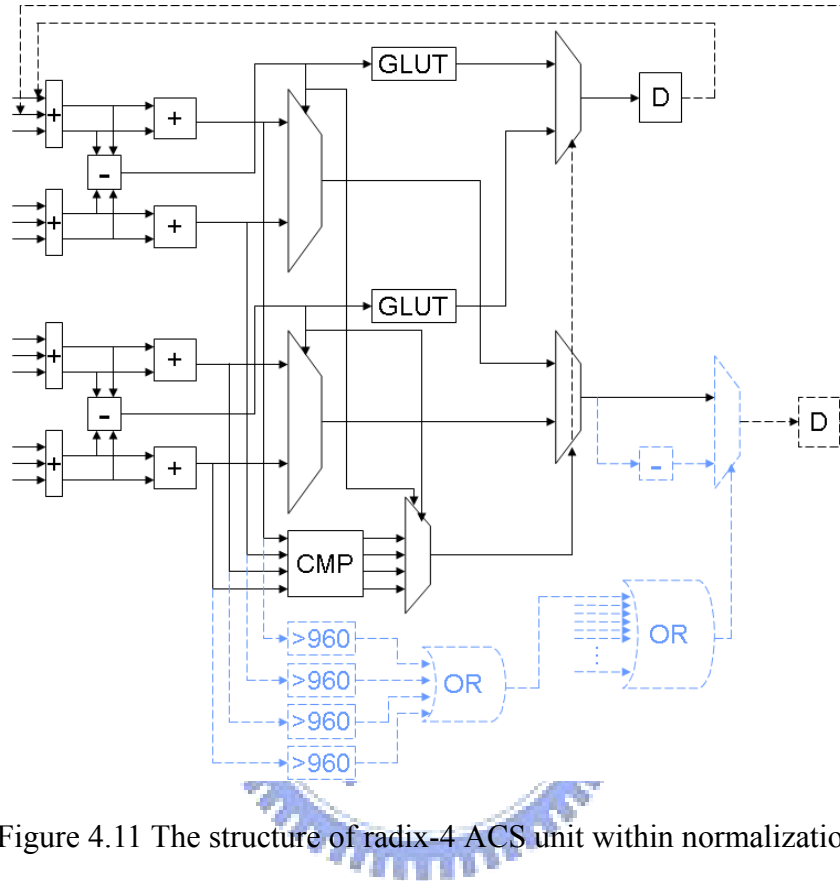


Figure 4.11 The structure of radix-4 ACS unit within normalization

4.4 The Structure of Branch Metric

According to the equation (2.22), the equation of the branch metric for code rate 1/3 and radix-4 log-MAP algorithm is derived as:

$$\begin{aligned}
\delta_k(s'', s', s) &\triangleq \gamma_k(s', s) + \gamma_{k-1}(s'', s') \\
&= \frac{1}{2} \cdot \left[u_k \cdot L_a(u_k) + \sum_{i=1}^n L_c \cdot y_k^{(i)} \cdot x_k^{(i)} \right] \\
&\quad + \frac{1}{2} \cdot \left[u_{k-1} \cdot L_a(u_{k-1}) + \sum_{i=1}^n L_c \cdot y_{k-1}^{(i)} \cdot x_{k-1}^{(i)} \right] \\
&= \frac{1}{2} \left[u_k \cdot L_a(u_k) + L_c \cdot (y_k^s \cdot x_k^s + y_k^p \cdot x_k^p) \right] \\
&\quad + \frac{1}{2} \left[u_{k-1} \cdot L_a(u_{k-1}) + L_c \cdot (y_{k-1}^s \cdot x_{k-1}^s + y_{k-1}^p \cdot x_{k-1}^p) \right] \\
&= \frac{1}{2} \left[u_k \cdot L_a(u_k) + u_{k-1} \cdot L_a(u_{k-1}) \right] \\
&\quad + \frac{1}{2} \left[L_c \cdot (y_k^s \cdot x_k^s + y_k^p \cdot x_k^p + y_{k-1}^s \cdot x_{k-1}^s + y_{k-1}^p \cdot x_{k-1}^p) \right]
\end{aligned} \tag{4.14}$$

The branch metric unit (BMU) for radix-4 log-MAP algorithm is shown in Figure 4.12. The MSB and LSB of the delta (δ) indices are at the time k-1 and k, respectively.

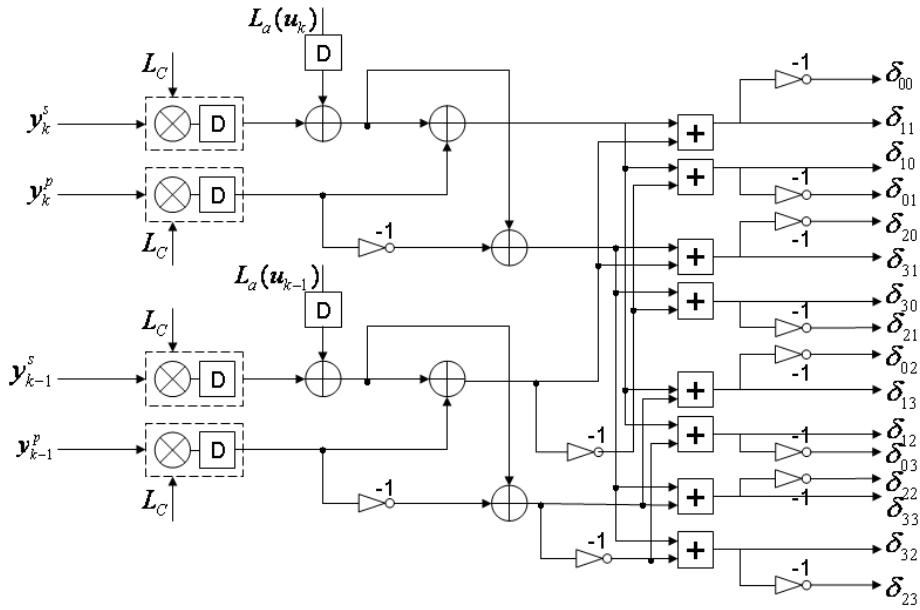


Figure 4.12 The branch metric unit (BMU) for radix-4 log-MAP algorithm

4.5 The Structure of Log-Likelihood Ratios (LLR)

4.5.1 Traditional LLR Computation Unit (LCU) Based on Radix-2 Log-MAP Algorithm

Based on (2.24), we have to compute the sum of the forward, backward state metrics and branch metrics. In order to decrease the critical path of LLR, we pipeline the outputs of the addition. We assume 4 trellis states RSC encoder as shown in Figure 2.6, in conventional architecture, a total number of 16 adders are used in the first pipelined stage to obtain the LLR value as shown in Figure 4.13, and need $2 * [M - 1]$ MAX* unit to compare and select the LLR_0 and LLR_1 values, where M is the number of trellis states. The number of pipelined stages is $(\log_2 M) + 2$.

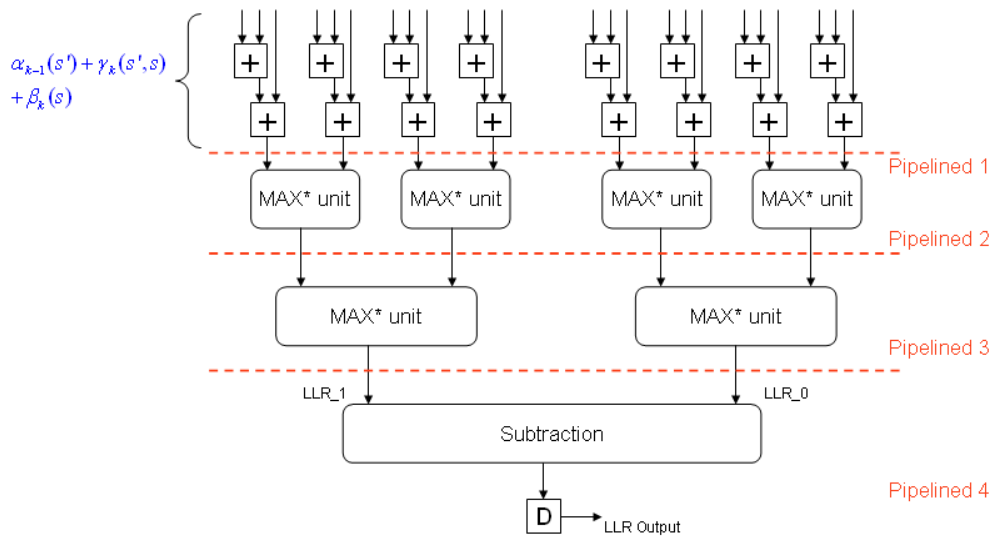


Figure 4.13 Traditional LLR Computation Unit (LCU) Based on Radix-2 Log-MAP Algorithm

4.5.2 LLR Computation Unit (LCU) Based on Radix-4 Log-MAP Algorithm

In this section, we also assume 4 trellis states RSC encoder as shown in Figure

4.13. For radix-2 log-MAP algorithm, there are 4 path candidates to compute the log-likelihood value (either LLR₁ or LLR₀) as shown in Figure 4.14 (a). However, for radix-4 log-MAP algorithm, according to (3.3), (3.4), there are both 8 path candidates to compute the log-likelihood value at time k and k-1 (also either LLR₁ or LLR₀), and the architecture of the LLR will be illustrated in Figure 4.15.

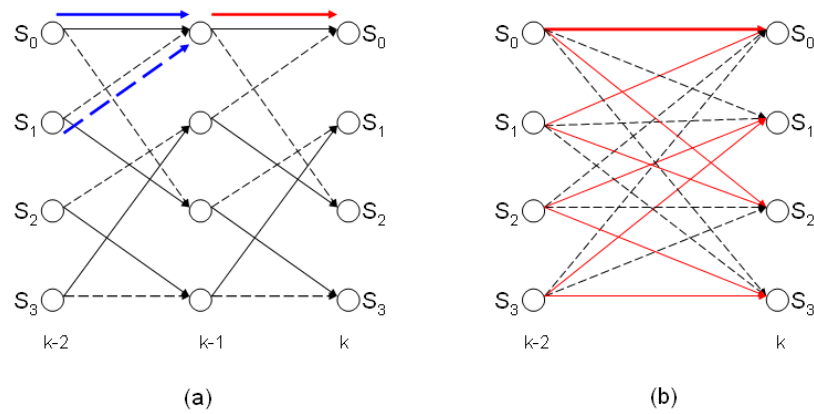


Figure 4.14 Trellis diagram (a) Radix-2 trellis (b) Radix-4 trellis

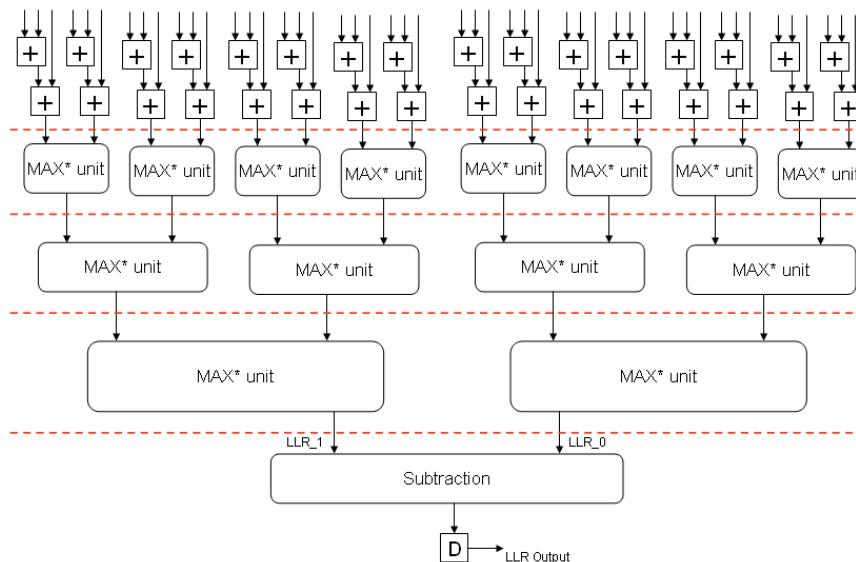


Figure 4.15 LLR Computation Unit (LCU) Based on Radix-4 Log-MAP

Algorithm

Chapter 5 System Simulation and Performance Analysis

In previous chapter, our discussions about turbo codes are based on a condition of the floating point. However, the floating point value should be bounded since infinite precision is impossible to be achieved for practical hardware implementation. A trade-off between hardware cost and the performance must be concerned since coding performance may suffer quantization loss due to internal bit-width limitation. In general, the hardware complexity of turbo code can be estimation in computing complexity and memory size which is proportional to bit-width. In this chapter, based on acceptable performance loss, the fixed point analysis and computing complexity is discussed. Besides, we also analyze the parameter of scaling factor under MATLAB tools. Note that in this chapter, only hardware complexities of the CCSDS standard with length-1784 interleaver. And the sliding window method for turbo decoder is assumed, where the length of sliding window is set as 32.

Due to we have briefly described turbo encoder and decoder structure in chapter 2 ~ 4, and the encoder is earlier than the decoder, this chapter will focus on the decoding simulation and performance analysis.

5.1 The Bit-Width Estimation of Soft-Input Information

Most Turbo decoder hardware implementations are based on fixed-point operations [26]. As a result, a significant amount of effort must be focused on dynamic range, number density, and normalization before choosing a number system. Since our aim is a fast turbo decoder design, we choose a 2's compliment integer representation.

For efficient implementation, we need to estimate the numerical range of the soft

inputs, various state metrics. In this section, we focused on the estimation of the soft inputs, while other state metrics are unconcerned. We simulate four different types of input and three various numbers of iterations under MATLAB for BER comparison in Figure 5.1. Figure 5.1 shows the BER performance of a code rate 1/3, 16 states, and frame size of 1784 bits on CCSDS standard. The MATLAB simulations were operating under the assumption of AWGN channel and BPSK modulation, and where (q, f) denotes a quantization scheme that uses q bits in total and f bits to represent the fractional part. We finally chose the fixed-point (5, 2) as hardware input though the performance of the fixed-point (6, 3) is a little better than the fixed-point (5, 2).

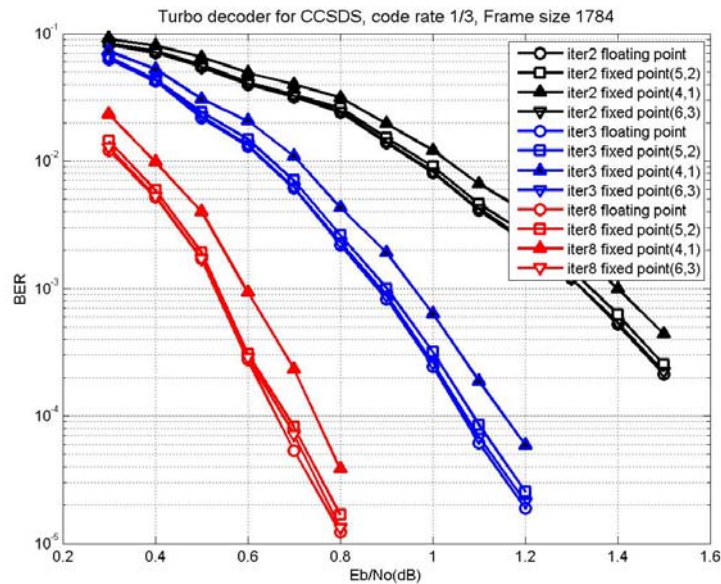


Figure 5.1 The comparison of BER performance for various soft inputs

5.2 The Bit-Width Estimation of Lex

After the last section estimation, we go on estimating extrinsic information. It is worth mentioning that if the entire range of extrinsic value is to be expressed, at least 7 bits for the integer part is needed. If we take 2 bits as the fractional part, then 9 bits are

needed to store an extrinsic value. In fact, the quantization scheme for the extrinsic value can be employed. The reason why the scheme can be used is described in [26].

At the beginning, we simulate the iterative decoding assuming that the soft-input information and other parameters are 5 bits (i.e.: (5, 2) bits) and ideal, respectively, except the representation of the extrinsic information. Figure 5.2 shows the simulated BER versus E_b/N_0 for different bit numbers of fraction part. From the curves, we see that the one bit of fraction part is very close to the floating point case. Although the BER performances of the two and three bits of fraction1 part have a little better than the one bit of fraction part, we choose the one bit to present the fraction part of the extrinsic information in hardware point of view.

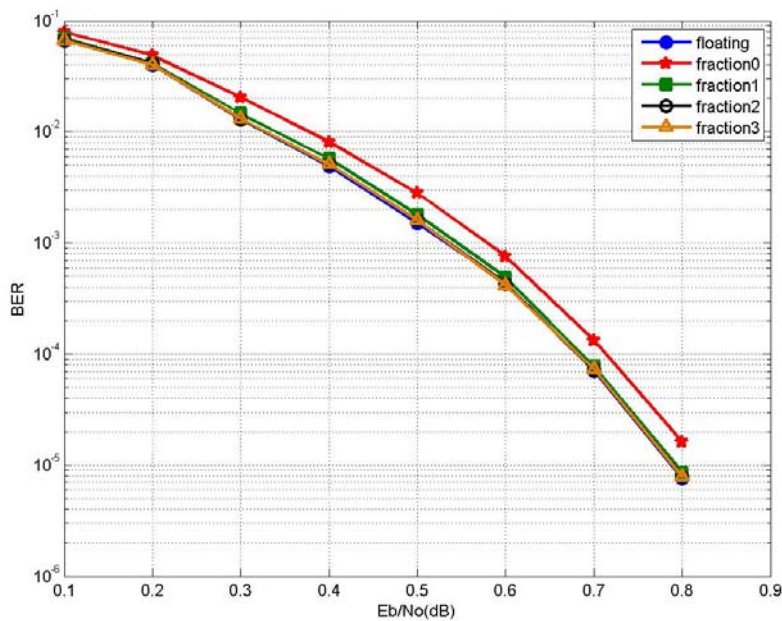


Figure 5.2 The comparison of BER performance for various soft inputs and extrinsic information

Then, Figure 5.3 shows the simulated BER versus E_b/N_0 for different bit numbers of integer part, while one bit is enough to indicate the fraction part. From the curves, we see that four bits presents the integer part is very close to the floating one.

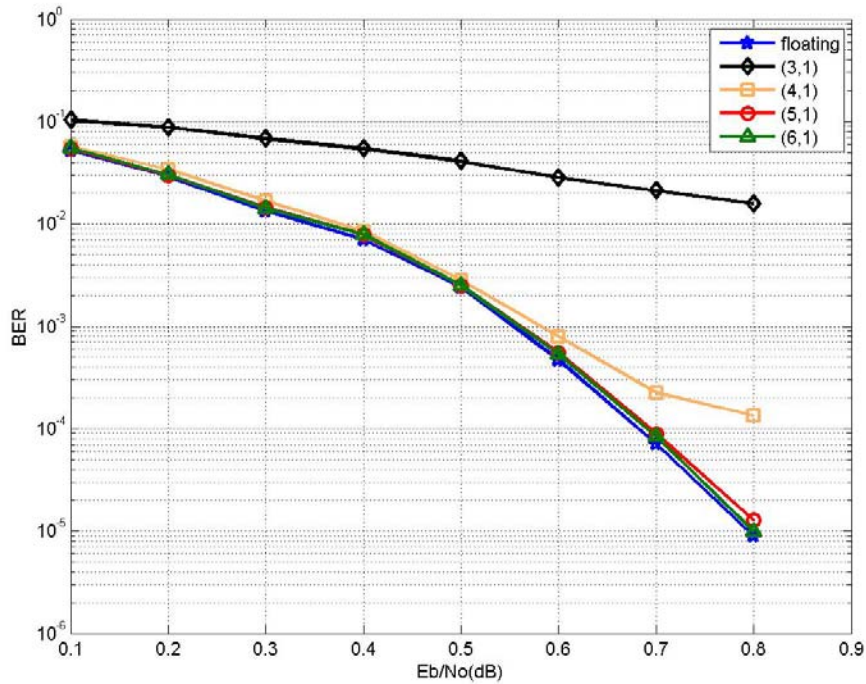


Figure 5.3 The comparison of BER performance for various soft inputs and

Figure 5.4 shows the simulated BER versus E_b/N_0 for different values of channel reliability. Theoretically, it is necessary to estimate the SNR when using log-MAP constituent decoder, while [30] reported that the differences are surprisingly small if a suitable parameter of channel reliability is selected. We use five hypothetical values of channel reliability to obtain the best result. Figure 5.4 present that 1.75 is better than others; therefore, we assume the channel reliability value is 1.75 for hardware implementation. The other simple specifications of the proposed turbo decoder are given in Table 5.1.

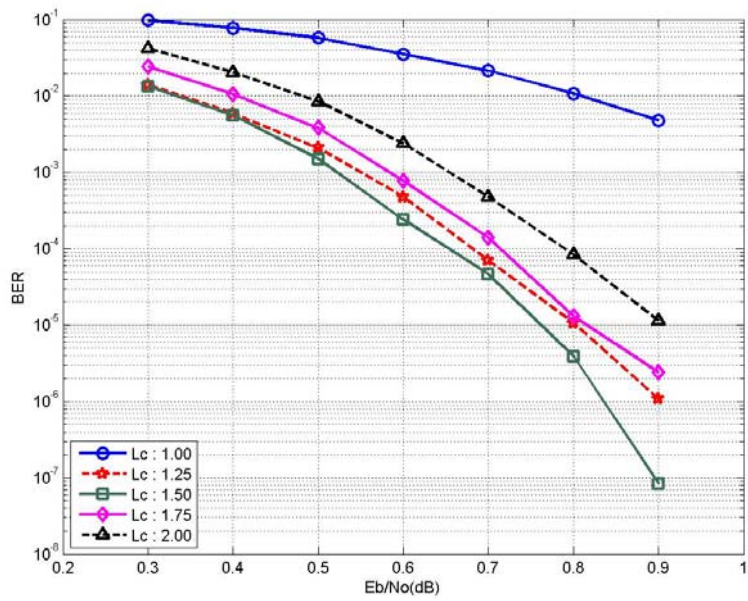


Figure 5.4 The comparison of various channel reliability

Table 5.1 Proposed Turbo Decoder Specification

Code polynomial	$\left[\begin{array}{c} 1+D+D^3+D^4 \\ 1+D^3+D^4 \end{array} \right]$		Note: because of HDA2 method is employed in our design, the average number of iterations is about 4.5.
Code rate	1/3		
Frame size	1784		
Window size	32		
Channel reliability	1.50		
Iteration number	≈ 4.5		
Scaling factor	The first three times: 0.75 Others: 1		
Data Width	Input	5(3,2)	
	$\alpha \cdot \beta \cdot \gamma$	10(8,2)	
	Lex	6(4,2)	

Chapter 6 Turbo Decoder Implementation in FPGA and ASIC

After the simulation and performance analysis in chapter 5, the bit-width of input symbol, branch metric, state metric and LLR is decided. This chapter will focus on our proposed turbo decoder for hardware implementation. Besides, in the last section, we will compare with other papers in hardware point of view.

6.1 The FPGA Implementation Results

In this section, we will first introduce the design and verify process, after that we report our FPGA implementation results.

First we write a MATLAB program to simulation the turbo decoding algorithm so that we can make sure we understand the flow of the process. Second, we develop a bit-accurate MATLAB model according to the architecture in the chapter three and chapter four. On the other hand, we write a RTL (Register Transfer Level) in Verilog code for hardware implementation. Then, we can verify our RTL code by MATLAB golden model, MATLAB model can help us to process Verilog HDL debugging easily. Third, after the functions of RTL code operate well, for the FPGA aspect, we use the Xilinx ISE 7.1i tools to produce the bit files and we can download the bit files to the FPGA develop board. Afterward we verify the hardware circuit by Vericomm tools. Beside, the ASIC process will be presented in the next section. Summarize our development and design flow is shown in Figure 6.1.

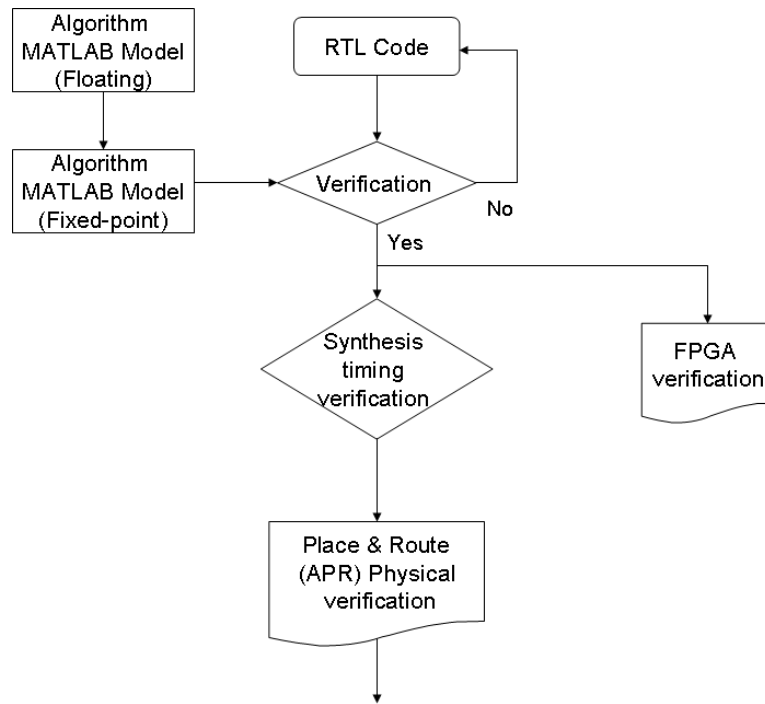


Figure 6.1 Development and design flow of the process

Figure 6.2 is a flow graph of turbo decoder which is based on one Radix-4 MAP decoder to implement. As the turbo decoder starts, the input data frame is stored in the ‘In Buffer’ memory. After that, MAP decoder fetches the input frame to calculate the LLR and extrinsic information. Then, the early stopping phase is beginning after two decoding iterations. The detail context is described in section 4.2. If the channel condition is not good, we also make a certain number of iteration to stop the decoding process. Finally, the decoding process make the hard decision according to the sign bit of LLR.

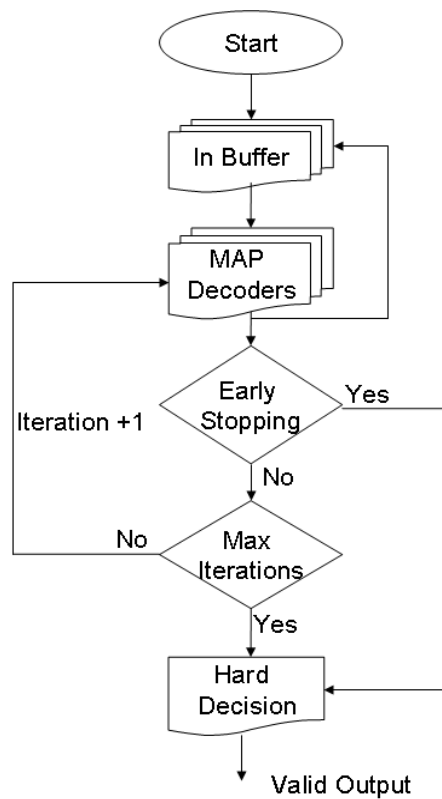


Figure 6.2 The flow graph of turbo decoder

For convenience, we pack the decoder as a processing core and indicate the input/output ports in Table 6.1. When this processing core is used, we just need to configure those pins adequately. The I/O diagram of this FPGA processing core is shown in Figure 6.3. The simulation environment is as follows:

FPGA development board: Xilinx Virtex-IV XC4VLX60-12FF1148

Simulation Software: Xilinx ISE 7.1i, VeriComm Pro.

HDL: verilog

Maximum clock frequency: 144.4 MHz

Therefore the turbo decoding rate is:

$$\text{Data rate: } R = \frac{K \cdot f \cdot M}{2 \cdot I \cdot \left(\frac{K}{N} + 2 \cdot W\right)} \approx 33.23 \text{ Mbps}$$

The turbo decoding latency is:

$$\text{Latency : } L = \frac{2 \cdot I \cdot \left(\frac{K}{N} + 2 \cdot W\right)}{f \cdot M} \approx 53.68 \mu\text{sec}$$

Where all parameters above are defined: frame size $K = 1784$; clock frequency $f = 144.4$ MHz; radix- 2^M (i.e. $M=2$) MAP decoder; the number of iterations $I \doteq 4.25$; number of MAP decoders $N = 1$; window size $W = 16$; input information data rate $R = 33.23$ Mbps; and the latency $L = 53.68$ μsec . The total occupied area is around 12904 slices (the input and output memory are included) from a total of 26624 slices.

Table 6.1 I/O ports definition

Port	I/O	Bit Width	Description
CLK	input	1	System clock
RESET	input	1	Reset the register contents
IN_VALID	input	1	Indicate the frame size of input data valid
SYSTEMATIC	input	5	The systematic input data
PARITY1	input	5	The parity input data (for in-order RSC encoder)
PARITY2	input	5	The parity input data (for re-permuted RSC encoder)
ITERATION	output	5	The iteration number
OUT_VALID	output	1	Indicate the decoder bit vialid
DECODER_OUT	output	1	Decode bit output

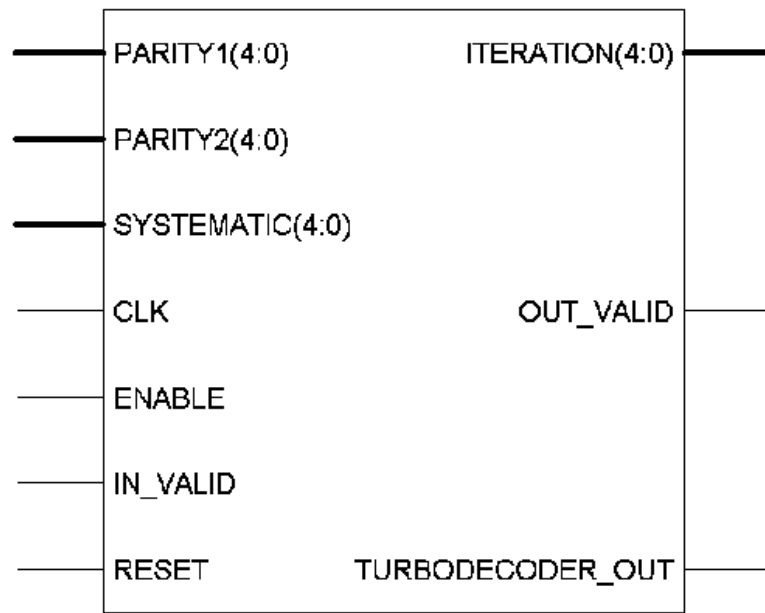


Figure 6.3 Turbo decoder I/O diagram under FPGA verification

6.2 The ASIC Implementation Results

We are interested in how many gate counts are used in the proposed turbo decoder, where single SISO decoder is employed of the turbo decoder. Table 6.2 shows the area and gate counts reports for each block components of SISO decoder. The ASIC verification flow is shown in Figure 6.4. The encoder sequence, BPSK (binary phase shift keying) modulation and the AWGN (additive white Gaussian noise) are generated by MATLAB tools and are written the information into TESTBENCH block. We can compare the results with the decoding bits by bit-accurate MATLAB decoding program. If “Error” outputs the other number but not zero, there should be something wrong in the decoding hardware.

Table 6.2 Area report for each component of SISO decoder

Component	Gate Count (Size)	Area (90 nm)
α state metric	16040.86	68975.7
β state metric	14597.56	62769.5
Dummy β state metric	13031.31	56034.66
γ branch metric	2950.86	12688.71
α memory	2560 bits	38335.0 x 4
Sliding window memory	1024 bits	20761.0 x 4
LLR	44949.86	193284.4
Total	152243.35	654646.4

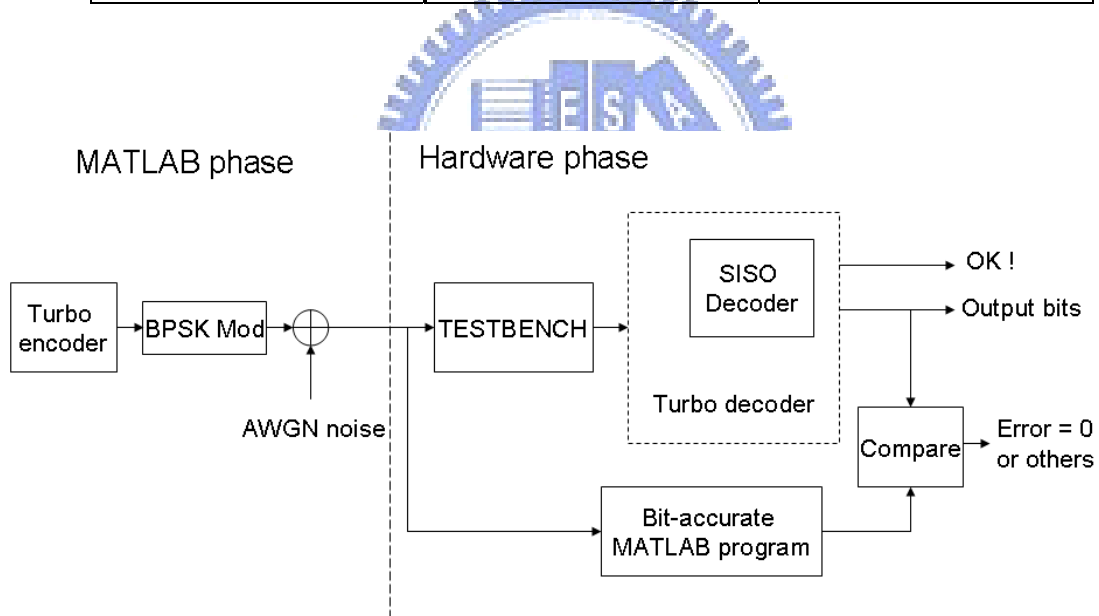


Figure 6.4 ASIC verification flow

We use SOC_Encounter as APR (automatically place & route) tool and layout is shown in Figure 6.5. The chip density and core size for the decoder are 64.6% and $1.26 \times 1.26 \text{ mm}^2 = 1.5876 \text{ mm}^2$, respectively. The detail ASIC simulation environment is as below:

HDL: Verilog

Compiler tool: NC-Verilog

Debug tool: Debussy

Synthesis tool: synopsys

Process: UMC 90 nm

The maximum clock rate for proposed turbo decoding process is 357.14 MHz, and the turbo decoding rate is:

$$\text{Data rate : } R = \frac{K \cdot f \cdot M}{2 \cdot I \cdot \left(\frac{K}{N} + 2 \cdot W\right)} \approx 77.62 \text{ Mbps}$$

The turbo decoding latency is:

$$\text{Latency : } L = \frac{2 \cdot I \cdot \left(\frac{K}{N} + 2 \cdot W\right)}{f \cdot M} \approx 22.98 \mu \text{ sec}$$

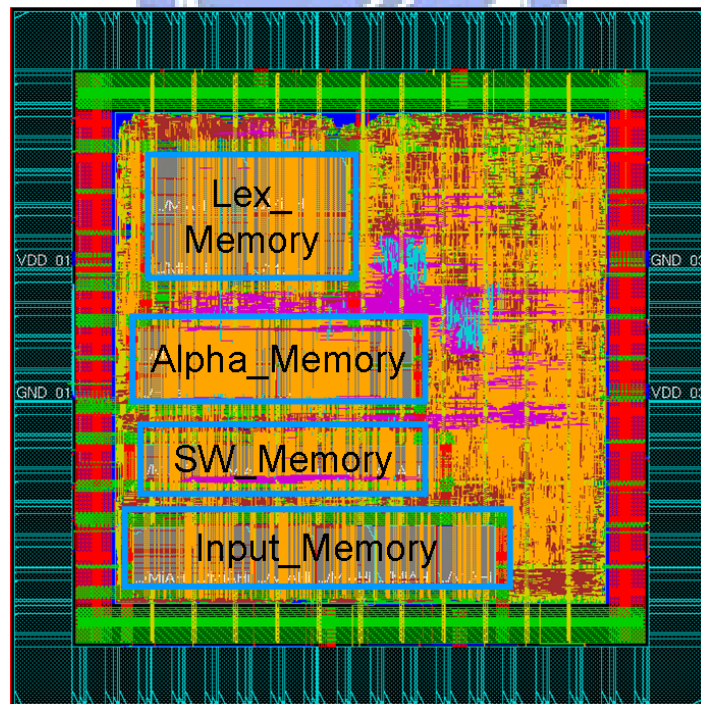


Figure 6.5 Chip layout of turbo decoder with single SISO decoder

Moreover, in order to further improve the turbo decoder speed, the improved radix-4

recursion unit, HDA2 early stopping criterion and parallel SISO decoders are shown in Figure 6.6. On the other hand, in order to solve the collision issue, the modified annealing method is introduced and that results in contention free and no any extra buffers are needed.

Finally, the 17.64mm^2 core area can support the maximum data rate is:

$$\text{Data rate : } R = \frac{K \cdot f \cdot M}{2 \cdot I \cdot \left(\frac{K}{N} + 2 \cdot W\right)} \approx 884.91\text{Mbps}$$

The turbo decoding latency is:

$$\text{Latency : } L = \frac{2 \cdot I \cdot \left(\frac{K}{N} + 2 \cdot W\right)}{f \cdot M} \approx 2.016\mu\text{sec}$$

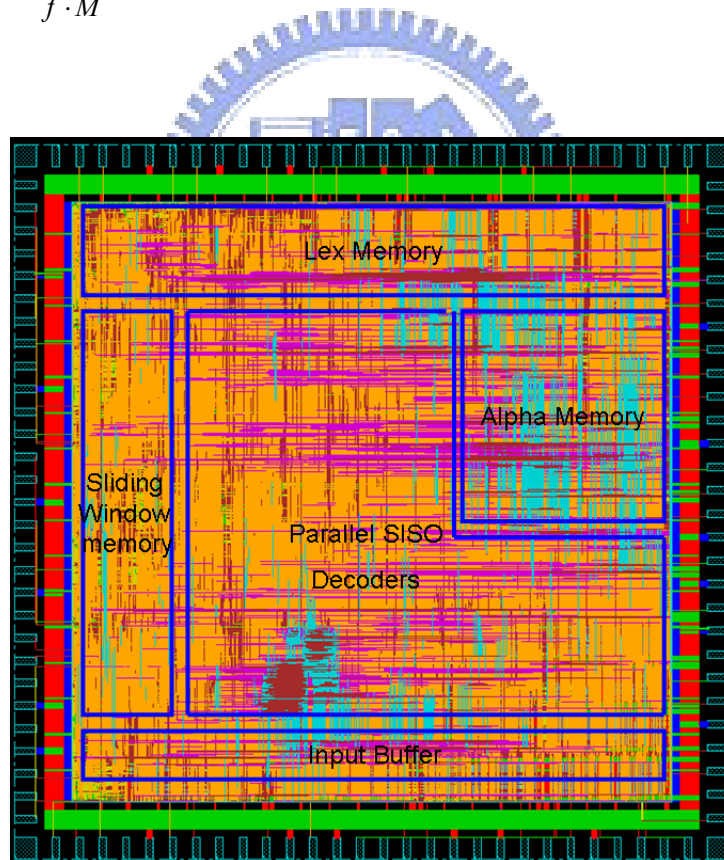


Figure 6.6 Chip layout of parallel turbo decoder by SoC Encounter

6.3 Comparison

In this section, we summarize the BER performance, area, timing, and others comparison. The synthesis result is shown in Table 6.3, Arch-T denotes the traditional radix-2 ACSO architecture; Arch-L denotes the modified radix-4 architecture [9]; Arch-W the radix-4 architecture proposed by Wang [10]; and Arch-C the proposed architecture, having the highest throughput among all recursion units.

Table 6.3 Comparison of four recursion architectures

	Timing (ns)	Relative area	Relative throughput
Arch-T	1.80	1	1
Arch-L	2.40	1.61	1.50
Arch-W	2.22	1.96	1.62
Arch-C (proposed)	2.01	1.94	1.80

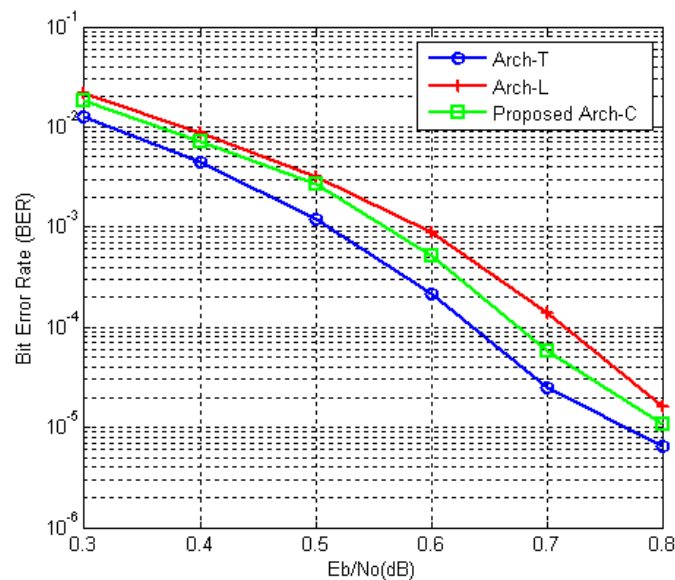


Figure 6.7 Performance comparisons among those three architectures

Figure 6.7 shows the BER performance of a code rate 1/3, 16 states, and frame size of 1784 bits on CCSDS standard. The number of total iterations is eight. The MATLAB simulations are operating under the assumption of AWGN channel and BPSK modulation. We could see that the traditional radix-2 architecture has the best performance due to least approximation, and the other two approximation architecture: Arch-L and Arch-C resulting to about 0.1 and 0.05dB performance loss, respectively.

The proposed design is compared with [27], [28], and the results shown in Table 6.4. Those three designs are all based on the CCSDS single-MAP decoding architecture for telemetry channel coding. Due to the high-radix structure (the early stopping rule is employed), the proposed design is the fastest one among all in Table 6.4.

Table 6.4 Comparison of CCSDS turbo decoders

	Refer to [27]	Refer to [28]	Proposed architecture
Board type	TITMS320C6000 (DSP x 8)	Xilinx Virtex-V XC5VLX30-3	Xilinx Virtex-IV XC4VLX60
Area	NA.	3411 Slices	13504 Slices
Speed	365 K bps	9.81 M bps	33.23 M bps
Clock rate	200 MHz	~100 MHz	144.4 MHz
Code rate	1/6	1/2 ~ 1/7	1/3
Frame size	8920	1784~16056	1784
Iteration numbers	10	5	~4.25
Note: the input/output buffer and interleaver address generator are not included in [28]			

Chapter 7 Conclusions

In this thesis, a hardware implementation for the CCSDS turbo decoder is presented. This implementation based on high throughput radix-4 recursion architecture. In order to increase the clock frequency, our proposed architecture “Arch-C” uses four comparators to fetch the maximum value of the four inputs. Besides, the hybrid 4-inputs subtraction method is presented to avoid becoming the critical path. On the other hand, in order to further increase the decoder rate, the HDA2 early stopping rule is employed with an insignificant hardware overhead and performance loss. Additionally, due to the approximate radix-4 MAP algorithm, we need to estimate the scaling factor to compensate for the performance loss. The better choice that the scaling factor is 0.75 for the first three iterations, and other iterations is 1. By the way, because we do not have the information of the channel reliability (even though some papers had approached methods to estimate the channel reliability, in this thesis we do not to do so.), we need to select a constant as the channel reliability. After MATLAB simulation as shown in Figure 5.3, we select an appropriate value 1.5 as the channel reliability. After chip implementation in 90nm process, the maximum clock rate 357.14MHz can be achieved, and the 17.64mm² core area can support the maximum data rate 884.91MS/s of turbo decoder with fourteen MAP decoders.

REFERENCE

- [1] C. Berrou, A. Glavieux and P. Thitimajshima, “Near Shannon limit error-correcting coding and decoding: Turbo-codes,” in Proc. ICC '93, Geneva, Switzerland, May 1993, pp. 1064–1070.
- [2] C. E. Shannon, “A Mathematical Theory of Communication,” *Bell System Technical Journal*, pp. 379-427, 1948.
- [3] *IEEE Std 802.16e-2005, 802.16 TGe*, Feb. 2006.
- [4] 3GPP Specifications. 3rd generation partnership project. [Online]. Available: <http://www.3gpp.org>
- [5] Consultative Committee for Space Data Systems, Recommendation for Telemetry Channel Coding, CCSDS 101.0-B-6, Blue Book, October 2002.
- [6] L. Bahl, J. Cocke, F. Jelinek and J. Raviv, “Optimal decoding of linear codes for minimizing symbol error rate,” *IEEE Trans. on Information Theory*, vol. 20, pp. 284-287, May 1974.
- [7] J. Hagenauer and P. Hoehner, “A Viterbi algorithm with soft-decision outputs and its applications,” in Proc. IEEE Globecom Conf., Nov. 1989, pp. 1680-1686.
- [8] A. J. Viterbi, “An intuitive justification and a simplified implementation of the MAP decoder for convolutional codes,” *IEEE J. Select. Areas Communication*, vol. 16, pp. 260-264, Feb. 1998.
- [9] M. Bicherstaff, L. Davis, C. Thomas, D. Garrett, and C. Nicol, “A 24Mb/s Radix-4 LogMAP Turbo Decoder for 3GPP-HSDPA Mobile Wireless,” in *IEEE ISSCC Dig. Tech. Papers*, 2003, pp. 150 – 151.
- [10] Z. Wang, “High-speed recursion architecture for MAP-based Turbo decoders”, in *IEEE Trans. VLSI Syst*, vol 14, No. 4, pp. 470-474, April 2007.

- [11] C. Zhang, X. Wang, F. Ye and J. Ren, "A 400Mb/s radix-4 MAP decoder with fast recursion architecture" in *IEEE ICACT 2008*, vol. 2, 17-20 Feb. 2008 paper(s): 1339-1342.
- [12] E. Boutillon, W.J. Gross and P.G. Gulak, "VLSI architectures for the MAP algorithm," *IEEE Transactions on Communications*, vol. 51(2), pp. 175 - 185, Feb. 2003.
- [13] J. Ertel, J. Vogt, A. Finger, "A high throughput Turbo Decoder for an OFDM-based WLAN demonstrator," in proceedings of 5th International ITG Conference on Source and Channel Coding (SCC), Jan. 2004.
- [14] A. Hekstra, "An alternative to metric rescaling in Viterbi decoders" *IEEE Trans. on Communications*, 37(11): 1220-1222, Nov 1989.
- [15] C. B. Shung, G. Ungerboeck and H. K. Thapar, "VLSI architectures for metric normalization in the Viterbi algorithm," in *Proc. IEEE Int. Conference Communications (ICC '90)*, vol.4, Atlanta, GA, Apr. 16-19, 1990, pp.1723-1728.
- [16] A. Worm, H. Michel, F. Gilbert, G. Kreiselmaier, M. Thul and N. When, "Advanced implementation issues of turbo-decoders" in *Proc. 2nd Int. Symp. on Turbo Codes*, Brest, France, Sept. 2000, pp. 351-354.
- [17] T.-H. Tsai, C.-H. Lin, and A.-Y. Wu, "A memory-reduced log-MAP kernel for turbo decoder," in *Prof. IEEE ISCAS*, 2005, pp. 1032-1035.
- [18] Ahmed and T. Arslan, "VLSI Design of Multi Standard Turbo Decoder for 3G and Beyond," 12th Asia and South Pacific Design Automation Conference (ASP-DAC 2007), pp. 589-594, Pacifico Yokohama, Yokohama, Japan, January 23-26, 2007.
- [19] Engin. N, "Turbo decoder architecture with scalable parallelism," in *Proceedings of IEEE Workshop on Signal Processing Systems*, 2004, pp. 298.303
- [20] A. Giulietti, L. Van der Perre, and A. Strum, "Parallel turbo coding interleavers:

Avoiding collisions in accesses to storage elements,” *Electron. Lett.*, vol. 38, pp. 232–234, Feb. 2002.

- [21] A. Tarable, S. Benedetto, G. Montorsi, “Mapping interleaving laws to parallel turbo and LDPC decoder architectures,” in *IEEE Transaction on*, vol 50, pp. 2002-2009, Sept. 2004.
- [22] J. Vogt, J. Ertel, and A. Finger, “Reducing bit width of extrinsic memory in turbo decoder realizations,” *Electron. Lett.*, pt. 20, pp. 1714–1716, Sept. 2000.
- [23] Z. Wang, Y. Zhang and K. K. Parhi, “Study of early stopping criteria for turbo decoding and their applications in WCDMA systems” in *Proc of ICASSP’06*, pp. III-1016-1019, May. 2006.
- [24] A. Matache, S. Dolinar, and F. Pollara, “Stopping rules for turbo decoders,” Tech. Rep., Jet Propulsion Laboratory, Pasadena, California, Aug. 2000.
- [25] T. M. N. Ngatched and F. Takawira, “Simple stopping criterion for turbo decoding,” *Electronics Letters*, vol. 37, no. 22, pp. 1350 – 1351, Oct. 2001.
- [26] Z. Wang, H. Suzuki, and K. K. Parhi, “Vlsi implementation issues of turbo decoder design for wireless applications,” in *Proc. of 1999 IEEE Workshop on Signal Processing Systems (SIPS’99)*, Oct. 1999, pp. 503–512.
- [27] Jeff B. Berner, Kenneth S. Andrews, “Deep Space Network Turbo Decoder Implementation” *Aerospace Conference, 2001, IEEE Proceedings*.
- [28] <http://www.sworld.com.au/pub/pcd04c.pdf>, Small World Communications.
- [29] Keshab K. Parhi, “VLSI Digital Signal Processing Systems: Desing and Implementation,” New York: Wiley, 1999.
- [30] A. Worn, Peter. Hoehner, Norbert. Wehn, “Turbo-decoding without SNR estimation” *IEEE Communications leter*, vol 4, NO 6, June 2000.