

國立交通大學

電信工程學系

碩士論文

用於平行渦輪碼之無衝突演算法

Contention Free Algorithm
For Parallel Turbo Decoder

研究生：曾凱信

指導教授：張振壹 教授

共同指導教授：方偉騏 教授

中華民國九十八年六月

用於平行渦輪碼之無衝突演算法
Contention Free Algorithm For Parallel Turbo
Decoder

研究生：曾凱信 Student: Kai-Hsin Tseng
指導教授：張振壹 博士 Advisor: Chen-Yi Chang
共同指導教授：方偉騏 博士 Co-Advisor: Wai-Chi Fang



A Thesis

Submitted to Institute of Communication Engineering
College of Electrical Engineering and Computer Science
National Chial Tung University

In Partial Fulfillment of the Requirements
for the Degree of
Master of Science

in

Communication Engineering

June 2009

Hsinchu Taiwan, Republic of China

中華民國九十八年六月

用於平行渦輪碼之無衝突演算法

研究生：曾凱信

指導教授：張振壹博士 共同指導教授：方偉騏博士

國立交通大學電信工程學系(研究所)碩士班

中文摘要

在此論文中我們利用退火模擬演算法(Simulated Annealing Algorithm)提出無衝突演算法去解決平行渦輪碼中記憶體碰撞問題。再者，對於平行渦輪碼中的非本質記憶體，我們提出有效使記憶體面積減少的兩種架構；其中一種架構是由平行單埠記憶體與一個緩衝暫存器所組成去取代原來須兩埠或雙埠記憶體所組成的架構。另外一個架構，我們基於前一個架構上再加上一個非本質函數的非線性映對器。在前兩種架構相較於傳統使用雙埠記憶體在 0.13 CMOS 聯電製程環境底下分別可以節省約 37 和 46 百分比記憶體使用量。

Contention Free Algorithm For Parallel Turbo Decoder

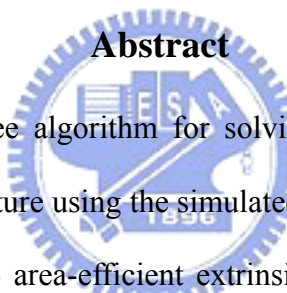
Student: Kai-Hsin Tseng

Advisor: Chen-Yi Chang Co -Advisor: Wai-Chi Fang

Institute of Communication Engineering

National Chial Tung University

Abstract

The logo of National Chial Tung University is a circular emblem with a blue border. Inside the circle, there are stylized Chinese characters and the acronym 'NCTU'. The logo is positioned behind the abstract text.

In this thesis, a contention free algorithm for solving memory collision problem of parallel Turbo decoder architecture using the simulated annealing algorithm is presented. Furthermore, we proposed two area-efficient extrinsic memory schemes based on the parallel contention free Turbo decoder. One of the proposed schemes employs only multiple single port memories with one temporary buffer instead of the original dual port or two port memories. And the other scheme further employs an additional non-linear extrinsic mapping architecture. The proposed schemes lead to approximately 37% and 46% memory area reduction, respectively, for 16-parallel Turbo decoder in comparison to the conventional dual port memory scheme under the UMC 0.13- μm CMOS process.

誌謝

這幾年碩士的求學生涯是一段豐富的旅程。雖然這段求學路走來波折不斷，不過在師友及家人的協助下逐一地克服困難進而更上一層樓。我要非常感謝張振壹教授和方偉騏教授，在我遇到求學期間最困難的時候，兩位老師們給我機會及諄諄教誨令我擁有面對更多挑戰的信心。

SOCLab 一個和諧的團隊合作，同學之間一起討論研究、解決問題和互相的鼓勵，讓我在此學習成長，吸收許多的寶貴的意見。感謝 正煌、秋國、盛弘學長，翔琮同學，志文、源煌、少彥、宗翰、致中和鴻溝學弟以及在 SOCLab 每一位成員，特別是翔琮和少彥同學在研究上和英文上給予相當大的幫忙和鼓勵。

最後誠摯地感謝我的家人(爸爸、媽媽和姊姊)，當我最大的後盾。還時常為我擔憂學業情形及是否溫飽，這股關懷更是我最大前進的動力。最後，我更要感謝已在天國的阿公和阿嬤，感謝您兩位老人家對凱信我的疼愛與深深地期盼。

在每一個困境中，家人的支持是我最大的動力，教授的指導給予我一盞明燈，實驗室的團隊合作是我的最大的力量。最後，感謝交通大學給予我這麼有系統的專業知識及學習做人處事的環境。

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 Motivation	1
1.2 Thesis Organization.....	2
Chapter 2 Turbo Code	3
2.1 System Overview.....	3
2.2 Turbo Encoder	4
2.2.1 Turbo Encoder Process.....	4
2.2.2 Recursive Systematic Convolution (RSC).....	5
2.2.3 Trellis-Termination.....	7
2.2.4 Puncturing	9
2.3 Interleaver.....	10
2.3.1 Block Interleaver.....	10
2.3.2 Prime Interleaver.....	11
2.3.3 Random Interleaver.....	12
2.3.4 S - Interleaver.....	12

2.3.5	Characteristic of Interleaver	13
2.4	Channel Model	14
2.5	Turbo Decoder Process	15
2.6	SISO Decoding Algorithm.....	17
2.6.1	Log-MAP Algorithm	17
2.6.2	Max-Log-MAP Algorithm	19
2.6.3	Initialized Procedure for Both Log-MAP and Max-Log-MAP Algorithm	19
2.7	Error Probability for Turbo Code	20
2.8	Turbo Code Application on Telemetry and Deep Space Communications ..	21
2.9	Simulation and Results	24
Chapter 3	VLSI Architecture of Turbo Decoder	28
3.1	Sliding Window Approach.....	29
3.2	VLSI Architecture of SISO Decoder for CCSDS Standard.....	31
3.2.1	Log-MAP Decoder	32
3.2.2	Interleaver Address Calculation Unit	38
3.2.3	Extrinsic Information Quantization.....	39
3.3	Serial SISO Structure.....	39
3.4	Parallel SISOs Structure	40
Chapter 4	Solving Memory Collision Problem For Parallel Turbo Decoder	43
4.1	Introduction	43
4.2	Memory Collision Problem for Parallel Turbo Decoder	43
4.3	Solving Memory Collision Problem Using Temporal Buffer Architecture ..	44
4.4	Proposed Memory Contention Free Scheme for Parallel Turbo Decoder	45
4.4.1	Definition of Memory Collision Problem	46

4.4.2	Solution to Graph Coloring Problem by Simulated Annealing Algorithm	48
4.4.3	The Extrinsic Memory Collision Free VLSI Architecture Design.....	52
4.4.4	Simulation and Experiment Results	53
4.5	An Approach for Reducing Memory Area of Parallel Turbo Decoder.....	63
4.5.1	Classical Extrinsic Memory Access for Single Turbo Decoder	63
4.5.2	An Area-Efficient Extrinsic Memory Scheme for Parallel Turbo Decoder	63
4.5.3	Analysis of the Required Memory Size	66
Chapter 5	Conclusion	69
	Bibliography	70



LIST OF FIGURES

Fig. 2-1 Digital communication system	3
Fig. 2-2 Turbo encoder diagram	4
Fig. 2-3 Block diagram of the RSC	6
Fig.2-4 Trellis expression the relationship of current states and next states with different input sequence.....	6
Fig. 2-5 Turbo decoder diagram	17
Fig. 2-6 CCSDS Turbo encoder.....	22
Fig. 2-7 Turbo code for different code rates.....	23
Fig. 2-8 Performance results of Turbo code for different turbo decoding algorithm ...	25
Fig. 2-9 Performance results of Turbo code for different block sizes	26
Fig. 2-10 Performance results of Turbo code for different iterations.....	26
Fig. 2-11 Performance results of Turbo code for different code rate	27
Fig. 3-1 Turbo decoder architecture	28
Fig. 3-2 The operation of sliding window approach	31
Fig. 3-3 Block diagram of the branch metric calculation.....	32
Fig. 3-4 The forward and backward recursive calculating by the trellis expression....	33
Fig. 3-5 Block diagram of the ACSO and OACS architecture.....	33
Fig. 3-6 Block diagram of LLR calculation architecture.....	34
Fig. 3-7 The change of BMC block placement in back of sliding window memory ...	36
Fig. 3-8 The sliding memories size requirement between traditional sliding window scheme [24] and our modified method for different code rate. @ ($SW_{depth} =$ $32, bit(L_i) = 6, bit(received\ codeword) = 5, \gamma_{width} = 6, two\ sliding\ window$	

memories)	37
Fig. 3-9 Block diagram of on-line interleaver pattern calculation	38
Fig. 3-10 Non-linear quantization for extrinsic information	39
Fig. 4-1 An example of memory collision event.	44
Fig. 4-2 Avoiding conflicting using temporal memory architecture	45
Fig. 4-3 (a) An example of natural order and interleaving order for 4-parallel Turbo decoder. (b) Converting the example of memory collision problem with graph expression.	48
Fig. 4-4 Using simulated annealing algorithm for solving memory collision problem.	51
Fig. 4-5 (a) The solution of the example of Fig. 4-3 obtaining from contention free algorithm. (b) Converting each color set to corresponding each node element.	52
Fig. 4-6 Structure of proposed memory collision free architecture for the example of Fig. 4-3.	53
Fig. 4-7 BER performance of the Turbo decoder	54
Fig. 4-8 The change of cost functions of contention free algorithm for various parallel Turbo decoder applications.	57
Fig. 4-9 The solution of contention free algorithm for 8-parallel Turbo decoder	58
Fig. 4-10 The solution of contention free algorithm for 16-parallel Turbo decoder	59
Fig. 4-11 The solution of contention free algorithm for 32-parallel Turbo decoder	60
Fig. 4-12 The VLSI architecture implementation of Turbo decoder in the FPGA platform	61
Fig. 4-13 The comparison of the output values of golden model from matlab [®] with the output values of FPGA	61
Fig. 4-14 The block diagram for the proposed contention free parallel Turbo decoder	62

Fig. 4-15 The waveform expression between the single log-MAP decoder and the external storage components which consist of input buffer and extrinsic memory.64

Fig. 4-16 The waveform expression between the multiple SISO decoders and the external storage components which consist of input buffers and extrinsic memories.65

Fig. 4-17 Structure of proposed an area-efficient extrinsic memory scheme for parallel Turbo decoder architecture.65

Fig. 4-18 Comparison of area requirements for different organization of extrinsic memory architecture (@ UMC 0.13- μ m CMOS Process Measured and latency L=104 cycles measured).68



LIST OF TABLES

Table 2-1	The Output encoded sequence with different input information corresponding to each state.....	7
Table 2-2	Interleaver algorithm for the Turbo code of CCSDS standard.....	23
Table 3-1	The minimum linear combination sets of branch metric values for various code rates	36
Table 3-2	The sliding window memories storage comparison of our modified method and traditional SW scheme	37
Table 4-1	Summary of parameters for Turbo code simulation.....	54
Table 4-2	Parallel Turbo decoder area and through for various number of SISO decoders at clock frequency 200MHz.	62
Table 4-3	Summary of area requirements for various organization of extrinsic memory architecture	67

Chapter 1 Introduction

1.1 Motivation

Turbo code has outstanding error correcting capacity, which was first introduced in 1993 [1], and its performance closely approaches the Shannon limit for *Bit Error Rate* (BER). The fundamental turbo decoder comprises interleaver and constituent (*Soft-In/Soft-Out*) SISO decoders. The SISO decoder performs iterative decoding based on *maximum a posterior* (MAP) probability algorithm, which often transfers into logarithm domain as log-MAP in the consideration of implementation complexity [2].

Since the Turbo decoder requires a certain number of iterations to achieve the desired performance, the iteratively decoding causes the lower throughput and higher latency for the Turbo decoder process. To apply for high speed and low latency application, a feasible method is to adopt the parallel SISO decoder architectures. However, one of the parallel SISO decoder architecture's existing problems is that there are probably more than one data to access the same memory destination simultaneously, also called the memory collision problem [3][4].

An available method of solving memory collision is to use extra storage devices for storing the collision dates until the destination memories are in idle state and can be accessed [20]. However, the above solution method requires an extra temporary buffer and collision handling time in view of hardware aspects. Therefore, the objective of the present memory collision free algorithm is to distribute the extrinsic dates from parallel SISO decoders into the storage elements without memory collision happening. The proposed memory collision free algorithm can support various Turbo standards as well as arbitrary the number of parallel high radix SISO architecture.

1.2 Thesis Organization

The thesis is organized as follows. Chapter II shows the concept of Turbo coding, including Turbo encoder / decoder structure, Log-MAP algorithm and Max-Log-MAP algorithm. The sliding window approach and the difference between the serial SISO structure and parallel structure are discussed in Chapter III. Chapter IV illustrates the parallel turbo decoder using simulated annealing algorithm achieving memory collision free requirement and supporting arbitrary parallel parameter P . Finally, the conclusions are given in Chapter V



Chapter 2 Turbo Code

This chapter introduces the components of turbo code, including turbo encoder, turbo decoder, interleaver and given an example for the specification of turbo code of (Consultative Committee for Space Data Systems) CCSDS standards [5]. Finally, performance results are compared between the max-log-MAP and log-MAP decoding algorithm, various code rates, different block sizes and iteration numbers.

2.1 System Overview

Fig. 2-1 shows the Turbo code application in the digital communication system which includes four parts: 1.) channel, 2.) modulation, de-modulation, DAC, ADC and Front End parts, 3.) synchronizer and channel estimation (Equalizer), 4) error correction and detection. Channel involves non-idea effects and distortion in the modulated continuous waveform. Demodulator and ADC convert the distorted analog waveform into digital samples. Error correction recovers these samples and renders decoded sequences. The error detection is primary used to verify the correctness of decoded sequences. This thesis assumes that the perfect synchronization and channel estimation in the receiver aspects.

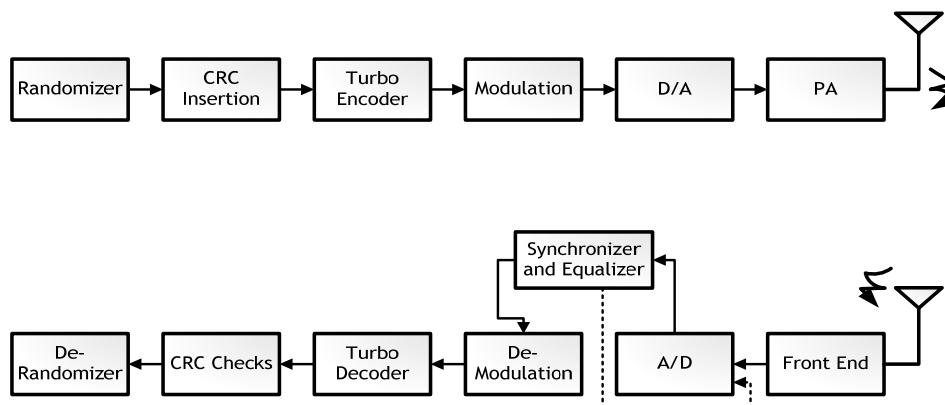


Fig. 2-1 Digital communication system

2.2 Turbo Encoder

2.2.1 Turbo Encoder Process

The turbo encoder consists of two parallel Recursive Symmetric Code (RSC) encoders, an interleaver and a puncture device (see Fig. 2-2). The interleaver is used for permuting the information u_k , which is an influencing factor in the performance of Turbo code. The information $u = \{u^1, u^2, \dots, u^N\}$ are transmitted through two identical structure RSC encoder, where encoder structure depends on the definition of code generator polynomial.

For the two RSC encoders, the information directly sending into upper RSC encoder produce upper encoded codeword $\{X_s, X_{p-siso1}\}$; the lower encoded codeword $\{X_{p-siso2}\}$ is obtained from the permuting information bits u_{int} passing through the lower RSC encoder. The outputs X_s is identical to information bits u , referred to as the systematic bits. The second output $X_{p-siso1}$ denotes the parity check bits, which will be used for the even sub-iteration of MAP decoding. Similarly, the other parity check bits $X_{p-siso2}$ will also be used to odd sub-iteration of MAP decoding. Finally, the puncturing block could support various code rates by multiplexing the encoded codeword sequence to obtain effective bandwidth utilization.

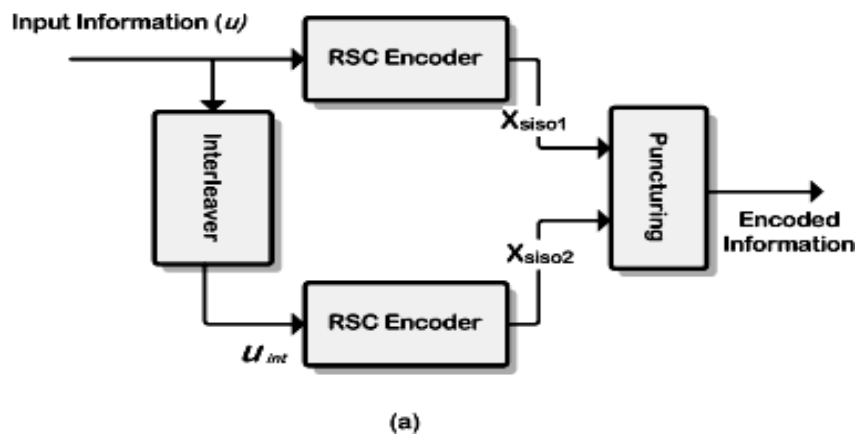


Fig. 2-2 Turbo encoder diagram

2.2.2 Recursive Systematic Convolution (RSC)

Good turbo codes have been constructed using short constraint length and infinite impulse response (IIR) convolutional codes instead of the more familiar finite impulse response (FIR) convolutional codes. The major reason for above finding is that the impulse response for IIR structure has more long free distance relative to FIR structure, resulting the more better performance for the IIR encoder structure [6].

Furthermore, several articles in [7] shown that the constituent convolution codes with primitive feedback polynomials can achieve larger minimum distance than applying other polynomials. As a result, the IIR encoder structure with primitive feedback polynomials is always employed for the constituent encoder of Turbo code.

These IIR convolutional codes are also referred to as recursive convolutional codes, because previously encoded information bits are fed back to the input of constituent encoder. For instance, the generator polynomial $G(D)$ form for constituent encoder shown in Fig. 2-3 is

$$G(D) = \begin{bmatrix} 1 & \frac{P_{F1}}{P_b} & \frac{P_{F2}}{P_b} & \frac{P_{F3}}{P_b} \end{bmatrix} \quad (2.1)$$

with the constraint length $\nu=5$ (constraint length $\nu =$ memory order $q + 1$), where P_b indicates feedback polynomial $(1+D^3+D^4)$, which fed previously encoder back to mix with new information sequence. P_{Fi} is forward polynomial corresponding to i -th output of encoder, here $(1+D+D^3+D^4)$, $(1+D^2+D^4)$ and $(1+D+ D^2+D^3+D^4)$ for $1 \leq i \leq 3$.

The constituent encoder total has $2^{\nu-1}$ distinct state, where each state expresses the temporal value of register components. When input sequence is fed, the temporal value of register components are affected by the input sequence and feedback information, leading to the change of register components. For previously example, let the all register values to be zeros, the update register values update into “1000”(called as S8) if

the high level of input information bit is sent. On the other hand, the current state remains to hold the all-zero state. Furthermore, the results for each state changing with all possible input patterns can be shown in Fig. 2-3 for the trellis expression, and the corresponding output encoded bits can look up in Table 2-1.

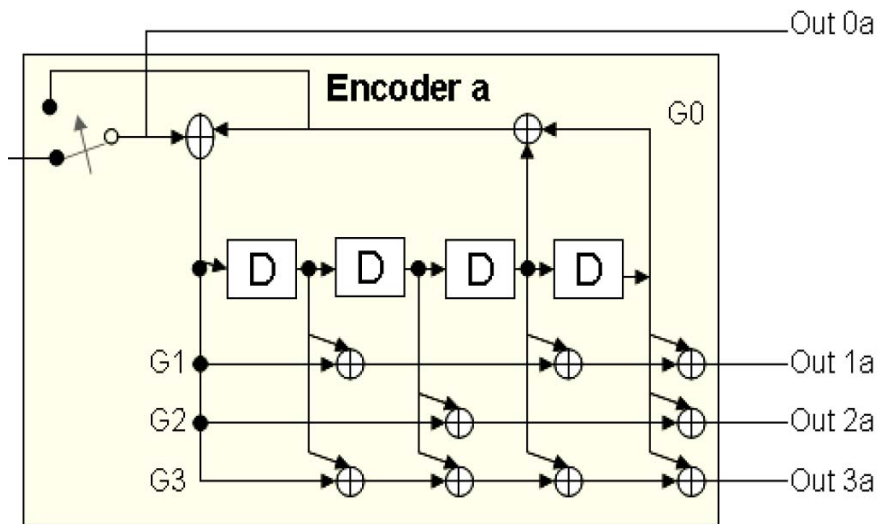


Fig. 2-3 Block diagram of the RSC

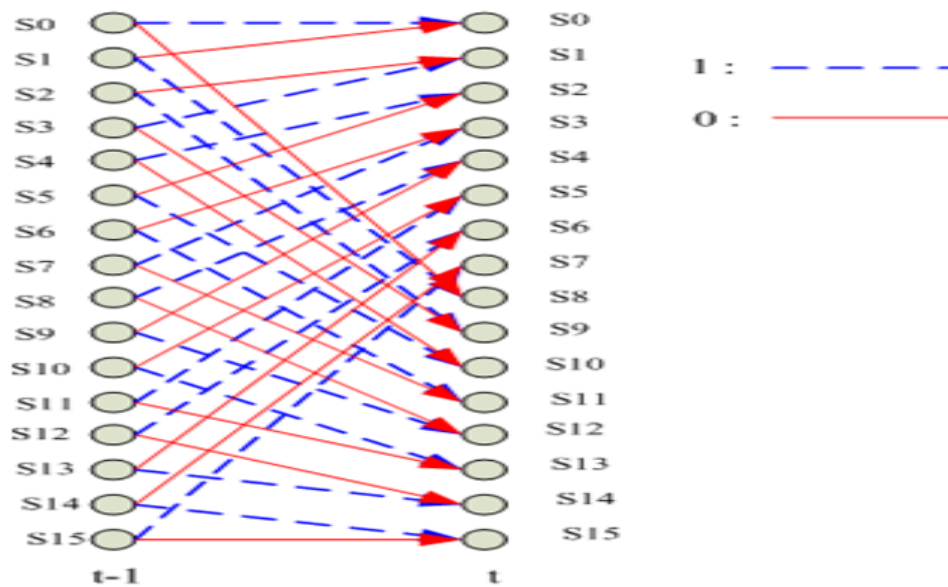


Fig.2-4 Trellis expression the relationship of current states and next states with different input sequence

Table 2-1 The Output encoded sequence with different input information corresponding to each state

	Input info.	Out 0a	Out 1 a	Out 2a	Out 3a
S0	0: 1:	0 1	0 1	0 1	0 1
S1	0: 1:	0 1	0 1	0 1	0 1
S2	0: 1:	0 1	0 1	1 0	0 1
S3	0: 1:	0 1	0 1	1 0	0 1
S4	0: 1:	0 1	0 1	1 0	1 0
S5	0: 1:	0 1	0 1	1 0	1 0
S6	0: 1:	0 1	0 1	1 0	1 0
S7	0: 1:	0 1	0 1	1 0	1 0
S8	0: 1:	0 1	1 0	0 1	1 0
S9	0: 1:	0 1	1 0	0 1	1 0
S10	0: 1:	0 1	1 0	1 0	1 0
S11	0: 1:	0 1	1 0	1 0	1 0
S12	0: 1:	0 1	1 0	1 0	0 1
S13	0: 1:	0 1	1 0	1 0	0 1
S14	0: 1:	0 1	1 0	0 1	0 1
S15	0: 1:	0 1	1 0	0 1	0 1



2.2.3 Trellis-Termination

Trellis termination process is to drive the encoder to the all-zero state at the end of the block. In generally, the beginning of state is assumed as all-zero states for constituent encoder.

- *Both encoders terminated with individual tail symbols*

The ending of state, due to employing the MAP algorithm for Turbo decoding, usually is known as all zero state (non-zeros state also can be assume) to perform feedback recursively decoding. Here, a tail bits driven from any probably state (2^q numbers) to any target state no longer than q bits when the recursive convolutional encoder consists of q registers.

Due to the excursiveness property of encoder, the required M tail bits cannot be “predetermined”. Thus, first, we observe the register values relationship with feedback

and input information as

$$\text{Register}_1 = \text{feedback information} \oplus \text{input information} \quad (2.2)$$

$$\text{Register}_2 = \text{Register}_1 \quad (2.3)$$

$$\text{Register}_3 = \text{Register}_2 \quad (2.4)$$

$$\text{Register}_4 = \text{Register}_3 \quad (2.5)$$

where \oplus symbols the modulo-2 addition.

Except from the first register value, the others register value are obtained from the previously register. However, there is no input sequence required to be encoded when performing termination for turbo encoder. Therefore, the simplest obtaining zero value for register is to use previously feedback information for performing self-cancellation. Furthermore, the other register values also obtain zero values by one after another when the first register has been zero value. The whole termination process can be expressed as follows

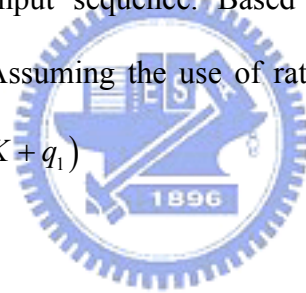
$$\text{Terminated_Register}_1 = \text{feedback information} \oplus \text{feedback information} = '0' \quad (2.6)$$

Compared to the case where none of trellis is terminated, the minimum distance here is increased from terminated bit. However, this trellis-termination method probably yield low minimum distance codeword because both trellis are terminated independently [8]. Assuming the use of rate-1/2 convolutional encoder, the overall code rate is $R_c = K / (3K + 2q_1 + 2q_2)$, where q_1 and q_2 indicate the memory order of first and second constituent encoder, respectively. It is observed that this type of termination is the reduction in code rate, especially for short interleaver.

- *Only first encoder terminated*

A common trellis termination method found in the literature is to terminate ENC1 and to leave ENC2 unterminated. The v_1 tail bits makes that only the ending stage of ENC1 is fed back all zero state after encoding K information symbols. Note that these tail bits are included in the sequence, thus, the interleaver size is $K+q_1$. The interleaved sequence, of length being $K+q_1$, is fed to ENC2 which starts encoding in the all-zero state and is left unterminated in an unknown state.

The minimum distance is guaranteed to be caused by an input sequence of weight greater than or equal to 2. A good spread interleaver, it is unlikely that both nonzero symbols in the un-interleaved input sequence are interleaved to positions near the encoded of the interleaved input sequence. Based on above reasons, most small distances are eliminated [9]. Assuming the use of rate-1/2 convolutional encoder, the overall code rate is $R_c = K / 3(K + q_1)$



2.2.4 Puncturing

Puncturing is the process which removes certain bits from the codeword. The purpose of puncturing is to increase the overall code rate for Turbo code. In general, the common operation of puncturing is to remove the parity check bits from the first and second encoders periodically.

However, a significantly improved puncturing approach has been presented by [10]. This type of puncturing probably could obtain a longer minimum distance if a small number of systematic bits are punctured. It is well known that the minimum distance is caused by input sequence with low input weight. This means that the puncturing systematic bits are increased without or with a small loss in the contribution of systematic part to the overall minimum distance. Further, increasing the number of

puncturing systematic bits means that fewer number of parity check bits are punctured. This results in an improvement in the distance properties because the minimum distance is mainly dominated by the contribution of parity check bits, especially for well designed interrelavers.

2.3 Interleaver

The purpose of the interleaver in turbo codes is to ensure that information patterns that cause low weight words for the first encoder is not interleaved to low-weight patterns for the second encoder, thus improving the code weight spectrum [11].

Consequently, the excellent interleaver is an essential condition for achieving good distance properties. Note that achieving good distance properties require not only the excellent interlever, but also recursive constituent encoders. In this thesis, the interleaver is referred to a vector π . Here $\pi(i)$ is the interleaved position after the information at position i_{th} is interleaved in the nature order. In other hands, $\pi^{-1}(i)$ is defined as de-interleaver, which is a converse operation of the interleaver. This is, the de-interleaver implies that the interleaved order information $\pi(i)$ is conversely interleaved into the nature order at i_{th} position. In other words, Considering the block size N in the original information sequence $\mathbf{u}=(u_0, u_2, \dots, u_{N-1})$ are interleaved into the interleaved information $\mathbf{u}_\pi = (u_{\pi(0)}, u_{\pi(1)}, \dots, u_{\pi(N-1)})$

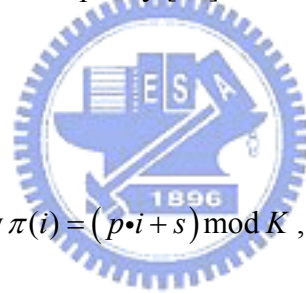
2.3.1 Block Interleaver

A simple structured interleaver is block interleaver, often also called as rectangular interleaver in the literature. It is constructed by a rectangular of M rows by N columns, where the interleaver size is $K=M*N$. This interleaving is performed as follows. From the beginning of upper left corner of rectangular, the data are in turns written into the

rectangular with column by column, and then read the interleaved data with row by row, or vice versa. Further, the block interleaver can be expressed as $\pi(i) = \left(i \cdot N + \left\lfloor \frac{i}{M} \right\rfloor \right) \bmod K$, where $\lfloor x \rfloor$ is the floor function, which means the largest integer of x .

However, the block interleaver is not an excellent interleaver. From the view of codeword weight, this interleaver produces a larger number of long distance codewords caused by input sequences of weight 2 and 3, but yields a large number of low distance codewords caused by that of weight 4. This is, for block interleaver, both the distance properties and error performance constrained by the input sequences of weight 4, leading to no significantly improving BER capacity [12].

2.3.2 Prime Interleaver



The permutation is defined by $\pi(i) = (p \cdot i + s) \bmod K$, where s, p are known as offset and step size, respectively. Note that the value of p must be chosen relatively prime to block size K , ensuring that the element in the interleaver differ from each other. This interleaver is also referred as circular-shifting interleaver in the literature.

For the view of distance properties, this interleaver can permute the low distance codeword for the first recursive encoder into the high distance codeword for the other recursive encoder. However, this type of interleaver is less likely to permute an input sequence of weight higher than 2 with low codeword weights into another input sequence with high codeword weights.

2.3.3 Random Interleaver

Random interleaver is generated by a random manner without any restriction on the selected element. This interleaver is also referred as pseudo-random interleaver in the literature. Modified random interleaver with some useful criterion is likely to achieve better performance. Usually, the performance of this type is significantly increased as the block size increases.

2.3.4 S - Interleaver

The S interleaver is one of spread interleavers. Usually, the codeword of minimum distance are contributed by the input pattern with low weight. The goal of these types is to spread the low weight input patterns, generating higher weight codewords. Here, the

Spread factor S is usually chosen less than or equal to $\sqrt{\frac{K}{2}}$. This interleaver can be described as follows. Select a random element from the selected set $\{0,1,\dots,K-1\}$ as the first element in the interleaver and delete it from the set. Then, each subsequent elements are moved from the selected set if current candidate position is selected within $\pm S$ range compared with the previous selected element. Otherwise, current candidate is rejected until the selection criterion is satisfied. Repeat this process until all K integers are selected.

This interleaver can achieve better performance than average to generate higher weight codewords. Unfortunately, the search time increases with the designed amount of separation, S, and the interleaver length K. Another drawback is that there is no guarantee that the search process will finish successfully. Further, another design criterion based on the constituent encoders adding into S interleaver is presented in [13] [14]. This goal of modified S interleaver is to eliminate low-weight codewords with

significant contributions to the error performance. In general, the elimination of a specific codeword can be done by breaking up the input pattern. This modified S interleaver, however, is no guarantee to eliminate all low-weight codewords and find a properly solution.

Usually, the random-like intelreaver structure, their performance degradation is significantly sharp than that of structure interleaver, such as the prime interleaver, in the highly codeword puncturing.

2.3.5 Characteristic of Interelaver

Minimum distance of the interleaver algorithm is a major factor which affects the error floor as defined as duo-distance between position i and j for a given interleaver:

$$d_{duo}(i, j) = |i - j| + |\pi(i) - \pi(j)| \quad (2.7)$$

where $\pi(i)$, $\pi(j)$ are “interleaved positions” of i, j , where $i, j = 0, 1, 2, \dots, K-1$ (K is the number of the interleaver block), and $i \neq j$ [15].

A better interleaver algorithm design usually should have three characteristics as the following:

- The d_{duo} should be as large as possible in order to “lower correlation” between input sequences and interleaver output sequences.
- The distances between any two input information bits before and after the interleaver, denoted $d(i, j) = |i - j|$ and $d(\pi(i) - \pi(j))$, $i, j = 0, 1, 2, \dots, N-1$ should not be multiple of the intrinsic period to avoid the change of the feeding self-terminating weight-2, where the intrinsic period are $2^v - 1$, if the memory of the RSC encoder is v . Due to the intrinsic period has a significant effect on the performance of turbo code, a better generator function of Turbo

encoder usually chooses an appropriate primitive polynomial (of degree v) as the feedback polynomial $g_0(D)$.

- The positions of any input information bit before and after interleaver, i.e., i and $\pi(i)$ ($0 \leq i \leq K-1$), should not be both near the end of the interleaver block in order to avoid edge effects. This is, if i is nearly K , then both $\pi(i)$ and $\pi^{-1}(i)$ should be much smaller than K .

One of Interleaver designs that is optimum in the sense of breaking up the weigh-2 input sequences was introduced in [16]. However, it is also noted in [16] that braking up only the weight-2 input sequences is not sufficient to achieve good distance properties. This is because input sequences of weigh higher than 2 are not broken up and can still lead to low codeword weights.

For achieving good distance properties, this suggests an additional design criterion based on the correlation between the extrinsic information. This is, an interleaver with good properties is designed to minimize correlation between the extrinsic information of constituent decoder and input sequence [17].

2.4 Channel Model

It is known channel models which could primary be divided into three types. First, AWGN is common non-fading channel model to simulate pure Gaussian noise, including thermal noise, uncertain effects, and so on. Second, the second type of channel model is defined static fading channel model, including (Line of Sign) LOS and NLOS types based on the signal propagation circumvent between transmitter and receiver. Finally, this channel model primary simulates the Doppler-effects and attenuation of fading mobile channel model, which can be simulated by Jake's Model

with different velocity requirement. AWGN channel model is primary discussed for Turbo code in this thesis.

- **AWGN (Additive White Gaussian Noise) Channel Model**

The power spectral density is independent of the operating frequency. The adjective white is used in the sense that light contains equal amounts of all frequencies within the visible band. We express the power spectral density of white noise, with a sample function denoted by $w(t)$, as $S_w(f) = \frac{N_0}{2}$. The parameter N_0 is usually referred to the input stage of the receiver of a communication system, expressing as $N_0 = kT_e$ where k is Boltzmann's constant and T_e is the equivalent noise temperature of the receiver.

Since the autocorrelation function is the inverse Fourier transform of the power spectral density, the autocorrelation function can be expressed as $R_w(\tau) = \frac{N_0}{2} \delta(\tau)$. This is, the autocorrelation function of white noise consists of a delta function weighted by the factor $N_0/2$ and occurring at $\tau = 0$.

2.5 Turbo Decoder Process

Fig. 2-5 shows that the Turbo decoder process employs two SISO decoders to estimate a posterior probability (APP) of each information u_k with a certain numbers of iterative computations such that the results have no significant BER performance loss. The Turbo decoding process is states as follows:

- (a) *Initialized phase:* the received signal codeword has to be stored into symmetrical buffer and parity check buffer due to iteratively decoding process. After the total N received data have been stored, the decoding

process starts to carry out iteratively *MAP* decoding, where the intrinsic information and iteration number *Iter.* are initialized as zero.

- (b) *1st half iteration phase*: the input codeword $(y_s, y_{p-siso1})$ from input buffer are sent into *SISO* decoder and then proceeds to *MAP* algorithm decoding, producing the extrinsic information which is written into extrinsic storage with natural order after a decoding latency. When the whole extrinsic information has been calculated, the stored extrinsic information with interleaving order are inputted as intrinsic information L_{i2} (*Interleaver operation*) for the 2^{nd} half iteration phase. The decoding process then jumps to the next *phase (c)*.
- (c) *2nd half iteration phase*: combining the interleaving order of systematic information u_{int} , parity check $y_{p-siso2}$ and L_{i2} are carried out the extrinsic information L_{ex2} . This outputted soft information L_{ex} are firstly stored by interleaving order into extrinsic storage and used by natural-order as intrinsic information L_{i1} for the *1st half iteration phase (De-interleaver operation)* as the 2^{nd} half iteration phase has finished. If the *Iter.* parameter is equal to the specified max-iteration, the decoding process phase jumps to *(d)*; otherwise, $Iter. = Iter. + 1$ and the decoding process returns into *phase (b)*.
- (d) *Output the estimated information u_k' phase*: the log-likelihood-ratio (*LLR*) information proceeds to the *De-interlaver operation*, and then obtains the estimated information u_k' through hard decision device. The hard decision operation is that if the sign of *LLR* is positive, the information u_k' are decided as *1*; otherwise, the information u_k' are decided as *0*.

Based on 1^{st} and 2^{nd} decoding phase, the operation of *SISO* decoder is identical and

extrinsic storage is performed through interleaver/de-interleaver procedure. Therefore, the above SISO decoder, interleaver procedure can be implemented by the same hardware for twice half-iteration.

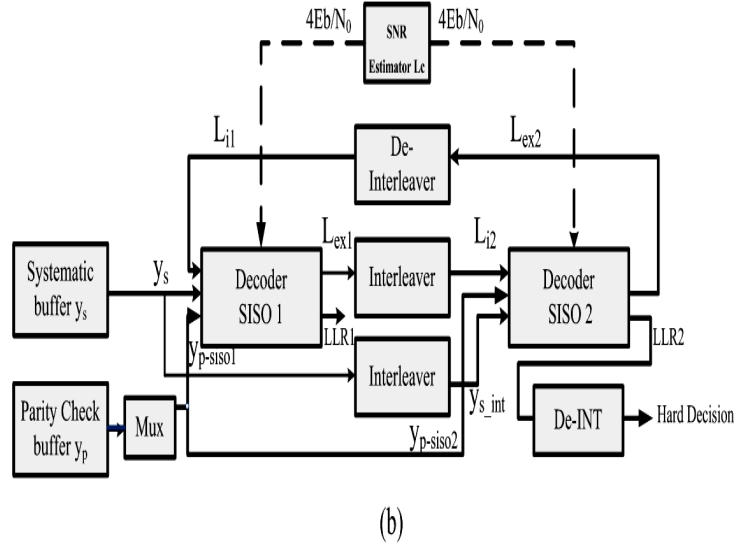


Fig. 2-5 Turbo decoder diagram

2.6 SISO Decoding Algorithm

2.6.1 Log-MAP Algorithm

The Turbo decoder iteratively decodes the parallel concatenated convolutional codes through log-MAP algorithm which decides the LLR of APP of each information bit u_k [2]. The MAP algorithm is based on the log-likelihood ratio a posterior defined as

$$L(\widehat{X}_k) = \ln \frac{\Pr(u_k = 1 | y)}{\Pr(u_k = 0 | y)} \quad (2.8)$$

where u_k are the source information bits.

The APP ratio $L(u_k)$ can be further represented in three terms:

$$L(\widehat{X}_k) = L_i(u_k) + L_c \cdot R_s + L_{ex}(u_k) \quad (2.9)$$

where $L_c \triangleq 4E_b / N_0$ and $L_c \cdot R_s$ are defined as the channel values. After interleaving or deinterleaving, the intrinsic information is calculated from the extrinsic information of

the other constituent decoder, as shown in Fig. 2-5. This means $L_{i1}(u_k) = L_{ex2}(u_k)$ and $L_{i2}(u_k) = L_{ex1}(u_k)$.

The arithmetic operations of the log-MAP are described as follows. For each trellis transitions leaving the state $k-1$ toward the state k , the branch metric value is formulated as:

$$\gamma_k(S_{k-1}, S_k) = \frac{1}{2} \left\{ x_k^s (L_i + L_c \cdot y_k^s) + \sum_i L_c \cdot y_k^p \cdot x_k^p \right\} \quad (2.10)$$

where (x_k^s, x_k^p) denotes the transmitted symmetrical and parity check bits, which takes values in $\{1, -1\}$. (y_k^s, y_k^p) represents the received symmetrical and parity check bits at the k -th time instant. At step k , for each trellis state S_k beginning from previous state S_{k-1} , the state metric can be calculated as:

$$\alpha_k(S_k) = \max_{S_{k-1}, S_k}^* (\alpha_{k-1}(S_{k-1}) + \gamma_k(S_{k-1}, S_k)) \quad (2.11)$$

On the other hand, on the step k , for each trellis state S_k beginning from the current state S_{k+1} , the backward metric calculation is:

$$\beta_k(S_k) = \max_{S_{k+1}, S_k}^* (\beta_{k+1}(S_{k+1}) + \gamma_{k+1}(S_k, S_{k+1})) \quad (2.12)$$

When Forward and Backward state metric are calculated, the APP ratio $L(u_k)$ can be re-written into:

$$L(\widehat{X}_k) = \max_{(S_k, S_{k-1}), u_k=1}^* (\alpha_{k-1}(S_{k-1}) + \gamma_k(S_{k-1}, S_k) + \beta_k(S_k)) - \max_{(S_k, S_{k-1}), u_k=0}^* (\alpha_{k-1}(S_{k-1}) + \gamma_k(S_{k-1}, S_k) + \beta_k(S_k)) \quad (2.13)$$

Here the definition of Max* function is:

$$\begin{aligned} \max^*(x, y) &= \ln(e^x + e^y) \\ &= \max(x, y) + \ln(1 + \exp^{-|x-y|}) \end{aligned} \quad (2.14)$$

where the corrective term $\ln(1 + \exp^{-|x-y|})$ can be implemented by a look-up table (LUT).

2.6.2 Max-Log-MAP Algorithm

The Max-Log-MAP is deduced from the Log-MAP decoder by substituting each \max^* -operation by a max-operation and shown in the following:

$$\text{Max}^*(x, y) = \ln(e^x + e^y) = \max(x, y) \quad (2.15)$$

Then, the correction function $\ln(1+e^{-|y-x|})$ in the $\max^*(.)$ operation are neglected in the Max-MAP decoder, which has less complexity due to eliminating the need of LUT unit. The correction term plays the important role of improving the capacity of correcting error code when operated in the low-signal to noise ratio environment, due to the difference is usually small.

Base on previous reason, the performance degradation is about 0.58dB compared to the Log-MAP algorithm [18]. However, the correction term worked in the high signal-to-noise-ratio environment always approximates as zero, since the difference has a more probability exceeding two. Another benefit for Max-Log-MAP is that Turbo-decoding does not require knowledge of the SNR [19].

2.6.3 Initialized Procedure for Both Log-MAP and Max-Log-MAP Algorithm

Initialization the state probability for α_0 and β_0 must be initialized as follows:

$$\begin{aligned} \alpha_0(S_0) &= 0 \\ \alpha_0(S_k) &= -\infty, \quad k \neq 0 \end{aligned} \quad (2.16)$$

$$\begin{aligned} \beta_N(S_0) &= 0 \\ \beta_N(S_k) &= -\infty, \quad k \neq 0 \end{aligned} \quad (2.17)$$

Except in the case for the decoder associated with the second encoder, where the trellis is simply left 'open' as follows

$$\begin{aligned}
\alpha_0(S_0) &= 0 \\
\alpha_0(S_k) &= -\infty, \quad k \neq 0 \\
\beta_N(S_k) &= \alpha_N(S_k)
\end{aligned} \tag{2.18}$$

In the second case the backward recursion uses the value of the state probabilities generated by the last forward recursion step.

2.7 Error Probability for Turbo Code

For a (N,K) Turbo code, the symbol error rate of Turbo code is bound by the union bound [1]:

$$P_w \leq \sum_{j=1}^{M-1} Q\left(\frac{2\sqrt{Ed^H(c_0, c_j)}}{\sqrt{2N_0}}\right) \tag{2.19}$$

where N indicates the interleaver length, K is the number of information bits and M is number of binary codeword. Here, $Q(x) = \int_x^{\infty} \frac{e^{-t^2}}{\sqrt{2\pi}} dt$ and $d^H(c_0, c_j)$ is codeword

Hamming distance. Furthermore, the function of Q(x) asymptotically approaches $e^{-\frac{x^2}{2}}$ as x approaches infinite. Therefore, the symbol error probability is upper bound by the sum of (M-1) exponentials. When the Hamming distance $d^H(c_0, c_j)$ grows, the error probability decays exponentially such that the minimum Hamming distance dominates the asymptotic symbol error probability, as shown follows:

$$P_w \approx A_{\min} Q\left(2d_{\min}^H \frac{E}{N_0}\right) \tag{2.20}$$

where A_{\min} is the number of codeword at d_{\min}^H . It follows that the asymptotic bit error probability is:

$$P_w \approx \frac{W_{\min}}{K} Q\left(2d_{\min}^H \frac{E}{N_0}\right) \quad (2.21)$$

where the W_{\min} is the number of input sequence causing d_{\min}^H . and $E = R_c \cdot E_{\text{non-encoded}}$ is the energy per code bit, resulting from the code rate R_c and the energy per encoded bit $E_{\text{non-encoded}}$. From the view of error probability, a turbo code achieves the better improvement performance, requiring more number of symbols to cause the minimum Hamming distance as large as possible.

2.8 Turbo Code Application on Telemetry and Deep Space Communications

In CCSDS recommendation for telemetry channel coding, the CCSDS encoder scheme could be seen two components, where input information of length k bits is held in a frame buffer, and then the bits in the buffer are read out in two orders of the two component encoders. The upper component encoder operates on the bits in un-permuted order (“*in a*”), while the second component encoder receives the same bits permuted by the Interleaver block (“*in b*”) as shown in Fig. 2-6. The nominal code rates of turbo codes are $1/2$, $1/3$, $1/4$, and $1/6$. In addition, the frame sizes of CCSDS turbo codes are *1784, 3568, 7136 and 8920*.

For turbo encoder termination operation, there are “four” terminal bits used to clear all the delay elements of the RSC encoders after the input information k bits have been delivered into the encoders, Thus, the actual code rate of the turbo encoder scheme is $n/n(k+4)$. The interleaving pattern is a fixed sequence, which is on a bit-by-bit level of the entire block of data, unlike the Reed-Solomon interleaving on a symbol-by-symbol level. The interleaver algorithm is described by the following algorithm in Table 2-2. (as

excerpted from [5]):

First express k as $k=k_1k_2$, where k_1 is eight. Next do the following operations for $s=1$ to k to obtain permutation numbers $\pi(s)$. In the equation below, $\lfloor x \rfloor$ denotes the largest integer less than or equal to x , and p_q denotes one of the following eight prime integers:

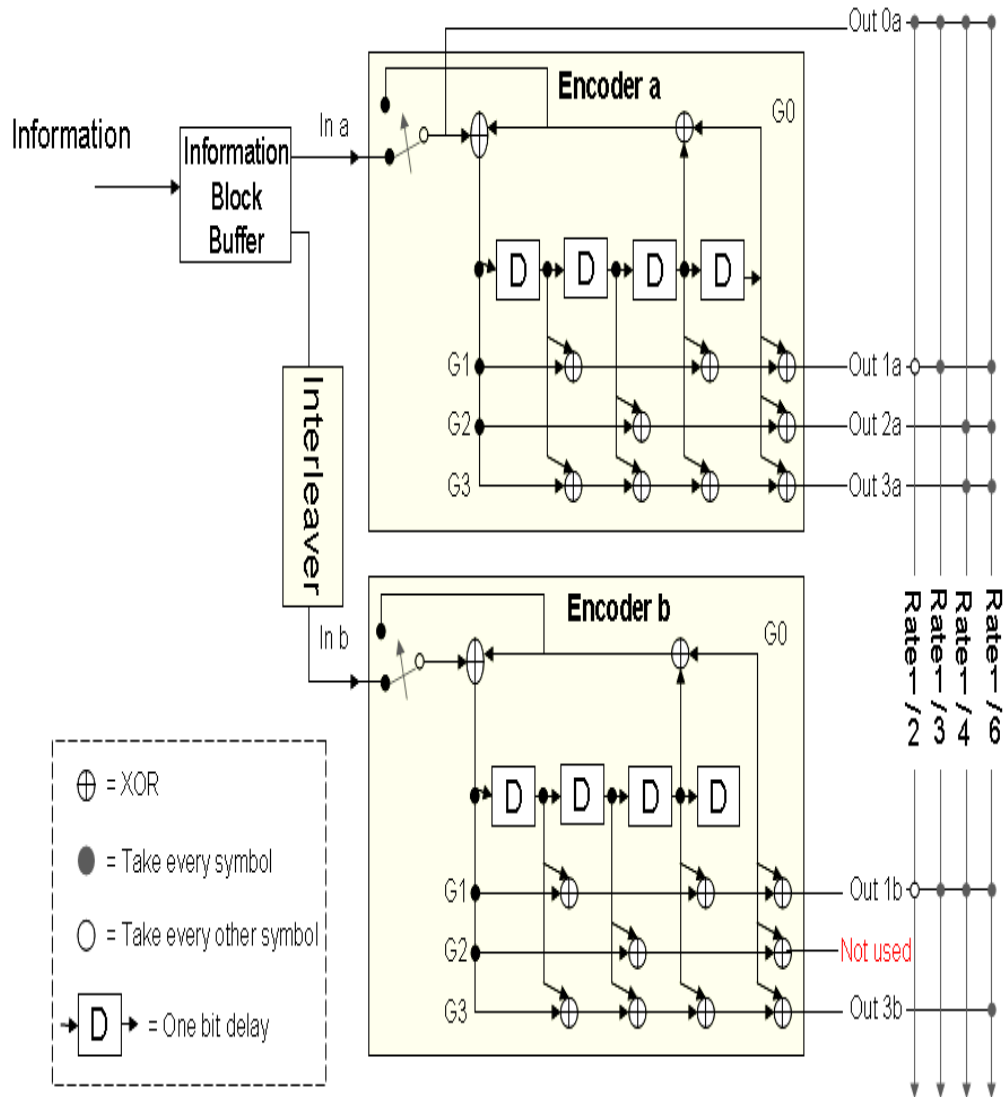


Fig. 2-6 CCSDS Turbo encoder

Table 2-2 Interleaver algorithm for the Turbo code of CCSDS standard

$$\begin{aligned}
 & p_1=31; p_2=37; p_3=43; p_4=47; p_5=53; p_6=59; p_7=61; p_8=67; \\
 & N_{\text{turbo}} = k_1 \times k_2 \\
 & k_1 = 8 \\
 & p_1 = 31 \quad p_2 = 37 \quad p_3 = 43 \quad p_4 = 47 \\
 & p_5 = 53 \quad p_6 = 59 \quad p_7 = 61 \quad p_8 = 67 \\
 & m = (s - 1) \bmod 2 \\
 & i = \left\lfloor \frac{s-1}{2k_2} \right\rfloor \\
 & j = \left\lfloor \frac{s-1}{2} \right\rfloor - ik_2 \\
 & t = (19i + 1) \bmod \frac{k_1}{2} \\
 & q = t \bmod 8 + 1 \\
 & c = (p_q j + 21m) \bmod k_2 \\
 & \pi(s) = 2(t + c \frac{k_1}{2} + 1) - m = 2(q + 4c) - m
 \end{aligned}$$

The turbo encoder codeblock outputs for various code rates are shown in Fig. 2-7. For each input to the delay elements, n symbols are outputted. The output sequence is a particularly periodic sampling from top to bottom of the outputs of Turbo decoder. (e.g., for code rate $1/3$, the output sequence is Out_0a, Out_1a, Out_1b).

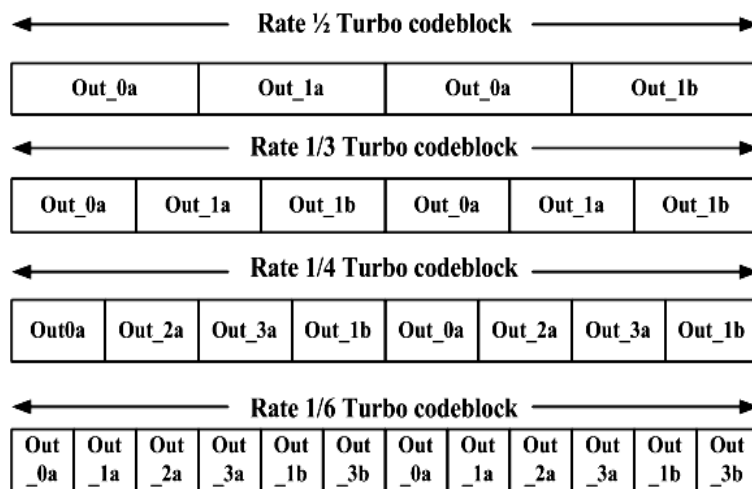


Fig. 2-7 Turbo code for different code rates

2.9 Simulation and Results

The generator polynomial of Turbo code defined as $(23, 33, 25, 37)_{oct}$, and interleaver lengths of 1784, 3568, 7136 have been simulated in Matlab[®] using Moto-Carlo method, where the *oct* expresses an abbreviation of octal. These different simulation conditions are performed as follows:

- For different turbo decoding algorithm: Max-Log-MAP versus Log-MAP
- For various block size (1784,3568,7136)
- For different number of iteration decoding
- For various code rate

The Fig. 2-8 shows the performance results of Log-MAP and Max-Log-MAP after 10-iterations for SNR value varying from 0.4dB to 0.8dB. Note that the performance of MAX-Log-MAP algorithm has larger degradation than that of Log-MAP algorithm over the low-SNR region. For low-SNR region, the corrected term is helpful for improving the turbo decoding in the Log-MAP algorithm.

The Fig. 2-9 shows the performance comparison of turbo code when Log-MAP algorithm is performed using different block sizes (1784, 3568, 7136) after 10 iterations. It is observed that the larger block size has more decoding gain relative to other smaller block sizes. This is due to the larger block size has a long free hamming distance d_{free} , and thus has more better capable of error correction capacity.

The Fig. 2-10 shows the performance results of turbo code when Log-MAP algorithm uses different number of iterations (3,5,7,9) at 1784 block size. Note that the performance results have more significantly performance improving in large iterations. This is to say, the Turbo code has to pass through highly iterative decoding process to meet the desirable performance result.

As far as the various code rate of turbo code concerning, the performance results of turbo code are shown in Fig. 2-11. The performance results shows that one codeword including more redundancy elements, such as code rate (1/6), has a greater coding gain relative to less redundancy elements (1/3). This is because more redundancy elements is helpful for correcting error codeword.

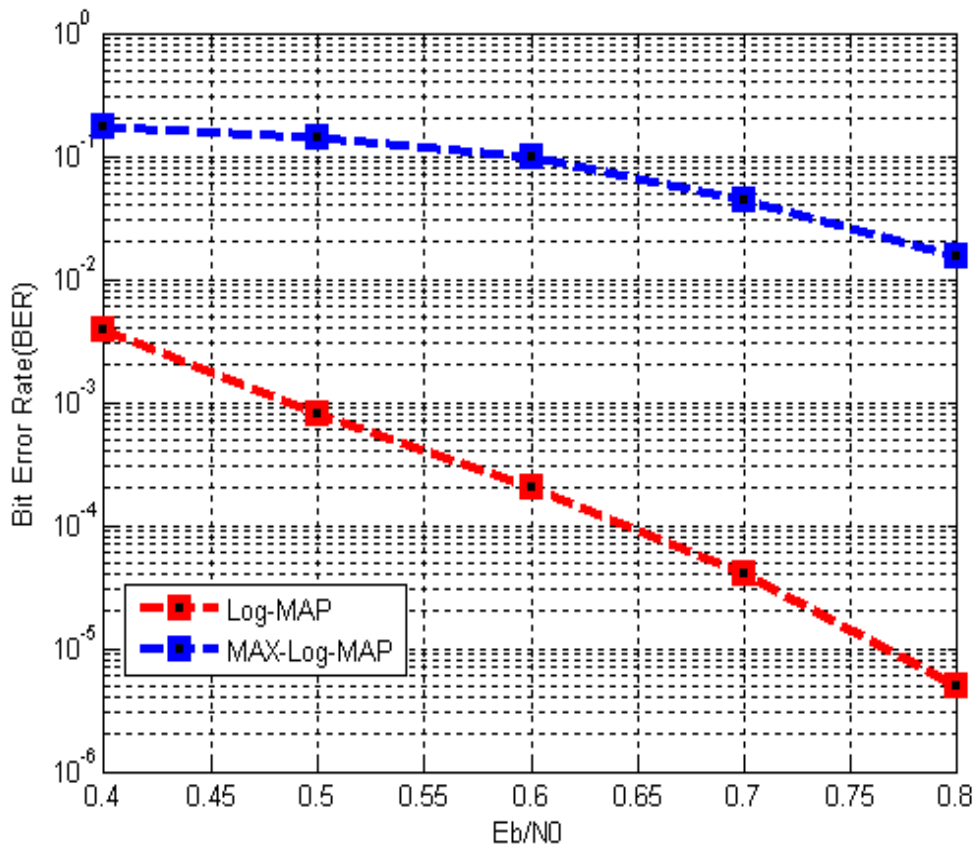


Fig. 2-8 Performance results of Turbo code for different turbo decoding algorithm

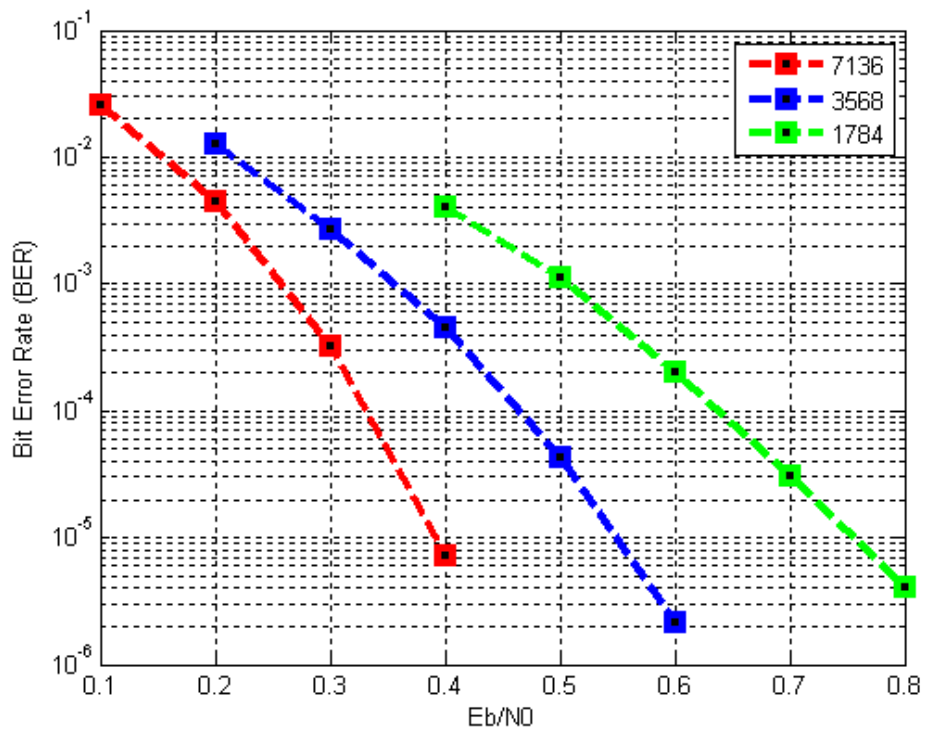


Fig. 2-9 Performance results of Turbo code for different block sizes

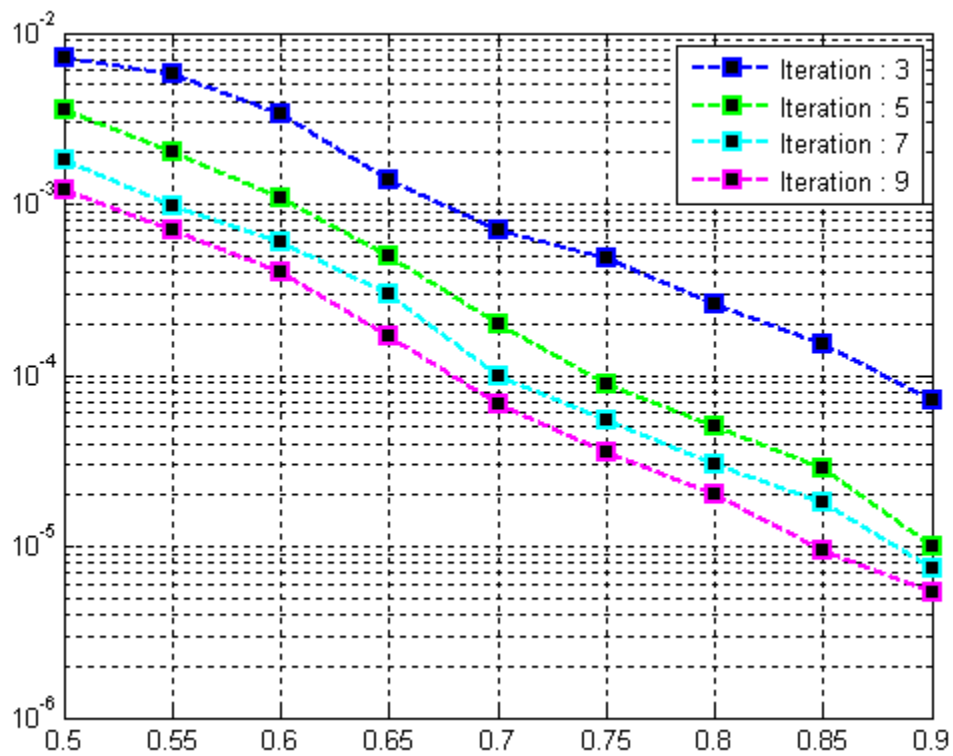


Fig. 2-10 Performance results of Turbo code for different iterations

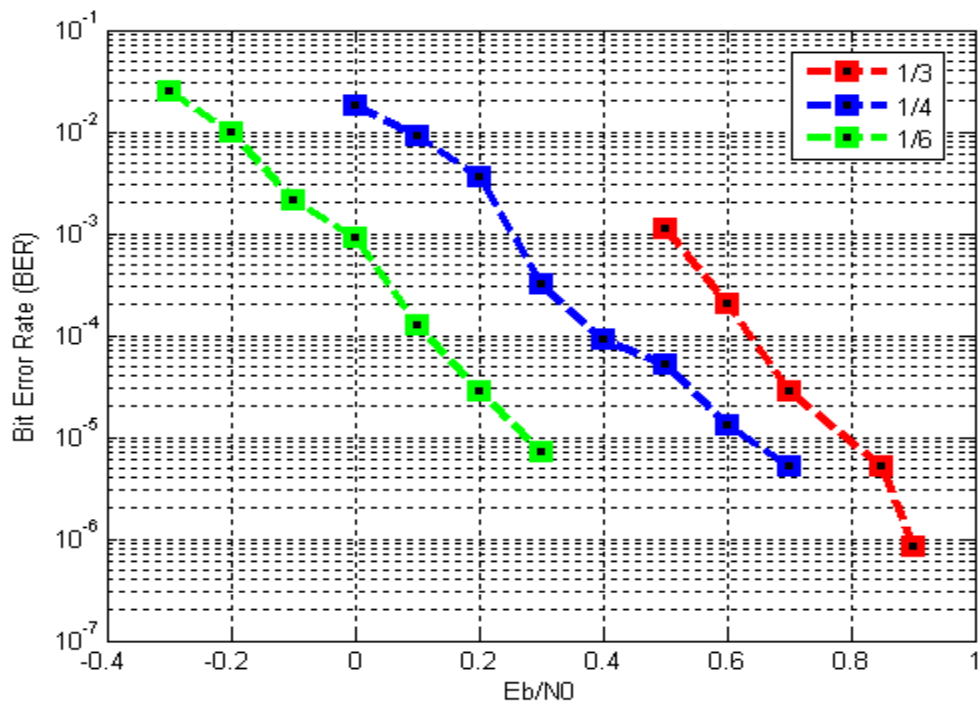


Fig. 2-11 Performance results of Turbo code for different code rate



Chapter 3 VLSI Architecture of Turbo Decoder

In this chapter, we will state the VLSI architecture of Turbo decoder. First, the sliding window approach divides large frame size N into equal-sized L sub-blocks to reduce the long decoding latency and storage memory. Then, the concurrent efficient SISO VLSI architecture is introduced, including OACS which is faster than original ACSO architecture, modified sliding window which is more suitable to high code rate (larger redundancy elements).

Next, for the Turbo code of CCSDS standard, the on-line fly interleaver address is implemented to save more memory area requirement relative to the *ROM* approach. And the extrinsic information quantization is used to reduce the bit-width of original extrinsic information. Finally, we discuss the throughput of series SISO decoder structure and parallel SISOs decoder. The whole Turbo decoder architecture is shown in Fig. 3-1.

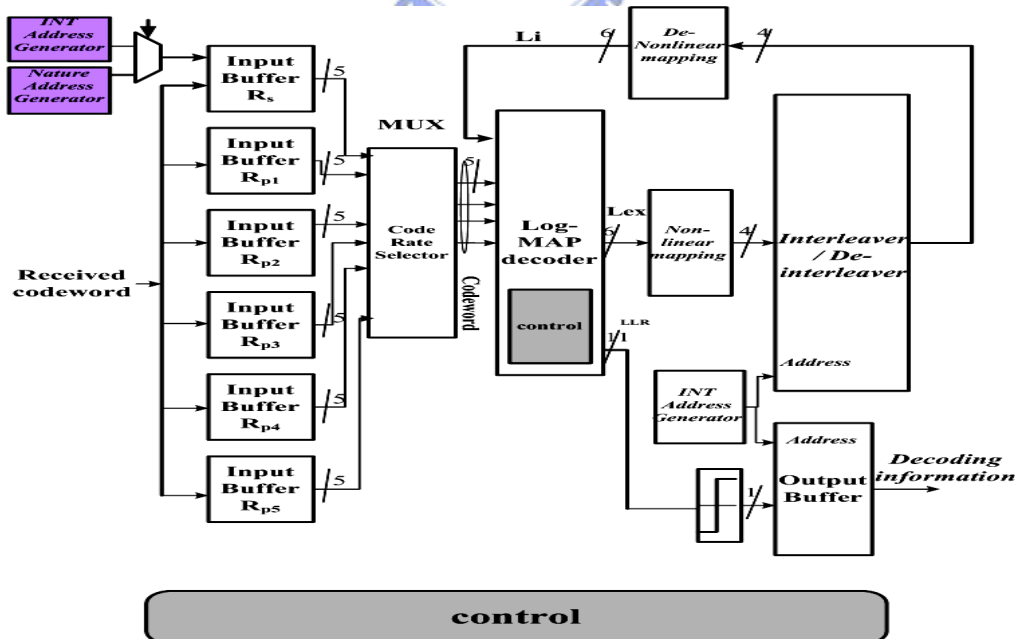


Fig. 3-1 Turbo decoder architecture

3.1 Sliding Window Approach

According to MAP algorithm, the decoder needs to store a large amount of memory and causes a long decoding latency during the forward and backward recursion calculations. The sliding window technique was introduced to reduce decoding latency and storage memory by dividing large frame size N into equal-sized L sub-blocks [20], where L is typically 5 to 10 times the constraint length (encoder memory order + 1). The appropriate warm up length L is determined by trading off the performance degradation and the relative sliding window memories size.

For SW approach, it is necessary to obtain a reliable initial value of state metric with a warm up phase, which recursively computes the state metrics from the previous L stages of estimated reliable point with all zeros values. In general, the initial alpha values are the last value of the previous window, and the initial beta values are obtained by the dummy beta calculation unit, which performs a warm up process with the same structure as beta processor.

Fig. 3-2 shows how sliding window approach calculation is done among Forward Processor (FP), Dummy Beta Processor (DBP) and Beta Processor (BP) working together, as stated as follows:

- (a) In time slot T_0 : the DB processor accesses the input buffer and recursively backward computes the SM values with all zeros values from data $L-1$ to 0 . On the other hand, the received input buffer data are stored in sliding window memory-bank₁.
- (b) In time slot T_1 : the valid initial value of backward state metrics are obtained by the DB processor calculating from data $2L-1$ to L , and then the received input data are stored into SW memory-bank₂. The FP processor accesses the

memory-bank₁ data ($L-1 \sim 0$) to calculate forward metrics values with an initial state. Because the soft output LLR calculation requires both the forward and backward state metrics, the output of FP has to be stored in forward SM storage due to the corresponding backward metrics being unready.

- (c) In time slot $T2$: the operation DB process for data ($3L-1$ to $2L$) and FP processor follows a similar procedure as in (b). The BP uses a valid initial value of SM from the last value of previous DP to compute backward SM for the data ($L-1 \sim 0$), and then combines with the associate forward state metrics from forward SM storage to decode soft output information.
- (d) Repeat (a)~(c) until the whole soft output information is obtained.

The input buffer can be built by multi-bank two port memory, whose storage size is defined by the word-length of total input soft information multiplied by memory-depth ($L+1$), where depth $L+1$ can allow contention free in practice. The latency of this sliding window structure can be approximate as $(2*L + C)$ cycles while throughput of the MAP-based SISO decoder is defined by the number of bits processed, N , divided by the latency L cycles, where C is the pipeline delay of SISO decoder.

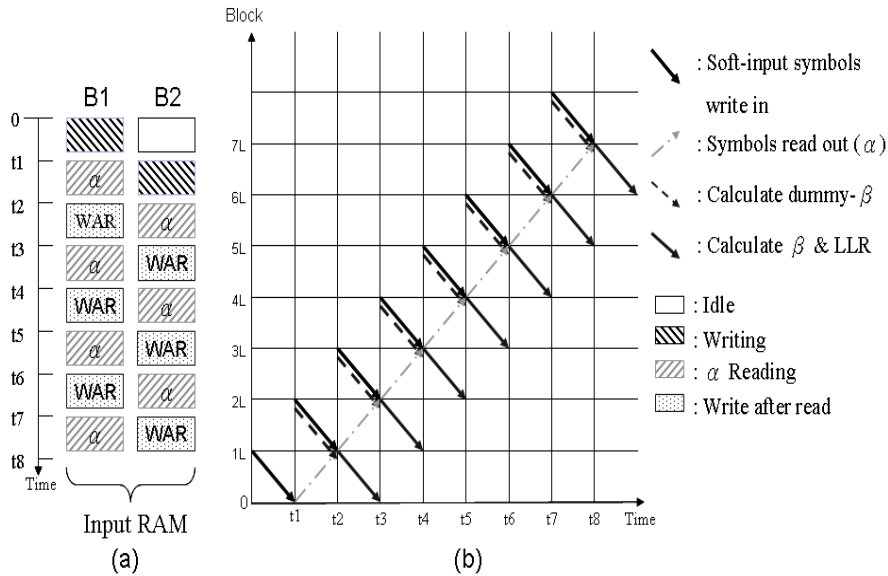


Fig. 3-2 The operation of sliding window approach

3.2 VLSI Architecture of SISO Decoder for CCSDS Standard

This Turbo decoder is composed of one log-MAP decoder, input buffer memory, extrinsic memory and interleaver address calculation unit. The log-MAP decoder is used to decide the LLR of each information bit u_k by the input a prior information. The input codeword sequential is stored in the input buffer to be used by iteratively decoding process. The extrinsic memory stores the extrinsic information from the output of log-MAP decoder, and then sends extrinsic information by an interleaving / de-interleaving order into the input of log-MAP decoder when all extrinsic values have been received. The interleaver / de-interleaver pattern can be calculated by an on-line interleaver address calculation unit, which can reduce significant area requirement relative to storing all interleaver values at the ROM table.

3.2.1 Log-MAP Decoder

(1) Branch Metric Calculation (BMC) Unit

The branch metric values are applied by computing the state metric values, LLR values and extrinsic values. When the input soft information is stored into the extrinsic and input buffer and the L_c value has been obtained by the SNR estimator [21]. The branch metric values are calculated according to (2.10). Due to different code rate requirement, to calculate branch metric values to send to the next state metric calculation unit, the maximum number of input sequences must be considered. The branch metric calculation unit is shown in Fig. 3-3, where the ‘Sat.’ means the saturation device that is used to avoid the overflow when the high SNR environment leading to L_c value become considerable large.

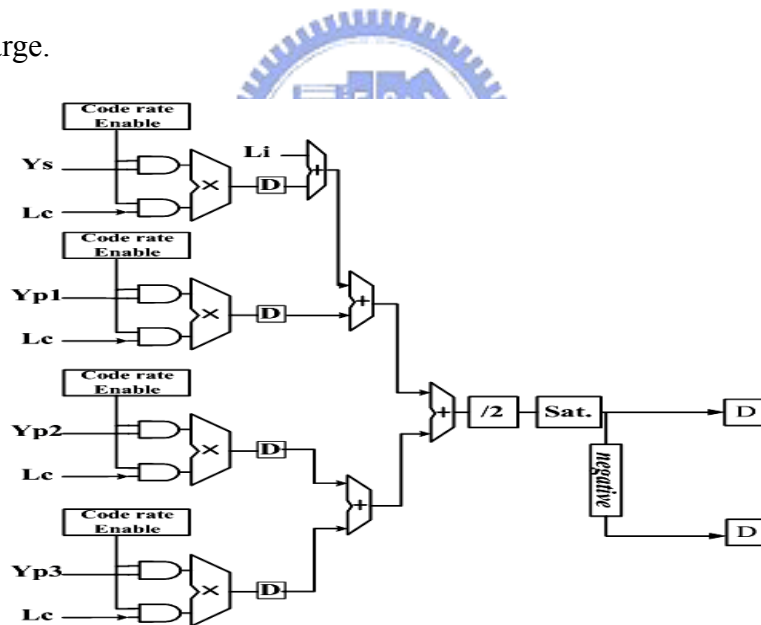


Fig. 3-3 Block diagram of the branch metric calculation

(2) State Metric Calculation (SMC) Unit

The forward and backward metric calculations are calculated according to (2.11) and (2.12). In the trellis expression Fig.3.4 (a), the updated forward state values $\alpha(S_{0,k-1})$ comes from the maximum summation value of previous forward state values with

relative branch values with a correcting term $\ln(1+\exp(-|\alpha(S_{0,k-1}) - \alpha(S_{1,k-1})|))$, which can be implemented by ACSO architecture, as shown in Fig. 3.5(a). In a similar way, the backward recursive calculating by the trellis expression are shown in Fig. 3.4(b). In order to avoid updated value overflow occurring, the state metric calculation usually uses the rescaling method [21] as the state metric value increase with the recursive calculation times. Because the speed of Turbo decoder is determined by the maximum clock rate achieved for ACS architecture, we employ the OACS architecture which uses retiming transformation of the ACS architecture to increase the clock rate as much as possible [22], as shown in Fig.3.5(b).

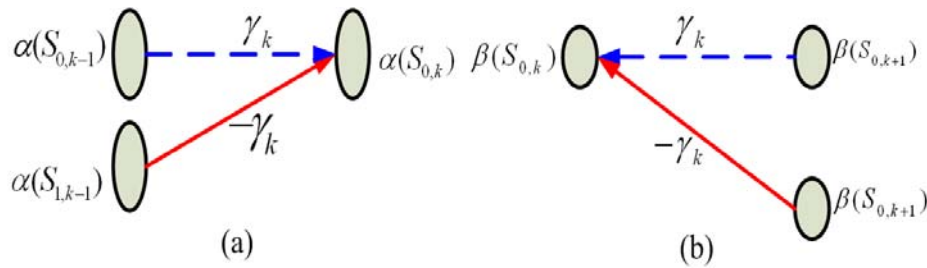


Fig. 3-4 The forward and backward recursive calculating by the trellis expression

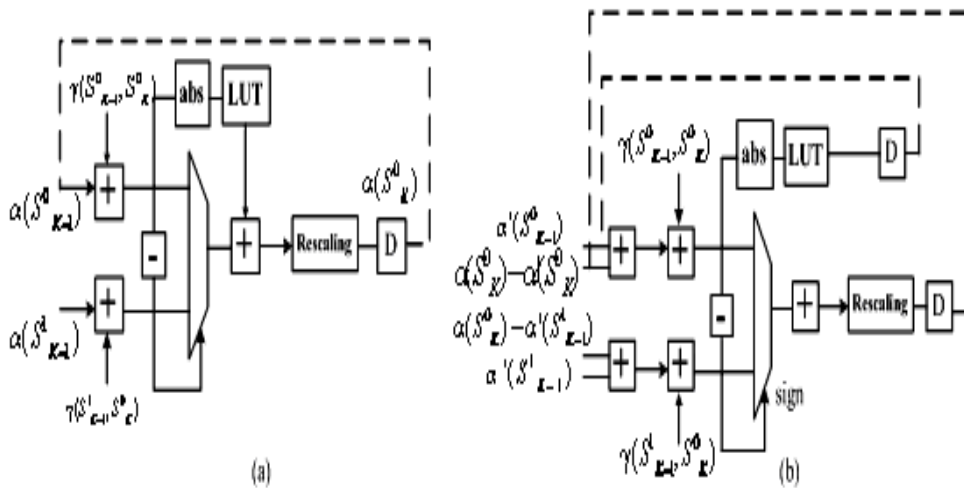


Fig. 3-5 Block diagram of the ACSO and OACS architecture

(3) Log-Likelihood Ratio Calculation (LLRC) unit

According to (2.13), the LLR value could be calculated by the difference between N -input \max^* function for $u_k = 1$ and N -input \max^* function for $u_k = 0$, where N denotes the total state metric values numbers. However, The N -input \max^* function can be transformed into the number of $\log_2(N)+1$ for parallel tree 2-input \max^* function. For example, when the N is *four*, the LLR architecture can be shown in Fig.3.6 due to the relationship of $\text{Max}^*(A,B,C,D) = \text{Max}^*(\text{Max}^*(A,B), \text{Max}^*(C,D))$.

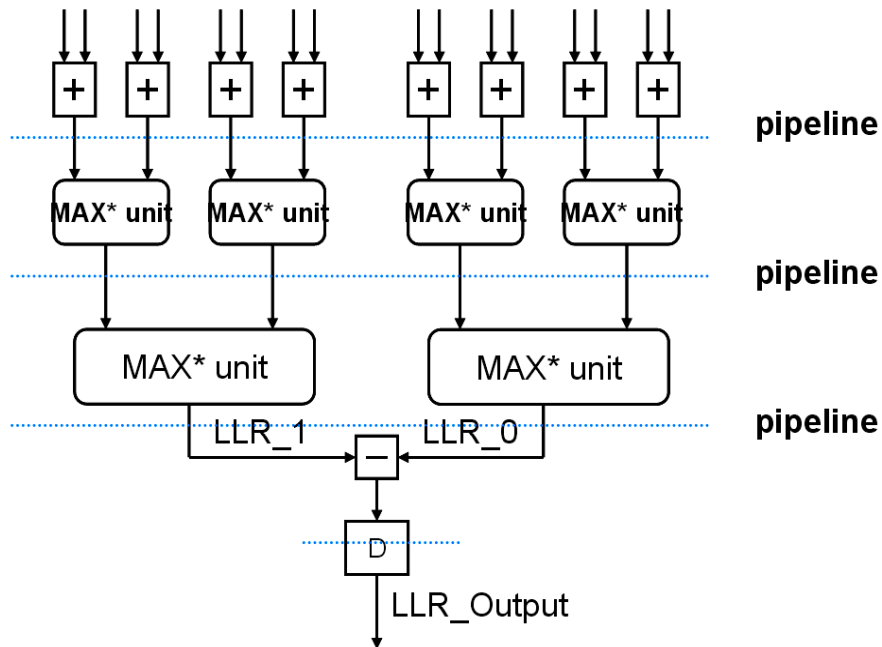


Fig. 3-6 Block diagram of LLR calculation architecture

(4) Modified Sliding Window Scheme

In the traditional log-MAP decoder architecture, the branch metric calculation unit is usually placed in front of the sliding window memories. Thus, the output branch of metric values needs to be stored into sliding window memories, which can then be read when calculating state metric. Further, we attempt to modify the placement of branch

metric for log-MAP decoder architecture such that the storing sliding memory size is less than that of traditional log-MAP decoder architecture.

The Fig. 3-7 shows the branch metric calculation block are placed in back of sliding window memory, in which results in the received codeword (Y_s, Y_p) and intrinsic information being stored, rather than the output of branch metric calculation. Although there are 16 sets branch metric value calculated for 16 states Turbo decoder, we can only compute the maximum independent sets of branch metric value to obtain all sets, as shown in Table 3-1. Note that the minimum linear combination branch metric sets sharply increase as the code rate of Turbo decoder increases from 1/2 to 1/6. This result causes the larger memory storage requirement for traditional architecture than that of modified architecture. The major reason is that the minimum independent branch metric sets [23] increase at a factor of 2 when the code rate (r) goes from 1/2 to 1/6.

The Table 3-2 lists the sliding window memories storage comparison of our modified sliding window scheme and traditional SW scheme and Fig. 3.8 shows that our modified sliding window architecture can reduce 53.13% sliding window memory area overhead comprised with of the traditional sliding window architecture for code rate ($r = 1/6$). Although the storing bit requirement of sliding window memories for modified method are larger (25%) than of traditional SW architecture, this reconfigurable Turbo decoder has to support various code rate. Thus, it is obviously that the modified architecture is suitable to be applied for reconfigurable Turbo decoder.

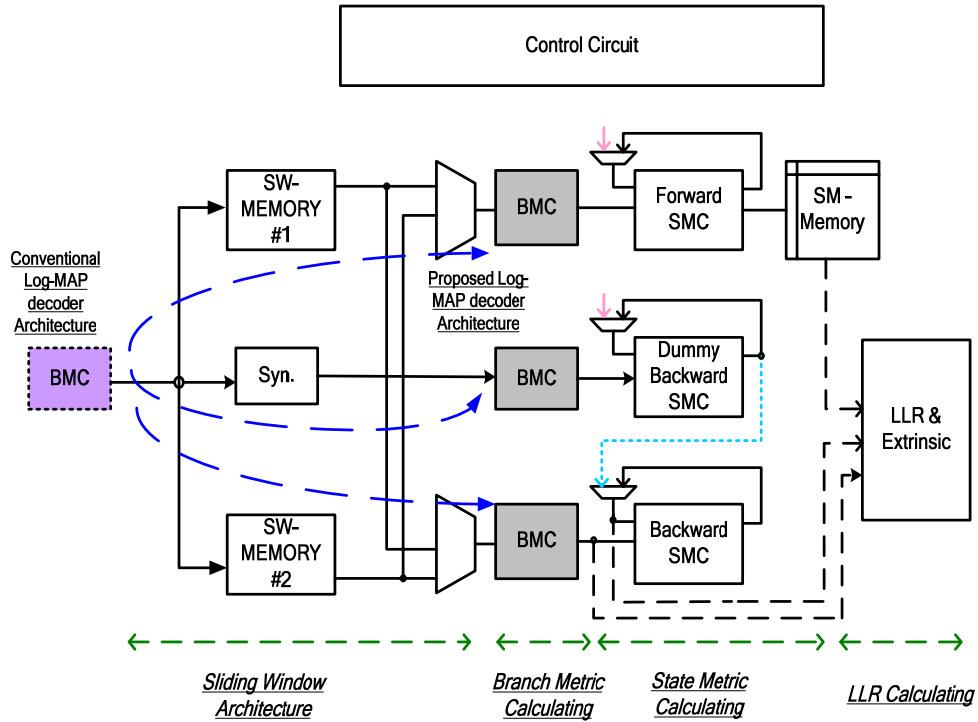


Fig. 3-7 The change of BMC block placement behind sliding window memory

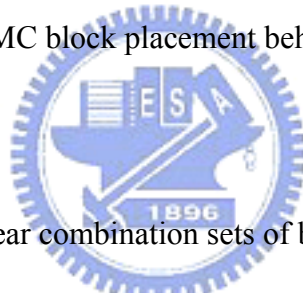


Table 3-1 The minimum linear combination sets of branch metric values for various code rates

Code rate	(1/2,1/3)	(1/4)	(1/6)
Input codeword	(x_k^s, x_k^{p1a})	$(x_k^s, x_k^{p2a}, x_k^{p3a})$	$(x_k^s, x_k^{p1a}, x_k^{p2a}, x_k^{p3a})$
All possible combinations of BM. value sets	$(+1,+1)$ $(+1,-1)$ $(-1,+1)$ $(-1,-1)$	$(+1,+1,+1)$ $(+1,+1,-1)$ $(+1,-1,+1)$ $(+1,-1,-1)$ $(-1,+1,+1)$ $(-1,+1,-1)$ $(-1,-1,+1)$ $(-1,-1,-1)$	$(+1,+1,+1,+1)$ $(+1,+1,+1,-1)$ $(+1,+1,-1,+1)$ $(+1,+1,-1,-1)$ $(+1,-1,+1,+1)$ $(+1,-1,+1,-1)$ $(+1,-1,-1,+1)$ $(+1,-1,-1,-1)$ $(-1,+1,+1,+1)$ $(-1,+1,+1,-1)$ $(-1,+1,-1,+1)$ $(-1,+1,-1,-1)$ $(-1,-1,+1,+1)$ $(-1,-1,+1,-1)$ $(-1,-1,-1,+1)$ $(-1,-1,-1,-1)$
Max. no. of BM value basis	2	4	8

Table 3-2 The sliding window memories storage comparison of our modified method and traditional SW scheme

<i>Architecture</i>	Traditional Sliding Window Memory Storage (For Each memory)	Proposed Sliding Window Memory Storage (For Each memory)
1/2	$\gamma_{width}^{radix^2} \cdot \frac{2}{min\ sets} \cdot SW_{length}$	$(bit(L_i) + 2 \cdot bit(y^s)) \cdot SW_{depth}$
1/3	$\gamma_{width}^{radix^2} \cdot \frac{2}{min\ sets} \cdot SW_{length}$	$(bit(L_i) + 2 \cdot bit(y^s)) \cdot SW_{depth}$
1/4	$(\gamma_{width}^{radix^2} + \frac{1}{avoiding\ overflow}) \cdot \frac{4}{min\ sets} \cdot SW_{length}$	$(bit(L_i) + 3 \cdot bit(y^s)) \cdot SW_{depth}$
1/6	$(\gamma_{width}^{radix^2} + \frac{2}{avoiding\ overflow}) \cdot \frac{8}{min\ sets} \cdot SW_{length}$	$(bit(L_i) + 4 \cdot bit(y^s)) \cdot SW_{depth}$

γ_{width} : The bit-width of branch metrics value.
min sets : the minimum linear combination sets for branch metrics values
bit(x) : The bit-width of variable x.
 L_i : Intrinsic information.
 y^s : Symmetrical codeword.
 SW_{depth} : Sliding Window Length.

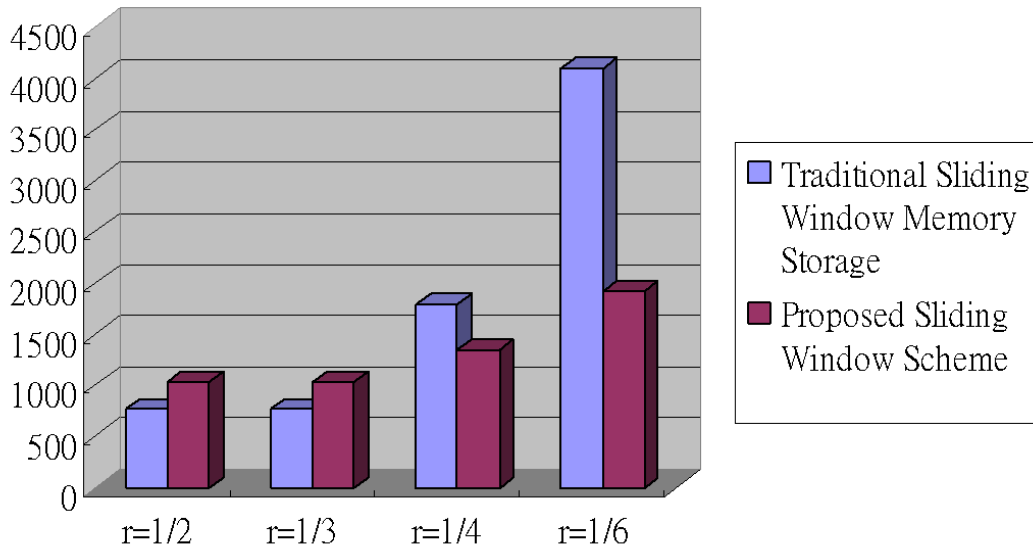


Fig. 3-8 The sliding memories size requirement between traditional sliding window scheme [24] and our modified method for different code rate. @ ($SW_{depth} = 32$, $bit(L_i) = 6$, $bit(\text{received codeword}) = 5$, $\gamma_{width} = 6$, two sliding window memories)

3.2.2 Interleaver Address Calculation Unit

Interleaver Address is used to shuffle the original sequence order to interleaving order or de-interleaving order for extrinsic information and systematic information (y_s). In practice, the interleaver address usually has two ways to implement. One way is to store interleaving patterns in the *ROM*, and the other way is to directly implement interleaving function on-line circuit.

Due to the number of interleaving patterns up to 8920 for *CCSDS* standard, we implement the reconfigurable interleaver address circuit which can select different interleaving sizes through control signal, as shown in Fig. 3.9. This on-line address calculation unit can save more area requirement relative to the *ROM* approach.

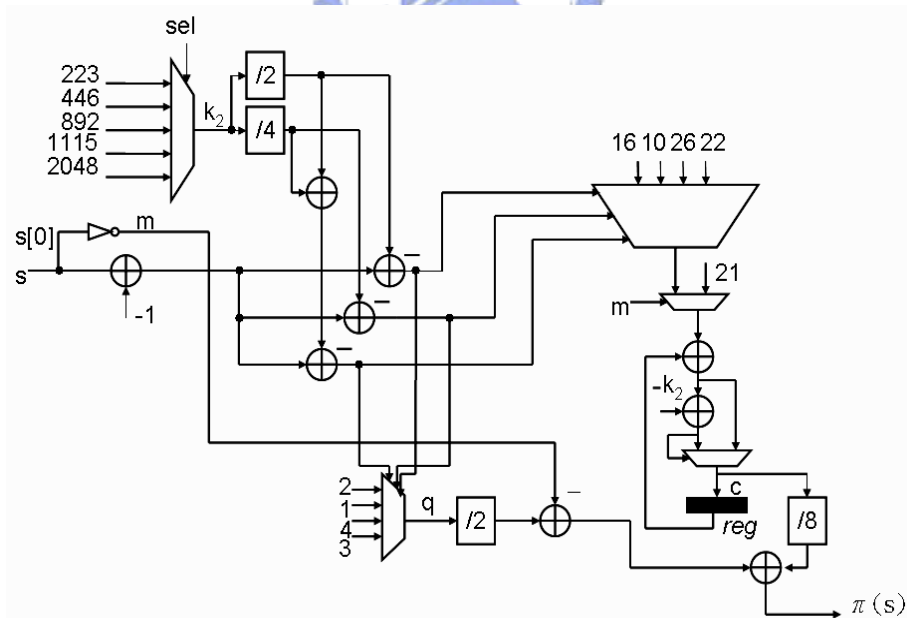


Fig. 3-9 Block diagram of on-line interleaving pattern calculation

3.2.3 Extrinsic Information Quantization

The quantization of extrinsic information technology proposed in [25], can reduce significant area requirement of extrinsic memory with a negligible performance loss. In the log-MAP algorithm, the extrinsic information is fed back to the branch metric calculator, which combines the input symbol with the extrinsic information. Minimizing the necessary chip area and power consumption is important, especially in mobile application.

The extrinsic values pass through the non-linear quantization mapping block, and then are stored into extrinsic memory, where the quantization mapping function is shown in Fig. 3.10. Note that the extrinsic information is compressed by non-linear mapping, leading to significantly the reduction of extrinsic memory area.

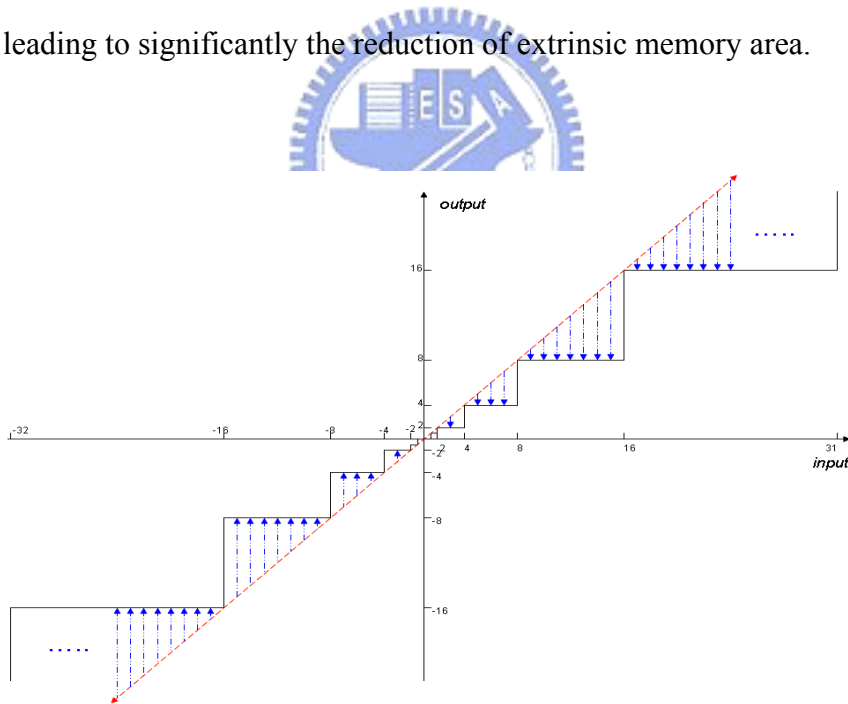


Fig. 3-10 Non-linear quantization for extrinsic information

3.3 Serial SISO Structure

In the single SISO decoder architecture, the estimated information is completely

outputted after approximately $2 \cdot Iter. \cdot (N + 2SW + C)$ cycles during decoding process, where N is the block size, SW denotes the sliding window length and C is the pipeline delay time. Therefore, for a given frame size N and sliding window length SW , the throughput of serial SISO Turbo decoder can be defined as follows:

$$Throughput_{single} \cong f_{clk} \cdot \frac{N}{2 \cdot Iter. \cdot (N + 2 \cdot SW_{length} + C)} \quad (3.1)$$

For instance, consider N to be 1784, the SW length to be 32, $Iter$ to be 5, where early stop method is applied to increase throughput, and assume the pipeline delay C can be neglected, the throughput of single SISO decoder could be approximately 19.3Mbps for clock rate 200MHz applied. However, this example demonstrates the serial SISO structure has a long latency 18480 cycles causing low data throughput. Thus, the low throughput and high latency for the serial structure is difficult to be applied in real-time media communication application.



3.4 Parallel SISOs Structure

Parallel decoding can significantly reduce the decoding processing latency relative to sequential decoding. The block size N is divided into P separate sub-blocks, $\{k, k + N/P, \dots, k + (P-1) \cdot N/P\}$, and each sub-block is performed by single SISO decoder. Thus, the throughput of parallel structure is faster than that of the serial SISO structure since the multiple structure is able to generate N extrinsic information currently relative one extrinsic information for the serial structure in each output unit time, and the latency is reduced to $2 \cdot Iter. \cdot (N/P) + 2L$ cycles.

Because the P -sets extrinsic are outputted from parallel SISOs simultaneously, the extrinsic memory requires P -distinct memory-banks with N/P depth to store it. However,

the beginning of each recursion state metrics requires a reliable value through “warm up training phase” for each forward and backward metrics per SISO decoder. To reduce the warm up training phase causing extra latency during decoder, there has been a well developed low latency initialized state metrics process for parallel SISOs structure as follows [26]:

- (a) The initial forward state metrics assume zero for all SISOs in the first decoding iteration.
- (b) The final forward state metrics of each SISO after iteration are stored and become the initial state metrics for its adjacent SISO for each iteration..
- (c) The initial value for the backward state metrics is to adopt the boundary backward state metrics value from the adjacent SISO decoder.

But, this initial process needs the additional memory storage to buffer the initial values for FP. Furthermore, the throughput of parallel SISO structure is a function of the parameters block size N , window size SW_{length} , the number of turbo iterations I , number of parallel workers N and clock frequency f_{clk} :

$$Throughput_{parallel \text{ and } high-radix} \cong f_{clk} \cdot \frac{N}{2 \cdot Iter. \left(\frac{N}{P} + 2 \cdot SW_{length} + C' \right)} \quad (3.2)$$

With the previous example, the throughput of 32-parallel structure could be achieved 297.95 Mbps and the latency are shorten as approximately 1198 cycles.

However, one of the parallel SISO decoder architecture’s existing problems is that there are probably more than one data to access the same memory destination simultaneously, also called the memory collision problem. There exists several issued interlaver, such as WCDMA, 3GPP and CCSDS, belonging to this “non-contention free interleaver”. As far as this non-contention free interleaver is concerning, we will

present an extrinsic information location strategy in the next chapter to resolve this problem.

On the other hand, many researchers have presented “contention free interleaver” to achieve high-parallelism with low complicated interleaver design. Even if the performance of contention free interleaver is superior to that of non-contention free ones since they have high spreading characteristic and high minimum distance to against the noise and interference over channel. So far, two excellent contention free interleaver QPP and APR interleaver , which has been be discussed in detail in [27] and [16], respectively.

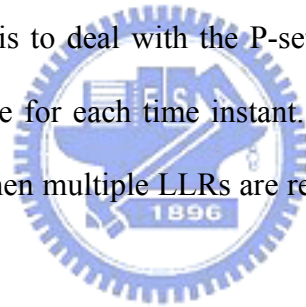


Chapter 4 Solving Memory Collision Problem For Parallel Turbo Decoder

4.1 Introduction

The well-known turbo decoder is an iteratively decoding process working between the soft-input soft-output decoder and interleaver / de-interleaver scrambler. Thus, a certain amount of memory is assigned to store channel information and extrinsic information.

For the serial Turbo structure, at each time instant, only one LLR is first written into extrinsic storage. As the whole decoding finishes, the *SISO2* begins to read intrinsic information from the extrinsic storage in serial. In contrast, for the P-parallel Turbo structure, its major difference is to deal with the P-set extrinsic values simultaneously writing into the distinct storage for each time instant. Unfortunately, the read or write access conflicts may occurs when multiple LLRs are read and written to the same target memory.



4.2 Memory Collision Problem for Parallel Turbo Decoder

The problem is best illustrated by taking the interleaver table shown in Fig. 4-1 for two concurrently produced LLRs and assigns its address to two individual RAMs. Table 1 shows the incoming data together with the associated targeted RAMs and relative addresses. In the first time-step from, one LLR is read from source RAM1 (Addr. 1) and written to garget RAM 2 (Addr. 3). At the same time, the other one is read from source RAM 2(Addr. 1) and written to target RAM1(Addr. 2), resulting in no conflicting event for the duration of write accesses. Unfortunately, in the second and third time-step, there are two data needed to be allocated at the memory simultaneously according to the

concept of interlaver table. Consequentially, it is needed to solve that the read or write access conflicts which may occur with multiple LLRs, otherwise, the exchanging information may fail in memory accesses.

Time-Step	Read accesses from Source RAM (Read address)		Write accesses into target RAM (Write address)		Memory Collision
1	M1 (Addr. 1)	M2 (Addr. 1)	M2 (Addr. 3)	M1 (Addr. 2)	X
2	M1 (Addr. 2)	M2 (Addr. 2)	M1 (Addr. 1)	M1 (Addr. 3)	○
3	M1 (Addr. 3)	M2 (Addr. 3)	M2 (Addr. 2)	M2 (Addr. 1)	○

Fig. 4-1 An example of memory collision event.

4.3 Solving Memory Collision Problem Using Temporal Buffer Architecture



The straightforward idea is to employ the buffer device for storing conflicting element, if the conflicting event occurs. Applying buffer device on the VLSI Turbo parallel decoder architecture, Norbert When (2002) [20] has been well developed in the Ring interconnect bottleneck breaker (RIBB) methodology, where the buffers are connected into a ring structure as shown in Fig. 4-2.

For each buffer, there are three different sources which come from local constituent decoder, the left-buffer distributor and the right-buffer distributor. When the output value of constituent decoder is sent into the buffer cell, the extrinsic information can either be fed through or stored to the local RAM. By the similar way, the decision for whether incoming data from the left or right slide is also determined by interlaver table.

As several data sets may have the same target, the buffers need to be capable of storing more than one data per cycle. Furthermore, the possible maximum read/write

ports for memory need to match the worst case for conflicting event. Otherwise, the buffer does not ensure to access all data sets correctly.

However, the above solution method requires an extra temporary buffer and collision handling time in view of hardware aspects. Therefore, the objective of the present memory collision free algorithm is to distribute the extrinsic dates from parallel SISO decoders into the storage elements without memory collision occurring. The proposed memory collision free algorithm can support various Turbo standards as well as arbitrary the number of parallel high radix SISO architecture in the following section.

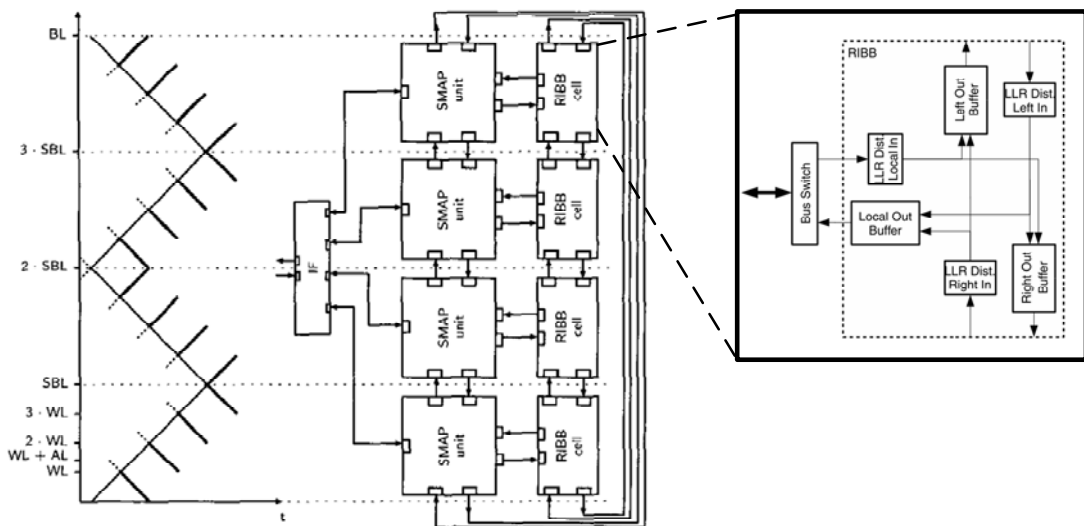


Fig. 4-2 Avoiding conflicting using temporal memory architecture

4.4 Proposed Memory Contention Free Scheme for Parallel Turbo Decoder

In order to obtain the maximum network flow (all memories are collision free), these extrinsic values must access distinct memory banks at every time instant; otherwise, the throughput of decoding process are delayed by occurring conflicting memory elements. Thus, we propose a solution for solving the extrinsic memory collision problem, as

discussed in the following concept.

4.4.1 Definition of Memory Collision Problem

Consider a P-parallel Turbo structure, whose frame size-N is divided into P number sub-blocks with length W defined by ceiling (N/P) , and each sub-block is performed by individual SISO decoder. The P-parallel SISOs structure are performed in parallel decoding through interleaver/de-interleaver, and the interleaving position corresponds into $\{\pi(i), \pi(W+i), \dots, \pi((P-1) \cdot W+i)\}$, where $\pi(i)$ represents the permuted position of i -th natural order data for $0 \leq i \leq W-1$.

For addressing the memory collision problem, the parallel structure can be distinguished into two aspects. For the interleaving aspect, when the P soft outputs are produced from the P-SISO decoders, it should be stored to distinct extrinsic memory banks. For the de-interleaver aspect $\pi^{-1}(i)$, which changes soft information ordering into the original sequence, the soft information are read separately from the distinct memory banks and used as intrinsic values for P-SISO decoding process.

In generally, the memory collision only occurs when accessing memory with an interleaving order. Consequently, for each time instant, they must access distinct memory banks and can be formulated as follows [3,4]:

$$\left\lfloor \frac{\pi(j+t \cdot W)}{W} \right\rfloor \neq \left\lfloor \frac{\pi(j+v \cdot W)}{W} \right\rfloor \quad (4.1)$$

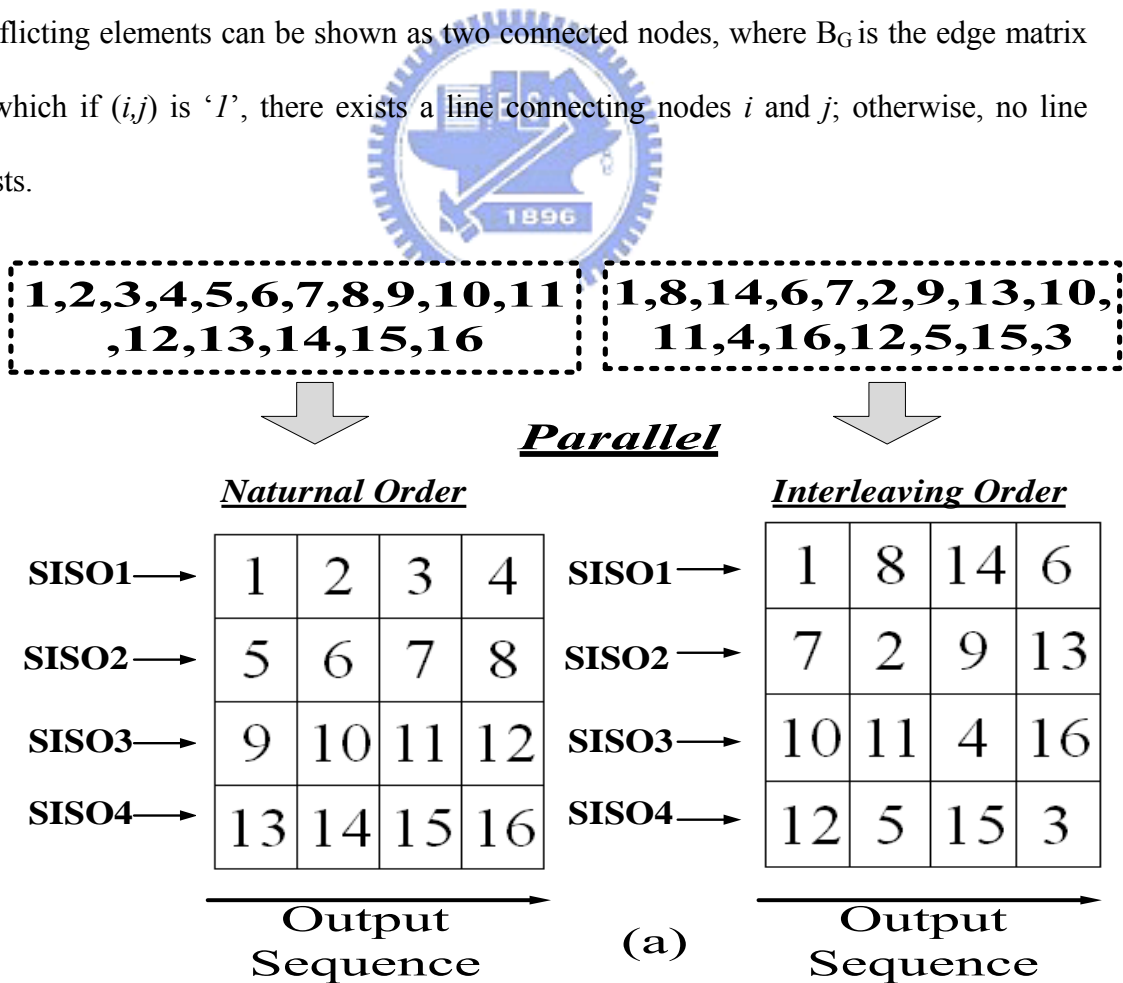
$$\left\lfloor \frac{\pi^{-1}(j+t \cdot W)}{W} \right\rfloor \neq \left\lfloor \frac{\pi^{-1}(j+v \cdot W)}{W} \right\rfloor \quad (4.2)$$

where $0 \leq j \leq W$, $0 \leq t, v \leq P-1$ and $t \neq v$.

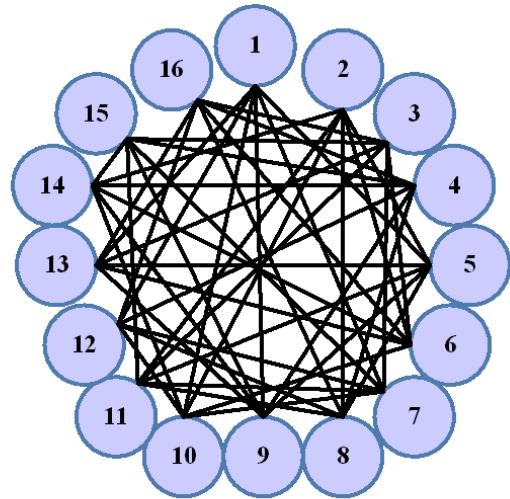
- Transforming Extrinsic Information Allocation Into Graphing Coloring Problem

In order to understand the relationship of (4.1) and (4.2) with soft information, for each time instant, the concurrent SISOs outputs are labeled as conflicting elements '1', which means not to be allocated into the same destination for combining all conflict cases from (4.1) and (4.2). And then converse the relationship between all conflict elements with the pictorial expression.

Fig. 4-3 demonstrates an example of frame-size ($N=16$) for four-parallel SISOs structure using graph expression of memory collision problem. Fig. 4-3 (a) shows the output sequence of natural order and interleaving order $\pi(i)$. According to (4.1) and (4.2), the output elements of each column in Fig. 4-3(a), seen as conflicting elements, should be allocated in parallel to different memory destinations. In Fig. 4-3(b) each pair of conflicting elements can be shown as two connected nodes, where B_G is the edge matrix in which if (i,j) is '1', there exists a line connecting nodes i and j ; otherwise, no line exists.



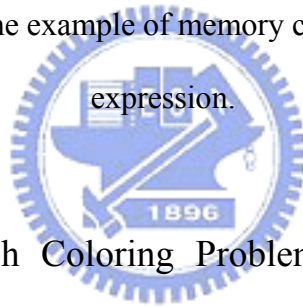
$$B_G = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & v_1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & v_2 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & v_3 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & v_4 \\ 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & v_5 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & v_6 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & v_7 \\ 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & v_8 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & v_9 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & v_{10} \\ 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & v_{11} \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & v_{12} \\ 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & v_{13} \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & v_{14} \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & v_{15} \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & v_{16} \end{bmatrix}$$



(b)

Fig. 4-3 (a) An example of natural order and interleaving order for 4-parallel Turbo decoder. (b) Converting the example of memory collision problem with graph

expression.



4.4.2 Solution to Graph Coloring Problem by Simulated Annealing Algorithm

Since memory collision problem can be analogous to robust graph coloring problem [27], which has been proved as NP-hardness problem, the simulated annealing algorithm [28] can be effectively used for reaching the goal of memory collision free.

- **Memory Collision Free Design with Simulated Annealing (SA) Algorithm**

First, we define an objective function to calculate the number of conflicting elements for given extrinsic information into extrinsic memory banks mapping (C). When the objective function reaches the zero value, the mapping C is an available memory collision mapping, the detail procedures are discussed as follows:

For a given graph topology $G = (V, E)$ with $|V| = N$, given the edge matrix B_G derived from the memory collision problem, where c indicates the number of memories and N denotes the frame size.

The objective function is defined as follows:

$$R(\text{MemoryMap}) \equiv \lambda \cdot \sum_{\forall (i,j), C(i)=C(j)} p_{i,j} \quad (4.3)$$

$$p_{i,j} \equiv \begin{cases} 1 & , (i,j) \in E \\ 0 & , \text{otherwise} \end{cases} \quad (4.4)$$

where λ denotes the penalty factor ($\lambda > 1$) and C is a coloring mapping which is identical to memory mapping in this case, i.e., $C: V \rightarrow \{1, 2, \dots, c\}$. This procedure is to obtain the solution C such that no two nodes are connected between the same memory set.

The memory collision free algorithm could be stated by the following four terms:

1. **Initialized procedure**: by choosing a random memory mapping to all vertexes (V_1, V_2, \dots, V_N) also or allocate all vertexes in the $Bank_l$. Then to set a very large value as the initial temperature, which has a larger probability to escape from local minimum (or maximum) value.
2. **Local search procedure**: The basic idea of the local search is that it starts from an initial solution and repeatedly replaces it with a better solution in its neighborhood until a better solution could not be found in the neighborhood. This is, if the new solution $MemoryMap_{new}$ is better than the current best solution $MemoryMap_{opt}$ in terms of the objective function value, the current best solution is updated; otherwise, the new solution is accepted by comparing the value of $exp((R(\text{new solution}) - R(\text{best solution}))/T)$ with a random number $U(0,1)$ generated from a uniform distribution on the interval $[0,1]$.

3. **Cooling procedure**: Next, the algorithm goes into the cooling schedule that decreases the temperature from T to $T*\alpha$ where the parameter attenuation constant α ($0<\alpha<1$) controls the speed of the convergence of the algorithm. This decreasing the temperature causes degenerative transition to be accepted with a lower probability as the algorithm progresses and corresponds to a lowering of temperature.
4. **Terminate memory collision free algorithm**: when the object function reaches zero value, the whole memory collision algorithm is terminated, and then the final memory mapping function C is outputted.

Fig. 4-4 shows a pseudo-code of memory collision free procedure using the simulated annealing algorithm.



Simulated Annealing Algorithm for Solving Memory Collision Problem

```

Procedure Memory_Collision_Free ( $c, n, MaxIt, T_m, \alpha$ )
 $T \leftarrow T_{initial}$ ;
 $V \triangleq \{V^1, \dots, V^n\}$ ;
 $Bank_1 \leftarrow \{V^1, \dots, V^n\}, Bank_2 \leftarrow \emptyset, \dots, Bank_c \leftarrow \emptyset$ ;
 $MemoryMap_{opt} \triangleq \{Bank_1, \dots, Bank_c\} \leftarrow$  Initial Solution Generation
REPEAT{
FOR  $I=1$  TO  $MaxIt$  DO
 $V^i \leftarrow$  Random choice of a vertex from  $\{V^1, \dots, V^n\}$ ;
Current allocated bank for  $V^i$  vertex  $\Rightarrow Bank_{current} \leftarrow Bank^{-1}\{V^i\}$ 
repeat{
 $Bank_{next} \leftarrow$  random choice of a memory bank from  $\{Bank_1, \dots, Bank_c\}$ 
} until  $\{(Bank_{current} \neq Bank_{next}) \ \& \ (|Bank_{next}| > 1) \text{ or } (|Bank_{current}| > 0)\}$ 
 $\Delta E = [R(MemoryMap_{new}) - R(MemoryMap_{opt})]$ 
IF  $(\Delta E < 0)$  OR  $(EXP(-\Delta E / T) > U[0,1])$  THEN
 $Bank_{current} \leftarrow Bank_{current} \setminus \{V^i\}, Bank_{next} \leftarrow Bank_{next} \cup \{V^i\}$ ,
 $|Bank_{current}| \leftarrow |Bank_{current}| - 1, |Bank_{next}| \leftarrow |Bank_{next}| + 1$ ,
 $MemoryMap_{opt} \leftarrow MemoryMap_{new}$ 
End IF
End FOR
 $T = \alpha \cdot T$ 
} UNTIL  $\{R(MemoryMap_{opt}) = 0\}$ 
return Final optimum solution  $\leftarrow \{Bank_1, \dots, Bank_c\}$ 

```

Fig. 4-4 Using simulated annealing algorithm for solving memory collision problem.

4.4.3 The Extrinsic Memory Collision Free VLSI Architecture Design

After simulated annealing algorithm is used for solving the memory collision problem, the final optimum solution guarantees that no two data can access the same memory concurrently. Fig. 4-5 demonstrates that the final optimum solution (color sets) obtained by SA algorithm to solve the memory collision of the previous example in Fig. 4-3. Fig. 4-5(a) shows that different colors correspond to different extrinsic memory banks. For the VLSI implementation, the memory mapping table can be stored into ROM table in advance as shown in Fig. 4-6.

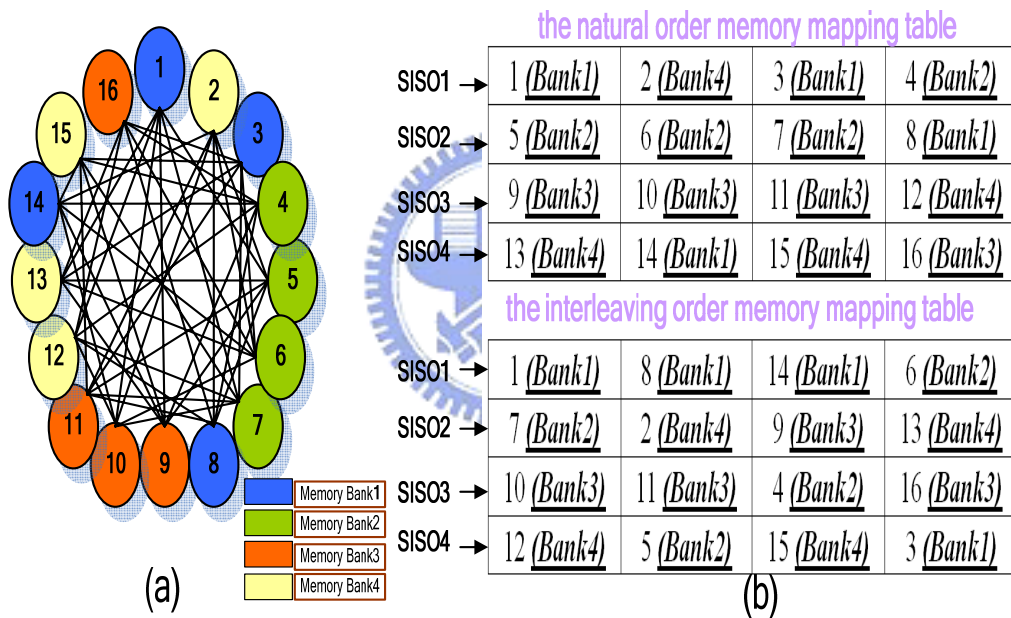


Fig. 4-5 (a) The solution of the example of Fig. 4-3 obtaining from contention free algorithm. (b) Converting each color set to corresponding each node element.

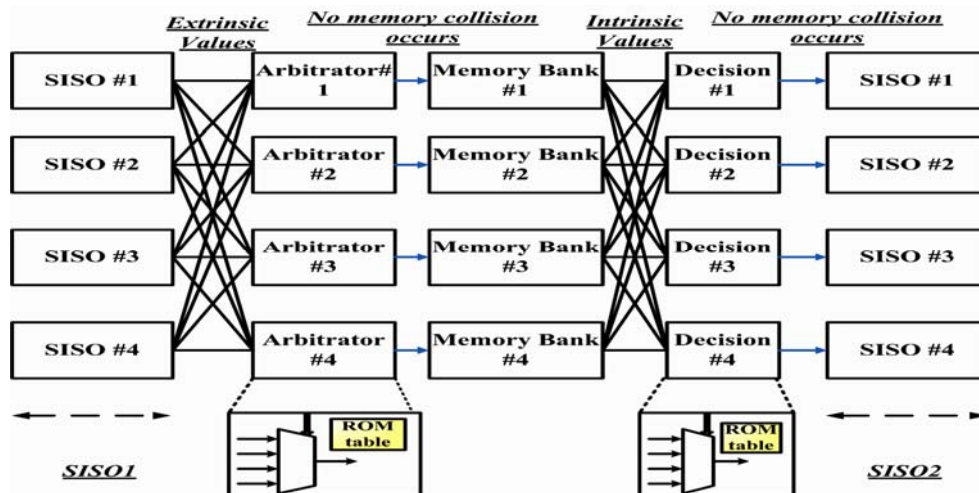


Fig. 4-6 Structure of proposed memory collision free architecture for the example of

Fig. 4-3

4.4.4 Simulation and Experiment Results

Since the quantization should be the trade-off between coding performance loss and hardware cost, the fix-point can be determined via *Monte-Carlo* simulation. The primary specifications of the Turbo decoder are given in Table 4-1, where the code polynomial follows the Consultative Committee for Space Data Systems (CCSDS) standard [5]. Fig. 4-7 demonstrates the Turbo decoding performance results after 8 iterations for different sliding window lengths (SW) and fixed point bit-width. We can see that the curve of (SW=32 & fixed point) has the minimum error performance loss relative to that of floating point simulation. On the other hand, the curve of (SW=32 & fixed point & 4-bits non-linear encoded [25]) leads to larger performance loss relative to that of floating point simulation, but less extrinsic memory requirement in the Turbo decoder is expected.

Our proposed contention free algorithm can be used for parallel Turbo decoder supporting the arbitrary SISO numbers and high radix VLSI architecture. Here, we take the specification of Table 4-1 into account to achieve the purpose of contention free for 8, 16 and 32-SISO numbers.

Table 4-1 Summary of parameters for Turbo code simulation

Application	CCSDS
Generator Polynomial G(D)	$\left[1 \frac{1+D+D^3+D^4}{1+D^3+D^4} \frac{1+D^2+D^4}{1+D^3+D^4} \frac{1+D+D^2+D^3+D^4}{1+D^3+D^4} \right]$
SISO Algorithm	Sliding window log-MAP
SISO Architecture	Radix-2
Block Size	1784
Sliding Window Size	32
The Number of Iteration	8 (Fixed)
Quantization	Received input: (5,2) Extrinsic information: (6,2) Branch metric: (6,2) State metric: (9,2)

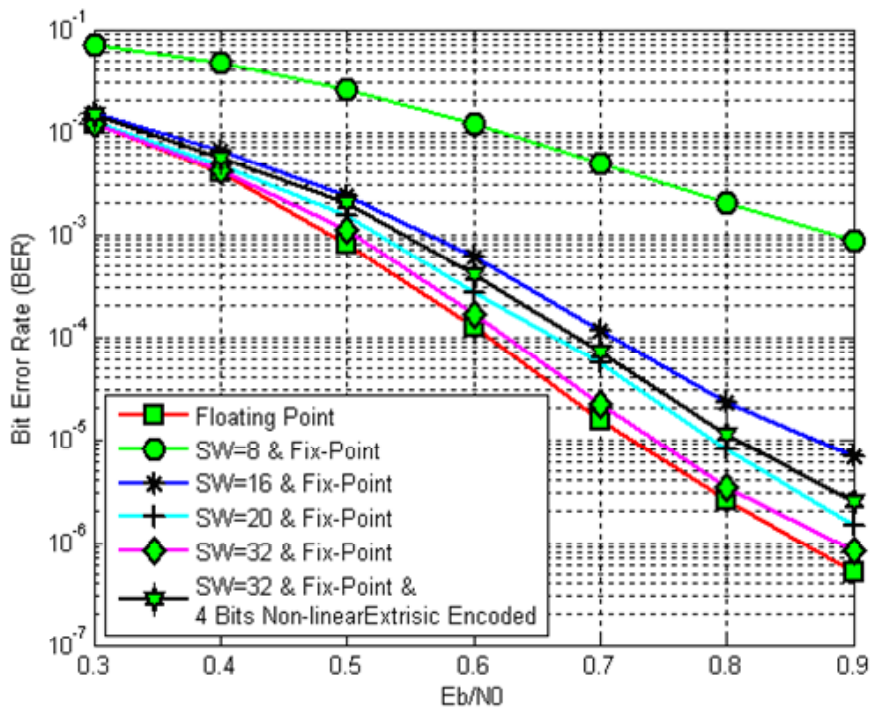


Fig. 4-7 BER performance of the Turbo decoder

The Fig. 4-8 demonstrates the change of cost functions of contention free algorithm for various parallel Turbo decoder applications. The contention free algorithm is terminated until the cost function reaches zero value. Then, the solution (color sets) is obtained with respect to memory banks such that each nodes (extrinsic value) has no occurrence of conflicting events. The Fig. 4.9-11 shows the solution of contention free algorithm for 8, 16 and 32-parallel Turbo decoder, where the horizontal axes denotes the time index (i.e., the order of output sequence of SISO decoder); and the vertical axes corresponds to the location of P-SISO decoders in the Turbo decoder. We can see that each column is drawn with different color. Thus, our proposed algorithm guarantees to achieve the purpose of contention free.

In general, the circuit function needs to be verified by function verification after synthesis or FPGA (Field Programmable Gate Array) platform. Due to the high expense in the IC manufacturing, the FPGA provides more cheaper programmable and reconfigurable ways to verify your circuit. Even the FPGA platform offers immediate real electronic signals to work together with other system platform (i.e., ARM) or measured from oscilloscope. The Fig. 4-12 shows that the output signals of Turbo decoder using the Xilinx Virtex-IV XC4VLX60 FPGA. For the function verification, we first store all output signals into text file and then compare the output values of golden model from MATLAB[®] with the output values of FPGA as shown in Fig. 4-13. When the error signal is raised, there exists some difference between golden model and output signals from FPGA. Otherwise, the output signals of FPGA platform are correct.

We have simulated and verified the design logic by comparing the output results to MATLAB[®] fixed-point simulation and performed synthesis targeted at UMC 130nm CMOS technology by the Synopsis[®] design compiler.

Fig. 4-14 shows the proposed architecture of contention free parallel Turbo decoder which major consists of multiple double-input-buffers, SISO decoders, Look-Up-Table (LUT) and some control circuits. Each SISO core consists of three recursion units for acquisition, forward and backward recursion which requires additional controllers for the state and branch metrics memories, where we assume eight iterations are performed for turbo decoding and clock rate is set 200MHz. There are two input memory-banks applied such that the decoding process could be able to continuously decode noisy codeword at different frame [20] and the extrinsic storage also employs P sets distinct memories to achieve the goal of memory collision free. Then, the results of memory collision free are stored into the LUT memories. One of LUT memories is used for the arbitrator device; the other is used for the decision device. Finally, the control circuit is employed such that the design can be more flexible.

Each SISO core consists of three recursion units for acquisition, forward and backward recursion which requires additional controllers for the state and branch metrics memories, where we assume eight iterations are performed for turbo decoding and clock rate is set at 200MHz.

Table 4-2 lists the area requirement of the proposed parallel collision free Turbo decoder implementation for various number SISO decoders. As a result, the high parallel Turbo decoder has larger total area size but relative its throughput also becomes faster than that of low parallel Turbo decoder. In practice, the hardware implementation should choose the appropriate parallel parameter P by achieving the throughput requirement and minimizing the area requirement. However, our proposed algorithm can support arbitrary parallel parameter P such that no conflicting element causes the degradation of whole throughput of Turbo decoder.

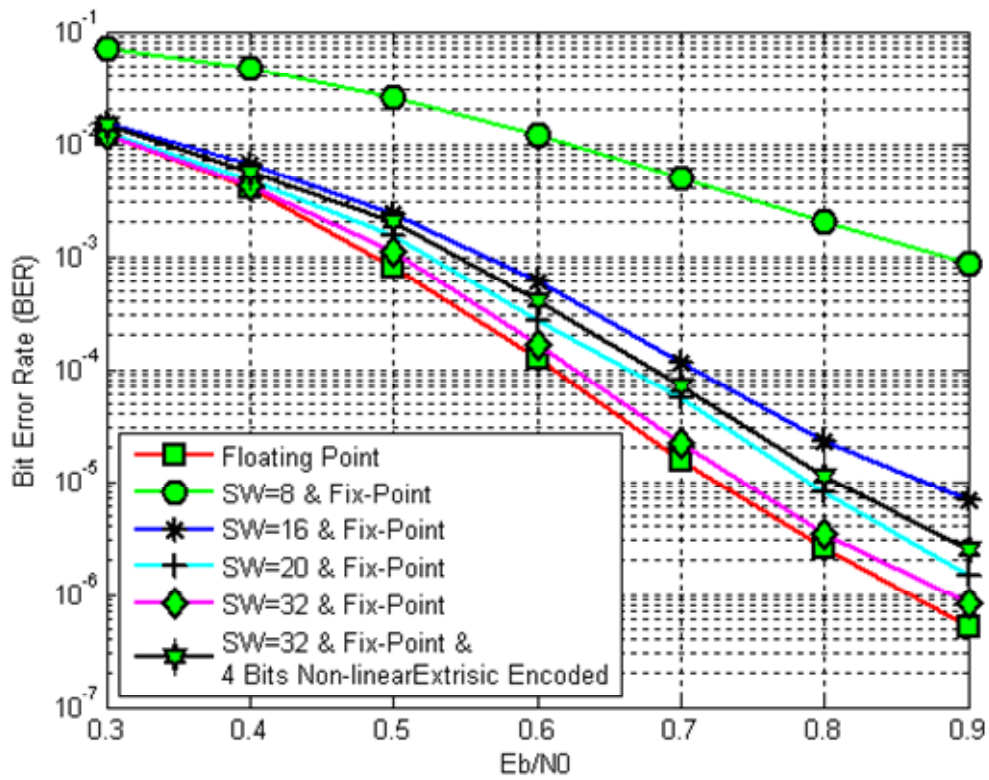


Fig. 4-8 The change of cost functions of contention free algorithm for various parallel Turbo decoder applications.

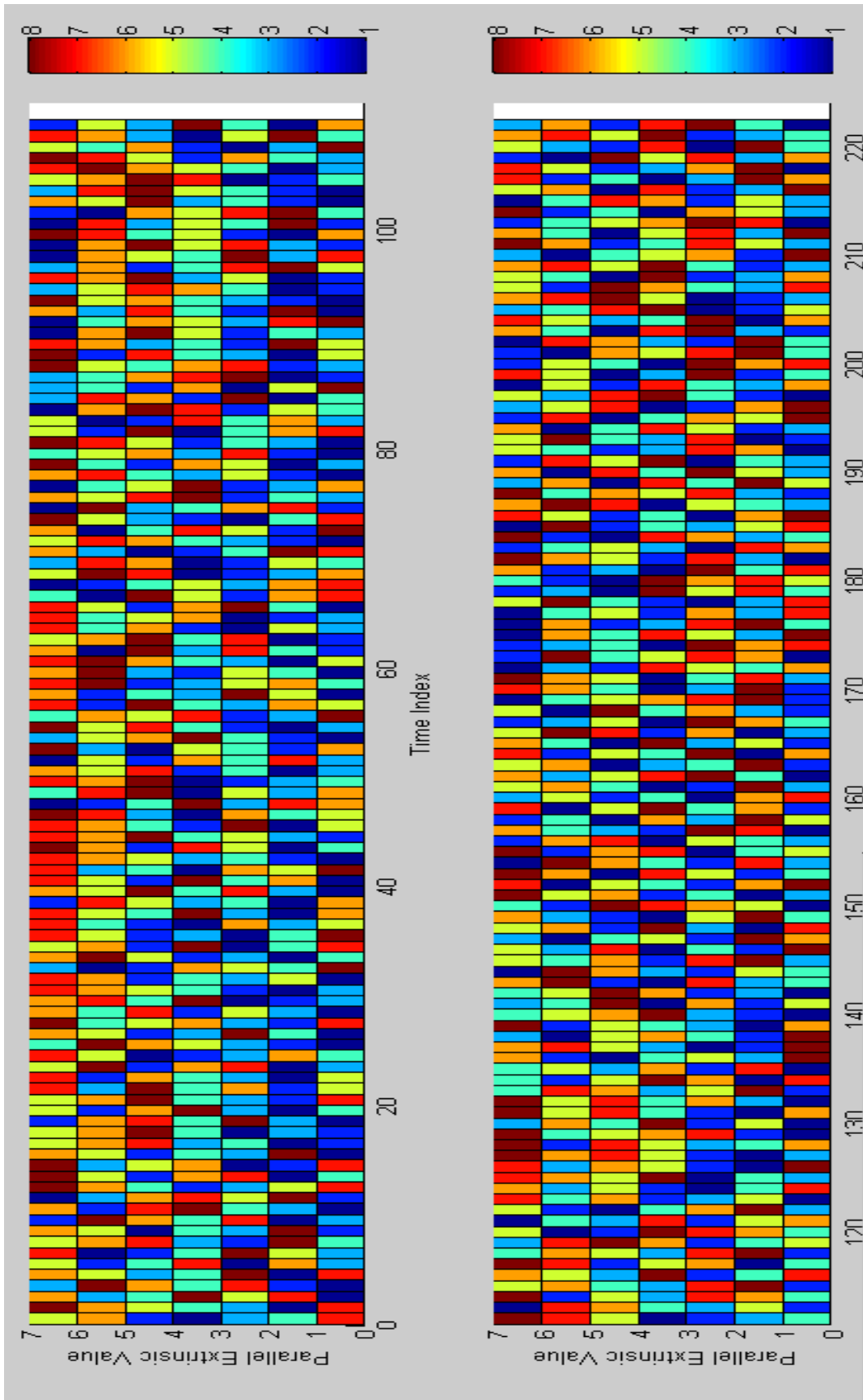


Fig. 4-9 The solution of contention free algorithm for 8-parallel Turbo decoder

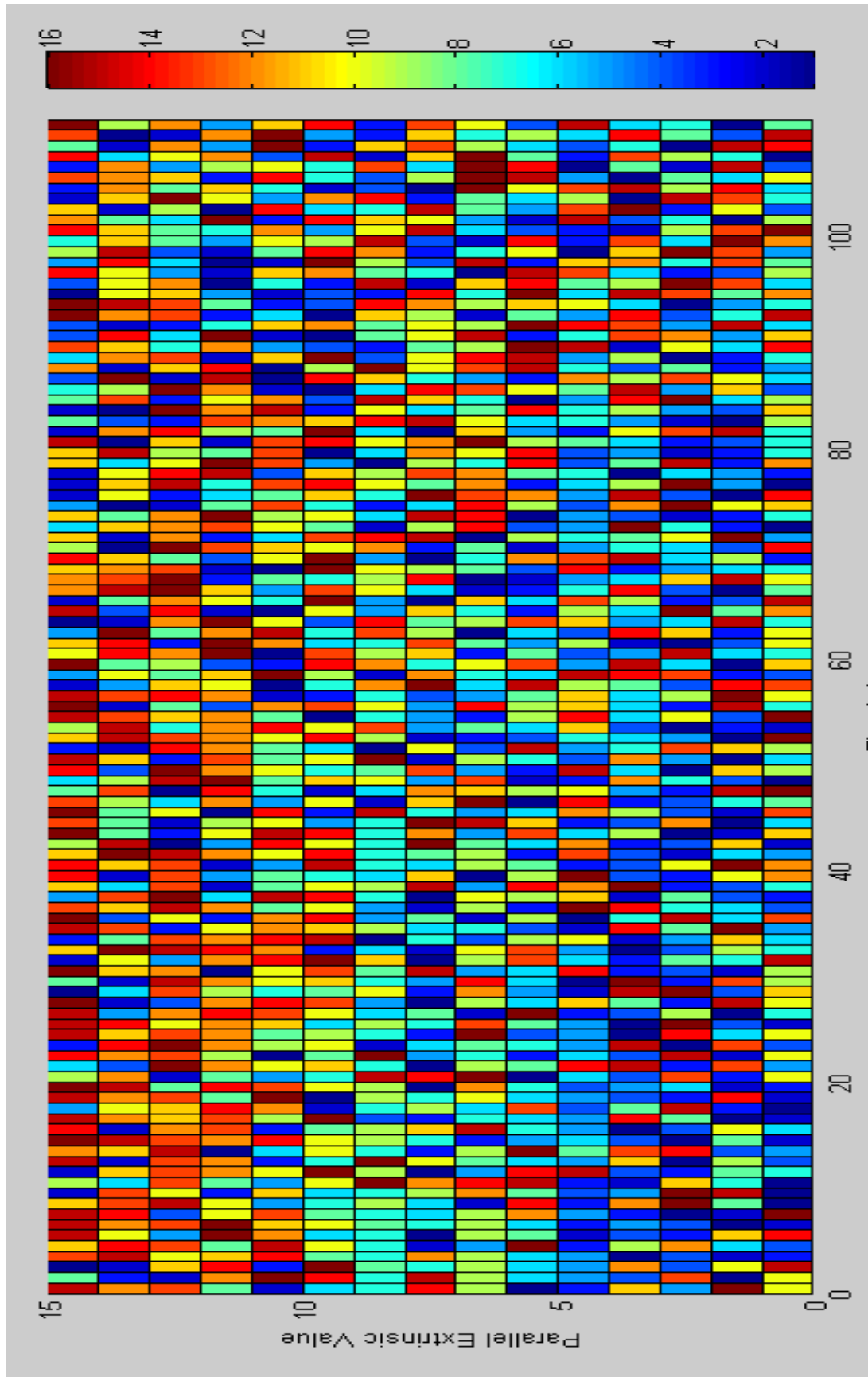


Fig. 4-10 The solution of contention free algorithm for 16-parallel Turbo decoder

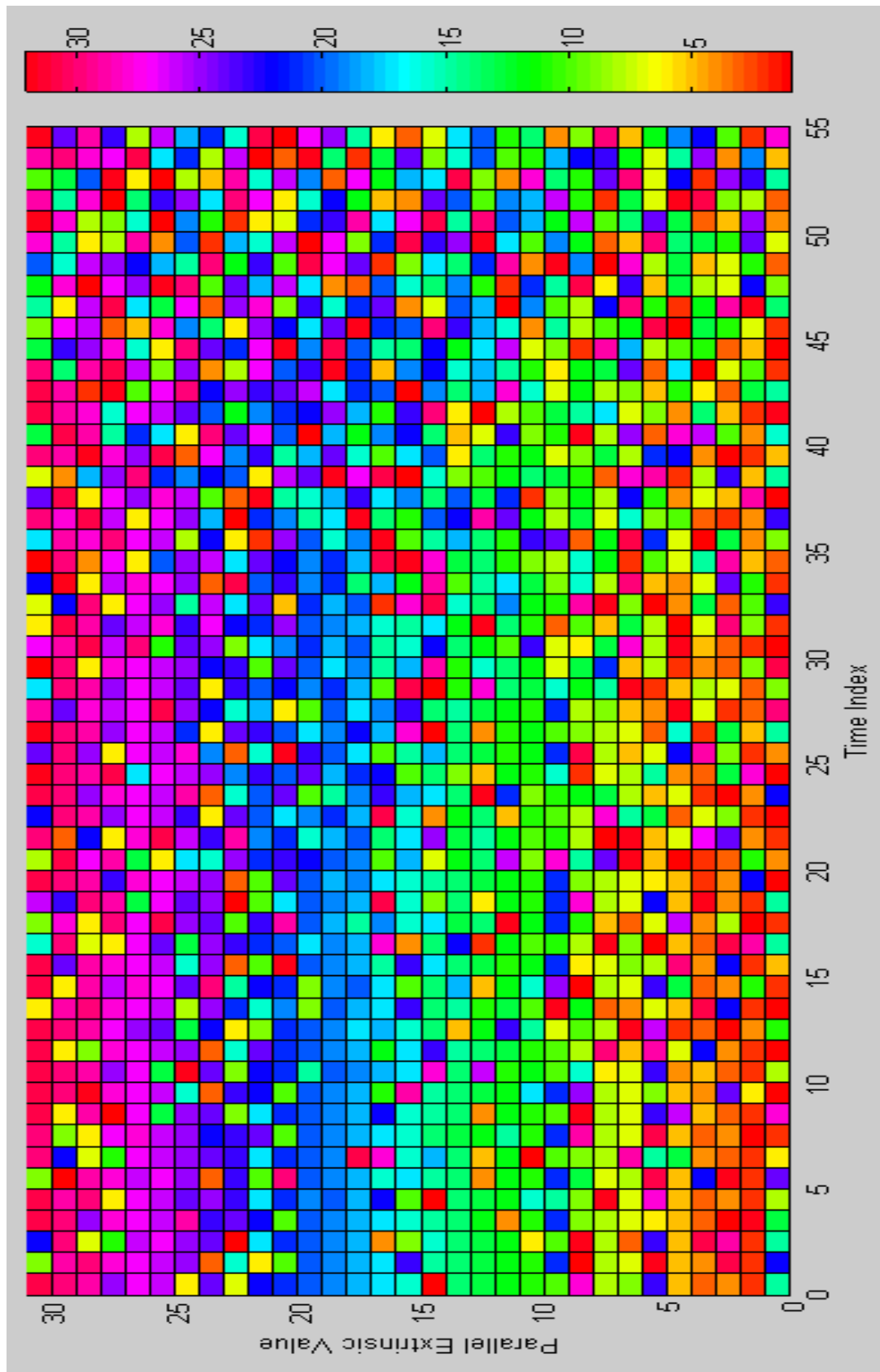


Fig. 4-11 The solution of contention free algorithm for 32-parallel Turbo decoder

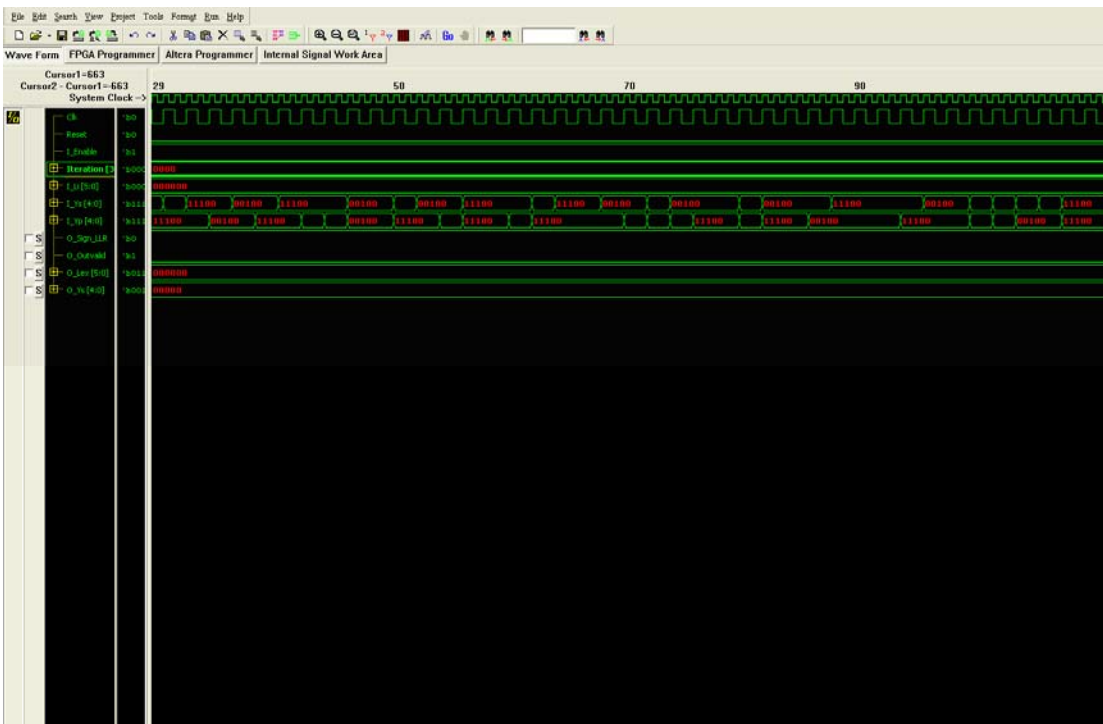


Fig. 4-12 The VLSI architecture implementation of Turbo decoder in the FPGA platform

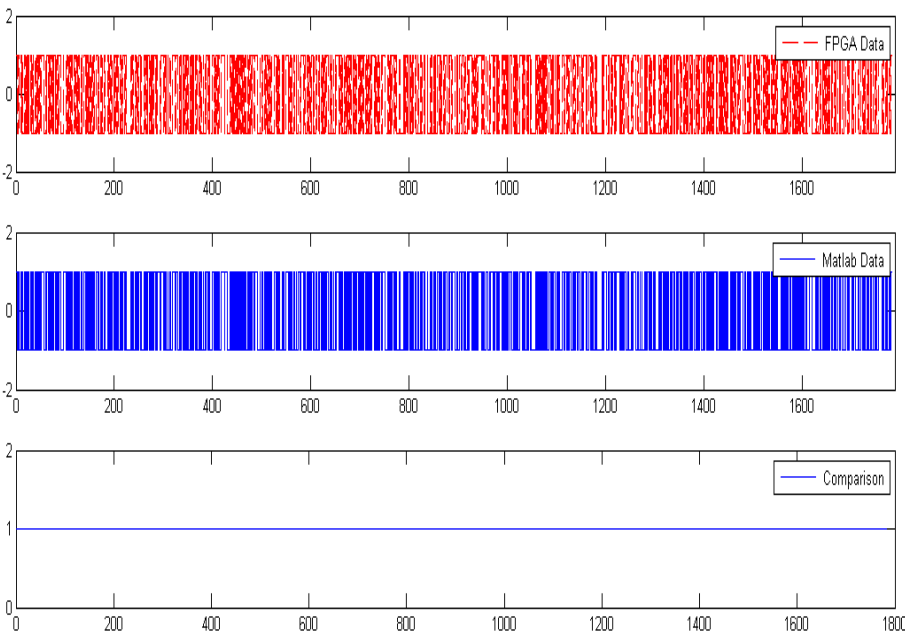


Fig. 4-13 The comparison of the output values of golden model from matlab[®] with the output values of FPGA

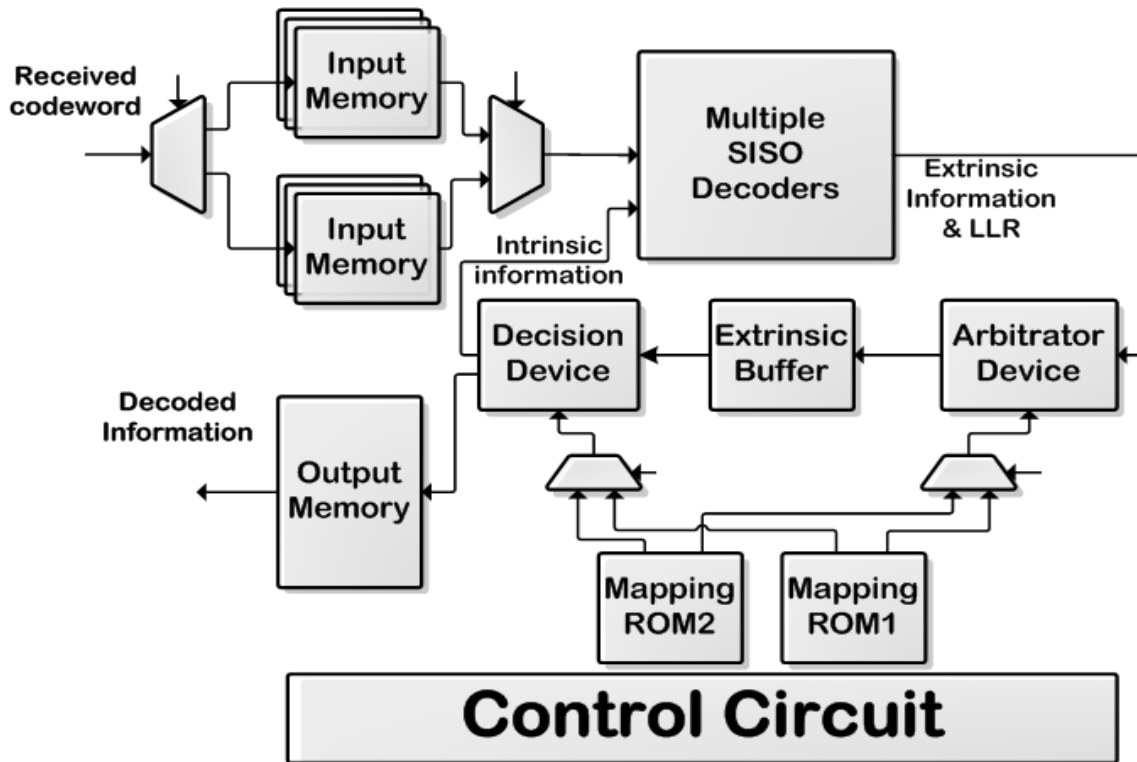


Fig. 4-14 The block diagram for the proposed contention free parallel Turbo decoder

Table 4-2 Parallel Turbo decoder area and through for various number of SISO decoders at clock frequency 200MHz.

<i># of parallel P [lock size N:1784]</i>	<i>1</i>	<i>8</i>	<i>16</i>	<i>32</i>
<i>SISO decoder (16 states OACS)</i>	<i>0.296</i>	<i>2.368</i>	<i>4.732</i>	<i>9.45</i>
<i>Extrinsic RAMs</i>	<i>0.061</i>	<i>0.184</i>	<i>0.272</i>	<i>0.425</i>
<i>I/O-Data RAM</i>	<i>0.245</i>	<i>0.784</i>	<i>1.344</i>	<i>2.688</i>
<i>Look Up Table Memory</i>	<i>No needs</i>	<i>0.558</i>	<i>0.960</i>	<i>1.344</i>
<i>Interface/Control</i>	<i>0.03</i>	<i>0.08</i>	<i>0.12</i>	<i>0.17</i>
<i>Total area (nm²)</i>	<i>0.632</i>	<i>3.974</i>	<i>7.428</i>	<i>14.077</i>
<i>Throughput rate @200MHz, iteration:8 (Mbps)</i>	<i>12.002</i>	<i>74.581</i>	<i>118.93</i>	<i>169.25</i>

4.5 An Approach for Reducing Memory Area of Parallel Turbo Decoder

For parallel Turbo decoding processing, the overall decoding throughput increases linearly together with the hardware complexity. In particular, the temporary memories occupy a significant portion of total hardware. Several papers have proposed different strategies for reducing memory area of SISO decoder, such as sliding window memory and state metrics memory [29]. In this section, we present two approaches that can reduce a considerable amount number of extrinsic memory area for parallel Turbo decoder.

4.5.1 Classical Extrinsic Memory Access for Single Turbo Decoder

Fig. 4-15 shows the waveform expression of single SISO decoder and the external storage components, which consist of input buffer unit and extrinsic memory. Note that the shaded region denotes concurrent read and write access to the extrinsic memory. The length of shaded region can be approximated as $N-L$, where N is codeword block size and L denotes the latency of log-MAP decoder. In practice, the extrinsic memory can be implemented by dual port memory [30], two port memory [31], or two single port memories.

4.5.2 An Area-Efficient Extrinsic Memory Scheme for Parallel Turbo Decoder

The extrinsic memories occupy a considerable amount of area in the parallel Turbo decoder due to multiple memory banks. The reduction of extrinsic memories is necessary for an area efficient Turbo decoder. Fig. 4-16 demonstrates the waveform of multiple SISO decoders and the extrinsic memories. We can see that the length of the shaded

region in parallel Turbo decoder scheme is shorter than that of single Turbo decoder scheme (Fig. 4-15), which can be approximated as $(N/P)-L$, where P assumes the number of multiple Turbo decoders.

The length of the shaded region can become much smaller, even reaching zero, through the use of a larger number of SISO decoders each employing the memory collision free algorithm presented in previously section. When the length of the shaded region is small enough, the extrinsic memories can be implemented by single port memories with one temporary buffer used for storing the extrinsic values. [25] had presented an extrinsic value mapping function, which could significantly reduce the bit-width of the extrinsic value. Fig. 4-16 shows an area-efficient extrinsic memory scheme for parallel Turbo decoder where the extrinsic memories and buffer are realized by signal port memories. Due to the multiple single port memories used for storing extrinsic values, there is significant area reduction in relation to use of multiple dual port or two port memories. Therefore, the presented area-efficient scheme has a memory area reducing benefit for the parallel Turbo decoder.

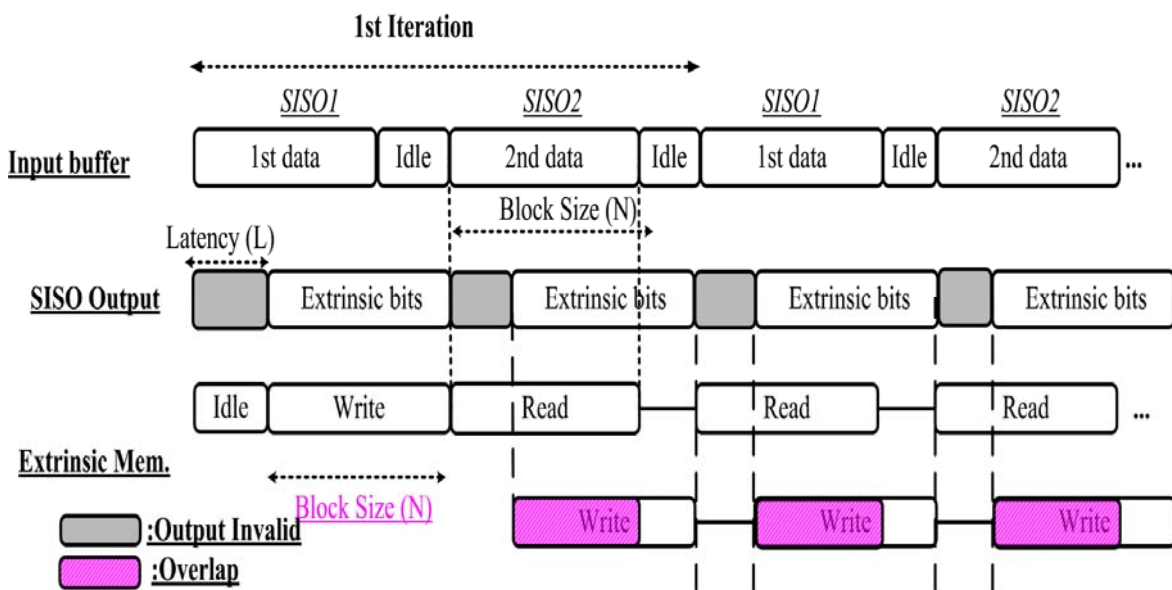


Fig. 4-15 The waveform expression between the single log-MAP decoder and the

external storage components which consist of input buffer and extrinsic memory.

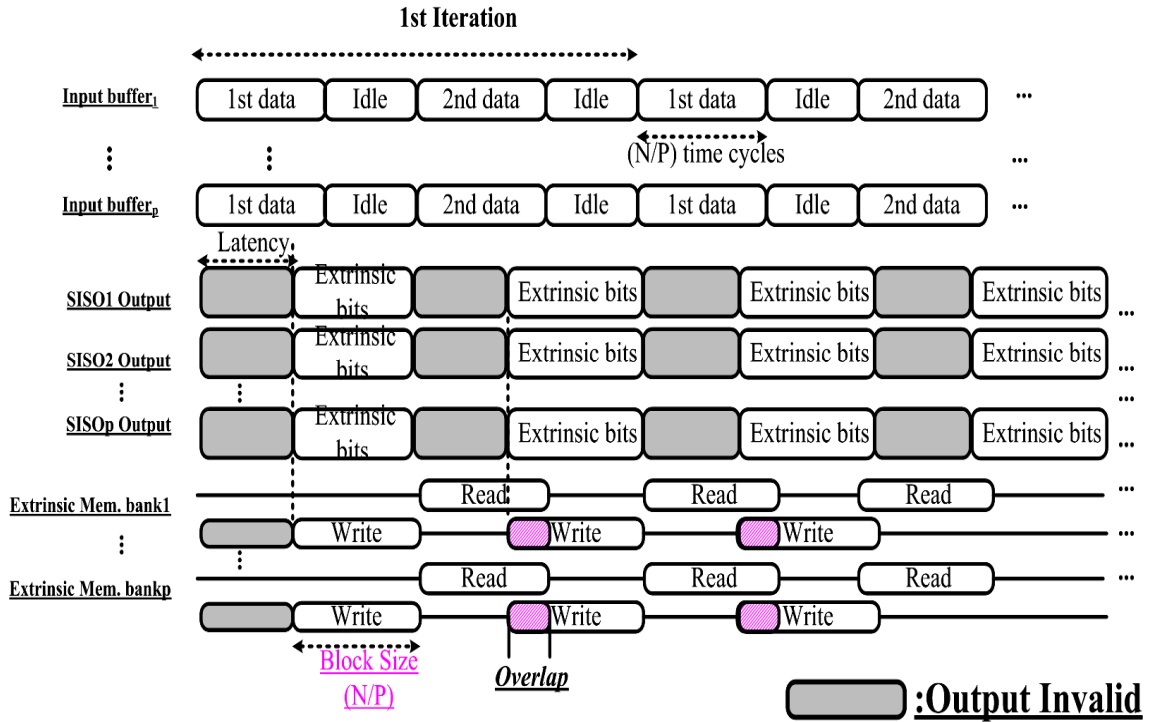


Fig. 4-16 The waveform expression between the multiple SISO decoders and the external storage components which consist of input buffers and extrinsic memories.

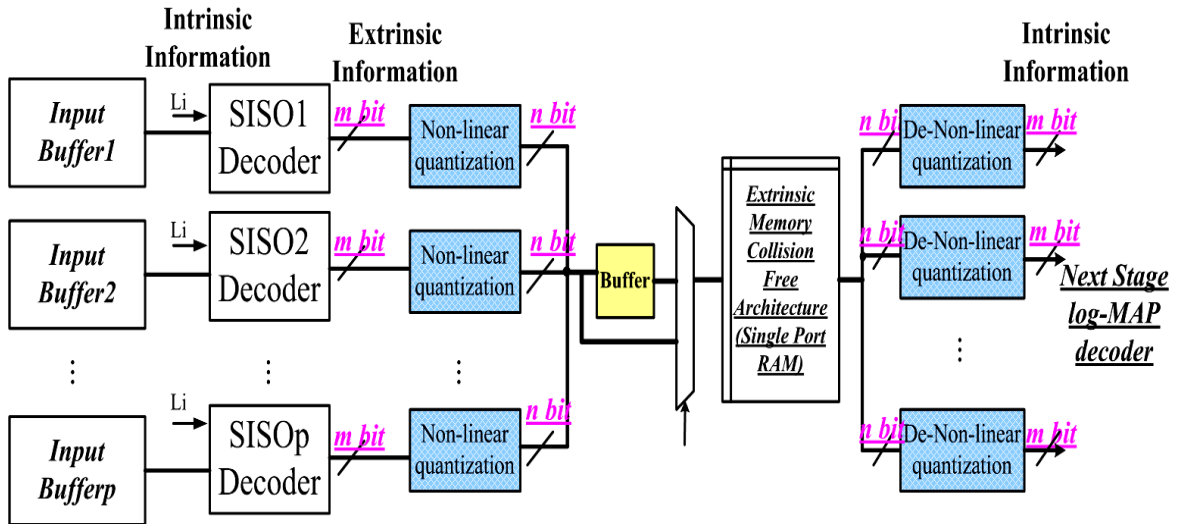


Fig. 4-17 Structure of proposed an area-efficient extrinsic memory scheme for parallel Turbo decoder architecture.

4.5.3 Analysis of the Required Memory Size

The hardware evaluation is obtained by using Verilog HDL codes synthesized with the standard cell library of UMC 0.13- μm CMOS process. Since the CCSDS standard does not support parallel contention free Turbo decoder architecture, we can apply memory collision free algorithm introduced in previously section to realize parallel turbo decoder architecture. In Table 4-3, the extrinsic memory requirement of four different parallel Turbo decoder configurations is evaluated with the latency L of SISO decoder as measured by HDL simulator. Note that the proposed extrinsic storage schemes achieve 31% area reduction(0.096 mm^2) without extrinsic quantization and 41% area reduction (0.125 mm^2) with extrinsic quantization relative to conventional extrinsic storage using dual port RAM, which was proposed in [30], for 8-parallel Turbo decoder architecture. The extrinsic memory requirement for 16-parallel Turbo decoder architecture and 32-parallel Turbo decoder architecture are also listed in Fig.4-18. Note that a larger number of parallel Turbo decoder can obtain a larger percentage of memory area reduction as the read and write access time of extrinsic memory does not overlap such that the temporary buffer is not necessary.

Table 4-3 Summary of area requirements for various organization of extrinsic memory architecture

Method	Organization	Area Requirement of Extrinsic Memory
Traditional Extrinsic Storage (mm^2)	(Dual Port RAM)	$\text{ceil}(N/P) * \text{Bit}\{Ex\}_{\text{sub-bank}} * (\text{No. of Banks})$
	(two Port RAM)	
Proposed Extrinsic Storage (mm^2)	(Single Port RAM+ Extrinsic temporary buffer)	$\text{ceil}(N/P) * \text{Bit}\{Ex\}_{\text{sub-bank}} * (\text{No. of Banks}) + \text{abs}(\text{ceil}(N/P-L)) * \text{Bit}(Ex * \text{No. of Banks})_{\text{buffer}}$
	(Single Port RAM+ Extrinsic Buffer+ Extrinsic mapping[11])	$\text{ceil}(N/P) * \text{Bit}\{map\{Ex\}\}_{\text{sub-bank}} * (\text{No. of Banks}) + \text{abs}(\text{ceil}(N/P-L)) * \text{Bit}(map\{Ex\} * \text{No. of Banks})_{\text{buffer}}$
<p>Note:</p> <p>L: the output extrinsic latency for half iteration of SISO decoder (104 cycles measured). ceil(x): the upper integer of x. N: Block size (1784+4 return zero bits). SW: 32 applied. Depth*Bit-width: the memory area requirement.</p>		

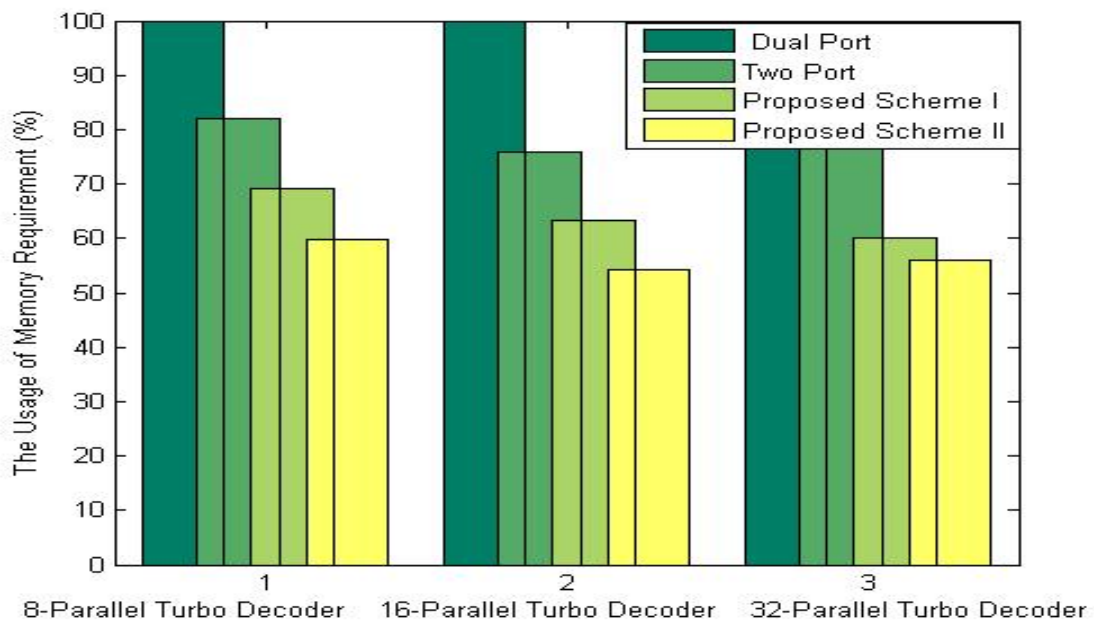


Fig. 4-18 Comparison of area requirements for different organization of extrinsic memory architecture (@ UMC 0.13- μm CMOS Process Measured and latency $L=104$ cycles measured).



Chapter 5 Conclusion

A memory collision free algorithm to achieve high parallel SISO decoders for turbo-decoding has been presented. The high parallel collision free Turbo decoder has been implemented for VLSI architecture using the UMC-90nm standard CMOS cell library. As a result, the throughput of 32-memory-collision free turbo-decoding could achieve up to 169.25Mbps with clock frequency 200MHz, which is faster than that of serial SISO decoder 12Mbps

This paper introduces a memory collision free algorithm using simulated annealing heuristic method for parallel Turbo decoder, in which a highly parallel structure is available. By applying memory collision free algorithm, we proposed two area-efficient extrinsic memory schemes achieving lower hardware cost for high parallel Turbo decoder structure. The experimental results in UMC 0.13- μm CMOS process show that the organization of our proposed extrinsic memory without extrinsic non-linear quantization can achieve around 40% memory area reduction for the 32-SISO parallel Turbo decoder relative to conventional extrinsic storage using dual port RAM for the Turbo code of CCSDS standard. On the other hand, the organization of our proposed extrinsic memory with extrinsic non-linear quantization can further achieve around 46% memory area reduction for the 16-SISO parallel Turbo decoder.

Bibliography

- [1]. C. Berrou, A. Glavieux, and P. Thitimajshima, “Near Shannon limit error-correcting coding and decoding: Turbo-codes,” in Proc. ICC, Geneva, Switzerland, May 1993, pp. 1064–1070.
- [2]. Boutillon, E. Douillard, C. Montorsi, G. “Iterative Decoding of Concatenated Convolutional Codes: Implementation Issues,” Proceedings of the IEEE, June 2007, Issue: 6, pp. 1201-1227.
- [3]. A. Nimbalkar, T. K. Blankenship, B. Classon, T. E. Fuja, D. J. Costello, Jr, “Contention-free interleavers,” Int. Symp. on Inf. Theory, June 2004.
- [4]. A. Tarable, G. Montorsi and S. Benedetto, “Mapping of interleaving laws to parallel turbo decoder architectures,” in Proc. 3rd Int. Symp. Turbo Codes Related Topics, Brest, France, Sep. 2003, pp. 153-156.
- [5]. Consultative Committee for Space Data Systems, CCSDS 131.0-B-1 Blue Book September 2003.
- [6]. S. Dolinar and D. Divsalar, “Weight Distributions for Turbo Codes Using Random and Nonrandom Permutations” TDA Progress Report 42-122, Jet Propulsion Laboratory, Pasadena, California, pp. 56–65, August 15, 1995. Primitive feedback polynomials.
- [7]. D. Divsalar and F. Pollara, “On the design of turbo codes”, TDA Progress Report 42-123, Jet Propulsion Laboratory, Pasadena, California, pp. 99–121, November 15, 1995.
- [8]. P. Guinand and J. Lodge, “Trellis termination for turbo encoders,” in Proc. 17th Biennial Symp. Commun., pp. 389–392, May 30-June 1 1994.

- [9]. J. Hokfelt, O. Edfors, and T. Maseng, "A survey on trellis termination alternatives for turbo codes," in Proc. IEEE Vehicular Technology Conf. (VTC'99), pp.2225–2229, May 1999.
- [10]. S. Crozier, P. Guinand, and A. Hunt, "On designing turbo-codes with data puncturing," in 9th Canadian Workshop on Inform. Theory (CWIT'05), pp. 32–35, June 2005.
- [11]. L. C. Perez, J. Seghers, and D. J. Costello, Jr., "A distance spectrum interpretation of turbo codes," IEEE Trans. Inform. Theory, vol. 42, pp.1698–1709, Nov. 1996.
- [12]. C. Berrou, "Turbo codes: some simple ideas for efficient communications", ESA-DSP 2001, Lisbon, Oct. 2001, and ESA-TTC 2001, Noordwijk, The Netherlands, Oct. 2001.
- [13]. D. Divsalar and F. Pollara, "Multiple Turbo Codes" MIL-COM'95, pp. 279-285, November 6-8, 1995.
- [14]. S. Benedetto and G. Montorsi, "Unveiling Turbo Codes: Some Results on Parallel Concatenated Coding Schemes", IEEE Trans. on Inform. Theory, Vol. 42, No. 2, pp.409-428, March 1996.
- [15]. S. Crozier, J. Lodge, P. Guinand and A. Hunt, "Performance of turbo codes with relatively prime and golden interleaving strategies", Proc. of 6th Int. Mobile Satellite Conf., pp. 268-275, Ottawa, Canada, June 1999.
- [16]. C. Berrou, Y. Saouter, C. Douillard, S. Kerouédan and M. Jézéquel, "Designing good permutations for turbo codes: towards a single model," in Proc. IEEE Int. Conf. Communications, Paris, France, June 2004.
- [17]. E. K. Hall, S. G. Wilson, "Stream-oriented turbo codes," IEEE Trans. Inform. Theory, vol.47, pp.1813-1831, July 2001.

- [18]. P. Robertson, E. Vilebrun, and P. Hoeher, "A comparison of optimal and sub-optimal MAP decoding algorithms operating in the log domain," in ZEEE Inc. on Communications (Seattle, WA, June 1995). pp. 1009-1013.
- [19]. A.Worm, P. Hoeher, and N. Wehn, "Turbo-Decoding without SNR Estimation," IEEE Communications Letters, vol. 4, no. 6, June 2000.
- [20]. M. J. Thul, F. Gilbert, and N. Wehn, "Concurrent Interleaving Architectures for High-Throughput Channel Coding," in Proc. 2003 Conference on Acoustics, Speech, and Signal Processing (ICASSP '03), Hong Kong, P.R.China, Apr. 2003, pp. 613-616.
- [21]. T. A. Summers and S. G. Wilson, "SNR mismatch and online estimation in turbo decoding," *IEEE Trans. Commun.*, vol. 46, no. 4, pp. 421-423, Apr. 1998.
- [22]. Zhongfeng Wang, Zhipei Chi and K. K. Parhi, "Area-efficient high speed decoding schemes for Turbo Decoders" IEEE Transactions on very large scale integration (VLSI) 2002.
- [23]. TH Tsai, CH Lin, AY Wu, "A memory-reduced log-MAP kernel for turbo decoder," Circuits and Systems, 2005. ISCAS 2005. IEEE International Symposium on Publication Date: 23-26 May 2005 On page(s): 1032- 1035 Vol. 2
- [24]. J.Ertel, J.Vogt and A.Finger, "A high throughput Turbo Decoder for an OFDM-based WLAN demonstrator," in proceedings of 5th International ITG Conference, Jan. 2004.
- [25]. D. Garrett, B. Xu and C. Nicol, "Energy Efficient Turbo Decoding for 3G Mobile," Proceedings of 2001 International Symposium on Low Power Electronic Design, pp 328-333.

- [26]. R. Dobkin, M. Peleg, and R. Ginosar, "Parallel interleaver design and VLSI architecture for low-latency MAP turbo decoders," *IEEE Trans. on VLSI Systems*, vol. 13, no. 4, pp. 427–438, Apr. 2005.
- [27]. J. Sun and O. Y. Takeshita, "Interleavers for turbo codes using permutation polynomials over integer rings," *IEEE Trans. on Inform. Theory*, vol. 51, no. 1, pp. 101–119, Jan. 2005
- [28]. J.Y. a~nez and J. Ram´irez, "The robust coloring problem," *European J. Oper. Res.*, vol. 148, no. 3, pp. 546–558, 2003.
- [29]. A.Lim and F.Wang, "Meta-heuristics for robust graph coloring problem," in *International Conference on Tools with Artificial Intelligence*. IEEE Computer Society, 2004, pp. 514–518.
- [30]. C.-H. Lin, and A.-Y. Wu, "Low-power traceback MAP decoding for double-binary convolutional turbo decoder," accepted for publication in *Proc. IEEE ISCAS 2008*, Seattle, USA, May 18-21, 2008.
- [31]. J.Ertel, J.Vogt, A.Finger, "A high throughput Turbo Decoder for an OFDM-based WLAN demonstrator," in *proceedings of 5th International ITG Conference*, Jan. 2004.
- [32]. Bougard, A. Giulietti, C. Desset, L. Van der Perre, and F. Catthoor, "A low power high speed parallel concatenated turbo-decoding architecture," in *Proc. Int. Symp. Turbo Codes and Related Topics*, Brest, France, Sep. 2003, pp. 511–514.