

國立交通大學

電子工程學系電子研究所

博士論文

運用平行架構及無競爭式交錯器之渦輪碼解碼器



Turbo Decoder with Parallel Architecture and
Contention-Free Interleaver

研究生：翁政吉

指導教授：張錫嘉

中華民國九十九年八月

運用平行架構及無競爭式交錯器之渦輪碼解碼器設計

Turbo Decoder with Parallel Architecture and Contention-Free
Interleaver

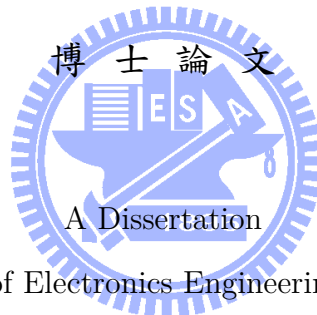
研 究 生：翁政吉

Student: Cheng-Chi Wong

指 導 教 授：張錫嘉 博士

Advisor: Dr. Hsie-Chia Chang

國立交通大學
電子工程學系電子研究所



Submitted to Department of Electronics Engineering & Institute of Electronics

College of Electrical and Computer Engineering

National Chiao Tung University

in Partial Fulfillment of the Requirements

for the Degree of Doctor of Philosophy

in

Electronics Engineering

August 2010

Hsinchu, Taiwan, Republic of China

中華民國九十九年八月

運用平行架構及無競爭式交錯器之渦輪碼解碼器設計

學生：翁政吉

指導教授：張錫嘉教授

國立交通大學電子工程學系電子研究所

摘要

本論文探討使用平行架構及無競爭式交錯器之渦輪碼解碼器來達到高速的資料輸出量。我們首先分析傳統平行架構的方法，整理各個方法的優缺點，也列出影響資料輸出量的關鍵因素。文中的討論主要是針對採用多個 soft-in soft-out (SISO) 處理器來對單一個接收到的字碼進行解碼這種技術所衍生出的議題。除了增加 SISO 處理器的數目之外，我們也結合了其他平行架構的方法來大幅提昇速度。然而，該方法將會導致相當高的硬體複雜度及降低處理器的運算效能。為了解決複雜度的問題，本論文介紹了兩種多層級的網路系統來負責平行解碼器中所有 SISO 處理器與記憶體之間的資料傳輸。這兩個網路系統分別支援採用 inter-block permutation (IBP) 交錯器及 quadratic permutation polynomial (QPP) 交錯器之渦輪碼解碼器。此多層級網路連結系統可以有效地降低實作時平行架構電路的繞線複雜度。至於另一個難題，則須透過調整處理器的執行程序來解決。我們提出兩套可防止資料相關性造成低運算效能的策略，並制定了各自對應的處理器執行程序。在這兩種高效能的程序中，其一是針對廣泛應用進行的設計，而另一則只能在特定的條件下使用。它們縮短了解碼流程中各個功能單元的閒置時間；因此，SISO 處理器的運算效能得到了改善。

基於上述之各項技術，我們實作了四個平行架構之渦輪碼解碼器。當中有兩個採用了 IBP 交錯器以及對應的多層級網路系統；它們皆使用了多個 SISO 處理器，且每個處理器在一個時脈週期中可處理複數筆資料；其中一個還利用了廣泛用途的高效能程序，讓硬體不會進入閒置的狀態。第三個解碼器則使用 QPP 交錯器和第二種網路連結系統；藉由這個裝置，再加上適當的控制電路，該解碼器最多可提供八個平行 SISO 處理器來支援在第三代合作伙伴計劃長期演進技術規格中所函括的全部區塊長度之渦輪

碼解碼。最後一個解碼器亦使用 QPP 交錯器和多層級網路；它比其他三個解碼器有更高的平行度；另一方面，因為符合另一個高效能程序的限制條件，它也具備最高的處理器運算效能；這個解碼器的資料輸出量可達到 1.4 Gb/s。實驗結果顯示我們所提的方法能夠得到預期的成效，也使得平行架構解碼器的速度有顯著的提昇。



Turbo Decoder with Parallel Architecture and Contention-Free Interleaver

Student: Cheng-Chi Wong

Advisor: Hsie-Chia Chang

Department of Electronics Engineering & Institute of Electronics

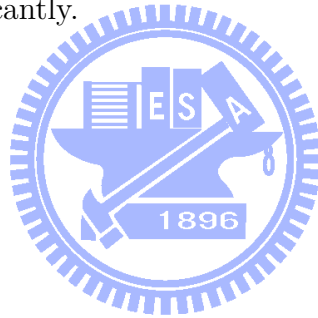
College of Electrical and Computer Engineering

National Chiao Tung University

ABSTRACT

This dissertation investigates the turbo decoders with parallel architecture and contention-free interleaver in pursuit of high throughput with reasonable cost. The benefits and disadvantages of conventional parallel schemes are examined; then the essential factors for throughput calculation are determined. Our discussions put emphasis on using multiple soft-in soft-out (SISO) decoders for single codeword. In addition to increasing the parallelism, the hybrid of parallel schemes is further applied for more speedup. However, the methodology leads to considerable complexity and inefficiency of processor. To reduce the complexity, we develop the multi-stage networks for the parallel data transmission in the turbo decoder. Two different types of apparatus are proposed for the designs using inter-block permutation (IBP) interleaver and quadratic permutation polynomial (QPP) interleaver, respectively. They can alleviate the routing congestion in the parallel design. To overcome the other difficulty, the processing schedule must be modified. We propose two different strategies to remove the data dependency and set their corresponding high-efficiency schedules. One of them is aimed for general application, whereas the other is designed for specific case. The inactive periods within the decoding flow are greatly shortened in these schedules. Hence, the efficiency of the SISO decoder can increase.

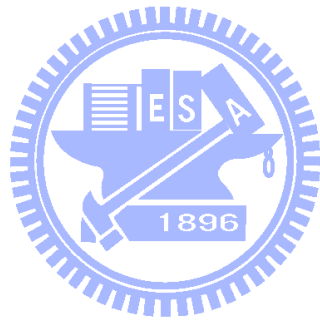
Four implemented works are presented in this dissertation. The multi-stage interconnection for IBP interleavers is applied to the first two parallel turbo decoders. Both the two designs contain multiple SISO decoders, each of which can process two or more symbols per cycle. One of them operates with the general high-efficiency schedule, and its idle time is completely removed. The third design exploits another interconnection for QPP interleavers. With such apparatus and appropriate control flow, it can use at most 8 SISO decoders to decodes all codeword blocks defined in the 3rd Generation Partnership Project Long Term Evolution standard. The remaining design also adopts QPP interleavers and the multi-stage network. It has higher parallelism than the other designs; moreover, its support of specific high-efficiency schedule results in the best efficiency. This design can achieve 1.4 Gb/s while decoding size-4096 blocks for 8 iterations. The implementation results reveal that the proposed methods work successfully in the parallel architecture and raise the throughput significantly.



誌

謝

在博士班的修業過程中，承蒙師長的指導、家人的支持以及夥伴們的協助，讓我能夠順利地抵達終點。非常感謝張錫嘉老師的提攜與提供的資源，給予我嘗試各式各樣研究方向的機會，即使當中有些結果不盡理想，依然獲得老師持續的支援；能夠身處於這樣的研究環境，是做為學生的幸福。接著要感謝的是我的家人，由於你們的伴隨和付出，我才能擁有堅定的力量邁步向前，去面對這段求學生涯中的一切挑戰。最後，由衷感激 OASIS 實驗室及 OCEAN 團隊的所有成員，在學業上，透過與大家的討論，得到了不少啟發，研究成果因此更趨完善；在生活中，也因為認識各位，締造了許多歡樂喜悅的回憶。



Contents

List of Tables	viii
List of Figures	ix
Chapter 1 Introduction	1
Chapter 2 Turbo Code	5
2.1 Overview of convolutional codes	6
2.1.1 Convolutional codes and encoders	6
2.1.2 The maximum a posteriori probability algorithm	10
2.2 Turbo code design	19
2.2.1 Parallel concatenated convolutional code	19
2.2.2 Iterative decoding flow	23
2.3 Practical turbo decoder architecture	27
Chapter 3 Parallel Architecture and Interleaver	37
3.1 Parallel turbo decoder architecture	37
3.1.1 Turbo decoder level	37
3.1.2 SISO decoder level	38
3.1.3 Trellis stage level	41
3.2 IBP interleaver and interconnection	45
3.3 QPP interleaver and interconnection	49
Chapter 4 High-Efficiency Processing Schedule	56
4.1 Interlaced half-iterations	56
4.2 Overlapping half-iterations	59
4.2.1 Interleaver constraints of such schedule	60
4.2.2 Performance and process of such schedule	65
Chapter 5 Implementation Results	70
5.1 Hybrid parallel design using IBP interleaver	70
5.2 Reconfigurable design for 3GPP LTE system	74
5.3 Full-efficiency design using QPP interleaver	81
Chapter 6 Conclusion	86
6.1 Summary	86
6.2 Future work	88
References	89

List of Tables

3.1	Equivalent gates count of two networks	54
4.1	Comparison between original and interlaced schedules	58
4.2	Properties of various QPP interleavers	65
5.1	Specifications of proposed turbo decoders with IBP interleaver	71
5.2	Throughput of selected modes with 275 MHz frequency	79
5.3	Comparison of different parallel designs for wireless application	80
5.4	Area of main components in different radix- $2^{\mathcal{P}\tau}$ SISO decoders	81
5.5	Comparison of different parallel turbo decoders using QPP interleaver . . .	85



List of Figures

1.1	Basic elements of a digital communication system	1
2.1	Basic framework of a turbo encoder	5
2.2	Block diagram of recursive convolutional encoder	6
2.3	The (2, 1, 2) recursive systematic convolutional encoder	8
2.4	State diagrams of the (2, 1, 2) recursive systematic convolutional code . . .	9
2.5	Trellis diagram of the (2, 1, 2) recursive systematic convolutional encoder .	9
2.6	Forward metric calculation and backward metric calculation	16
2.7	The MAP algorithm with sliding window technique	18
2.8	Trellis diagrams with different termination schemes	20
2.9	The turbo encoder specified in the 3GPP system	23
2.10	Basic framework of a turbo decoder	24
2.11	The bit error rate performance based on the iterative decoding	26
2.12	Practical architecture of a turbo decoder	27
2.13	Fundamental circuits for path metric and LLR calculations	29
2.14	Conventional SISO decoder and its schedule with three windows	31
2.15	Modified SISO decoder with β_d and its schedule with three windows	33
2.16	Modified SISO decoder without β_d and its schedule with three windows . .	35
3.1	Architecture with parallel turbo decoder level	38
3.2	Architecture with parallel SISO decoder level	39
3.3	An example of memory access in the parallel SISO decoder level	41
3.4	Architecture with parallel trellis stage level	42
3.5	High-radix ACS unit	43
3.6	An example of memory access in the parallel trellis stage level	44
3.7	An example of the contention-free interleaver with 4 sub-blocks.	46
3.8	Networks for connecting 4 SISO decoders and 4 memory modules.	47
3.9	Performance of turbo code with 3GPP interleaver and IBP interleaver . . .	49
3.10	Performance of turbo code with 3GPP interleaver and QPP interleaver . .	50
3.11	Fully-connected network for 8 SISO decoders and 8 memory modules . . .	51
3.12	Barrel-shift network for 8 SISO decoders and 8 memory modules	51
3.13	Parallel data transmission via the proposed network	53
4.1	Processes of independent codeword A and codeword B	57
4.2	Performance of turbo codes with various sizes and different QPP interleavers	66
4.3	Overlapping half-iterations for two windows ($\eta = 66.7\%$ with $\kappa = 2$) . . .	67
4.4	Overlapping half-iterations for four windows ($\eta = 100\%$ with $\kappa = 4$) . . .	68

5.1	Performance of the proposed Design-I and Design-II.	73
5.2	Photo of the proposed turbo decoders using IBP interleaver	73
5.3	Modified SISO decoder for the 3GPP LTE turbo decoder	75
5.4	Performance of the proposed 3GPP LTE turbo decoder with $N \leq 128$. . .	76
5.5	Performance of the proposed 3GPP LTE turbo decoder with small blocks .	77
5.6	Performance of the proposed 3GPP LTE turbo decoder with large blocks .	78
5.7	Photo of the proposed turbo decoder for 3GPP LTE application	79
5.8	Power consumption from measurement with 1.0 V and 275 MHz.	80
5.9	Performance of the proposed codes with various parameters	82
5.10	Performance with different formats of data width	83
5.11	Layout graph of the proposed design with 100% efficiency.	85



Chapter 1

Introduction

While sending digital information from a source to one or more destinations, several procedures will be involved to ensure efficient and reliable transmission. The elementary functional model of a digital communication system can be depicted as Fig. 1.1. In general, the information data are binary symbol sequences. The source encoder performs the compression by using a shorter bit sequence to replace the original sequence. The channel encoder further transforms these compressed data into a longer sequence. Some redundant symbols, also known as parity check symbols, are added here to minimize transmission errors. Then the modulator converts the channel coded data into analog signals which is suitable for transmitting over a channel. The modulated data will suffer from channel noise such as thermal noise, signal attenuation, distortion, and interference. In the receiver, the demodulator produces the quantized symbols based on the estimation of the transmitted data from channel outputs. Some of these received data might be erroneous due to the influence of channel. To correct the such errors, the channel decoder works on the coded data with parity check symbols. The outcome of an ideal channel decoder should be identical with the compressed data sequence. After the source decoder

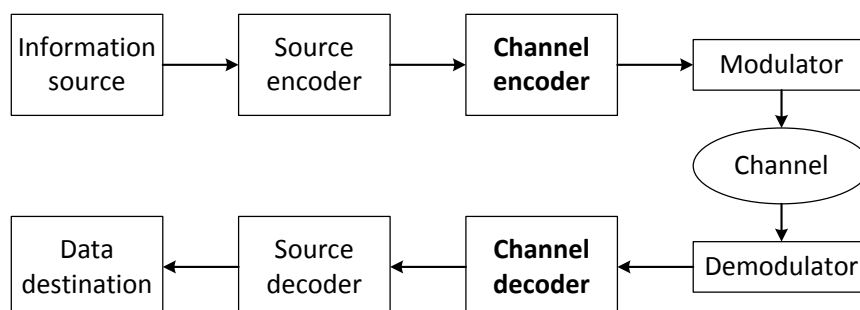


Figure 1.1: Basic elements of a digital communication system

decompresses such sequence, the information is transmitted to its destinations successfully.

The channel encoder and decoder are essential to the digital communication system for protecting the transmitted information against the channel noise. The study in such field is called *channel coding* or *forward error correction* (FEC). It begins with the landmark paper by C. E. Shannon in 1948 [1, 2]. Shannon's channel coding theorem indicated that arbitrary transmission can be asymptotically error-free by appropriate coding if the information rate is less than the channel capacity. With such theorem, the theoretical limits on performance can be calculated for various signaling schemes, rates, and channels. Lots of coding techniques are developed since then, and there are two major classes of codes: *block codes* and *convolutional codes* [3,4]. The ultimate objective is to approach the Shannon limit by a practical coding techniques. This investigation has lasted for several decades. In the 1990s, the advent of turbo codes in [5] and the rediscovery of low-density parity check codes in [6] made major breakthroughs in channel coding. Both of them can achieve the performance closed to Shannon limit at the expense of reasonable complexity. The outstanding features make these two codes receive a great deal of attentions.

Every coding technique has its own advantages, and it might meet the requirements in certain application. On the other hand, it also needs to face its unique design challenge. There are many researches attempting to solve different problems of each channel code. In this dissertation, we will only take turbo code into consideration. The basic model of turbo code is parallel concatenated convolutional codes separated by an interleaver [5]. Numerous publications had discussed various aspects of turbo code, including coding algorithm and circuit architecture. There is a thorough and comprehensive survey of turbo code in [4] and [7]. Some papers also give detailed overviews: general introduction of coding techniques, implementation issues of turbo decoder, applications in communication systems, and recent discoveries [8–11]. During the development of turbo code, several problems have been solved. Meanwhile, new design issues have arisen. One of them is to increase the throughput of decoding process. Turbo codes are adopted as the FEC techniques in the Third Generation Partnership Project (3GPP) standard and the Third

Generation Partnership Project 2 (3GPP2) standard [12,13]. The throughput requirement in either standard is less than 10 Mb/s. Some modern systems like IEEE 802.16 standard and 3GPP Long Term Evolution (LTE) standard also choose turbo codes [14, 15], but they would support much higher throughput. Therefore, the high-speed decoder is of practical interest nowadays.

In addition to the reliability, it is necessary to recover the erroneous data within a very short time. The utilization of parallel architecture is an intuitive method to achieve higher throughput. From [9], there are following levels of parallelism: using multiple decoders for multiple codewords, using multiple decoders for single codeword, and processing more data at each clock cycle. For all levels of parallelism, the component circuits of design will be duplicated so that they can deal with the received data simultaneously. However, the conventional structure of turbo code might pose obstacles to successful parallel processing. The extra hardware cost is also a critical issue. Moreover, the influence over error performance should be noticed. An improvement in one design factor might lead a degradation in another factor. To promise substantial quality in all aspects, the architecture design must connect the code structure, the decoding strategy, and the circuit complexity. This dissertation will put emphasis on these subjects. Several network topologies and processing flows are presented to support miscellaneous parallel turbo decoders with different interleavers. Three types of parallel designs are implemented. The first type exploits more than one parallelism level; the second type is aimed for 3GPP LTE application; and the third type can derive the maximum benefit of parallel processing. The corresponding results can prove the feasibility of the proposed designs.

The remainder of the dissertation is organized as follows. Chapter 2 includes a general introduction of turbo codes. It reviews the conventional code structure and decoding algorithm. In Chapter 3, it introduces the design issues of parallel turbo decoders at first. Then the interleavers with contention-free property are presented. We show that these interleaving methods can facilitate the implementation of parallel architecture. The major modifications to design are discussed in Chapter 4. To raise further the throughput

of parallel turbo decoders, some processing schedules with higher efficiency are proposed. Based on the approaches mentioned in these chapters, several parallel turbo decoders are implemented. Chapter 5 presents the implementation results and gives the state of the art. Finally, we conclude the dissertation in Chapter 6.



Chapter 2

Turbo Code

Turbo code, first proposed by Berrou [5] in 1993, is impressive with the near Shannon limit performance. Fig. 2.1 illustrates the general code structure of turbo code, where two component codes are separated by an interleaver. The two components are usually identical recursive convolutional encoders, so turbo code is also known as the parallel concatenated convolutional codes (PCCC). Besides, the interleaver can permute information sequence in a pseudo random way. The original data and the permuted data will be encoded by the two component encoders respectively. These parity checks and the original information, or referred to systematic data, form a complete turbo codeword. Although the permuted data are excluded from the codeword, they can be derived again at receiver by interleaving the received systematic data. The turbo decoder will deal with the two component codes alternatively. In [5], the maximum *a posteriori* probability (MAP) algorithm [16] is applied to calculate soft values of each component code. The soft output of one convolutional code will be treated as the *a priori* probability estimation

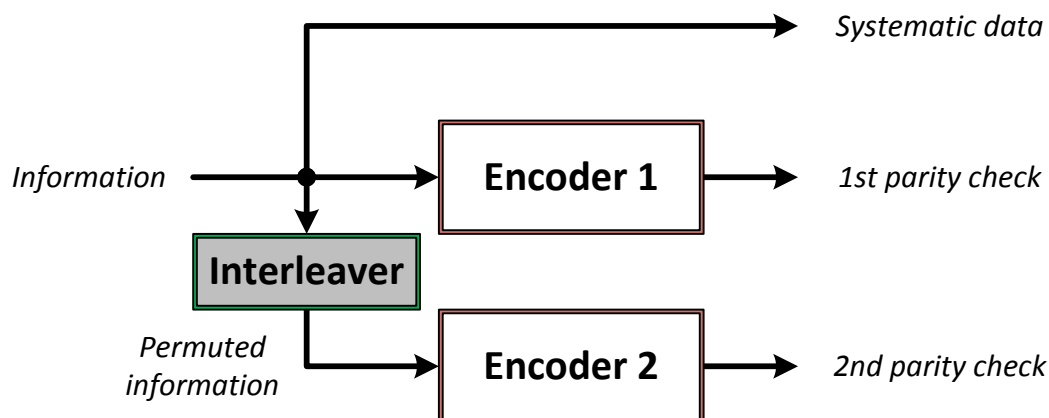


Figure 2.1: Basic framework of a turbo encoder

of the other one. Until certain criterion is satisfied, the decoding process will stop. With the random-like properties of turbo code, such iterative decoding method can be very efficient.

Such typical structure is widely utilized in research works and applications. Our discussions take only PCCC into consideration. A basic understanding of the component code would be helpful in this section. Therefore, a brief overview of convolutional code will be given at first. It involves the optimal and sub-optimal algorithms for soft output calculation. Then we focus on the turbo code, including the encoding and decoding flow, and there is a practical example of turbo encoder. Finally, the decoder architecture and implementation issues are discussed as well as the important factors in decoding speed.

2.1 Overview of convolutional codes

2.1.1 Convolutional codes and encoders

Convolutional codes were invented by Elias [17] in 1955, and the corresponding algebraic description were presented in [18] and [19]. One of its main ideas is the usage of memory. The encoder can keep previous information symbols in several storage elements, and it utilizes these data in memory and the current information symbols to generate convolutional codewords. For an (n, k, m) convolutional code, every k -tuple information symbol will be encoded into an n -tuple codeword symbol by the encoder with memory order m . The corresponding code rate R is k/n despite the length of the input sequence.

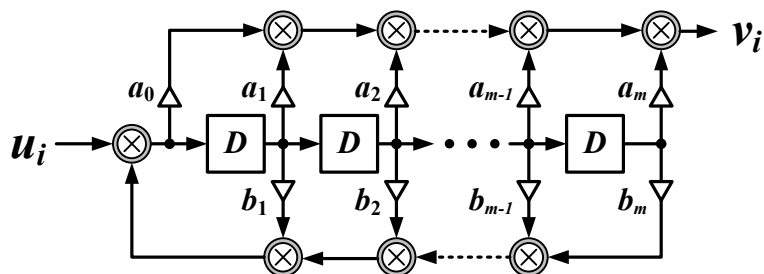


Figure 2.2: Block diagram of recursive convolutional encoder

Fig. 2.2 demonstrates the block diagram of recursive convolutional encoder. The $u_i = (u_i^{(0)}, u_i^{(1)}, \dots, u_i^{(k-1)})$ is the i -th input symbol, and the $v_i = (v_i^{(0)}, v_i^{(1)}, \dots, v_i^{(n-1)})$ is the i -th output symbol. In this figure, the delay elements denoted by \mathbf{D} construct a single input shift register. Besides, \mathbf{a}_x with $x = 0 \sim m$ and \mathbf{b}_y with $y = 1 \sim m$ determine the connections of circuit, and each of them is a $k \times n$ matrix that contains either 0 or 1. These binary values could be regarded as the weight of all data in memory during the encoding process. After the XOR gates perform modulo-2 additions of current input and weighted previous input, the codeword symbol is generated. We can use a fractional function $G(D)$ as (2.1) to express this encoder realization.

$$G(D) = \frac{\mathbf{a}_0 + \mathbf{a}_1 D + \dots + \mathbf{a}_m D^m}{1 + \mathbf{b}_1 D + \mathbf{b}_2 D^2 + \dots + \mathbf{b}_m D^m} \quad (2.1)$$

This function can transform the stream of k -tuple information into the stream of n -tuple codeword. We rewrite the transform function and get the generator matrix as (2.2). It is another well-known expression of the convolutional encoder. Each $g_i^{(j)}(D)$ adopts the format like (2.1), but all coefficients are binary number rather than matrix now.

$$G(D) = \begin{bmatrix} g_0^{(0)}(D) & g_0^{(2)}(D) & \cdots & g_0^{(n-1)}(D) \\ g_1^{(0)}(D) & g_1^{(2)}(D) & \cdots & g_1^{(n-1)}(D) \\ \vdots & \vdots & \ddots & \vdots \\ g_{k-1}^{(0)}(D) & g_{k-1}^{(2)}(D) & \cdots & g_{k-1}^{(n-1)}(D) \end{bmatrix} \quad (2.2)$$

When an informational symbol is fed into the encoder, the data in the memory will be updated, and the corresponding output symbol will be generated. The *state diagram* is one graphical representation to describe such behavior, and its basic elements are states and branches. The states are the contents of all delay elements. If the entire encoder has m memory, the total state number is 2^m . The branches stand for transitions caused by the input data. Generally, each state has 2^k incoming branches and 2^k outgoing branches. Here we use the (2, 1, 2) encoder as an example. The encoder realization with generator

matrix

$$G(D) = \begin{bmatrix} 1 & \frac{1 + D^2}{1 + D + D^2} \end{bmatrix} \quad (2.3)$$

is illustrated in Fig. 2.3, and the contents of this shift register structure are represented as d_0 and d_1 . All combinations of (d_0, d_1) are further symbolized as $S(j)$. There are $2^2 = 4$ states, and we let $S(0) = (0, 0)$, $S(1) = (0, 1)$, $S(2) = (1, 0)$, and $S(3) = (1, 1)$. Fig. 2.4(a) shows the state diagram of this example. The directed branches labeled with $u_i^{(0)}/v_i^{(0)}v_i^{(1)}$ indicate the changes in states and outputs of encoder with respect to input $u_i^{(0)}$. The state diagram can be also depicted as Fig. 2.4(b). This form would give a more clear illustration of the transitions among all states within one time unit.

Based on the state diagram, we can exploit the *trellis diagram* to trace all input sequences, codeword frames, and state transitions. This graphical representation is the concatenation of state diagrams throughout several successive time units. Fig. 2.5 shows the trellis diagram associated with the state diagram in Fig. 2.4; the solid branches and the dash branches indicate the input $u_i = 1$ and $u_i = 0$ respectively. Note that the binary input $u_i^{(0)}$ is simplified to u_i hereafter. We can observe all possible transitions within the first five time units ($t = 0 \sim 5$) in Fig. 2.5. The initial state is set to $S(0)$ ($(d_0, d_1) = (0, 0)$) because the contents of memory is always flushed before encoding process. If $(u_0, u_1, u_2, u_3, u_4)$ is $(1, 1, 0, 0, 0)$, then there will be a path going through $S(0), S(2), S(1), S(2), S(3), S(1)$ from $t = 0$ to $t = 5$, and the codeword sequence will be $(11, 10, 00, 01, 01)$. Every information sequence leads to a unique codeword as well as a

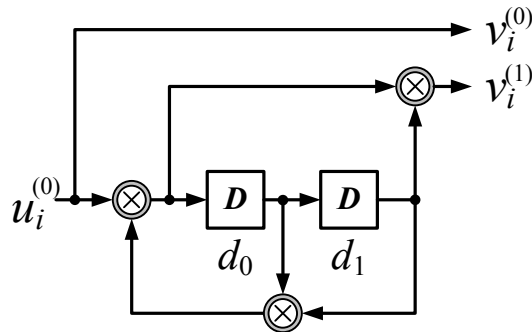


Figure 2.3: The $(2, 1, 2)$ recursive systematic convolutional encoder

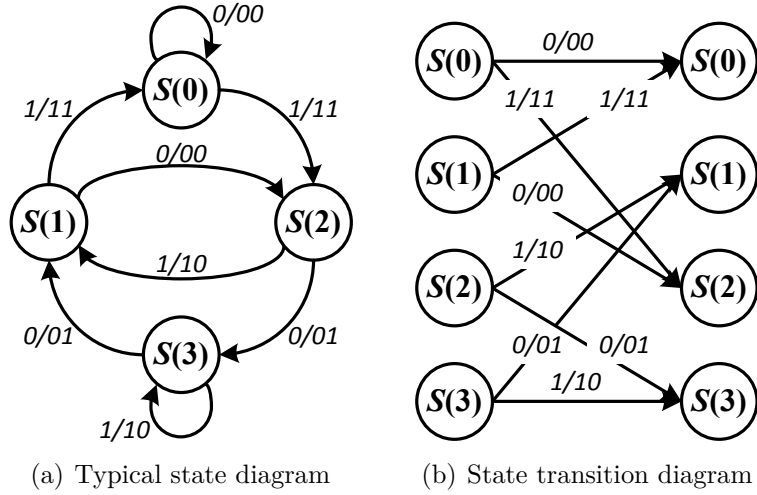


Figure 2.4: State diagrams of the (2, 1, 2) recursive systematic convolutional code

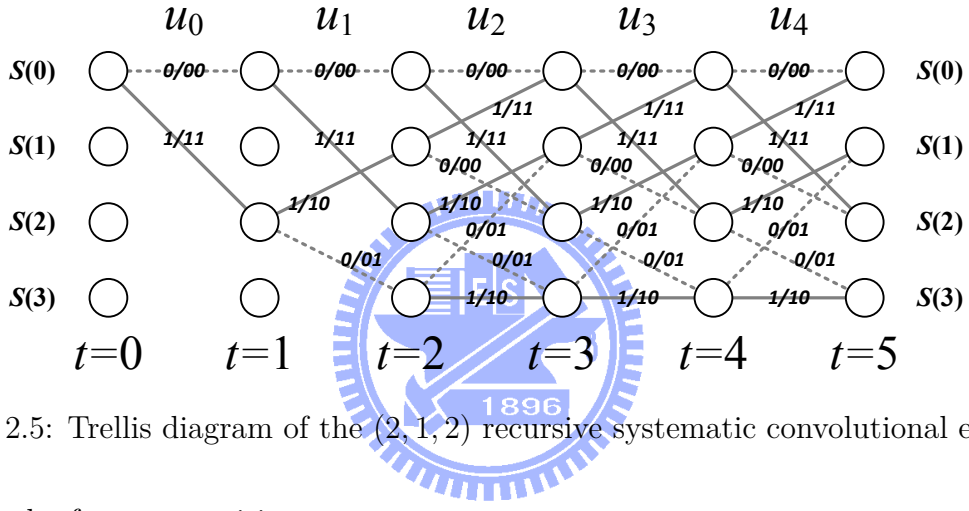


Figure 2.5: Trellis diagram of the (2, 1, 2) recursive systematic convolutional encoder

unique path of state transitions.

The trellis diagram is the basis of some decoding methods of convolutional codes. For a binary information sequence with length N , there are 2^N possible codewords and 2^N valid paths within the trellis. The Viterbi algorithm [20] is a maximum likelihood (ML) decoding technique. It uses a recursive approach to compute the difference between the received data sequence and all codewords, and then find the closest codeword out of 2^K candidates. The frame error probability can be minimized with the optimal ML algorithm for convolutional codes. The MAP algorithm [16] is another trellis-based decoding method. This technique calculates the probabilities of all state transitions from $t = i$ to $t = i + 1$. With the labels on branches, the most likely u_i can be determined. It minimizes the error probability of every information bit. In fact, the Viterbi algorithm and the MAP

algorithm have similar decoding results of convolutional code, either in frame error rate (FER) or bit error rate (BER). However, the MAP algorithm can get the soft value of every information bit. This feature is essential to turbo decoders, so more details about MAP algorithm are given later.

2.1.2 The maximum a posteriori probability algorithm

The maximum *a posteriori* probability (MAP) algorithm is a soft-output decoding algorithm. Since this technique is developed by Bahl, Cocke, Jelinek, and Raviv in 1974 [16], it is also termed *BCJR algorithm*. We have to make some assumptions about data transmission in advance. First, the code rate R is $1/n$; the input bit $u_t = (u_t^{(0)})$ will generate output symbol $v_t = (v_t^{(0)}, \dots, v_t^{(n-1)})$. Second, the binary phase shift keying (BPSK) modulation is applied to map each binary symbol into one of the modulation signal set. The coded signal $v_t^{(j)}$ will be mapped into modulated signal $y_t^{(j)}$ as (2.4) for $j = 0 \sim (n-1)$.

$$y_t^{(j)} = (-1)^{v_t^{(j)}} = \begin{cases} +1 & \text{if } v_t^{(j)} = 0 \\ -1 & \text{if } v_t^{(j)} = 1 \end{cases} \quad (2.4)$$

Third, the channel is an additive white-Gaussian-noise (AWGN) channel. While receiving the data $r_t = (r_t^{(0)}, \dots, r_t^{(n-1)})$ from channel, each $r_t^{(j)}$ can be viewed as the summation of modulated signal $y_t^{(j)}$ and the zero-mean white Gaussian noise $n_t^{(j)}$.

$$r_t^{(j)} = y_t^{(j)} + n_t^{(j)}. \quad (2.5)$$

The variance of $n_t^{(j)}$ is σ^2 , which is determined by symbol signal-to-noise ratio (SNR). The symbol SNR is usually denoted by E_s/N_0 . We can also use the bit SNR, E_b/N_0 , to calculate σ^2 due to $E_s = RE_b$. The transition probabilities of a size- N sequence are defined by

$$\Pr\{r_0, \dots, r_{N-1} \mid y_0, \dots, y_{N-1}\} \triangleq \prod_{t=0}^{N-1} \Pr\{r_t^{(0)}, \dots, r_t^{(n-1)} \mid y_t^{(0)}, \dots, y_t^{(n-1)}\}, \quad (2.6)$$

where

$$\Pr\{r_t^{(0)}, \dots, r_t^{(n-1)} \mid y_t^{(0)}, \dots, y_t^{(n-1)}\} \triangleq \prod_{j=0}^{n-1} \Pr\{r_t^{(j)} \mid y_t^{(j)}\} \quad (2.7)$$

and

$$\Pr\{r_t^{(j)} \mid y_t^{(j)}\} \triangleq \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{1}{2\sigma^2} \left(r_t^{(j)} - y_t^{(j)}\right)^2\right). \quad (2.8)$$

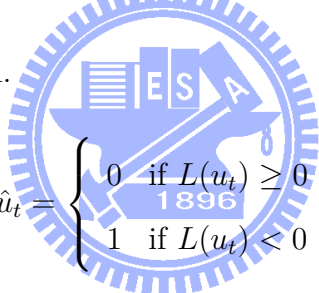
Given the received data sequence from channel, the MAP algorithm can generate the *a posteriori* probability (APP) of each transmitted symbol as

$$\Pr\{u_t \mid r_0, \dots, r_{N-1}\} \quad (2.9)$$

The APP is further used to compute the log-likelihood ratio (LLR)

$$L(u_t) \triangleq \ln \frac{\Pr\{u_t = 0 \mid r_0, \dots, r_{N-1}\}}{\Pr\{u_t = 1 \mid r_0, \dots, r_{N-1}\}} \quad (2.10)$$

and then make the hard decision.

$$\hat{u}_t = \begin{cases} 0 & \text{if } L(u_t) \geq 0 \\ 1 & \text{if } L(u_t) < 0 \end{cases} \quad (2.11)$$


The LLR can be rewritten as (2.12) by utilizing the characteristic of conditional probability.

$$\begin{aligned} L(u_t) &= \ln \frac{\Pr\{u_t = 0; r_0, \dots, r_{N-1}\} / \Pr\{r_0, \dots, r_{N-1}\}}{\Pr\{u_t = 1; r_0, \dots, r_{N-1}\} / \Pr\{r_0, \dots, r_{N-1}\}} \\ &= \ln \frac{\Pr\{u_t = 0; r_0, \dots, r_{N-1}\}}{\Pr\{u_t = 1; r_0, \dots, r_{N-1}\}} \end{aligned} \quad (2.12)$$

The $u_t = 0$ and $u_t = 1$ have their respective state transitions in the trellis diagram, so we have the equivalence as

$$\Pr\{u_t\} = \sum_{(S_t, S_{t+1})} \Pr\{u_t; S_t, S_{t+1}\}. \quad (2.13)$$

Note that S_t is the state at time t of the trellis diagram, and (S_t, S_{t+1}) represents the state transition from S_t to S_{t+1} . With (2.13), the LLR calculation is modified to

$$\begin{aligned} L(u_t) &= \ln \frac{\sum_{(S_t, S_{t+1})} \Pr\{u_t = 0; S_t, S_{t+1}; r_0, \dots, r_{N-1}\}}{\sum_{(S_t, S_{t+1})} \Pr\{u_t = 1; S_t, S_{t+1}; r_0, \dots, r_{N-1}\}} \\ &= \ln \frac{\sum_{(u_t=0; S_t, S_{t+1})} \Pr\{S_t, S_{t+1}; r_0, \dots, r_{N-1}\}}{\sum_{(u_t=1; S_t, S_{t+1})} \Pr\{S_t, S_{t+1}; r_0, \dots, r_{N-1}\}}. \end{aligned} \quad (2.14)$$

The joint probability $\Pr\{S_t, S_{t+1}; r_0, \dots, r_{N-1}\}$ is involved in the LLR calculation. If there is no transition from S_t to S_{t+1} , this probability will be zero. Otherwise, it can be decomposed as (2.15) with Bayes's rule.

$$\begin{aligned} \Pr\{S_t, S_{t+1}; r_0, \dots, r_{N-1}\} &= \Pr\{S_t; r_0, \dots, r_{t-1}\} \\ &\quad \times \Pr\{S_{t+1}; r_t \mid S_t; r_0, \dots, r_{t-1}\} \\ &\quad \times \Pr\{r_{t+1}, \dots, r_{N-1} \mid S_t, S_{t+1}; r_0, \dots, r_{t-1}, r_t\} \end{aligned} \quad (2.15)$$

We can simplify the two conditional probabilities in (2.15) by removing the redundant conditions. Since the S_t is given, the transition to S_{t+1} with r_t is independent of previous data (r_0, \dots, r_{t-1}) . Similarly, the condition S_{t+1} is sufficient for the last conditional probability.

$$\Pr\{S_{t+1}; r_t \mid S_t; r_0, \dots, r_{t-1}\} = \Pr\{S_{t+1}; r_t \mid S_t\} \quad (2.16)$$

$$\Pr\{r_{t+1}, \dots, r_{N-1} \mid S_t, S_{t+1}; r_0, \dots, r_{t-1}, r_t\} = \Pr\{r_{t+1}, \dots, r_{N-1} \mid S_{t+1}\} \quad (2.17)$$

Then the factorization of $\Pr\{S_t, S_{t+1}; r_0, \dots, r_{N-1}\}$ becomes

$$\Pr\{S_t; r_0, \dots, r_{t-1}\} \times \Pr\{S_{t+1}; r_t \mid S_t\} \times \Pr\{r_{t+1}, \dots, r_{N-1} \mid S_{t+1}\}. \quad (2.18)$$

Now we define three functions:

$$\alpha(S_t) = \ln \Pr\{S_t; r_0, \dots, r_{t-1}\} \quad (2.19)$$

$$\gamma(S_t, S_{t+1}) = \ln \Pr\{S_{t+1}; r_t \mid S_t\} \quad (2.20)$$

$$\beta(S_t) = \ln \Pr\{r_t, \dots, r_{N-1} \mid S_t\}, \quad (2.21)$$

where $\alpha(S_t)$ is named *forward metric*, $\gamma(S_t, S_{t+1})$ is *branch metric*, and $\beta(S_t)$ is *backward metric*. Thus (2.18) can be rewritten as

$$\Pr\{S_t, S_{t+1}; r_0, \dots, r_{N-1}\} = \exp(\alpha(S_t)) \times \exp(\gamma(S_t, S_{t+1})) \times \exp(\beta(S_{t+1})). \quad (2.22)$$

By substituting (2.22) for the APP in (2.14), the LLR will become

$$L(u_t) = \ln \left[\sum_{(u_t=0; S_t, S_{t+1})} \exp(\alpha(S_t) + \gamma(S_t, S_{t+1}) + \beta(S_{t+1})) \right] - \ln \left[\sum_{(u_t=1; S_t, S_{t+1})} \exp(\alpha(S_t) + \gamma(S_t, S_{t+1}) + \beta(S_{t+1})) \right] \quad (2.23)$$

On the other hand, the definition of (2.19) is extended to be

$$\begin{aligned} \exp(\alpha(S_t)) &= \Pr\{S_t; r_0, \dots, r_{t-1}\} \\ &= \sum_{S_{t-1}} \Pr\{S_{t-1}, S_t; r_0, \dots, r_{t-1}\} \\ &= \sum_{S_{t-1}} \Pr\{S_{t-1}; r_0, \dots, r_{t-2}\} \Pr\{S_t; r_{t-1} \mid S_{t-1}; r_0, \dots, r_{t-2}\} \\ &= \sum_{S_{t-1}} \Pr\{S_{t-1}; r_0, \dots, r_{t-2}\} \Pr\{S_t; r_{t-1} \mid S_{t-1}\} \\ &= \sum_{S_{t-1}} \exp(\alpha(S_{t-1})) \times \exp(\gamma(S_{t-1}, S_t)). \end{aligned} \quad (2.24)$$

Then we compute the natural logarithm of both sides in (2.24).

$$\alpha(S_t) = \ln \sum_{S_{t-1}} \exp(\alpha(S_{t-1}) + \gamma(S_{t-1}, S_t)) \quad (2.25)$$

The above equation just needs $\alpha(S_{t-1})$ and $\gamma(S_{t-1}, S_t)$ to get $\alpha(S_t)$. That is, the calculation of all $\alpha(S_t)$ with $0 \leq t \leq N$ is a forward recursion. Such recursive method needs an appropriate initial condition. If the encoder starts from $S(0)$, the condition will be

$$\alpha(S_0) = \begin{cases} 0 & \text{if } S_0 = S(0) \\ -\infty & \text{if } S_0 \neq S(0) \end{cases} \quad (2.26)$$

We can make the similar deduction about the backward metric $\beta(S_t)$. The first step is

$$\begin{aligned} \exp(\beta(S_t)) &= \Pr\{r_t, \dots, r_{N-1} \mid S_t\} \\ &= \sum_{S_{t+1}} \Pr\{S_{t+1}; r_t, \dots, r_{N-1} \mid S_t\} \\ &= \sum_{S_{t+1}} \Pr\{S_{t+1}; r_t \mid S_t\} \Pr\{r_{t+1}, \dots, r_{N-1} \mid S_t, S_{t+1}; r_t\} \\ &= \sum_{S_{t+1}} \Pr\{S_{t+1}; r_t \mid S_t\} \Pr\{r_{t+1}, \dots, r_{N-1} \mid S_{t+1}\} \\ &= \sum_{S_{t+1}} \exp(\gamma(S_t, S_{t+1})) \times \exp(\beta(S_{t+1})). \end{aligned} \quad (2.27)$$

After the computation of natural logarithm, (2.27) changes to

$$\beta(S_t) = \ln \sum_{S_{t+1}} \exp(\gamma(S_t, S_{t+1}) + \beta(S_{t+1})) \quad (2.28)$$

The calculation of $\beta(S_t)$ needs $\beta(S_{t+1})$ and $\gamma(S_t, S_{t+1})$, and it is a backward recursion to find all $\beta(S_t)$ with $0 \leq t \leq K$. If the encoder terminates at $S(0)$, the initial condition is

$$\beta(S_N) = \begin{cases} 0 & \text{if } S_N = S(0) \\ -\infty & \text{if } S_N \neq S(0) \end{cases} \quad (2.29)$$

Furthermore, the branch metric in (2.20) can be

$$\begin{aligned}
\exp(\gamma(S_t, S_{t+1})) &= \Pr\{S_{t+1}; r_t \mid S_t\} \\
&= \frac{\Pr\{S_t, S_{t+1}; r_t\}}{\Pr\{S_t\}} \\
&= \frac{\Pr\{S_t, S_{t+1}\}}{\Pr\{S_t\}} \times \frac{\Pr\{S_t, S_{t+1}; r_t\}}{\Pr\{S_t, S_{t+1}\}} \\
&= \Pr\{S_{t+1} \mid S_t\} \Pr\{r_t \mid S_t, S_{t+1}\} \\
&= \Pr\{u_t\} \Pr\{r_t \mid y'_t\}, \tag{2.30}
\end{aligned}$$

where u_t and y'_t are the corresponding information bit and modulated output on the state transition (S_t, S_{t+1}) . To find the $\Pr\{u_t\}$, we need the *a priori* information represented by (2.31). For simplicity, we use the modulated signal $u'_t = +1$ to replace $u_t = 0$ and $u'_t = -1$ to replace $u_t = 1$.

$$L_a(u'_t) \triangleq \ln \frac{\Pr\{u'_t = +1\}}{\Pr\{u'_t = -1\}} \tag{2.31}$$

We utilize $L_a(u'_t)$ to calculate the *a priori* probability

$$\Pr\{u'_t = \pm 1\} = \frac{e^{\pm L_a(u'_t)}}{1 + e^{\pm L_a(u'_t)}} = \frac{e^{-L_a(u'_t)/2}}{1 + e^{-L_a(u'_t)}} e^{u'_t L_a(u'_t)/2} = A_t e^{u'_t L_a(u'_t)/2}, \tag{2.32}$$

where A_t is independent of the actual value of u'_t . With (2.7) and (2.8), the $\Pr\{r_t \mid y'_t\}$ can be modified to

$$\begin{aligned}
\Pr\{r_t \mid y'_t\} &= \prod_{j=0}^{n-1} \left[\frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(r_t^{(j)} - y'_t^{(j)})^2}{2\sigma^2}} \right] \\
&= \left(\frac{1}{\sigma\sqrt{2\pi}} \right)^n \times e^{-\frac{\sum_{j=0}^{n-1} [(r_t^{(j)})^2 + (y'_t^{(j)})^2]}{2\sigma^2}} \times e^{\frac{\sum_{j=0}^{n-1} [r_t^{(j)} \times y'_t^{(j)}]}{\sigma^2}} \\
&= B_t \times e^{\frac{L_c \sum_{j=0}^{n-1} [r_t^{(j)} \times y'_t^{(j)}]}{2}}. \tag{2.33}
\end{aligned}$$

Because $r_t^{(j)}$ is the same for all state transitions from time t to $(t+1)$ and $y'_t^{(j)} = \pm 1$, the B_t is a constant. In addition, the channel reliability value L_c is $2/\sigma^2$, and it will be

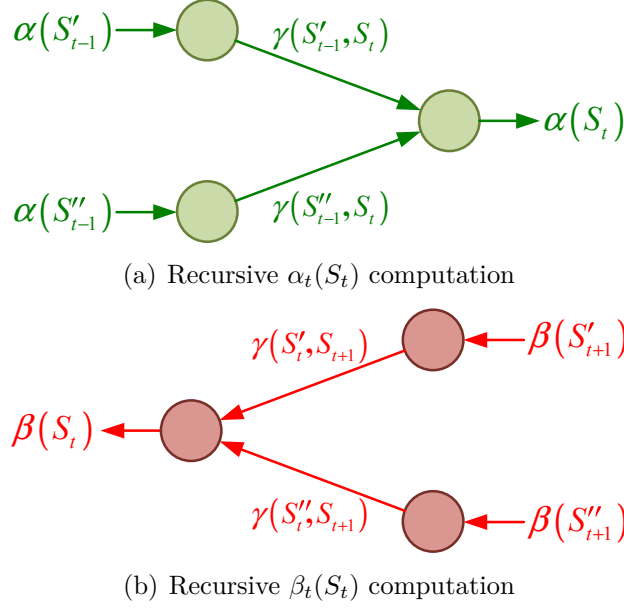


Figure 2.6: Forward metric calculation and backward metric calculation

$4E_s/N_0$ for the AWGN channel [21]. The (2.32) and (2.33) change the branch metric to

$$\gamma(S_t, S_{t+1}) = \ln A_t + \ln B_t + \frac{1}{2}u'_t L_a(u'_t) + \frac{1}{2}L_c \sum_{j=0}^{n-1} [r_t^{(j)} \times y_t^{(j)}]. \quad (2.34)$$

In fact, both A_t and B_t will be canceled out in the LLR of (2.23). We can drop A_t and B_t in the expression of $\gamma(S_t, S_{t+1})$ and get

$$\gamma(S_t, S_{t+1}) = \frac{1}{2}u'_t L_a(u'_t) + \frac{1}{2}L_c \sum_{j=0}^{n-1} [r_t^{(j)} \times y_t^{(j)}]. \quad (2.35)$$

Consequently, the MAP algorithm needs the forward metrics, backward metrics, and branch metrics to get all the LLR. It will initialize $\alpha(S_0)$ and $\beta(S_N)$ at first. After receiving the codeword symbol r_t , the decoder can derive $\gamma(S_t, S_{t+1})$ of each branch in the trellis diagram. Then the decoder use these branch metrics to calculate $\alpha(S_t)$ and $\beta(S_t)$ in a recursive way. The respective computations of forward metrics and backward metrics are described graphically in Fig. 2.6. Here we let each state at time t have two incoming branches from different states, S'_{t-1} and S''_{t-1} , and two outgoing branches to different states, S'_{t+1} and S''_{t+1} . As $\alpha(S_t)$ and $\beta(S_{t+1})$ are available, the LLR $L(u_t)$ and

decision \hat{u}_t can be further determined.

The MAP algorithm is often approximated to Log-MAP or Max-Log-MAP algorithm in order to reduce implementation complexity [22]. We use the Jacobian function [23]

$$\ln(e^{x_1} + e^{x_2}) \triangleq \max^*(e^{x_1}, e^{x_2}) = \max(e^{x_1}, e^{x_2}) + \ln(1 + e^{-|x_1 - x_2|}) \quad (2.36)$$

and its extension

$$\begin{aligned} \ln(e^{x_1} + e^{x_2} + e^{x_3} + \dots + e^{x_q}) &= \max^*(e^{x_1}, e^{x_2}, e^{x_3}, \dots, e^{x_q}) \\ &= \max^*(\dots \max^*(\max^*(x_1, x_2), x_3) \dots, x_q) \end{aligned} \quad (2.37)$$

to replace original computations in (2.23), (2.25), and (2.28). The value of $\ln(1 + e^{-|x_1 - x_2|})$ can be found via a lookup table in a practical design. If the logarithmic term is very small, it could be omitted. Then the normal max operations could replace the \max^* operations.

$$\max^*(e^{x_1}, e^{x_2}) \approx \max(e^{x_1}, e^{x_2}). \quad (2.38)$$

We express both (2.25) and (2.28) in a simpler form:

$$\alpha(S_t) = \max_{S_{t-1}} [\alpha(S_{t-1}) + \gamma(S_{t-1}, S_t)] \quad (2.39)$$

$$\beta(S_t) = \max_{S_{t+1}} [\beta(S_{t+1}) + \gamma(S_t, S_{t+1})]. \quad (2.40)$$

Thus, the $L(u_t)$ alters:

$$\begin{aligned} L(u_t) &= \max_{(S_t, S_{t+1}): u_t=0} [\alpha(S_t) + \gamma(S_t, S_{t+1}) + \beta(S_{t+1})] \\ &\quad - \max_{(S_t, S_{t+1}): u_t=1} [\alpha(S_t) + \gamma(S_t, S_{t+1}) + \beta(S_{t+1})] \end{aligned} \quad (2.41)$$

If the algorithm still uses \max^* operations, it is named Log-MAP algorithm, and its performance is equivalent to that of MAP algorithm. The approximation in (2.38) leads

to the Max-Log-MAP algorithm using max operations. Because the logarithmic term in (2.36) is discarded, there will be some performance degradation. However, the Max-Log-MAP algorithm contains only addition, comparison, and selection functions. It is more suitable for circuit implementation. Moreover, its recursive metric calculations is similar to the critical add-compare-select (ACS) operation of Viterbi algorithm [20]. As a result, the decoder with Max-Log-MAP algorithm can adopt many techniques which originally support Viterbi decoder.

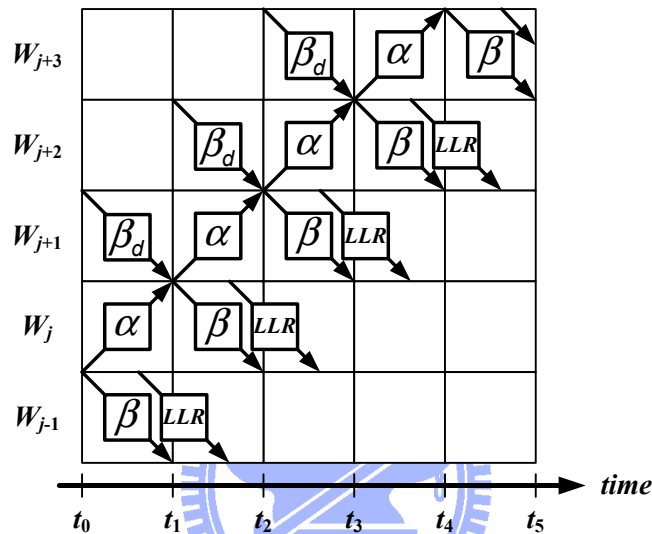


Figure 2.7: The MAP algorithm with sliding window technique

The optimal or suboptimal MAP algorithm will encounter another difficulty in implementation while the block size N is large. The data (r_0, \dots, r_{N-1}) are usually sent to the decoder in ascending order, and then the forward metrics can be derived soon with the initial condition $\alpha(S_0)$. After the whole sequence has been received, the recursive calculation of backward metrics can start from its only known state S_N . Both $\alpha(S_t)$ and $\beta(S_{t+1})$ are necessary to calculate $L(u_t)$ with $t = 0 \sim (N - 1)$. Hence, all forward metrics must be kept during such decoding procedure. The memory requirement would be considerable, and the decoder would become impractical. To reduce this hardware overhead, the sliding window technique [24, 25] exploits a dummy calculation to provide reliable metric initialization at any time. As shown in Fig. 2.7, the codeword block is divided into $\lceil N/L \rceil$ windows of length L , and the W_j stands for the j -th window. The dummy

backward recursion β_d is an operation similar to the β . Except the last window, the initial β_d within each window is unknown. We set the β_d of all 2^m states in the $(j+1)$ -th window equally probable:

$$\beta_d(S_{(j+2)L}) = \ln \frac{1}{2^m} \quad \text{for } S_{(j+2)L} \in \{S(0), S(1), \dots, S(2^m - 1)\} \quad (2.42)$$

The β_d in the last window is the same as $\beta(S_N)$. As the β_d process in the $(j+1)$ -th window finishes, the initial metrics $\beta(S_{(j+1)L})$ in the j -th window are available for the β recursion. The exact operations from t_0 to t_1 in Fig. 2.7 can be expressed as follows:

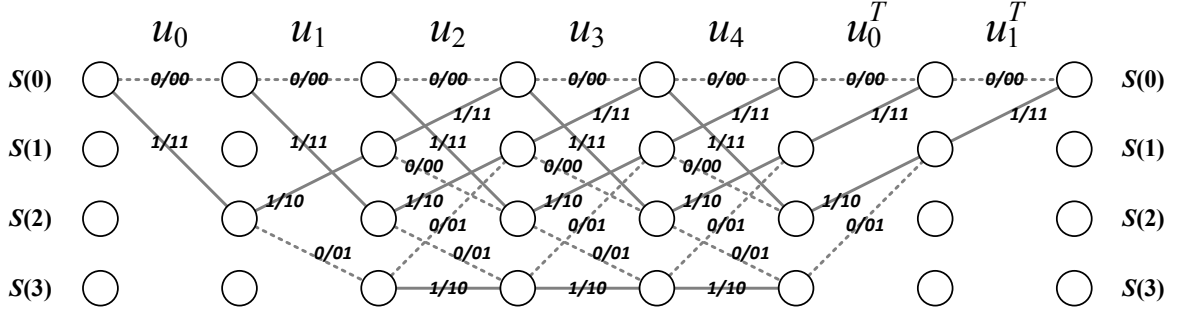
$$\left\{ \begin{array}{l} \beta_d: \quad S_{(j+2)L} \rightarrow S_{(j+2)L-1} \rightarrow \dots \rightarrow S_{(j+1)L+1} \rightarrow S_{(j+1)L} \\ \alpha: \quad S_{(j+0)L} \rightarrow S_{(j+0)L+1} \rightarrow \dots \rightarrow S_{(j+1)L-1} \rightarrow S_{(j+1)L} \\ \beta: \quad S_{(j+0)L} \rightarrow S_{(j+0)L-1} \rightarrow \dots \rightarrow S_{(j-1)L+1} \rightarrow S_{(j-1)L} \\ LLR: \quad u_{(j+0)L} \rightarrow u_{(j+0)L-1} \rightarrow \dots \rightarrow u_{(j-1)L+1} \rightarrow u_{(j-1)L} \end{array} \right. \quad (2.43)$$

During the β_d operation of the $(j+1)$ -th window, the decoder performs concurrently the following operations: the α of the j -th window, the β and the $L(u_t)$ of the $(j-1)$ -th window. The calculation of $L(u_t)$ is possible because all α results of the $(j-1)$ -th window had been completed and stored in the memory. We also use the same memory to store the α of the j -th window. In the subsequent process between t_1 and t_2 , the $L(u_t)$ of the j -th window can be derived with the α in the memory, the β in computing, and the corresponding branch metrics. Instead of keeping $(N \times 2^m)$ α metrics, the decoder with sliding window technique requires a smaller memory for $(L \times 2^m)$ α metrics.

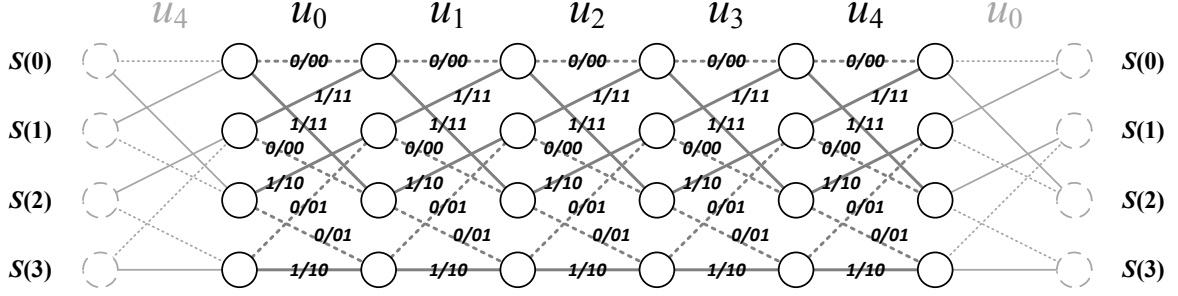
2.2 Turbo code design

2.2.1 Parallel concatenated convolutional code

The turbo code can utilize an iterative decoding process to achieve the near Shannon limit performance [5, 26]. The typical encoder structure, which consists of two recursive



(a) Trellis with tail bits



(b) Trellis with tail-biting technique

Figure 2.8: Trellis diagrams with different termination schemes

convolutional encoders and one interleaver, is given in Fig. 2.1. One of the convolutional encoders encodes the information in original order, while the other encodes the information in interleaved order. If the first parity and the second check sequences have N_1 and N_2 bits respectively, the overall code rate will be

$$R = \frac{N}{N + N_1 + N_2}. \quad (2.44)$$

In the encoding process of turbo code, the trellis termination is an important issue. It will provide the information of the beginning state S_0 and the ending state S_N , so the trellis-based decoding algorithm can get reliable initial conditions: $\alpha(S_0)$ and $\beta(S_N)$. There is one common termination method whose trellis is starting from all-zero state and ending at all-zero state. The encoder can get $S_0 = S(0)$ by flushing the memory contents, but the S_N is undetermined due to various input sequences. To force the ending state to $S(0)$, several tail bits are attached after encoding the size- N information block. In a recursive convolutional encoder, the input is replaced with the duplicated feedback path

during the termination flow. Then we can make certain that the content of the first delay element will be 0. This 0 will further propagate to the succeeding delay elements, and it takes m additional input bits to reset all m delay elements. Since these redundant bits and their corresponding parity check bits must be sent to decoder, it will cause the rate loss, especially for short block sizes. The tail-biting technique is another trellis termination method [27]. It let the trellis begin and end at the same state, that is, $S_0 = S_N$. The information block has to be encoded twice to achieve this target. In the first encoding process, the trellis diagram starts from $S(0)$ and stops at one of the 2^m states, symbolized by S'_N . According to the last state S'_N , the block size N , and the code generator matrix $G(D)$, we can compute an appropriate state S''_0 for the second encoding process. As the S_0 of the trellis is set to S''_0 , encoding the same information will make the S_N equal S''_0 , too. The trellis with tail-biting technique can be viewed as a loop. Although the decoder lacks the exact S_0 and S_N , it can get reliable $\alpha(S_0)$ from the final received data and get reliable $\beta(S_0)$ from the initial received data. Here the encoder in Fig. 2.3 with $N = 5$ is used to illustrate these methods. Fig. 2.8 depicts the two different trellis diagrams. We assume that the information $(u_0, u_1, u_2, u_3, u_4)$ is $(1, 0, 0, 1, 1)$. In Fig. 2.8(a), this input sequence can generate a path stopping at $S(2)$. The tail bits (u_0^T, u_1^T) , derived from $d_0 \otimes d_1$, force the path back to $S(0)$ and decrease the code rate to $5/14$. In Fig. 2.8(b), the initial state in the second encoding process will be set as

$$\left\{ \begin{array}{ll} S''_0 = 0 & \text{if } S'_N = 0 \\ S''_0 = 3 & \text{if } S'_N = 1 \\ S''_0 = 1 & \text{if } S'_N = 2 \\ S''_0 = 2 & \text{if } S'_N = 3 \end{array} \right.$$

While starting from $S(1)$, the above input sequence causes the trellis also end at $S(1)$. No extra bits are introduced, so the code rate is still $1/2$.

The consistent code structure and the interleaving method are critical to the turbo code performance. We can evaluate the performance bound with some fundamental properties

of turbo code. The first property is the *Hamming weight*; it is the number of nonzero components of each codeword. The second property, *Hamming distance*, is the number of locations in which any two codewords differ. Among all Hamming distances of the code, the smallest one is called *free distance*, also denoted by d_{free} . Due to the linearity of this code, d_{free} is equivalent to the minimum weight of all non-zero codewords. The third property is the weight distribution or the distance spectrum. For a turbo code with information block size N and code rate R over the AWGN channel, its codeword error probability $P(E)$ and bit error probability $P_b(E)$ are bounded by [7, 28]

$$P(E) \leq \sum_{d \geq d_{\text{free}}} A_d Q \left(\sqrt{\frac{2dRE_b}{N_0}} \right) \quad (2.45)$$

$$P_b(E) \leq \sum_{d \geq d_{\text{free}}} \frac{B_d}{N} Q \left(\sqrt{\frac{2dRE_b}{N_0}} \right), \quad (2.46)$$

where A_d is the total number of codewords of weight d , and B_d is the total information weight of all codewords of weight d . The function $Q(x)$ is the Gaussian error integral:

$$Q(x) \triangleq \frac{1}{\sqrt{2\pi}} \int_x^{\infty} \exp\left(-\frac{x^2}{2}\right) dx. \quad (2.47)$$

When the SNR is high, these bounds become very tight, and they are usually dominated by the first few terms of (2.45) and (2.46). Therefore, we prefer the constituent codes and interleavers that can lead to large d_{free} and good distance spectrum.

To meet various requirements, every turbo code specification contains specific component code, block sizes, trellis termination scheme, and interleaving method. Here we introduce the turbo encoder specified both in the 3GPP standard [12], and Fig. 2.9 demonstrates its architecture. Its constituent code has the following generator matrix:

$$G(D) = \left[\begin{array}{c|c} 1 & \frac{1 + D + D^3}{1 + D^2 + D^3} \end{array} \right]. \quad (2.48)$$

The block sizes range from 40 to 5144. A modified block interleaver is used to permuted the

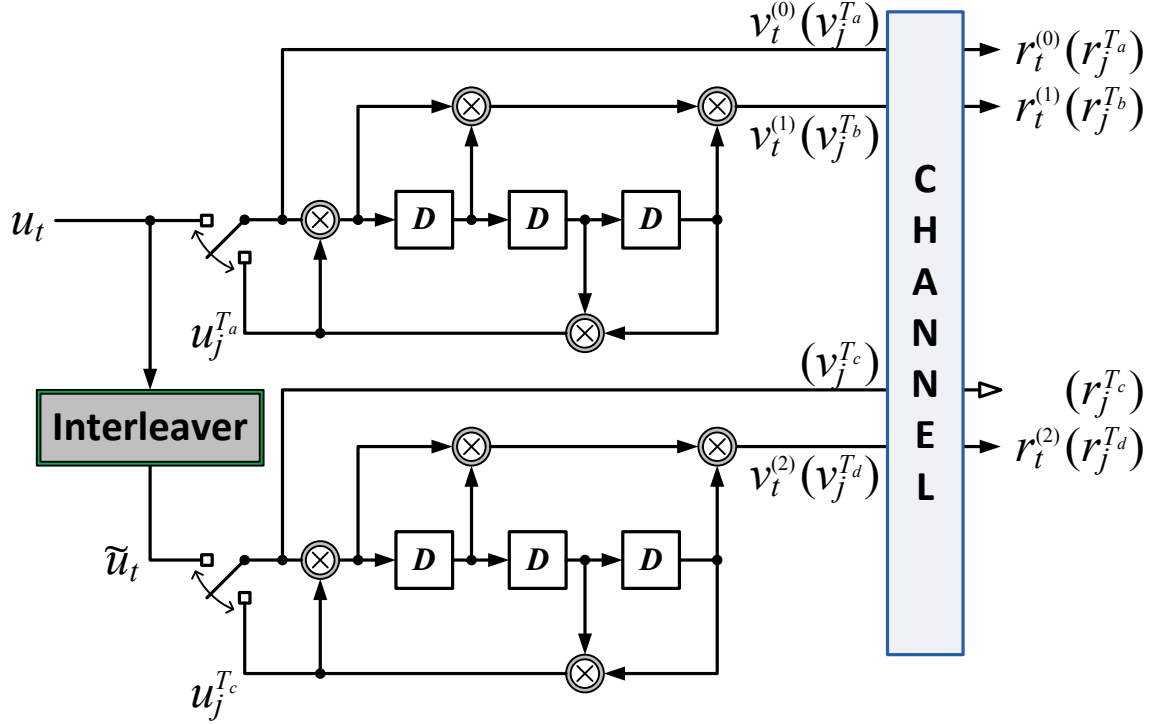


Figure 2.9: The turbo encoder specified in the 3GPP system

input sequence (u_0, \dots, u_{N-1}) , and the permuted sequence is symbolized as $(\tilde{u}_0, \dots, \tilde{u}_{N-1})$. The transmitted codeword includes the systematic data $v_t^{(0)}$, the first parity check $v_t^{(1)}$, and the second parity check $v_t^{(2)}$ for $t = 0 \sim (N-1)$. Besides, some bits from feedback path, $u_j^{T_a}$ and $u_j^{T_c}$, are used for trellis termination. There are 12 additional bits, $(v_j^{T_a}, v_j^{T_b}, v_j^{T_c}, v_j^{T_d})$ with $0 \leq j \leq 2$, attaching to the codeword, so the overall code rate is $N/(3N + 12)$. All codewords are distorted by channel noise before arriving the receiver. From these received data $(r_t^{(0)}, r_t^{(1)}, r_t^{(2)})$ and $(r_j^{T_a}, r_j^{T_b}, r_j^{T_c}, r_j^{T_d})$, the turbo decoder can get an estimate of each information bit.

2.2.2 Iterative decoding flow

For a turbo code with two rate-1/2 constituent codes, the decoding flow involves the process for the first component, $\{r_t^{(0)}, r_t^{(1)}\}$, and the process for the second component, $\{\tilde{r}_t^{(0)}, r_t^{(2)}\}$. Note that $(\tilde{r}_0^{(0)}, \dots, \tilde{r}_{N-1}^{(0)})$ is the interleaved version of $(r_0^{(0)}, \dots, r_{N-1}^{(0)})$. If the trellis is terminated with tail bits, these extra bits should be considered. The soft-in soft-out (SISO) decoder is used to calculate the soft values of each component code [21].

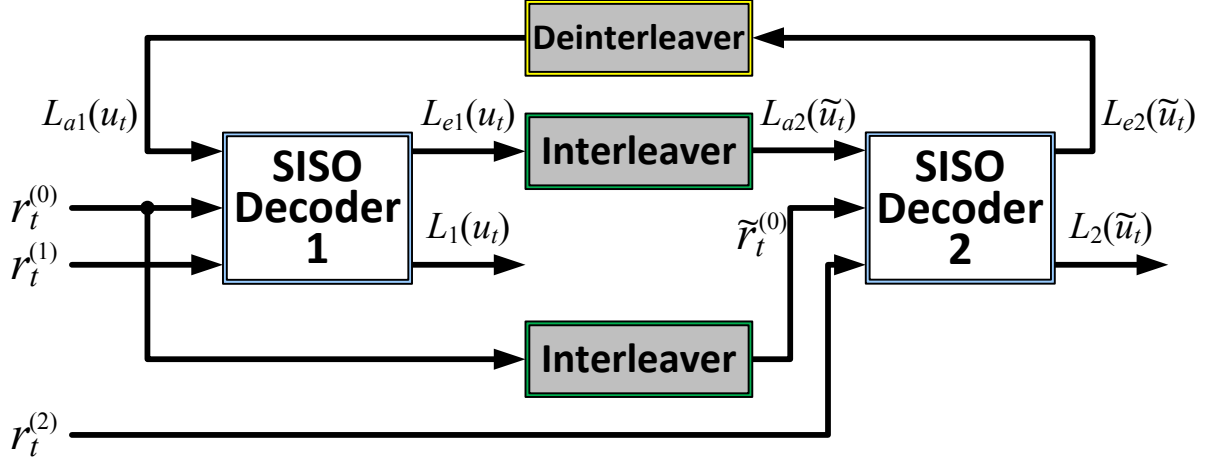


Figure 2.10: Basic framework of a turbo decoder

By substituting (2.35) for the $\gamma(S_t, S_{t+1})$ of (2.23), we have another general expression of LLR:

$$\begin{aligned}
L(u_t) &= \ln \frac{\sum_{(u_t=0; S_t, S_{t+1})} \left[e^{\left(\alpha(S_t) + \gamma(S_t, S_{t+1}) + \beta(S_{t+1}) \right)} \right]}{\sum_{(u_t=1; S_t, S_{t+1})} \left[e^{\left(\alpha(S_t) + \gamma(S_t, S_{t+1}) + \beta(S_{t+1}) \right)} \right]} \\
&= \ln \frac{\sum_{(u_t=0; S_t, S_{t+1})} \left[e^{\frac{1}{2} \left((+1)L_a(u_t) + (+1)L_c r_t^{(0)} \right)} \right] \left[e^{\left(\alpha(S_t) + \frac{1}{2} L_c \sum_{j=1}^{n-1} r_t^{(j)} \times y_t^{(j)} + \beta(S_{t+1}) \right)} \right]}{\sum_{(u_t=1; S_t, S_{t+1})} \left[e^{\frac{1}{2} \left((-1)L_a(u_t) + (-1)L_c r_t^{(0)} \right)} \right] \left[e^{\left(\alpha(S_t) + \frac{1}{2} L_c \sum_{j=1}^{n-1} r_t^{(j)} \times y_t^{(j)} + \beta(S_{t+1}) \right)} \right]} \\
&= L_a(u_t) + L_c r_t^{(0)} + \ln \frac{\sum_{(u_t=0; S_t, S_{t+1})} \left[e^{\left(\alpha(S_t) + \frac{1}{2} L_c \sum_{j=1}^{n-1} r_t^{(j)} \times y_t^{(j)} + \beta(S_{t+1}) \right)} \right]}{\sum_{(u_t=1; S_t, S_{t+1})} \left[e^{\left(\alpha(S_t) + \frac{1}{2} L_c \sum_{j=1}^{n-1} r_t^{(j)} \times y_t^{(j)} + \beta(S_{t+1}) \right)} \right]} \\
&= L_a(u_t) + L_c r_t^{(0)} + L_e(u_t). \tag{2.49}
\end{aligned}$$

The term $L_e(u_t)$ is the *extrinsic* information corresponding to the information bit u_t [5,26]. In above equation, the systematic part $r_t^{(0)}$ is extracted from the original branch metric. During the decoding flow, the sequence $(r_0^{(0)}, \dots, r_{N-1}^{(0)})$ is shared by both component codes. There is a weak correlation between $L_e(u_t)$ and $r_t^{(0)}$, so $L_e(u_t)$ is helpful for the other component code to estimate its *a priori* information.

Fig. 2.10 shows the iterative decoding flow of turbo code. There are two SISO decoder

for the two constituent convolutional codes. The initial *a priori* information $L_{a1}(u_t)$ for the first SISO decoder is set to zero. Based the corresponding trellis diagram, the SISO decoder performs the BCJR algorithm to compute γ , α , and β metrics. It can further get the *a posteriori* information $L_1(u_t)$. From (2.49), the extrinsic information $L_{e1}(u_t)$ can be obtained

$$L_{e1}(u_t) = L_1(u_t) - L_c r_t^{(0)} - L_{a1}(u_t), \quad (2.50)$$

where we assume $L_{a1}(u_t) = 0$ initially. The $L_{e1}(u_t)$ is regarded as the *a priori* information $L_{a2}(\tilde{u}_t)$ in the second SISO decoder. This SISO decoder also does the trellis-based decoding procedure with the following inputs: $\tilde{r}_t^{(0)}$, $r_t^{(2)}$, and $L_{a2}(\tilde{u}_t)$. Then it evaluates the *a posteriori* $L_{a2}(\tilde{u}_t)$ and the extrinsic information $L_{e2}(\tilde{u}_t)$ of the second component code by

$$L_{e2}(\tilde{u}_t) = L_2(\tilde{u}_t) - L_c \tilde{r}_t^{(0)} - L_{a2}(\tilde{u}_t). \quad (2.51)$$

The $L_{e2}(\tilde{u}_t)$ is passed back to the first SISO decoder. A de-interleaver rearranges the sequence order so that $L_{e2}(\tilde{u}_t)$ can be the *a priori* information $L_{a1}(u_t)$ for the first component code. When all $L_{a1}(u_t)$ for $t = 0 \sim (N-1)$ are updated, the first SISO decoder performs BJCR algorithm again. Such soft value calculation of each component code is named as a half-iteration, and two successive processes form one complete iteration. The decoding flow alternates between these components until the stopping criteria are reached. These criteria may be the maximum iteration number or a correctly decoded codeword. Finally, the hard decisions are made from the LLR at the last half-iteration by (2.11).

In Fig. 2.11, the BER performance versus SNR diagram of 3GPP turbo code with $N = 4096$ is presented [12]. The simulation results based on Max-Log-MAP algorithm include the performance with different iteration numbers. The error performance will improve as the iteration number increases. At the first few iterations, the improvement is considerable. However, the correlation between the APP estimates and the received data will become stronger after the first half-iteration. The benefit of the extrinsic information

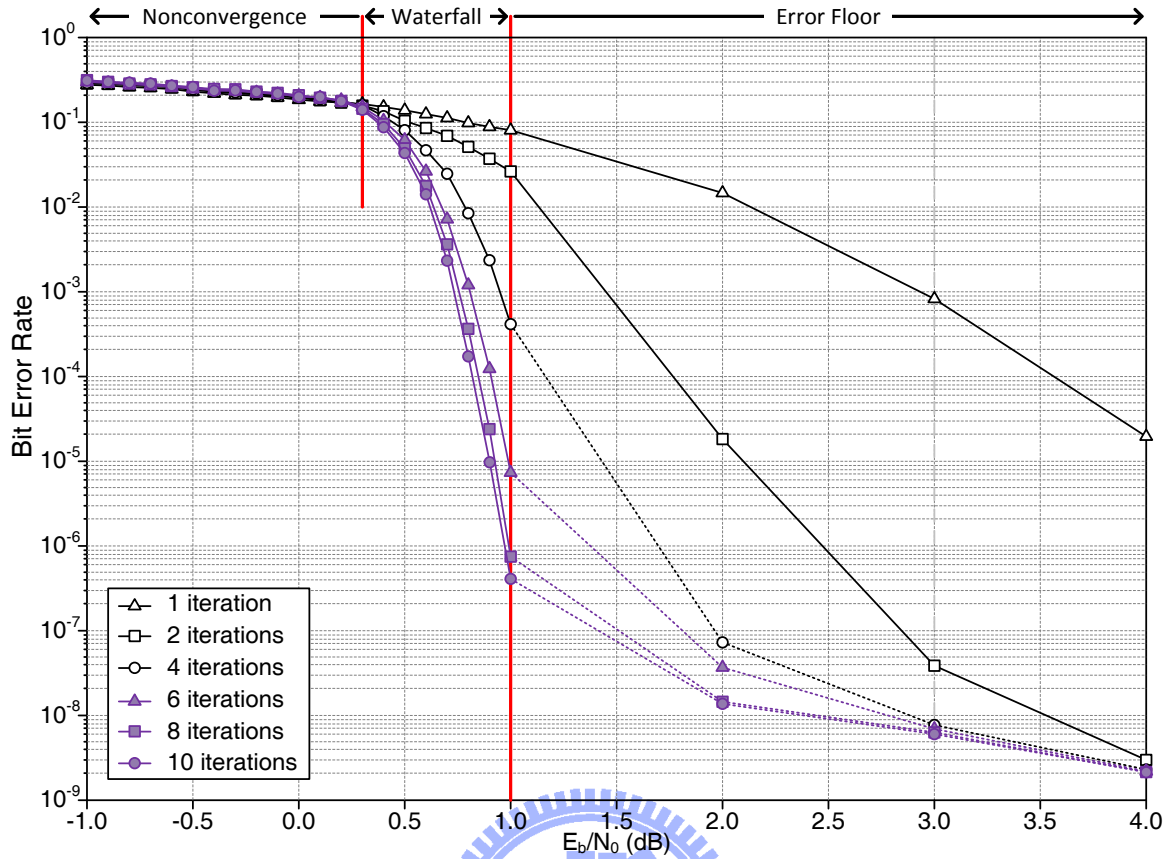


Figure 2.11: The bit error rate performance based on the iterative decoding

diminishes gradually during the succeeding decoding process. In this example, there is little improvement if the iteration number exceeds 6. Actually, each result with more than 6 iterations is similar to the typical BER curve of turbo code. As shown in Fig. 2.11, the BER curve can be divided into three regions [29]. The initial region where error rate remains high and almost constant is the non-convergence region; the middle region where error rate sharply drops to a lower value is the waterfall region; the last region where the decrease of error probability slows down is the error floor region. The interleaver design is closely related to the waterfall region and the error floor region. The random-like properties of interleaver determines the efficiency of APP information exchange. If the interleaver can promise the decorrelation between the constituent codes, there will be a more rapid decline of error probability. Moreover, the distance spectrum, also affected by the interleaver, determines the performance bound at high SNR's [26]. The reason why error floor region occurs is that the BER curve approaches the performance bound. To

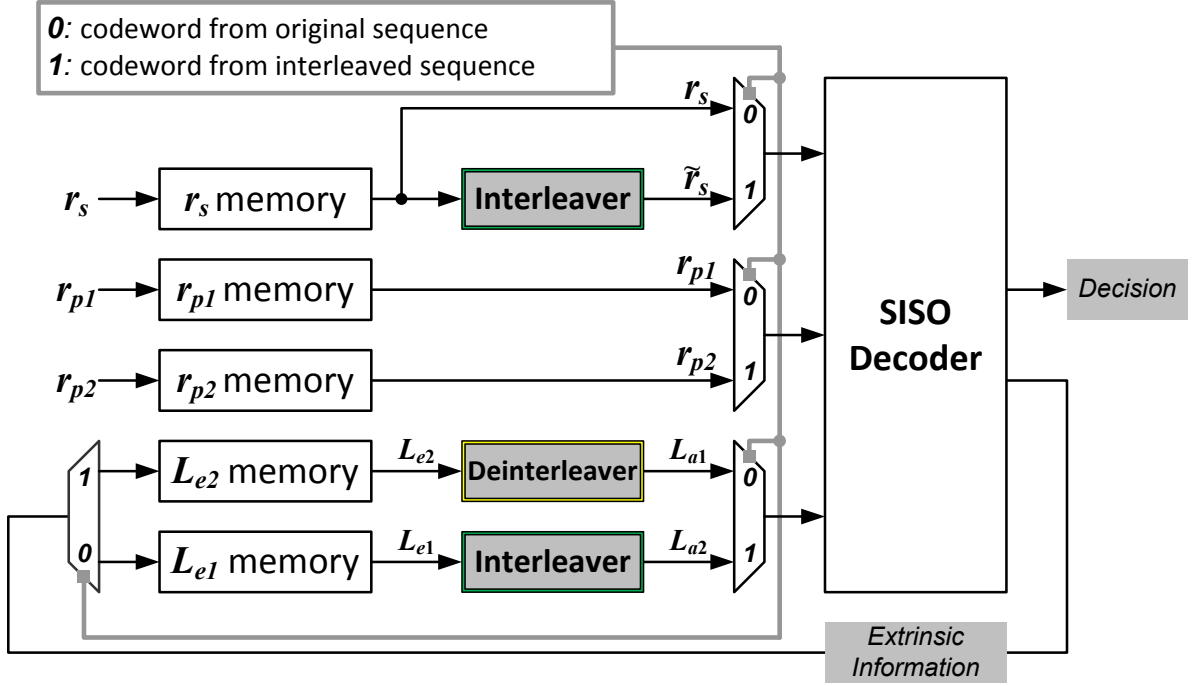


Figure 2.12: Practical architecture of a turbo decoder

achieve further improvement, both the distance spectrum optimization of the code and the randomness of the interleaving rule are two main design criteria [30]. Generally, the convergence rate can be enhanced by increasing the block size N .

2.3 Practical turbo decoder architecture

Due to the permutation by interleaver and de-interleaver, the last few extrinsic information of one constituent code may be the first few probability estimate for the other code. Every half-iteration cannot start until the required *a priori* information is available. For conventional turbo decoders like Fig. 2.10, one SISO decoders is always idle as the other is in execution. Practical decoder architecture takes advantage of such property, and the two component codes will share the same SISO decoder. Fig. 2.12 shows the corresponding architecture, which consists mainly of one SISO decoder and memories. Here \mathbf{r}_s is the received systematic sequence $(r_0^{(0)}, \dots, r_{N-1}^{(0)})$; \mathbf{r}_{p1} is the received first parity sequence $(r_0^{(1)}, \dots, r_{N-1}^{(1)})$; and \mathbf{r}_{p2} is the received second sequence $(r_0^{(2)}, \dots, r_{N-1}^{(2)})$. All received codewords and temporary decoding results are stored in these memories. Both

interleaver and de-interleaver are implemented as the address generators for the memory module. In fact, we can merge the memories for L_{e1} and L_{e2} into a smaller one with elaborate controls on memory access. The multiplexers determine which component code will be processed in each half-iteration. The SISO decoder usually utilizes Log-MAP or Max-Log-MAP algorithm [22] rather than the MAP algorithm [16] for less complexity. In the Log-MAP algorithm, the calculation of branch metric in (2.35) includes the channel reliability value $L_c = 4E_s/N_0$. To guarantee high performance, the precise SNR estimation for the AWGN channel is required [31]. Nevertheless, it is difficult to get the real channel information. Besides, the range of the logarithmic term $\ln(1 + e^{-|x_1-x_2|})$ in (2.36) is very huge. A large lookup table is necessary for such nonlinear function. While the Max-Log-MAP algorithm is applied, the mentioned overhead can be avoided. After simplifying \max^* to \max in (2.38), the logarithmic term is removed, and the lookup table is needless. The value L_c also becomes ineffective because of the maximum function [32]. Although it will lead to performance degradation, the problem can be alleviated by scaling the extrinsic information [33, 34].

The bit number of quantization, also called data width, is another design issue during the fixed-point implementation. It affects the performance and the design area. Both α and β will accumulate vast branch metrics γ over a period of time. It is impossible to represent their values with infinite bit number. On the other hand, too few data width might cause the overflow. We must minimize the necessary data width without performance loss. The works in [35, 36] prove that the quantities of the SISO decoder can be limited. According to the width of received data, the difference between any two $\gamma(S_t, S_{t+1})$ at the same time instant is bounded; the maximal differences of $\alpha(S_t)$ and of $\beta(S_t)$ can be further determined; then the sufficient width for $L(u_t)$ can be obtained [35]. The modulo normalization [37, 38], which makes use of the bounded differences, can be applied to prevent overflow. As a result, we can implement a turbo decoder with reasonable overhead and good performance.

The major computation of an SISO decoder with Max-Log-MAP algorithm includes

γ in (2.35), α in (2.39), β in (2.40), and LLR in (2.41). Because the value of $y_t^{(j)}$ is ± 1 , the corresponding circuits for γ calculation are common adders. The α , β , and LLR calculations have to find the maximal one among several summations. Such maximum function can be achieved by add-compare-select (ACS) units. Fig. 2.13(a) demonstrates the basic ACS unit for the forward path metric calculation in Fig. 2.6(a). The comparator performs the subtraction so that the sign of difference can indicate the larger input and choose it by the multiplexer. With appropriate substitution of inputs, this functional unit can perform the backward path metric calculation. The practical SISO decoder usually adopts the sliding window method for less overhead [24]. This method does the dummy backward path metric β_d to provide reliable initialization for β in each window, and the typical process has been introduced in Fig. 2.7. We let α -ACS mean the circuits for forward path metrics of all states. Obviously, there are also β -ACS and β_d -ACS in the SISO decoder. In Fig. 2.13(b), another ACS unit for the APP value is presented. Its addition function involves α , γ , and β . There will be at least 2^m summations with respect to the same u_t . The comparator and multiplexer will choose the maximal one out of all candidates. When the maximal APP with $u_t = 0$ and the maximal APP with $u_t = 1$ are available, the LLR and the extrinsic information can be computed soon. The total circuits for LLR calculation are called LLR unit here. Since the design computes β_d of W_{j+1} , α of W_j , and β of W_{j-1} in the same time, the received data of three windows must

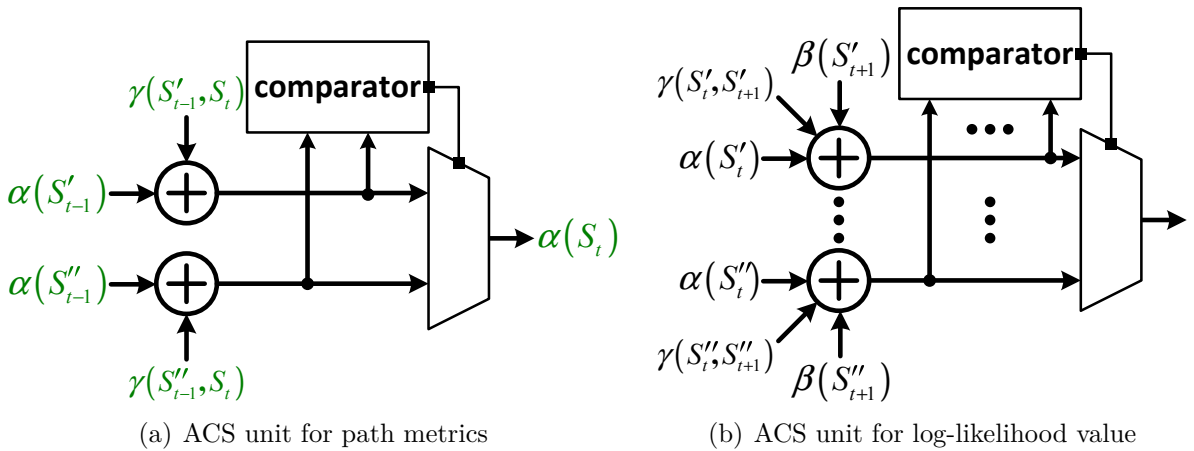
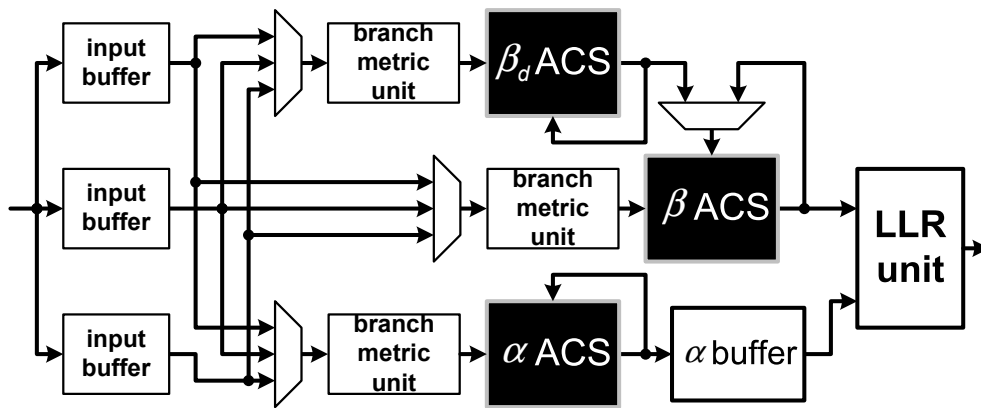


Figure 2.13: Fundamental circuits for path metric and LLR calculations

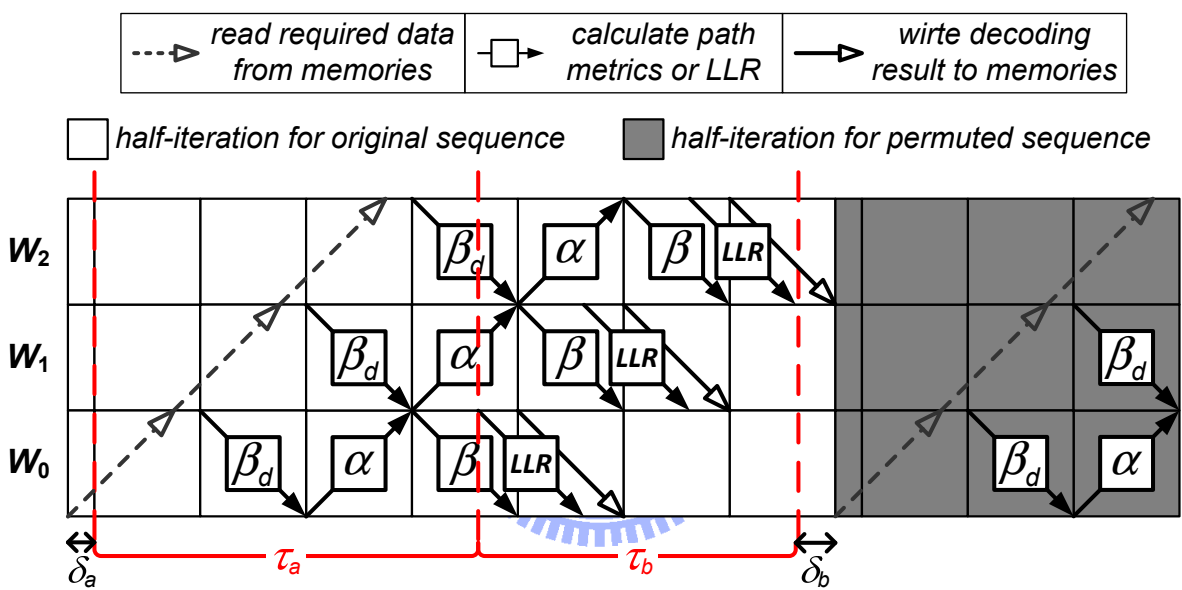
be kept until all their relevant executions are finished. The input cache within the SISO decoder can facilitate the access to data. Moreover, it needs an interior buffer to store α temporarily. Therefore, the complete SISO decoder comprises input buffers, branch metric units, β_d -ACS, α -ACS, β -ACS, α buffer, and LLR unit.

Among all component circuits, the ACS units for recursive path metric calculation lead to the speed bottleneck of the trellis-based decoding process. The data dependency of successive trellis stages makes the pipelining technique difficult to insert registers within these ACS units. Although the LLR calculation also needs the ACS unit, the corresponding data path can be shorten with the pipelining technique. Many researches improve the critical path delay by modifying the ACS circuit. The design in [39] shifts the normalization circuit, and the design in [40] applies the double state technique. With the modifications, the turbo decoder can operate at higher frequency. Besides frequency, the total iteration number is essential for throughput. Fig. 2.11 indicates that the performance improvement is small as the iteration exceeds a certain number. For some erroneous blocks, there is faster convergence in their performance during the iterative process, so we can exploit an efficient stopping criterion to reduce the average iteration number [21, 41, 42]. Most early stopping rules examine the difference between the temporary results of two consecutive iterations or half-iterations. If the difference is less than a given threshold, the decoding process of current data block terminates. The choice of temporary result affects the iteration number, the circuit complexity, and the performance loss. These methods work well at high SNR, and the decoder could achieve the similar performance while costing half or less iterations.

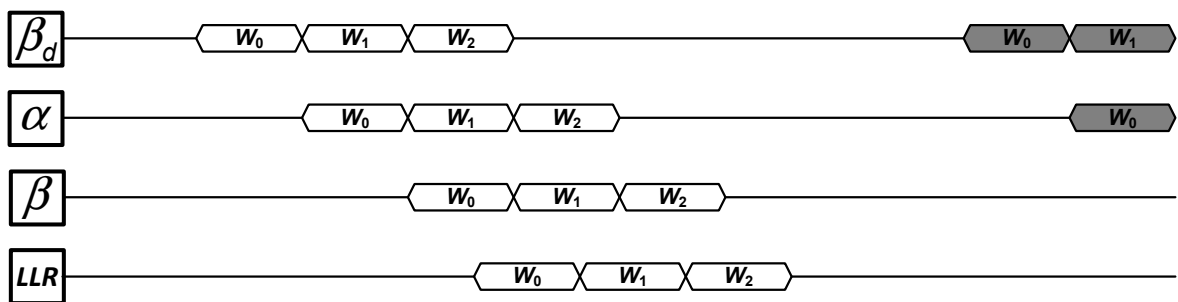
Fig. 2.14(a) shows the traditional SISO decoder architecture with three separate input buffers [43], and Fig. 2.14(b) illustrates its processing schedule. The SISO decoder acquires the received data in ascending order. As the data of entire window are ready, they will be sent from the input buffers to the β_d -ACS, α -ACS, and β -ACS; so these functional units are inactive in the first \mathcal{T}_W cycles. Each half-iteration can be divided as follows: both δ_a and δ_b are pipeline delay time and memory access time; τ_a is the interval for initial metric



(a) SISO decoder architecture



(b) Processing schedule



(c) Corresponding active periods

Figure 2.14: Conventional SISO decoder and its schedule with three windows

calculation between receiving the first input and producing the first output; and τ_b is the time to output all LLR and decisions. Fig. 2.14(c) shows when the major operations of every window are executed. After all extrinsic information are written back to memory, the following half-iteration will start. These component functional units stay idle for $(\delta_a + \delta_b + \tau_a)$ across two successive half-iterations. It takes τ_b out of total execution time to generate decoding results, and the ratio is viewed as the *operating efficiency*, symbolized as η , in (2.52) during the throughput calculation.

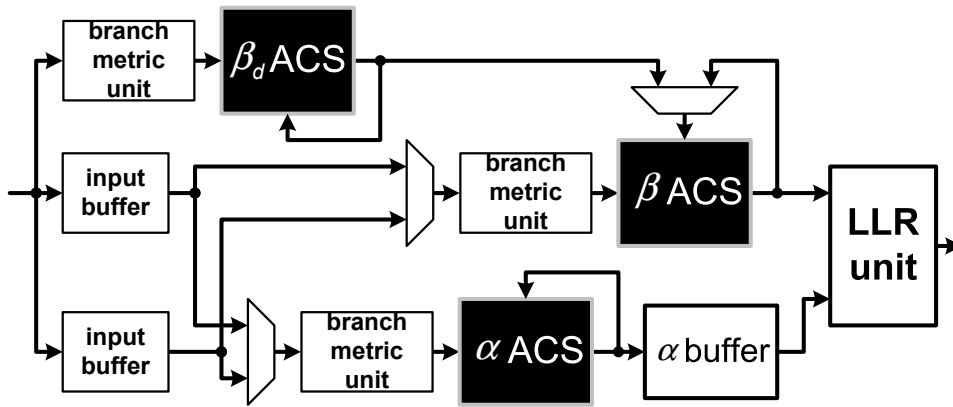
$$\eta = \frac{\tau_b}{\delta_a + \delta_b + \tau_a + \tau_b} \quad (2.52)$$

The value of δ_a , δ_b , τ_a , and τ_b are affected by window length and decoder architecture. From Fig. 2.14(b), the necessary cycles of these execution periods can be expressed as

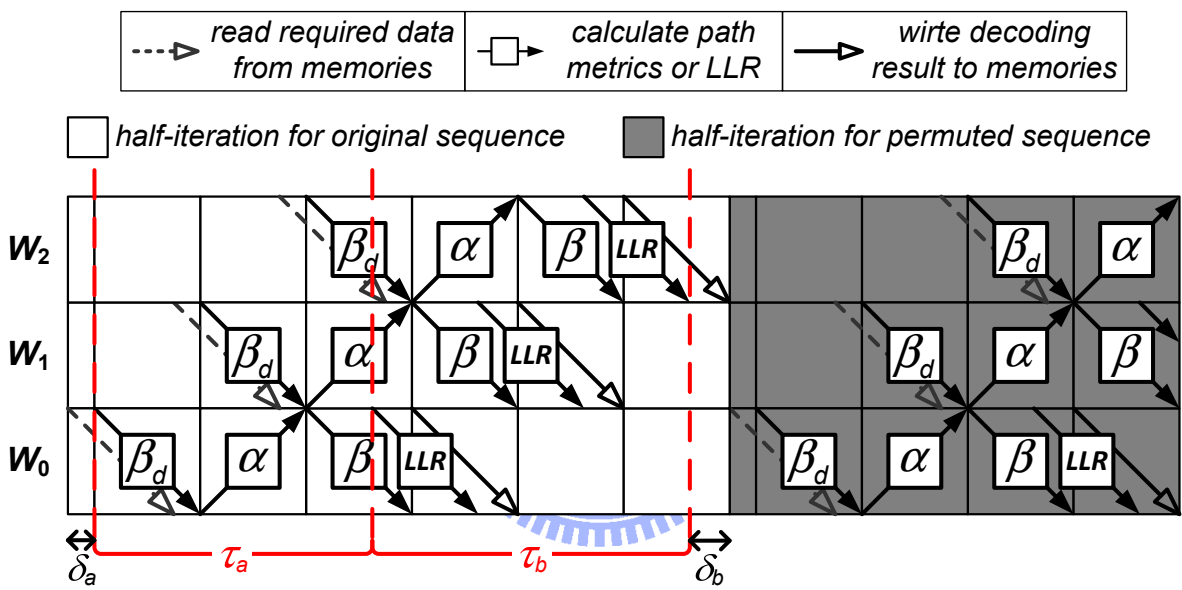
$$\left\{ \begin{array}{l} 0 \leq \delta_a, \delta_b \leq \mathcal{T}_W \\ 3\mathcal{T}_W \leq \tau_a \leq 4\mathcal{T}_W \\ \tau_b = 3\mathcal{T}_W, \end{array} \right. \quad (2.53)$$

where \mathcal{T}_W is the cycle number that each functional unit takes to process one window. In general, δ_a , δ_b , and τ_a are constant for any block size, but τ_b will be in proportion to the window number. When the SISO decoder has to process κ ($= N/L$) windows, only τ_b becomes $(\kappa \times \mathcal{T}_W)$ cycles. We assume that the summation of δ_a , δ_b , and τ_a is $4\mathcal{T}_W$, causing the η equal to $\kappa/(\kappa + 4)$ with the traditional schedule. If the block size is large, the influence of η will be slight; otherwise, the decoding process will be inefficient. The η of the conventional design is merely 42.9% with $\kappa = 3$.

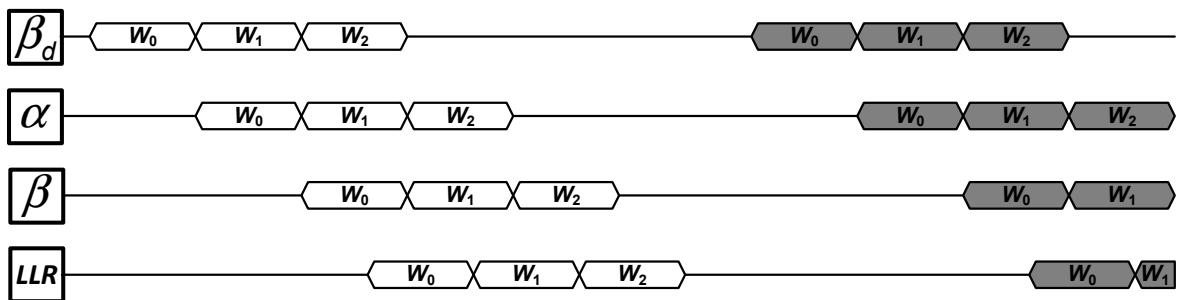
The δ_a , δ_b , and τ_a dominate the operating efficiency η . A trivial solution for raising η is to shorten any of them. The work in [44] modifies the way how the data are input to the SISO decoder. For each window, the received symbols are sent in descending order. The corresponding architecture and schedule are given in Fig. 2.15. There are two input buffers connecting to α -ACS and β -ACS. Rather than waiting for the input buffers, the



(a) SISO decoder architecture



(b) Processing schedule



(c) Corresponding active periods

Figure 2.15: Modified SISO decoder with β_d and its schedule with three windows

β_d -ACS can get its required data sequence immediately and start the backward recursive operations. Compared to the conventional SISO decoder, it costs fewer storage elements and less processing time. This architecture avoids the initial redundant time, and τ_a becomes

$$2\mathcal{T}_W \leq \tau_a \leq 3\mathcal{T}_W. \quad (2.54)$$

The η changes to $\kappa/(\kappa + 3)$. For this schedule with $\kappa = 3$, its η is improved to 50%.

In [45] and [46], an initialization approach is proposed to reduce \mathcal{T}_W cycles from τ_a . Instead of the dummy calculation, the boundary α and β from previous iteration are utilized to initialize the α and β in current iteration. Fig. 2.16 depicts the modified SISO decoder without β_d operations. The branch metric units and ACS units for β_d are removed, and only one input buffer module is exploited to support β -ACS. However, there are extra buffers for previous boundary path metrics in this SISO decoder. Such additional overhead is in proportion to the window number κ , so this architecture is suitable for processing small blocks. Its major advantage is the shortened range of τ_a :

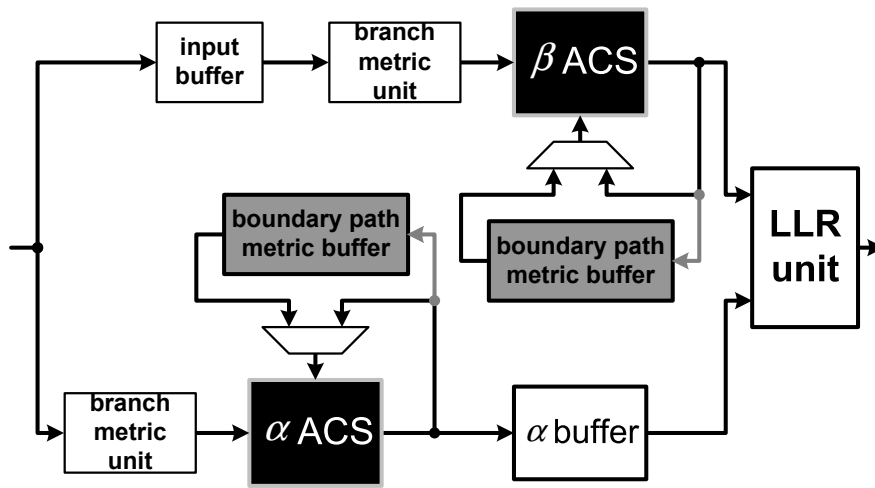


$$\mathcal{T}_W \leq \tau_a \leq 2\mathcal{T}_W. \quad (2.55)$$

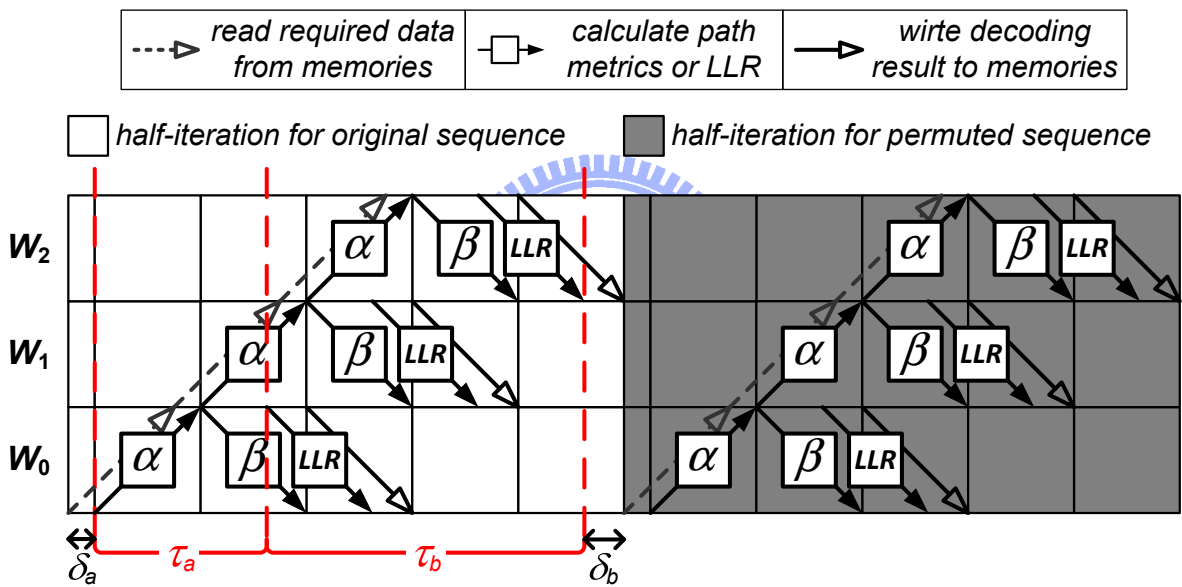
The general expression of η is $\kappa/(\kappa + 2)$ now. When κ is 3, the η is 60%. This method achieves the best operating efficiency.

The turbo decoder will choose the architecture and schedule that lead to the greatest benefits in specific application. If the decoder only deals with large blocks, the SISO decoder in Fig. 2.15 is preferred due to reasonable area and tolerable η . Conversely, the SISO decoder in Fig. 2.16 has superior η , and it may also bring about less overhead. While the application involves both small κ and large κ , we have to make a trade-off between the cost and the efficiency. The cycle number of one half-iteration is

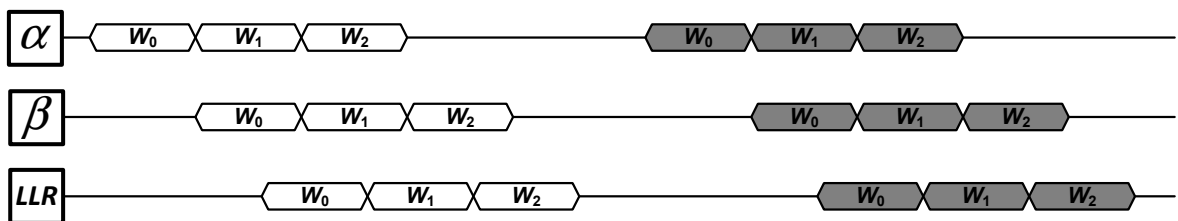
$$\delta_{\mathcal{I}} = \frac{\kappa \times \mathcal{T}_W}{\eta} = \frac{N}{\eta}. \quad (2.56)$$



(a) SISO decoder architecture



(b) Processing schedule



(c) Corresponding active periods

Figure 2.16: Modified SISO decoder without β_d and its schedule with three windows

From [47], all of the critical path delay, iteration number, and operating efficiency are essential for decoding speed. The throughput of a normal turbo decoder can be calculated via

$$\frac{N}{2 \times \mathcal{I} \times (N/\eta) \times (1/\mathcal{F})} = \frac{\mathcal{F} \times \eta}{2 \times \mathcal{I}}. \quad (2.57)$$

where \mathcal{F} is the clock frequency, and \mathcal{I} is the iteration number. Based on these basic architectures introduced above, many designs are developed to pursue higher throughput and less decoding time [9]. Most of the previous works concerns the process of large blocks [48, 49]. The corresponding η usually approaches 100%, and the \mathcal{I} is relatively larger than that of small blocks. For these cases, the \mathcal{F} will be the principal factor in throughput. However, it is impossible to increase \mathcal{F} infinitely. There will be an upper bound of the maximal throughput in traditional turbo decoders. With a growing interest in higher throughput, several techniques are developed, including architecture, algorithm, and decoding flow. These methodologies bring better speedup to turbo decoders, but they also pose new challenges.



Chapter 3

Parallel Architecture and Interleaver

For conventional turbo decoders, both \mathcal{F} and \mathcal{I} are important to throughput calculation. The two factors can be improved by modified ACS circuits and early stopping rules respectively. However, it is a challenge to provide a stable clock signal with high frequency, and the iteration number is unchanged at worst case. We need other methods to raise the decoding speed further even with slow clock or low SNR. Exploiting parallel architecture is an intuitive solution. There are three levels of parallelism: the turbo decoder level, the SISO decoder level, and the trellis stage level [9, 50]. In the turbo decoder level, multiple dedicated turbo decoders are used to decode different codeword blocks independently. In the SISO decoder level, multiple SISO decoders are responsible for the decoding process of single codeword block simultaneously. In the trellis stage level, the computation units inside the trellis-based decoder would process more than one trellis stage every clock cycle. To lessen the implementation complexity of the parallel turbo decoders, appropriate interleaver designs are necessary. In the following descriptions, we give more detailed features of each parallel level; and we also introduce the inter-block permutation (IBP) interleaver [51] and the quadratic permutation polynomial (QPP) interleaver [52] as well as their corresponding circuits.

3.1 Parallel turbo decoder architecture

3.1.1 Turbo decoder level

The parallel turbo decoder level is achieved by simply duplicating the circuits in Fig. 2.12. Fig. 3.1 shows the design with this parallel level, which contains several individ-

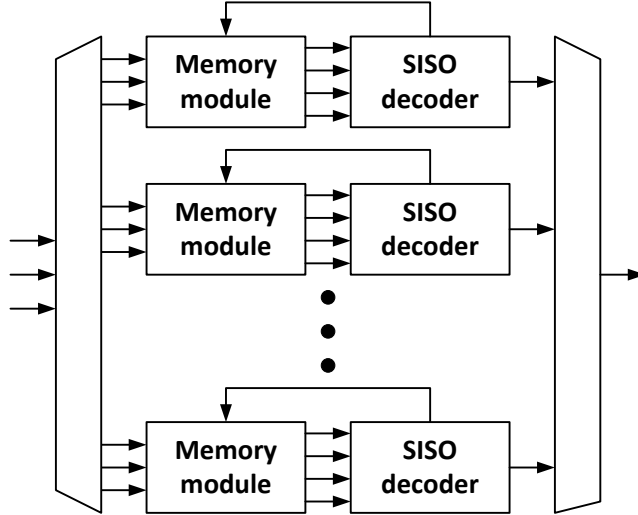


Figure 3.1: Architecture with parallel turbo decoder level

ual pairs of memory module and SISO decoder. While receiving an erroneous codeword, the whole data block are sent to one memory module via the de-multiplexer; then this memory module and its corresponding SISO decoder will start their executions. During the above decoding process, the design can also decode newer received blocks by assigning them to other unoccupied turbo decoders. The parallel design with \mathcal{P}_C turbo decoders can process at most \mathcal{P}_C codewords concurrently, so its throughput increases from (2.57) to

$$\frac{\mathcal{P}_C \times \mathcal{F} \times \eta}{2 \times \mathcal{I}}, \quad (3.1)$$

but the decoding latency for each codeword remains as (2.56). This level is the simplest parallelization, and it can support any turbo decoder with arbitrary interleavers. Nevertheless, it results in considerable overhead due to extra memories and SISO decoders for multiple codewords. Besides, the unchanging decoding time is a drawback. The parallel turbo decoder level is seldom applied to practical designs [9].

3.1.2 SISO decoder level

In the parallel SISO decoder level, \mathcal{P}_S SISO decoders are utilized to process one erroneous codeword. Fig. 3.2 depicts the architecture. Unlike the parallel turbo decoder level,

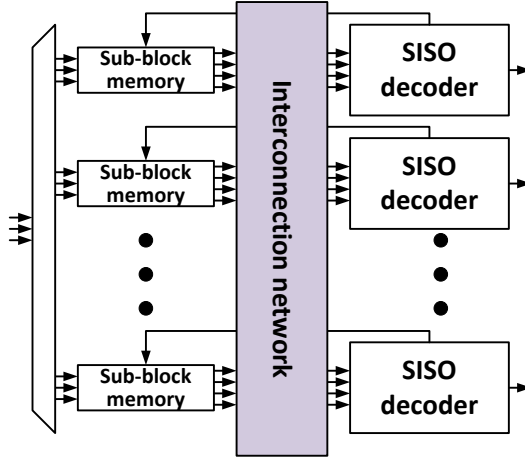


Figure 3.2: Architecture with parallel SISO decoder level

only the SISO decoders are duplicated here. Every received size- N block will be divided into \mathcal{P}_S sub-blocks, and each sub-block has M ($M = \lceil N/\mathcal{P}_S \rceil$) successive data. These sub-blocks are stored in separate smaller memory modules. We represent the M symbols of one sub-block as $(r_{iM}, r_{iM+1}, \dots, r_{(i+1)M-1})$. During the half-iteration for original sequence, each SISO decoder continues accessing the same sub-block memory. At the other half-iteration, the interleaved sequence, $(\tilde{r}_{iM}, \tilde{r}_{iM+1}, \dots, \tilde{r}_{(i+1)M-1})$, is required. The data belong to different sub-blocks, so the SISO decoder will access various sub-block memories. This design requires an interconnection network to handle the data transmission between multiple memories and multiple SISO decoders.

The SISO decoder level gets the most attention for its profits in recent years. In this parallel level, one memory module might be accessed by several SISO decoders at the same time, and such collision problem is the major design issue [9]. Fig. 3.3 gives an example of size-16 block. According to the mapping rule in Fig. 3.3(a), the sequence (r_0, \dots, r_{15}) is reordered and then labeled as $(\tilde{r}_0, \dots, \tilde{r}_{15})$. For the design with $\mathcal{P}_S = 4$, four sub-blocks are stored in separate memory modules in the nature order. Fig. 3.3(b) illustrates a successful parallel data access. As the SISO decoders acquire their first data of the permuted sequence, the $\{\tilde{r}_0, \tilde{r}_4, \tilde{r}_8, \tilde{r}_{12}\}$ come from four different sub-block memory modules. Unfortunately, these memory modules will encounter collision problem while accessing $\{\tilde{r}_1, \tilde{r}_5, \tilde{r}_9, \tilde{r}_{13}\}$. Fig. 3.3(c) shows that the last sub-block memory module has

trouble with the simultaneous requests for \tilde{r}_1 and \tilde{r}_9 . Spreading concurrent requests over several cycles [53] and storing data by specific rules [54] are two solutions. Both techniques can be compatible with conventional interleavers, but they require some hardware to deal with complicated data flow. For large blocks or high parallelism, the corresponding cost is very high. Current studies solve the problem by designing the contention-free interleavers that allow instant access and trivial mapping for all sub-blocks [51, 52, 55–59]. These interleavers result in relatively lower overhead, and they also possess outstanding error-correcting capability. Furthermore, some contention-free interleavers relieve the complexity of interconnection between SISO decoders and memory modules [60, 61].

Each SISO decoder in this parallel architecture processes shorter sub-block, so the decoding time per half-iteration can be reduced, and the throughput can be increased. However, the operating efficiency is sensitive to the block size. The throughput calculation must take the variation of η into account. We assume that window number κ is divisible by \mathcal{P}_S , and one sub-block has κ/\mathcal{P}_S windows. If the parallel design uses the SISO decoder in Fig. 2.16, its operating efficiency will equal

$$\eta_S \approx \frac{\tau_b/\mathcal{P}_S}{\delta_a + \delta_b + \tau_a + \tau_b/\mathcal{P}_S} \approx \frac{\kappa/\mathcal{P}_S}{\kappa/\mathcal{P}_S + 2}. \quad (3.2)$$

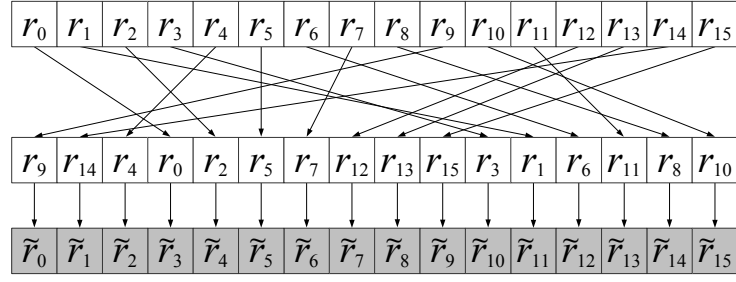
Thus, its throughput is improved to

$$\frac{\mathcal{P}_S \times \mathcal{F} \times \eta_S}{2 \times \mathcal{I}}, \quad (3.3)$$

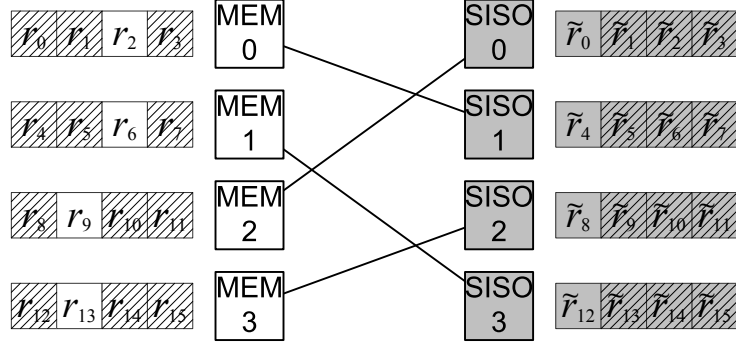
and the execution cycles per half-iteration are reduced to

$$\delta_{\mathcal{I}} = \frac{\kappa \times \mathcal{T}_W}{\mathcal{P}_S \times \eta_S} = \frac{N}{\mathcal{P}_S \times \eta_S}. \quad (3.4)$$

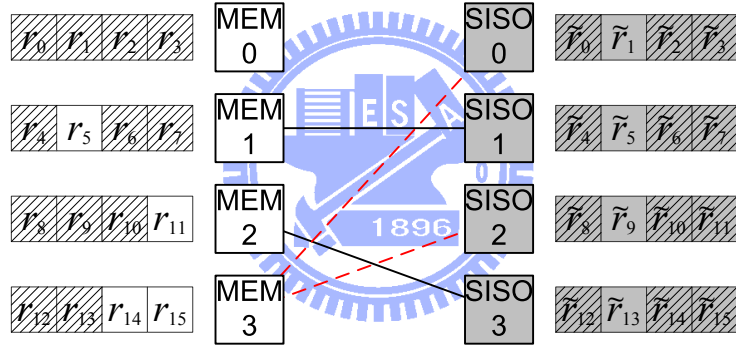
The actual speedup is $(\mathcal{P}_S \times \eta_S/\eta)$, which is smaller than \mathcal{P}_S . The lower operating efficiency is a side effect of parallel SISO decoder level, and it makes high parallelism ineffective [47, 62, 63].



(a) Interleaving rules



(b) Successful access



(c) Collision problem

Figure 3.3: An example of memory access in the parallel SISO decoder level

3.1.3 Trellis stage level

In this parallel level, we primarily modify the circuits of the SISO decoder so that the design can process \mathcal{P}_T consecutive data per clock cycle. Based on the path metrics at S_t , this design uses the received symbols to compute $\{\gamma(S_t, S_{t+1}), \dots, \gamma(S_{t+\mathcal{P}_T-1}, S_{t+\mathcal{P}_T})\}$ and get the path metrics at $S_{t+\mathcal{P}_T}$ in one cycle. As shown in Fig. 3.4, there is a chain of ACS units associated to a series of trellis stages. Apparently, the trivial architecture has a very long data path: \mathcal{P}_T additions, \mathcal{P}_T 2-to-1 comparisons, and \mathcal{P}_T 2-to-1 selections. It needs larger clock period and diminishes the benefit of parallelization. This design costs extra

overhead, but the overall throughput might remain the same. To make this parallel level profitable, the trellis compacting technique is usually applied. More than two consecutive trellis stages are merged into a single one. The calculation of distinct branch metrics can be combined as

$$\gamma(S_t, S_{t+\mathcal{P}_T}) = \sum_{i=0}^{\mathcal{P}_T-1} \left[\frac{1}{2} u'_{t+i} L_a(u'_{t+i}) + \frac{1}{2} L_c \sum_{j=0}^{n-1} (r_{t+i}^{(j)} \times y_{t+i}^{(j)}) \right], \quad (3.5)$$

and it can be computed in advance. Then, α and β can be derived by

$$\alpha(S_t) = \max_{S_{t-\mathcal{P}_T}} [\alpha(S_{t-\mathcal{P}_T}) + \gamma(S_{t-\mathcal{P}_T}, S_t)] \quad (3.6)$$

$$\beta(S_t) = \max_{S_{t+\mathcal{P}_T}} [\beta(S_{t+\mathcal{P}_T}) + \gamma(S_t, S_{t+\mathcal{P}_T})]. \quad (3.7)$$

If each state has 2 incoming branches in the original trellis, it will have $2^{\mathcal{P}_T}$ in the compact trellis. Therefore, we need a radix- $2^{\mathcal{P}_T}$ ACS unit as Fig. 3.5 to choose the maximal summation among $2^{\mathcal{P}_T}$ candidates. The data path includes one addition, one $2^{\mathcal{P}_T}$ -to-1 comparison, and one $2^{\mathcal{P}_T}$ -to-1 selection. In general, the execution time of comparison and selection circuits in Fig. 3.5 is similar to that in Fig. 3.4. Since all additions are operated in parallel, the radix- $2^{\mathcal{P}_T}$ ACS unit has shorter path delay and gains more benefits from the parallel trellis stage level. Many trellis-based decoders adopt such parallelization to increase throughput [48, 64–68].

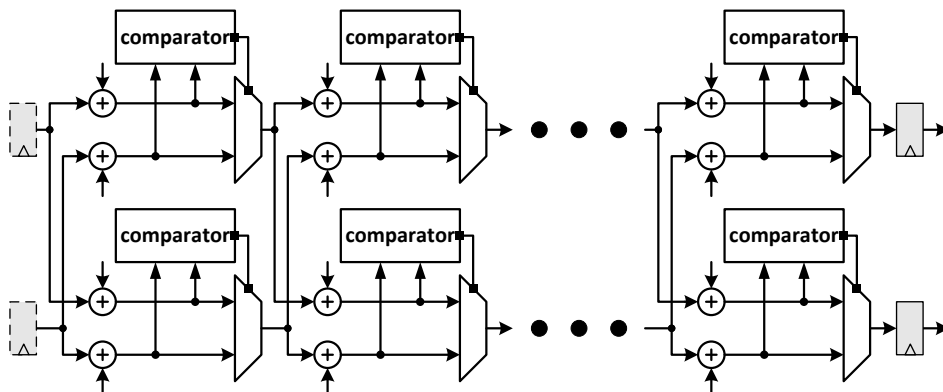


Figure 3.4: Architecture with parallel trellis stage level

This parallel design needs simultaneous access to $(r_t, \dots, r_{t+\mathcal{P}_\mathcal{T}-1})$ or $(\tilde{r}_t, \dots, \tilde{r}_{t+\mathcal{P}_\mathcal{T}-1})$. For this purpose, the received block is partitioned into multiple groups depending on their indexes modulo $\mathcal{P}_\mathcal{T}$. There are $\mathcal{P}_\mathcal{T}$ separate memory modules for these groups, and each one stores $\{r_t, r_{t+\mathcal{P}_\mathcal{T}}, \dots, r_{t+N-\mathcal{P}_\mathcal{T}}\}$ if $\mathcal{P}_\mathcal{T}$ is a factor of N . The collision problem also exists in the parallel trellis stage level. We take the interleaver in Fig. 3.3(a) for example again. Fig. 3.6(a) illustrates a successful data access with $\mathcal{P}_\mathcal{T} = 4$, the high-radix SISO decoder can get $\{\tilde{r}_4, \tilde{r}_5, \tilde{r}_6, \tilde{r}_7\}$ from the four memory modules without hazard. In Fig. 3.6(b), the collision problems occur at the second and the last memory modules while accessing $\{\tilde{r}_8, \tilde{r}_9, \tilde{r}_{10}, \tilde{r}_{11}\}$. With appropriate adjustments, the solutions to the collision problem in parallel SISO decoder level can be effective here.

The execution time of component circuits will be scaled by a factor of $(1/\mathcal{P}_\mathcal{T})$, but the pipeline delay and memory access time are unaffected. Compared to (2.52), the operating efficiency in Fig. 2.16 with the parallel trellis stage level becomes

$$\eta_\mathcal{T} \approx \frac{\tau_b/\mathcal{P}_\mathcal{T}}{\delta_a + \delta_b + \tau_a/\mathcal{P}_\mathcal{T} + \tau_b/\mathcal{P}_\mathcal{T}} \approx \frac{\kappa/\mathcal{P}_\mathcal{T}}{(\kappa + 1)/\mathcal{P}_\mathcal{T} + 1}, \quad (3.8)$$

where we let the summation of δ_a and δ_b be \mathcal{T}_W . As $\mathcal{P}_\mathcal{T}$ is larger than 1, the corresponding operating efficiency degrades. The critical path lengthens, so the maximal frequency decreases to $\mathcal{F}_\mathcal{T}$ ($\mathcal{F}_\mathcal{T} < \mathcal{F}$). Consequently, the throughput changes to

$$\frac{\mathcal{P}_\mathcal{T} \times \mathcal{F}_\mathcal{T} \times \eta_\mathcal{T}}{2 \times \mathcal{I}}, \quad (3.9)$$

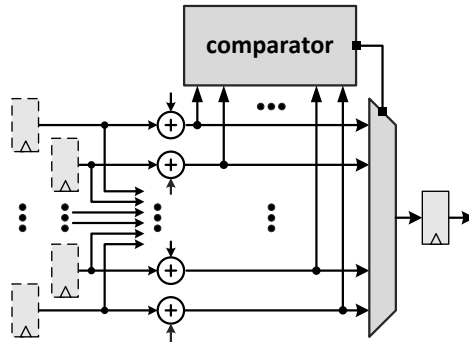


Figure 3.5: High-radix ACS unit

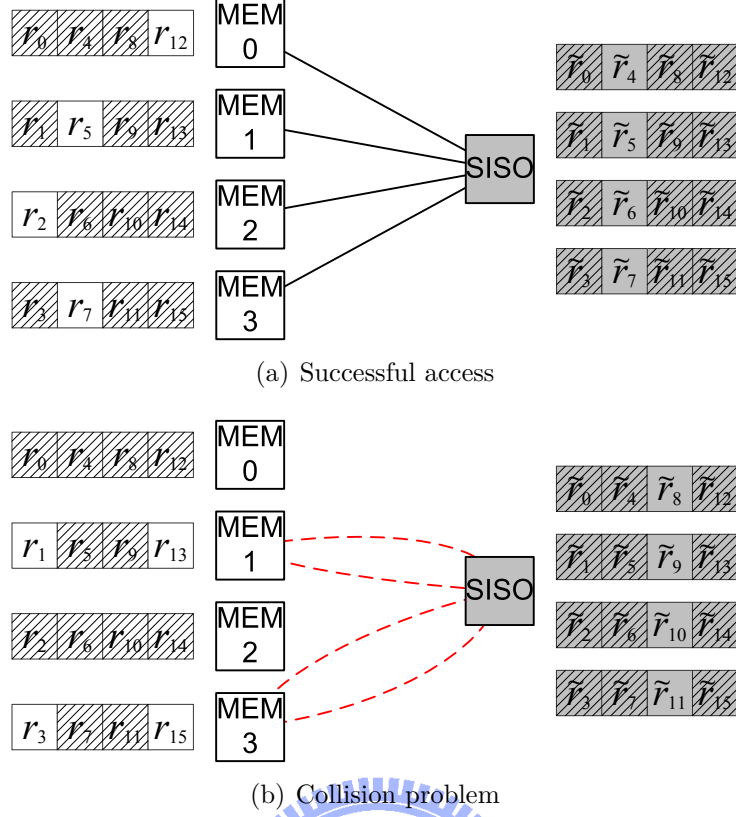


Figure 3.6: An example of memory access in the parallel trellis stage level

and the cycle number of one half-iteration changes to

$$\delta_{\mathcal{I}} = \frac{\kappa \times \mathcal{T}_W}{\mathcal{P}_{\mathcal{T}} \times \eta_{\mathcal{T}}} = \frac{N}{\mathcal{P}_{\mathcal{T}} \times \eta_{\mathcal{T}}}. \quad (3.10)$$

The overall speedup is $(\mathcal{P}_{\mathcal{T}} \times \mathcal{F}_{\mathcal{T}} \times \eta_{\mathcal{T}} / \mathcal{F} \times \eta)$. In fact, the $\mathcal{P}_{\mathcal{T}}$ is always small because this approach causes exponential growth in overhead. The original radix-2 structure has $2^m \times 2$ branches between two trellis stages; there are $2^m \times 2$ adders, 2^m 2-to-1 comparators, and 2^m 2-to-1 multiplexers. After parallelization with $\mathcal{P}_{\mathcal{T}}$, the compact radix- $2^{\mathcal{P}_{\mathcal{T}}}$ structure has $2^m \times 2^{\mathcal{P}_{\mathcal{T}}}$ branches between S_t and $S_{t+\mathcal{P}_{\mathcal{T}}}$; there are $2^m \times 2^{\mathcal{P}_{\mathcal{T}}}$ adders, 2^m $2^{\mathcal{P}_{\mathcal{T}}}$ -to-1 comparators, and 2^m $2^{\mathcal{P}_{\mathcal{T}}}$ -to-1 multiplexers in total. Hence, most designs choose the parallel architecture with $\mathcal{P}_{\mathcal{T}} = 2$ for reasonable cost [48, 64]. For $\mathcal{P}_{\mathcal{T}} > 2$, we can exploit the *two-dimensional* technique to reach a compromise between overhead and enhancement [69–71].

3.2 IBP interleaver and interconnection

The design may utilize a hybrid parallelism rather than single parallel level to achieve higher throughput. Considering the advantages of each level, the combination of parallel SISO decoder level and parallel trellis stage level can get better speedup and shorter decoding latency. However, the collision problem will become more serious. It is inefficient to solve it by enlarging the storage bandwidth or using sophisticated control system. The contention-free interleavers are preferable to such hybrid parallelism since all data are stored in nature order. Undoubtedly, the interleaving rules must avoid all collision patterns with respect to the given \mathcal{P}_S and \mathcal{P}_T . Another issue is the parallel data transmission between memory modules and SISO decoders. In order to make the parallelization easier, the design should consider architecture and algorithm jointly.

This interleaver design originates from the IBP interleaver in [51], and now it takes both the hybrid parallelism and implementation issue into account. Actually, many contention-free interleaver can be regarded as a combination of the interleaving operation within one sub-block and the interchange operation among all sub-blocks. The practical design needs an address generator for each memory module to perform the intra-block permutation and an interconnection network to complete the parallel data transmission. If the original \mathcal{P}_T successive data could be one-to-one mapped onto the \mathcal{P}_T distinct groups after interleaving, the design can support parallel trellis stage level. The intra-block permutation is the key to achieve this objective. For parallel SISO decoder level, the interconnection could be either the hierarchical memory structures [47, 62] or the network topologies suitable for connecting multiple sources to multiple destinations [60, 61]. The inter-block permutation plays an important role in reducing the storage for buffering data and lessening cycles for transmitting data.

Before introducing the methodologies for the intra-block and inter-block permutations, the following example illustrates the interleaving steps. In the design with \mathcal{P}_S size- (N/\mathcal{P}_S) sub-blocks, the y -th symbol in the x -th sub-block is labeled as r_y^x . An example with one size-16 block and $\mathcal{P}_S = 4$ is given in Fig. 3.7. Fig. 3.7(a) demonstrates that the first step is

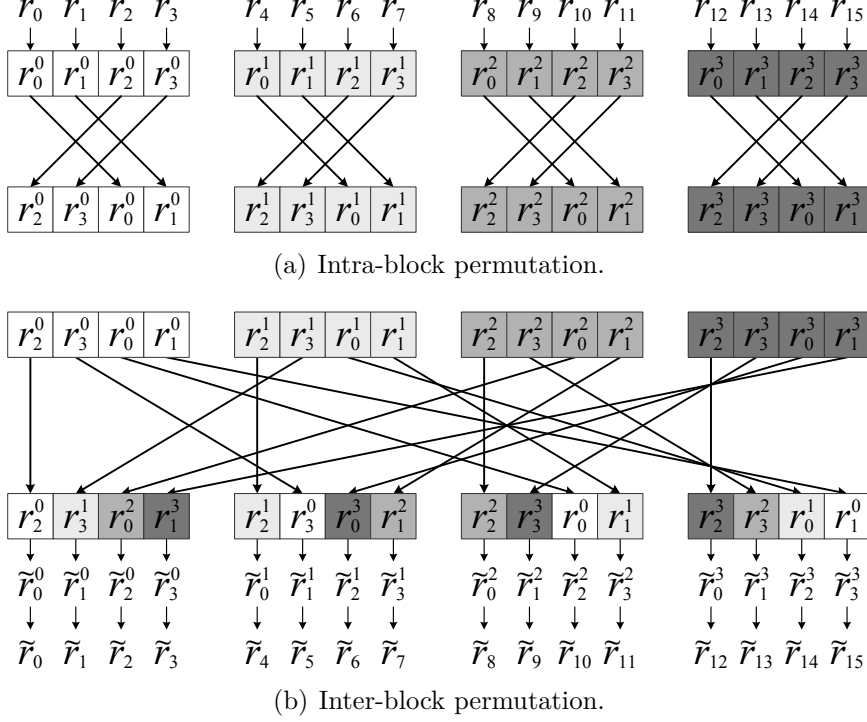


Figure 3.7: An example of the contention-free interleaver with 4 sub-blocks.

reordering the data in each sub-block from $(r_0^x, r_1^x, r_2^x, r_3^x)$ to $(r_2^x, r_3^x, r_0^x, r_1^x)$ for $x = 0 \sim 3$. The intra-block permutation must be constrained to make \tilde{r}_y^x and its nearby data from different banks so that the collision problem for $\mathcal{P}_T \geq 2$ can be prevented. Fig 3.7(b) shows that the second step is swapping the symbols with the same index y with each other. Thus, it establishes a mapping between the original r_y^x 's and the interleaved \tilde{r}_y^x 's. Both the $\{r_y^0, r_y^1, r_y^2, r_y^3\}$ and $\{\tilde{r}_y^0, \tilde{r}_y^1, \tilde{r}_y^2, \tilde{r}_y^3\}$ can be accessed by four parallel SISO decoders concurrently without contention.

In intra-block permutation, only the sequence index y inside each sub-block is concerned. The prime interleaver $\pi(\cdot)$ in (3.11) whose factor ϵ must be relatively prime to (N/\mathcal{P}_S) is one possible solution for parallel trellis stage level [72].

$$\pi(y) = (y \times \epsilon + \vartheta) \pmod{N/\mathcal{P}_S}. \quad (3.11)$$

The offset ϑ is another essential factor in randomness. By assigning appropriate param-

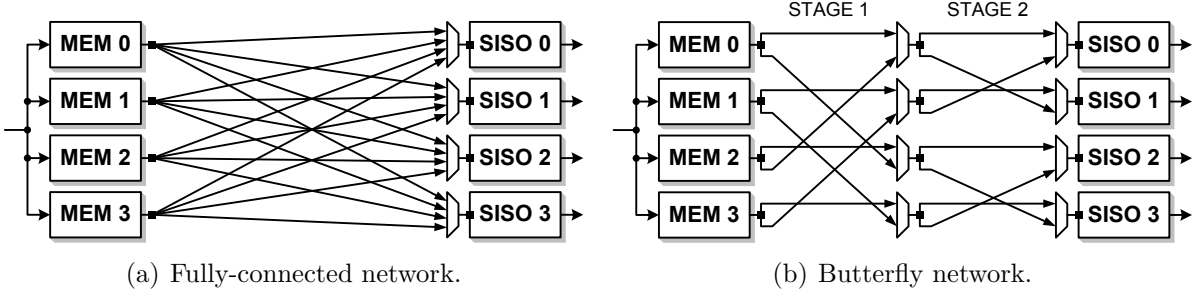


Figure 3.8: Networks for connecting 4 SISO decoders and 4 memory modules.

eters, the indexes y and $(y + j)$ can satisfy (3.12) after interleaving.

$$\pi(y) \not\equiv \pi(y + j) \pmod{\mathcal{P}_T}, \quad 0 \leq j < \mathcal{P}_T. \quad (3.12)$$

We could use padding bits to let the sub-block length be a multiple of \mathcal{P}_T , and the desired parameters for (3.12) could be determined easily. For example, the ϵ will be an odd number with $\mathcal{P}_T = 2$, and the $\pi(y)$ and $\pi(y + 1)$ are not congruent modulo 2. Based on the prime interleaver, an alternative method that supports higher \mathcal{P}_T but has fewer constraints is given in (3.13).

$$\pi(y) = \begin{cases} 2(\lfloor y/2 \rfloor \times \epsilon) + 1 \pmod{N/2\mathcal{P}_S}, & y \text{ is odd} \\ 2(\lfloor y/2 \rfloor \times \epsilon + \vartheta) \pmod{N/2\mathcal{P}_S}, & y \text{ is even} \end{cases} \quad (3.13)$$

After partitioning each sub-block into two groups by the index x modulo 2, the odd-indexed data and the even-indexed data are permuted separately. The idea can be generalized to more partitions for various \mathcal{P}_T 's. The basic double prime method is suitable for our designs with $\mathcal{P}_T = 2$ and $\mathcal{P}_T = 4$. Both the interleaver and de-interleaver can be expressed in (3.13) with different parameters.

Since the fully-connected network can provide any arbitrary interconnection, it is a trivial solution for parallel architectures. Fig. 3.8(a) shows the fully-connected network with $\mathcal{P}_S = 4$. Such network can support various contention-free interleavers by assigning proper control signals to these \mathcal{P}_S -to-1 multiplexers. However, it has some difficulties in implementing the network in high parallelism. As \mathcal{P}_S increases, the area overhead

of multiplexers increases rapidly, and the routing congestion becomes more severe. The network complexity depends on both the parallelism and the characteristic of interleaver. If we can design the inter-block permutation based on a simpler network, interconnecting patterns can be constrained so that the network complexity can be alleviated.

Our design exploits the multi-stage network that consists primarily of 2-to-1 multiplexers to transmit concurrent \mathcal{P}_S sub-blocks. The multi-stage network must be constructed according to the following principle. The output port of every multiplexer or every memory module must connect to the first input port of one multiplexer and to the second input port of another multiplexer in next stage. We let the multiplexers with common input source share the same 1-bit control signal. Thus, each of the \mathcal{P}_S data can be sent to next stage via exactly one of the two paths. The inter-block permutation will follow the behavior of the network, and a systematic interconnection will facilitate the implementation.

Fig. 3.8(b) demonstrates the multi-stage structure with $\mathcal{P}_S = 4$, where the butterfly network topology is utilized. This network can swap x -th sub-block data with the $(x + 2^{((\log_2 \mathcal{P}_S) - i)})$ -th sub-block data in the i -th stage for $i = 1 \sim \log_2 \mathcal{P}_S$. For example, the x -th and the $(x + 2^0)$ -th sub-block data can be exchanged in the second stage of the $\mathcal{P}_S = 4$ network. It gives a low-complexity solution without performance degradation [73]. The external control signals can be determined in advance and stored in a small look-up table. Using periodic assignment on these control signals can reduce the table size. With the help of this approach, the parallel design not only takes lower routing effort but also avoids complex control circuits.

Although the proposed interleaver seems regular, its randomness can promise good decoding performance. Fig. 3.9 shows the floating point simulation results of turbo codes with the same generator matrix (2.48) but different interleavers. The processing schedule in Fig. 2.15 with $L = 32$ and Max-Log-MAP algorithm is applied. The bit error rate with the modified IBP interleaver with well-searched parameters is similar to the 3GPP turbo code in these block sizes. When N is 4096, our strategy can outperform the specification

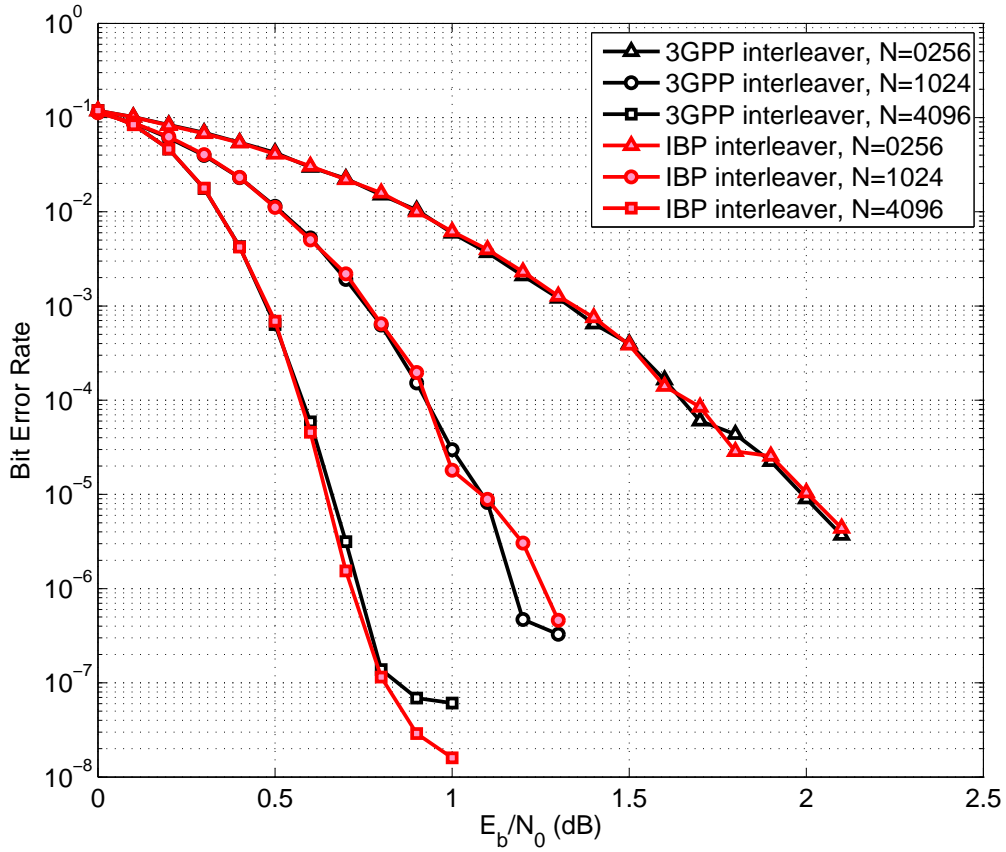


Figure 3.9: Performance of turbo code with 3GPP interleaver and IBP interleaver

interleaver around $E_b/N_0 = 1$.

3.3 QPP interleaver and interconnection

Besides developing a novel interleaver with simple network, we can also design an interconnection with low complexity for famous interleavers. The 3GPP LTE standard adopts QPP interleaver, whose contention-free property allows multiple SISO decoders to decode one codeword for higher throughput and lower latency [15, 52]. The QPP interleaver of a size- N block is expressed as

$$F(t) = f_1 t + f_2 t^2 \pmod{N}. \quad (3.14)$$

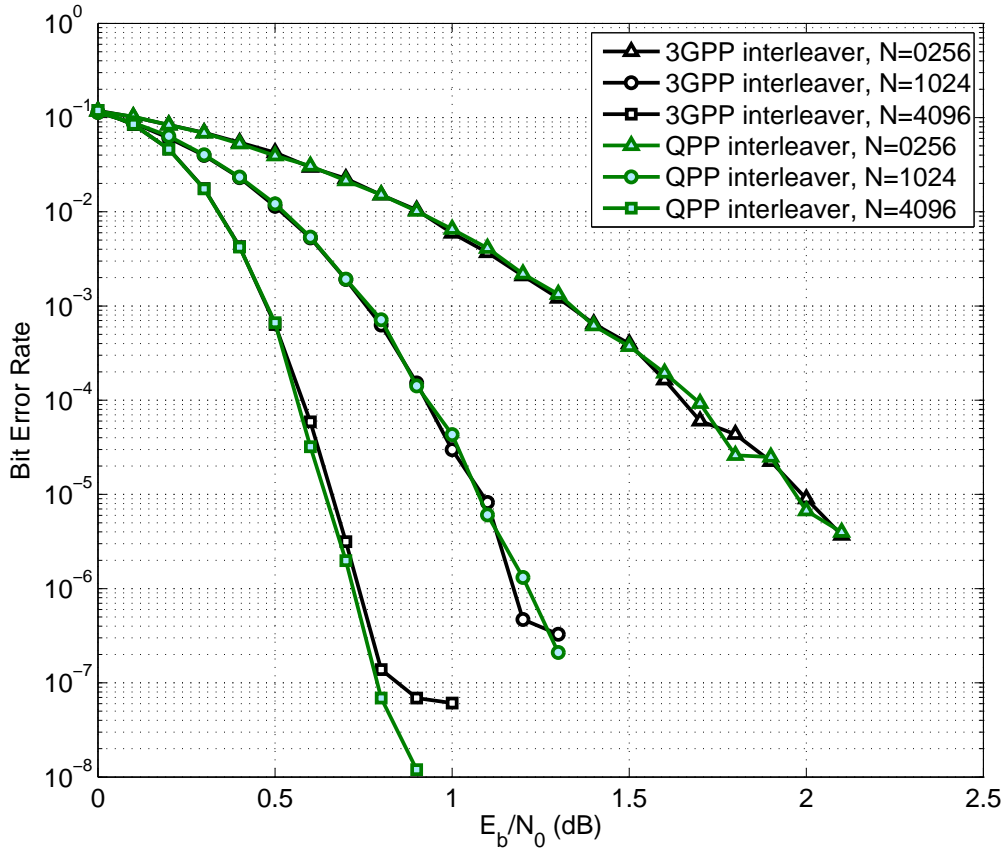


Figure 3.10: Performance of turbo code with 3GPP interleaver and QPP interleaver

The t stands for the original address, whereas $F(t)$ is the interleaved address. The determination of f_1 and f_2 is related to the block size [52]. The quadratic inverse polynomial, $F^{-1}(t)$, can be derived to obtain the corresponding de-interleaving address [74]. From Fig. 3.10, the performance of turbo code in 3GPP LTE standard is comparable to that in 3GPP standard; and it is even better with $N = 4096$.

There are \mathcal{P}_S size- M ($M = N/\mathcal{P}_S$) sub-blocks. For a proper address expression in the parallel architecture, we replace the t in (3.14) with $(xM + y)$, indicating the y -th data in the x -th sub-block. After the substitution, the interleaving address is rewritten as the index q_y in the Q_x -th sub-block:

$$\begin{aligned}
 F(xM + y) &= f_1xM + f_2x^2M^2 + 2f_2xMy + f_1y + f_2y^2 \\
 &= Q_xM + q_y \pmod{N}.
 \end{aligned} \tag{3.15}$$

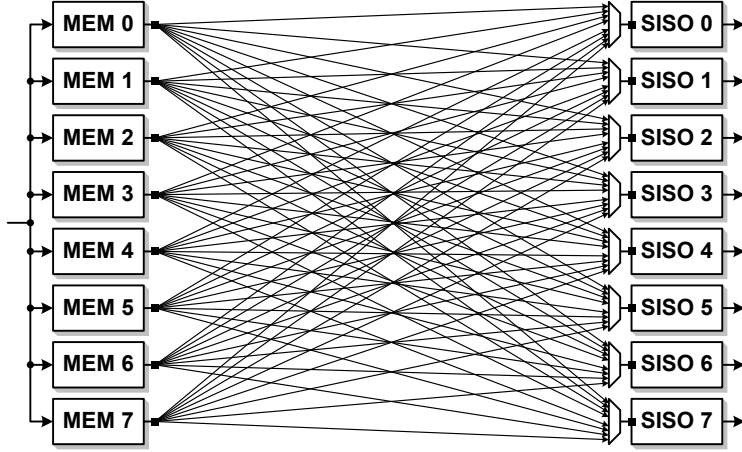


Figure 3.11: Fully-connected network for 8 SISO decoders and 8 memory modules

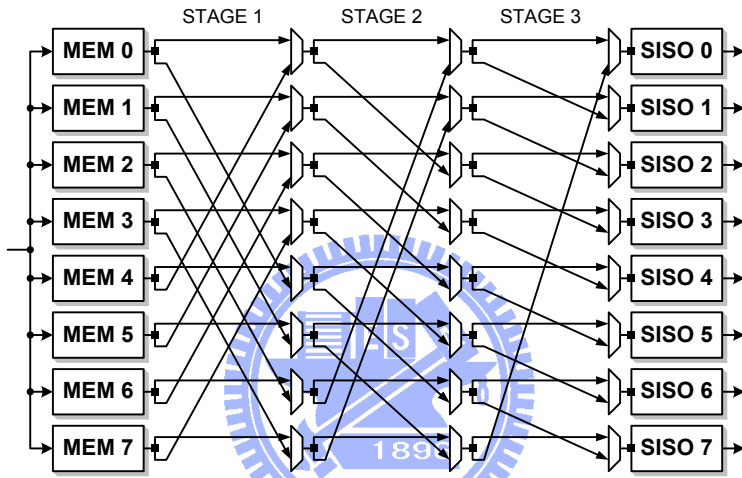


Figure 3.12: Barrel-shift network for 8 SISO decoders and 8 memory modules

Note that $0 \leq x, Q_x < \mathcal{P}_S$ and $0 \leq y, q_y < M$.

Each sub-block will be stored in one individual memory. In the parallel architecture, the data will be transmitted from x -th memory to the Q_x -th SISO decoder. In [75], a recursive approach for generating interleaving addresses $F(t)$ on the fly is illustrated. The computation of y and q_y can be realized with a small address generator rather than a large look-up table. The fully-connected network, which supports arbitrary interconnections, is a trivial solution for parallel data transmission. However, its area overhead grows rapidly as \mathcal{P}_S increases, and the routing congestion would be another critical design issue. Fig. 3.11 shows the fully-connected network with $\mathcal{P}_S = 8$, where each SISO decoder uses one 8-to-1 multiplexers to access the quantized signal from 8 memory modules.

According to the interleaving parameters and parallelism, we can determine how to transmit parallel data simultaneously through the interconnection. The concept of such approach is similar to the work in [76], but the parameter conditions and interconnecting mechanism are different. From the characteristics of QPP interleaver, a multi-stage network based on the barrel shifter is developed for parallel data transmission [77]. Fig. 3.12 shows the network structure with $\mathcal{P}_S = 8$. For $i = 0 \sim 2$, the proposed network can shift data 2^i locations in stage $(3 - i)$ and accomplish the transmission by using appropriate selection signals upon these 2-to-1 multiplexers. The behavior of linking each memory module to its corresponding SISO decoder can be regarded as shifting each sub-block by a certain offset. The following theorem demonstrates the relationship among the offset of all \mathcal{P}_S sub-blocks, and it implies that the network can support parallel design using QPP interleaver. We will use the notation $a \mid b$ if a divides b and use $a \nmid b$ otherwise.

Proposition 3.1. If the QPP interleaver parameters satisfy $\mathcal{P}_S = 2^\ell$ with positive integer ℓ , $\mathcal{P}_S \mid N$, $2 \nmid f_1$, and $2 \mid f_2$, then the offset of the x -th sub-block, which is from x to Q_x , is congruent to the offset of the $(x + 2^i)$ -th sub-block modulo 2^{i+1} .

Proof:

a) From (3.15), the interleaved index q_y can be expressed as

$$\begin{aligned} q_y &= f_1 y + f_2 y^2 \pmod{M} \\ &= f_1 y + f_2 y^2 - \lambda M \end{aligned} \quad (3.16)$$

where λ is independent of x .

b) We can find $Q_x M$ by substituting (3.16) for the q_y in (3.15). Then Q_x can be derived by following steps:

$$Q_x M = f_1 x M + f_2 x^2 M^2 + 2 f_2 x M y + \lambda M \pmod{N}. \quad (3.17)$$

$$Q_x = f_1 x + f_2 x^2 M + 2 f_2 x y + \lambda \pmod{\mathcal{P}_S}. \quad (3.18)$$

c) Let Δx be the difference between x and Q_x .

$$\begin{aligned}\Delta x &= Q_x - x \pmod{\mathcal{P}_S} \\ &= (f_1 - 1)x + f_2x^2M + 2f_2xy + \lambda \pmod{\mathcal{P}_S}.\end{aligned}\tag{3.19}$$

d) Similarly, the $\Delta(x + 2^i)$ is further calculated as

$$\begin{aligned}\Delta(x + 2^i) &= (f_1 - 1)(x + 2^i) + f_2(x + 2^i)^2M + 2f_2(x + 2^i)y + \lambda \pmod{\mathcal{P}_S} \\ &= \Delta x + (f_1 - 1)2^i + f_2(2^{2i} + 2^{i+1}x)M + 2f_22^iy \pmod{\mathcal{P}_S}.\end{aligned}$$

e) Because $(f_1 - 1)$ and f_2 are even numbers in the standard, and 2^{i+1} is a factor of \mathcal{P}_S ,

$$\Delta x \equiv \Delta(x + 2^i) \pmod{2^{i+1}}.\tag{3.20}$$

■

The $\Delta x \pmod{\mathcal{P}_S}$ is the shift amount after passing these $\log_2 \mathcal{P}_S$ stages, and its binary expression determines whether the data of the x -th sub-block are rotated in every stage or not. The congruence in (3.20) indicates that the last $(i + 1)$ bits of Δx and $\Delta(x + 2^i)$ are equivalent, so the two sub-blocks can share the same selection signal in

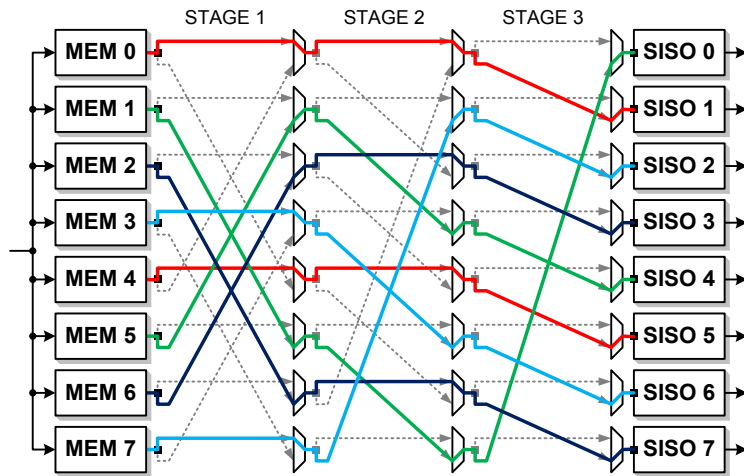


Figure 3.13: Parallel data transmission via the proposed network

Table 3.1: Equivalent gates count of two networks

Parallelism	Fully-connected network	The proposed network
$\mathcal{P}_S = 8$	0.57k (100%)	0.38k (66.7%)
$\mathcal{P}_S = 16$	2.43k (100%)	1.07k (44.0%)
$\mathcal{P}_S = 32$	9.63k (100%)	2.58k (26.8%)

Synthesis result with 90 nm technology: parallel 6-bit data transmission and path delay ≤ 1 ns.

stage $(\log_2 \mathcal{P}_S - i)$. Fig. 3.13 illustrates an example of parallel data transmission with $N = 64$, $f_1 = 7$, $f_2 = 16$, $\mathcal{P}_S = 8$, and $y = 2$. Data from these eight memories $\{0 \sim 7\}$ are sent to SISO decoders $\{1, 0, 7, 6, 5, 4, 3, 2\}$ via the proposed network. The Δx 's of eight sub-blocks are $\{1, 7, 5, 3, 1, 7, 5, 3\}$, and these values satisfy (3.20). The multiplexers with common input sources are controlled by the same 1-bit signal, so there are 4, 2, 1 controlling bits for the three stages respectively.

Our multi-stage network leads to lower complexity in the parallel turbo decoder with QPP interleaver. TABLE 3.1 shows the overhead of two interconnecting networks as \mathcal{P}_S is 8, 16, and 32. Both mechanisms have short path delay so that the data can be transmitted immediately. The proposed network can get a significant area saving, especially in higher parallelism. In addition, less routing effort can be achieved for all necessary interconnections as well.

The QPP interleaver can support $\mathcal{P}_T = 4$ with the following constraints: $4 \mid N$, $2 \nmid f_1$, and $2 \mid f_2$. This feature is verified by interleaving four successive indexes as

$$\left\{ \begin{array}{ll} F(t+0) \equiv f_1 t + f_2 t^2 & \equiv \varphi \pmod{4} \\ F(t+1) \equiv f_1 t + f_2 t^2 + f_1 + 2f_2 t + f_2 & \equiv \varphi + f_1 + f_2 \pmod{4} \\ F(t+2) \equiv f_1 t + f_2 t^2 + 2f_1 + 4f_2 t + 4f_2 & \equiv \varphi + 2 \pmod{4} \\ F(t+3) \equiv f_1 t + f_2 t^2 + 3f_1 + 6f_2 t + 9f_2 & \equiv \varphi + f_1 + f_2 + 2 \pmod{4}, \end{array} \right. \quad (3.21)$$

where the φ is congruent to $F(t)$ modulo 4; $2f_1$ is congruent to 2 modulo 4; and $2f_2$ is divisible by 4. Since $(f_1 + f_2)$ is congruent to 1 or 3 modulo 4, the four indexes can satisfy

(3.12). If the constraint $4 \mid N$ is replaced by $4 \mid M$, the hybrid parallelization can be applied to the turbo decoder using QPP interleaver, too.



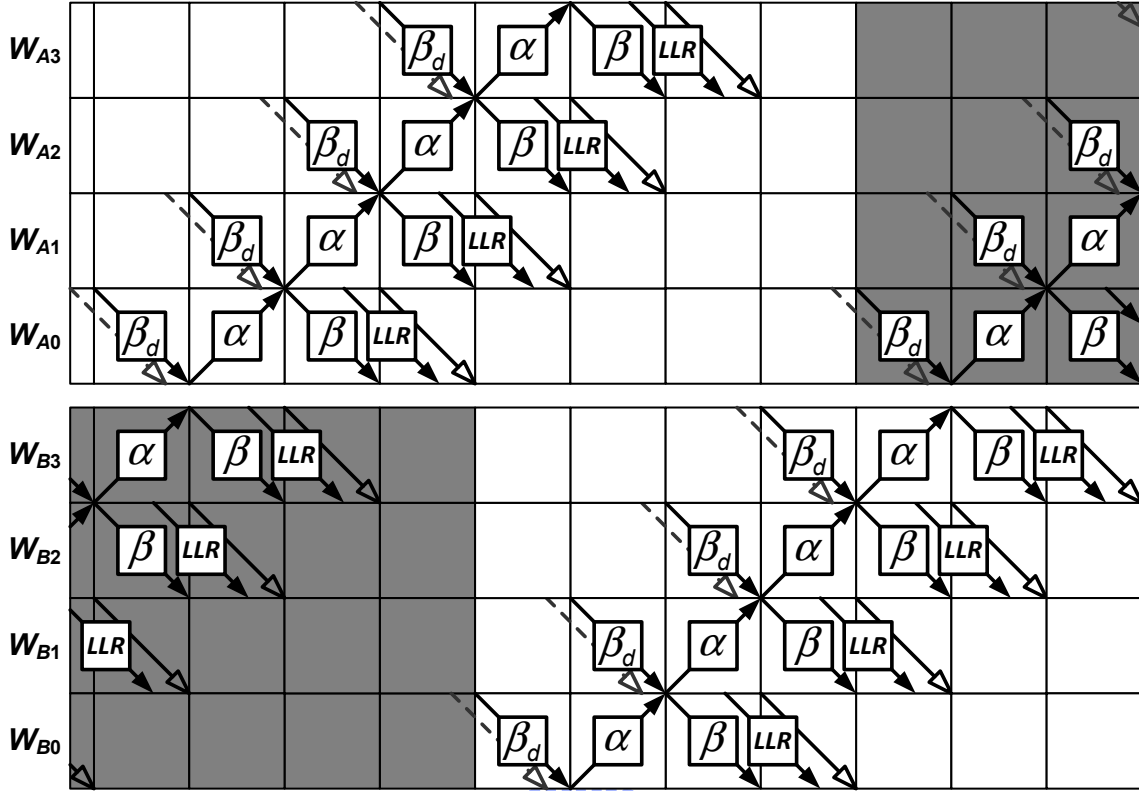
Chapter 4

High-Efficiency Processing Schedule

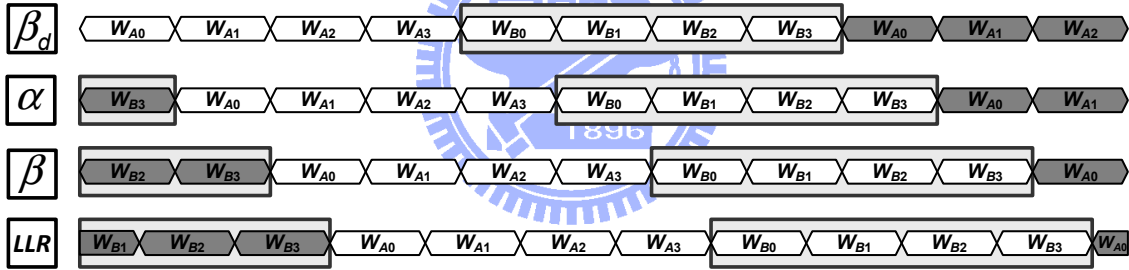
The impacts over operating efficiency in the parallel architecture are (3.2) and (3.8), which are caused by parallel SISO decoder level and parallel trellis stage level respectively. In fact, the \mathcal{P}_T is usually small to avoid too much extension of the critical path delay. Current designs usually prefer large \mathcal{P}_S , but they also suffer from the declining operating efficiency as (3.2). For example, a size-4096 block with $L = 32$ and $\kappa = 128$ has 97.7% operating efficiency while the SISO decoder in Fig. 2.16 is applied. After using parallel 32 SISO decoders, the η_S of each size-128 sub-block decreases down to 66.7% ($\kappa/\mathcal{P}_S = 4$). The actual speedup is about 18 times rather than expected 32 times; almost half benefits of \mathcal{P}_S are lost. In order to overcome such drawback, two processing schedules whose operating efficiency are superior to conventional designs are presented. For simplicity, we consider using one SISO decoder for small blocks and use η rather than η_S . The first proposed schedule is compatible with all traditional schedules (Fig. 2.14, Fig. 2.15, and Fig. 2.16) and any interleaver at the expense of additional storage elements. The second proposed schedule is used in conjunction with the schedule in Fig. 2.16 and specific QPP interleavers only, but it requires less overhead than the first one. These high-efficiency schedules can be effortlessly employed in the parallel architecture.

4.1 Interlaced half-iterations

Because of the randomness of interleaver, the decoding process of one constituent code has to wait for *a priori* probability estimation of the whole block. The waiting time is $(\delta_a + \delta_b + \tau_a)$, and it is also the idle period of all component circuits. If the functional



(a) Partial processing schedule of one SISO decoder.



(b) Corresponding active periods of main components.

Figure 4.1: Processes of independent codeword **A** and codeword **B**

units could be reactivated during their originally idle period, the decoder will become more efficient. Considering the dependency of constituent codes, we let the SISO decoder decode several codewords concurrently by utilizing the idle time of each independent codeword. When any functional unit finish its executions at any half-iteration of one codeword, the unit can be used for the process of other codeword. The method interlaces the decoding processes among two or more codewords, so it is called the *interlaced half-iterations* here. The interlaced half-iterations can increase the η and is harmless to performance, and such concept can be regarded as the *pseudo parallel turbo decoder level*. Its overhead is the

Table 4.1: Comparison between original and interlaced schedules

κ	Original schedule (Single Codeword)		Interlaced schedule (Double Codewords)	
	η	$\delta_{\mathcal{I}}$	η	$\delta_{\mathcal{I}}$
1	25.0%	$4\mathcal{T}_{\mathcal{W}}$	50.0%	$4\mathcal{T}_{\mathcal{W}}$
2	40.0%	$5\mathcal{T}_{\mathcal{W}}$	80.0%	$5\mathcal{T}_{\mathcal{W}}$
3	50.0%	$6\mathcal{T}_{\mathcal{W}}$	100%	$6\mathcal{T}_{\mathcal{W}}$
4	57.1%	$7\mathcal{T}_{\mathcal{W}}$	100%	$8\mathcal{T}_{\mathcal{W}}$
5	62.5%	$8\mathcal{T}_{\mathcal{W}}$	100%	$10\mathcal{T}_{\mathcal{W}}$
6	66.7%	$9\mathcal{T}_{\mathcal{W}}$	100%	$12\mathcal{T}_{\mathcal{W}}$
7	70.0%	$10\mathcal{T}_{\mathcal{W}}$	100%	$14\mathcal{T}_{\mathcal{W}}$
8	72.7%	$11\mathcal{T}_{\mathcal{W}}$	100%	$16\mathcal{T}_{\mathcal{W}}$

storage for extra received codeword, extrinsic information, and decisions.

Fig. 4.1(a) illustrates the partial procedure based on the schedule in Fig. 2.15 and the interlaced half-iterations. Here one SISO decoder deals with codeword **A** and codeword **B**, each of which comprises 4 windows. Fig. 4.1(b) gives the corresponding active periods of main functional units. After any functional unit is released by the process of codeword **A**, it is soon occupied by the process of codeword **B**. All functional units are fully utilized so that η is 100%. Nevertheless, the interlaced schedule may extend the execution time of one half-iteration. As shown in Fig. 4.1(a), the permuted round of codeword **A** is delayed because the required components are occupied by codeword **B**. The interlaced schedule for double codewords makes the $\delta_{\mathcal{I}}$ extend to $8\mathcal{T}_{\mathcal{W}}$ cycles. For $\kappa = 4$, the throughput is increased due to higher η , but the overall latency $2\mathcal{I} \times \delta_{\mathcal{I}}$ also increases.

TABLE 4.1 compares the original schedule and the interlaced schedule with double codewords, and both schedules originate from the Fig. 2.15. The interlaced half-iterations is applied to improve η of small sub-blocks in parallel architecture, so this comparison includes window number $\kappa = 1 \sim 8$ only. This table reveals that the data block size is an significant factor for η and $\delta_{\mathcal{I}}$ in the interlaced schedule. For $\kappa < 3$, the η 's grow greatly, and the $\delta_{\mathcal{I}}$ remain the same. Since the η 's are still less than 100%, interlacing more than two blocks with the same κ is a possible solution for this condition. For $\kappa = 3$,

it has an excellent trade-off between the 100% efficiency and unchanged $\delta_{\mathcal{I}}$. For $\kappa > 3$, the η 's in this proposed schedule are raised to 100% with growing $\delta_{\mathcal{I}}$. Interlacing one block whose $\kappa > 3$ and one block whose $\kappa < 3$ can achieve high η with fewer increments of $\delta_{\mathcal{I}}$. In fact, the choice of fundamental processing schedules dominates the improvement on η and increment of $\delta_{\mathcal{I}}$. The best window number, which can achieve 100% efficiency with least increasing latency, is $\kappa = 4$ for the schedule in Fig. 2.14; while it is $\kappa = 2$ for the schedule in Fig. 2.16.

4.2 Overlapping half-iterations

The operating efficiency in small blocks is still far from 100 % even though there is shorter τ_a in the modified schedule like Fig. 2.16. It is difficult to reduce the necessary execution time (δ_a , δ_b , and τ_a) to zero within each individual half-iteration. However, starting the succeeding half-iteration in advance would achieve the same effect and more improvement. The design could compute the LLR of one component code and the path metric of another component code at the same time [63]. For simplicity, this method is named as *overlapping half-iterations*. Its corresponding interleaver must guarantee no mapping between the processed data of different constituents involved in the overlapping region; otherwise, several problems would happen. First, the *a priori* probability estimation for the new half-iteration might be unavailable, and it would cause a catastrophic performance loss. Second, writing outputs of unfinished half-iteration to memory and reading inputs for new half-iterations from memory are performed simultaneously. The design would encounter a hazard if the memory addresses for these two accesses coincide.

To avoid above problems, the turbo decoder needs appropriate interleaving rules and processing schedules. In our approach, the initial step is classifying all original data into two groups according to their window index, and the permuted data are also divided into a couple of window groups. Then the interleaver lets one group in original sequence be exactly mapped onto one group in permuted sequence. The last step takes the advantage

of processing schedule in [45] and [46], where all windows can be processed in arbitrary order by utilizing previous α 's and β 's. After arranging the execution order of all windows properly, one window group in original sequence and its uncorrelated group in permuted sequence can be decoded at the same time. Thus, the decoding process with overlapping half-iterations can be accomplished. In this section, two types of interleaver constraints will be introduced. As the parameters meet either of them, the interleaver can possess the mentioned characteristic. Besides, the turbo code with such restricted interleaving rule could also get comparable performance as conventional turbo code.

4.2.1 Interleaver constraints of such schedule

First of all, a proper address expression is necessary in the proposed window-based strategy. The (3.18) is rewritten as (4.1) by replacing sub-block size M with window length L .

$$Q_x = f_1x + f_2x^2L + 2f_2xy + \left\lfloor \frac{f_1y + f_2y^2}{L} \right\rfloor \pmod{\frac{N}{L}} \quad (4.1)$$

From (3.16), the q_y is the remainder of this division.

$$q_y = f_1y + f_2y^2 \pmod{L} \quad (4.2)$$

The interleaver design involves classification methods and mapping rules, and only x and Q_x are under consideration. All windows are categorized according to x and Q_x modulo 2; then even x 's and odd x 's must be mapped onto different groups of Q_x 's after interleaving. Three basic restrictions are set in this dissertation: $2L \mid N$, $2 \nmid f_1$, and $2 \mid f_2$. The first restriction, $2L \mid N$, promises that κ is an even number. Because of $2 \nmid f_1$ and $2 \mid f_2$, it is trivial that $f_1x \equiv x \pmod{2}$, $f_2x^2L \equiv 0 \pmod{2}$, and $2f_2xy \equiv 0 \pmod{2}$. With these properties, the (4.1) can be simplified to

$$Q_x \equiv x + \left\lfloor \frac{f_1y + f_2y^2}{L} \right\rfloor \pmod{2} \quad (4.3)$$

The term within the floor function determines the relation between x and Q_x . It is essential that, for all possible y 's, the floor function will produce only even numbers or only odd numbers; then either of $Q_x \equiv x \pmod{2}$ or $Q_x \not\equiv x \pmod{2}$ can hold. The following constraints on f_1 and f_2 are used for $Q_x \equiv x \pmod{2}$.

Proposition 4.1. If the f_1 , f_2 , and L can satisfy (4.4a), (4.4b), and (4.4c), then Q_x and x can be congruent modulo 2.

$$\begin{cases} L \mid (f_1 - 1) & (4.4a) \\ L \mid f_2 & (4.4b) \\ (f_1 - 1)/L \equiv f_2/L \pmod{2} & (4.4c) \end{cases}$$

Proof:

a) The floor function in (4.3) can be rewritten as

$$\left\lfloor \frac{f_1 y + f_2 y^2}{L} \right\rfloor = \left\lfloor \frac{(f_1 - 1)y}{L} + \frac{f_2 y^2}{L} + \frac{y}{L} \right\rfloor. \quad (4.5)$$

b) With (4.4a) and (4.4b), both $(f_1 - 1)y/L$ and $f_2 y^2/L$ are integers. If y is an odd number, $(f_1 - 1)y/L$ and $f_2 y^2/L$ are congruent modulo 2 after applying (4.4c); otherwise, they are both even integers. In either case, these constraints make their summation always be an even integer. Then they could be moved out of the floor function:

$$\frac{(f_1 - 1)y}{L} + \frac{f_2 y^2}{L} + \left\lfloor \frac{y}{L} \right\rfloor. \quad (4.6)$$

c) Due to $0 \leq y < L$, the range of y/L is $0 \leq y/L < 1$, and $\lfloor y/L \rfloor$ will be eliminated by floor function.

d) As a result, the last term in (4.3) is divisible by 2 and can be removed; and $Q_x \equiv x \pmod{2}$ holds true for $x = 0 \sim (N/L - 1)$ and $y = 0 \sim (L - 1)$.

■

In the proof, decomposing (4.3), checking summation of $(f_1 - 1)y/L$ and f_2y^2/L , and removing $\lfloor y/L \rfloor$ are critical steps. If we replace (4.4a) by $(f_1 - i)y/L$ with another integer i , the decomposition will generate $\lfloor i \times y/L \rfloor$, and the elimination will fail. Without (4.4c), an odd number y might result in an odd summation of $(f_1 - 1)y/L$ and f_2y^2/L , and it would destroy the consistency of $Q_x \equiv x \pmod{2}$ for all y 's. This set of constraints lets all even x 's map to even Q_x 's and odd x 's map to odd Q_x 's. The window length L is an important factor in finding the suitable interleaver parameters. This search process can start with setting $(f_1, f_2) = (L + 1, L)$ or $(f_1, f_2) = (2L + 1, 2L)$. By adding multiples of $2L$ to f_1 and f_2 , we could get many usable (f_1, f_2) : $(L + 1 + 2L \times i_1, L + 2L \times i_2)$ and $(2L + 1 + 2L \times i_1, 2L + 2L \times i_2)$, where i_1 and i_2 are arbitrary integers. The final step is to verify the corresponding performance of the turbo code with these parameters.

If the tail-biting technique is applied [27], we can have more choices of interleaving parameters for overlapping half-iterations. Another constraint set that promises $Q_x \not\equiv x \pmod{2}$ in an alternative way will be used for this object. Since the initial state and the ending state are the same, the whole trellis can be regarded as a loop. The decoding procedure for original sequence, $(r_0, r_1, \dots, r_{N-1})$, could be rotated as $(r_\rho, \dots, r_{N-1}, r_0, \dots, r_{\rho-1})$. Similarly, the permuted sequence also could be changed from $(\tilde{r}_0, \tilde{r}_1, \dots, \tilde{r}_{N-1})$ to $(\tilde{r}_\rho, \dots, \tilde{r}_{N-1}, \tilde{r}_0, \dots, \tilde{r}_{\rho-1})$. The cyclic shift with offset ρ reorganizes the window classification and allows for different constraints.

To comply with the following discussion, ρ is set to 1. New expressions, $(x'L + y')$ and $(Q'_xL + q'_y)$, are defined for rotated normal sequence and interleaved sequence, respectively. We can further find the relation between the modified indexes and the original indexes as follows:

$$y' = y - 1 \pmod{L}, \quad (4.7)$$

$$q'_y = q_y - 1 \pmod{L}, \quad (4.8)$$

$$x' = x + \left\lfloor \frac{y-1}{L} \right\rfloor = \begin{cases} x - 1 \pmod{\frac{N}{L}} & \text{if } y = 0 \\ x + 0 \pmod{\frac{N}{L}} & \text{if } y \neq 0, \end{cases} \quad (4.9)$$

and

$$Q'_x = Q_x + \left\lfloor \frac{q_y - 1}{L} \right\rfloor = \begin{cases} Q_x - 1 \pmod{\frac{N}{L}} & \text{if } q_y = 0 \\ Q_x + 0 \pmod{\frac{N}{L}} & \text{if } q_y \neq 0. \end{cases} \quad (4.10)$$

Each of the $\lfloor (q_y - 1)/L \rfloor$ and $\lfloor (y - 1)/L \rfloor$ has two possible values: -1 and 0 . Besides, in (4.2), $y = 0$ implies

$$q_y|_{y=0} = 0. \quad (4.11)$$

Hence, the two terms are equivalent to each other.

$$\left\lfloor \frac{q_y - 1}{L} \right\rfloor = \left\lfloor \frac{y - 1}{L} \right\rfloor = \begin{cases} -1 & \text{if } y = 0 \ (q_y = 0) \\ 0 & \text{if } y \neq 0 \ (q_y \neq 0). \end{cases} \quad (4.12)$$

With the modified indexes, the interleaving parameters for $Q'_x \not\equiv x \pmod{2}$ can be proven as well as for $Q_x \not\equiv x' \pmod{2}$.

Proposition 4.2. If the f_1 , f_2 , and L can satisfy (4.13a), (4.13b), and (4.13c), then $Q'_x \not\equiv x \pmod{2}$ and $Q_x \not\equiv x' \pmod{2}$ are true.

$$\begin{cases} L \mid (f_1 + 1) & (4.13a) \\ L \mid f_2 & (4.13b) \\ (f_1 + 1)/L \equiv f_2/L \pmod{2} & (4.13c) \end{cases}$$

Proof:

a) By substituting Q'_x for Q_x in (4.3), we get

$$Q'_x \equiv x + \left\lfloor \frac{q_y - 1}{L} \right\rfloor + \left\lfloor \frac{f_1 y + f_2 y^2}{L} \right\rfloor \pmod{2}. \quad (4.14)$$

b) The second floor function in (4.14) can be rewritten as

$$\left\lfloor \frac{(f_1 + 1)y}{L} + \frac{f_2 y^2}{L} - \frac{y}{L} \right\rfloor. \quad (4.15)$$

Due to (4.13a), (4.13b) and (4.13c), the summation of the first two terms is an even

integer and can be moved out of the floor function. Such outcome lets (4.14) change to

$$\begin{aligned} Q'_x &\equiv x + \left\lfloor \frac{q_y - 1}{L} \right\rfloor + \frac{(f_1 + 1)y}{L} + \frac{f_2 y^2}{L} + \left\lfloor -\frac{y}{L} \right\rfloor \\ &\equiv x + \left\lfloor \frac{q_y - 1}{L} \right\rfloor + \left\lfloor -\frac{y}{L} \right\rfloor \pmod{2}. \end{aligned} \quad (4.16)$$

- c) The $\lfloor (q_y - 1)/L \rfloor$ is given in (4.12). If y is 0, $\lfloor -y/L \rfloor$ is 0; otherwise, its value is -1 . Therefore, (4.16) can be further expressed as $Q'_x \equiv (x - 1) \pmod{2}$ in (4.17).

$$Q'_x \equiv \begin{cases} x - 1 + 0 \pmod{2} & \text{if } y = 0 \\ x + 0 - 1 \pmod{2} & \text{if } y \neq 0 \end{cases} \quad (4.17)$$

- d) The relation of Q_x and x' can be found by substituting x' for x in (4.3) and simplifying the equation with the proposed constraints and mentioned properties.

$$\begin{aligned} Q_x &\equiv x' - \left\lfloor \frac{y - 1}{L} \right\rfloor + \left\lfloor \frac{f_1 y + f_2 y^2}{L} \right\rfloor \\ &\equiv x' - \left\lfloor \frac{y - 1}{L} \right\rfloor + \left\lfloor -\frac{y}{L} \right\rfloor \pmod{2} \end{aligned} \quad (4.18)$$

Then, the value of $\lfloor (y - 1)/L \rfloor$ and $\lfloor -y/L \rfloor$ are assigned.

$$Q_x \equiv \begin{cases} x' + 1 + 0 \pmod{2} & \text{if } y = 0 \\ x' - 0 - 1 \pmod{2} & \text{if } y \neq 0 \end{cases} \quad (4.19)$$

- e) Finally, (4.17) and (4.19) indicate that both $Q'_x \not\equiv x \pmod{2}$ and $Q_x \not\equiv x' \pmod{2}$ hold true. ■

The constraints set is close to the preceding one, but $(f_1 - 1)$ is used to replace $(f_1 + 1)$. It will generate $\lfloor -y/L \rfloor$ by the decomposition as (4.16) or (4.18). Although this term has two possible values, the substitution of x' or Q'_x can offer a complementary number and get

Table 4.2: Properties of various QPP interleavers

(f_1, f_2)	Spread Factor Minimum Distance		
	(449, 384)	(191, 128)	(31, 64)
$N = 0512$	16 31	16 36	32 33
$N = 1024$	32 38	32 38	32 43
$N = 2048$	64 44	64 44	32 44
$N = 4096$	64 44	64 44	32 44

a steady result. With (4.13a), (4.13b), and (4.13c), the required property for overlapping process in the circular trellis structure is allowed. Here we take the example of size-16 blocks with $L = 4$, and we represent the original and permuted sequence as (r_0, \dots, r_{15}) and $(\tilde{r}_1, \dots, \tilde{r}_{15}, \tilde{r}_0)$, respectively. As the interleaver meets

$$\begin{cases} 4 \mid (f_1 + 1) \\ 4 \mid f_2 \\ (f_1 + 1)/4 \equiv f_2/4 \pmod{2}, \end{cases}$$

these two windows, $\{r_0, r_1, r_2, r_3\}$ and $\{r_8, r_9, r_{10}, r_{11}\}$ ($x = 0, 2$), will be mapped to $\{\tilde{r}_5, \tilde{r}_6, \tilde{r}_7, \tilde{r}_8\}$ and $\{\tilde{r}_{13}, \tilde{r}_{14}, \tilde{r}_{15}, \tilde{r}_0\}$ ($Q'_x = 1, 3$). On the other hand, the other windows in original sequence, $\{r_4, r_5, r_6, r_7\}$ and $\{r_{12}, r_{13}, r_{14}, r_{15}\}$ ($x = 1, 3$), are mapped to $\{\tilde{r}_1, \tilde{r}_2, \tilde{r}_3, \tilde{r}_4\}$ and $\{\tilde{r}_9, \tilde{r}_{10}, \tilde{r}_{11}, \tilde{r}_{12}\}$ ($Q'_x = 0, 2$). During the search for proper parameters, (f_1, f_2) could be $(L - 1, L)$ or $(2L - 1, 2L)$ at first. The subsequent flow is examining the performance of turbo code while the (f_1, f_2) equals $(L - 1 + 2L \times i_1, L + 2L \times i_2)$ or $(2L - 1 + 2L \times i_1, 2L + 2L \times i_2)$ with any integers i_1 and i_2 .

4.2.2 Performance and process of such schedule

Fig. 4.2 shows the floating point simulation with Max-Log MAP algorithm, 8 iterations, and three sets of (f_1, f_2) in $N = 512, 1024, 2048$, and 4096. The simulation utilizes the turbo code generator polynomial in 3GPP LTE standard and the tail-biting method [15, 27]. Here we apply the typical processing schedule with $L = 32$ in Fig. 2.15 [44].

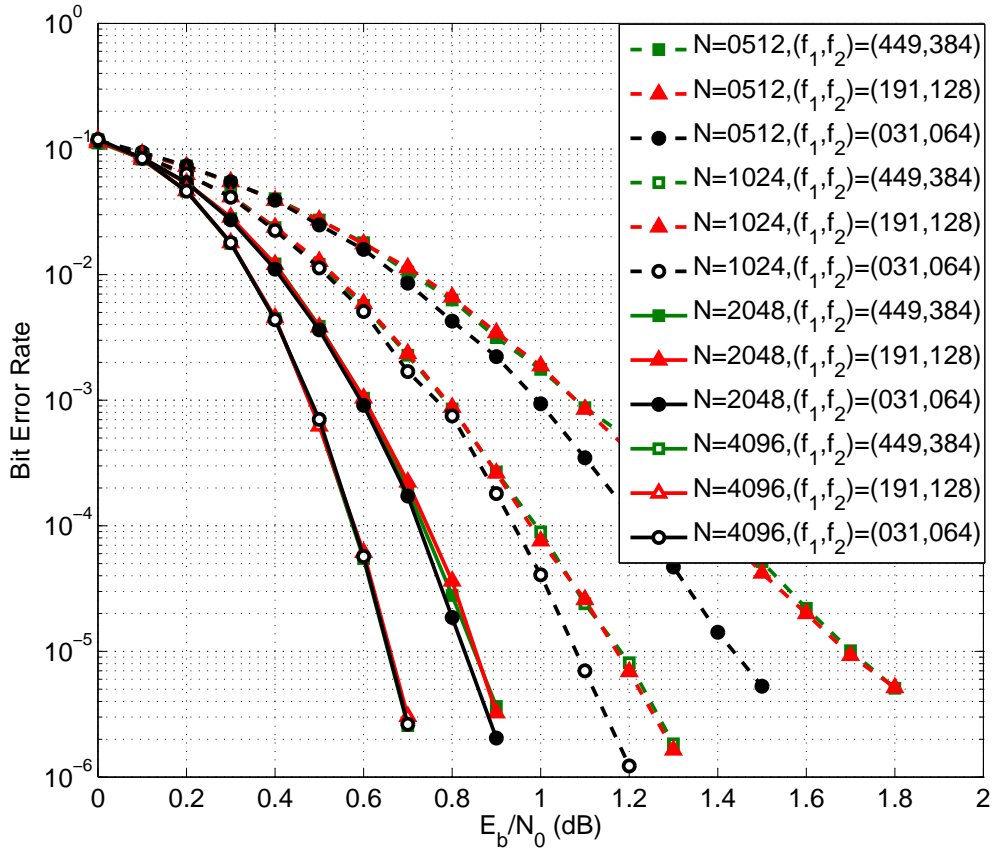


Figure 4.2: Performance of turbo codes with various sizes and different QPP interleavers

For each block size, the $(f_1, f_2) = (449, 384)$ can meet (4.4a), (4.4b), and (4.4c); the $(f_1, f_2) = (191, 128)$ can meet (4.13a), (4.13b), and (4.13c); and the $(f_1, f_2) = (31, 64)$ is the parameters defined in [15]. The selection of these parameters mainly depends on their spread factors, which can filter inferior interleavers quickly [52, 78, 79]. Both $(f_1, f_2) = (449, 384)$ and $(f_1, f_2) = (191, 128)$ have higher spread factors than most interleavers under their respective constraints. TABLE 4.2 shows the properties of these interleavers, including spread factor and minimum distance [80]. Note that these data are helpful to remove the interleavers that result in very poor performance, and an accurate estimation of performance must consider their spectra. At 10^{-5} error rate, the performance loss caused by both restricted interleavers is about 0.3 dB in $N = 512$ and 0.1 dB in $N = 1024$, while the loss becomes insignificant as N is 2048 or 4096. Actually, if the window length is 16, the specification parameters $(f_1, f_2) = (31, 64)$ can also satisfy the

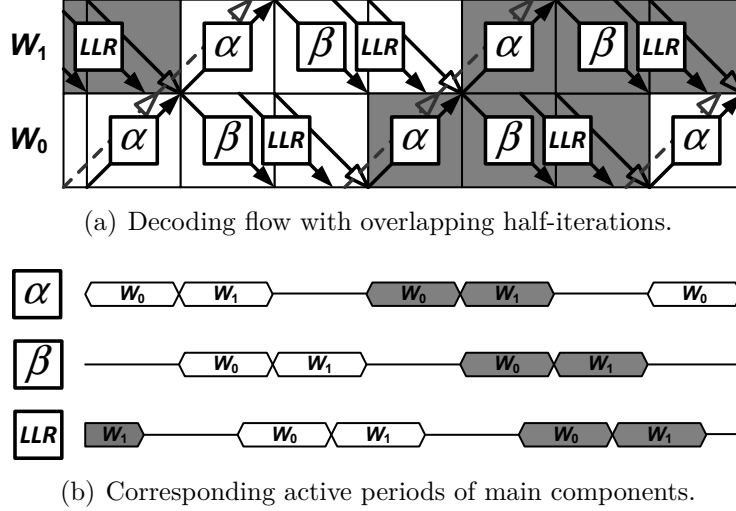
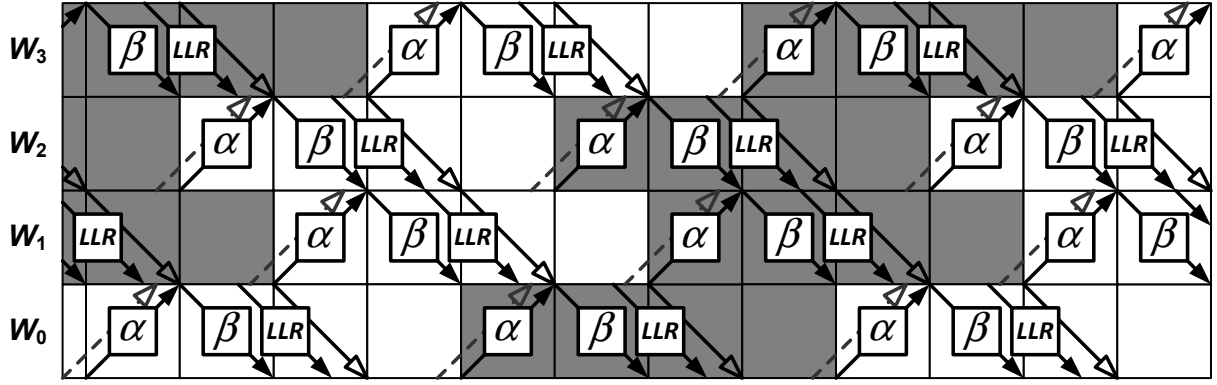


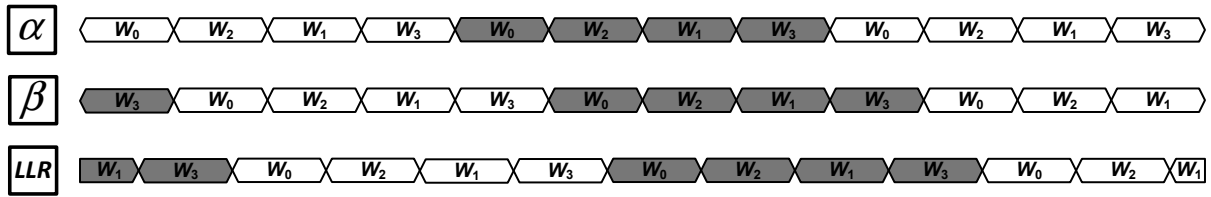
Figure 4.3: Overlapping half-iterations for two windows ($\eta = 66.7\%$ with $\kappa = 2$)

second proposition. The corresponding decoder could support overlapping half-iterations and get better operating efficiency. However, the short window might damage error-correcting capability. Compared to $L = 32$ in $N = 4096$, the code with $L = 16$ has 0.1 dB loss at 10^{-5} error rate. To get both benefits of performance and operating efficiency, sufficient window length plays a key role in the search of parameters.

Here we only take (4.4a), (4.4b), and (4.4c), which promise $Q_x \equiv x \pmod{2}$, into account while introducing the overlapping process. The schedule for another situation can be derived from the same statements after rotating either the original sequence or the permuted sequence. The inactive time of main component circuits is $(\delta_b + \delta_a + \tau_a)$ cycles, and it is also the maximal period which two successive half-iterations can overlap. In fact, the number and order of processed windows dominate the exact overlapping duration. These two issues influence the time when the data in the subsequent half-iteration can access reliable extrinsic information. According to the restricted mapping between x and Q_x , the processed windows within the overlapping region should avoid mapping correlation. Hence, we let those windows whose indexes are identical in modulo-2 calculation be executed before this region. Assuming that the group that contains even-indexed windows is processed first in current half-iteration, then only the odd-indexed windows are involved in the overlapping duration. The interleaving property, $Q_x \equiv x$



(a) Decoding flow with overlapping half-iterations.



(b) Corresponding active periods of main components.

Figure 4.4: Overlapping half-iterations for four windows ($\eta = 100\%$ with $\kappa = 4$)

(mod 2), implies that initial processed windows in subsequent half-iteration must have even indexes.

Two fundamental cases of the proposed schedule are demonstrated here. In Fig. 4.3, the decoder decodes only two windows. It is the smallest window number κ (or κ/P) in our discussion because of $2L \mid N$. If the data of W_0 in original sequence are mapped to W_0 in permuted sequence only, the executions of the permuted-ordered W_0 can start L cycles in advance. The LLR of original-ordered W_1 and the α of permuted-ordered W_0 are computed simultaneously. In comparison to the modified schedule in Fig. 2.16(b), the equivalent η with $\kappa = 2$ increases from 50% to 66.7%. As there are at least four windows in every data block, the SISO decoder can achieve 100% efficiency. Fig. 4.4 shows the schedule with $\kappa = 4$ and activities of the main component circuits. The execution order of windows in either half-iteration is $W_0, W_2, W_1,$ and W_3 . After deriving the LLR of W_0 and W_2 , the SISO decoder continues the unfinished process of W_1 and W_3 . Meanwhile, the executions of W_0 and W_2 in different-ordered sequence begin immediately. All component circuits are fully utilized now. With the help of overlapping half-iterations, the η for the

case with $\kappa = 4$ grows rapidly from 66.7 % to 100 %.



Chapter 5

Implementation Results

The implementation works may exploit more than one parallel level, so we use a general expression for the throughput as

$$\frac{\mathcal{P}_{\mathcal{H}} \times \mathcal{F}_{\mathcal{H}} \times \eta_{\mathcal{H}}}{2 \times \mathcal{I}}, \quad (5.1)$$

where $\mathcal{P}_{\mathcal{H}}$, $\mathcal{F}_{\mathcal{H}}$, and $\eta_{\mathcal{H}}$ are the factors affected by the hybrid parallel levels. Their values are


$$\mathcal{P}_{\mathcal{H}} = \mathcal{P}_c \times \mathcal{P}_s \times \mathcal{P}_T, \quad (5.2)$$

$$\mathcal{F}_{\mathcal{H}} = \mathcal{F}_T, \quad (5.3)$$

and

$$\eta_{\mathcal{H}} \approx \frac{\tau_b / (\mathcal{P}_s \times \mathcal{P}_T)}{\delta_a + \delta_b + \tau_a / \mathcal{P}_T + \tau_b / (\mathcal{P}_s \times \mathcal{P}_T)}. \quad (5.4)$$

The parameters are determined according to the application target of each design as well as the corresponding performance. Our implementation includes two highly parallel turbo decoders using IBP interleaver [73, 81], one parallel turbo decoders for 3GPP LTE application [77, 82], and one highly parallel turbo decoder using QPP interleaver.

5.1 Hybrid parallel design using IBP interleaver

TABLE 5.1 lists the specifications of our proposed parallel turbo decoders using IBP interleaver. Design-II is a modified version from Design-I, so they have many charac-

Table 5.1: Specifications of proposed turbo decoders with IBP interleaver

	Design-I	Design-II
Code Polynomial	$1 \frac{1 + D + D^3}{1 + D^2 + D^3}$	
Code Rate	1/2	1/3
Block Size	128, 256, 512, 1024, 2048, 4096	256, 512, 1024, 2048, 4096
Window Length	32	
Iterations	8 (16 half-iterations)	
Data Width	Received Symbol: 6(3.3) Extrinsic Information: 6(4.2) Path Metric: 8(6.2)	
Double Prime Method	$(\epsilon, \vartheta) = (15, 23)$	
Inter-Block Permutation (32 Periodic Sequences)	08, 19, 12, 18, 17, 14, 29, 05, 10, 21, 06, 23, 03, 26, 20, 22, 30, 04, 25, 15, 00, 02, 11, 09, 28, 24, 31, 27, 01, 13, 16, 07.	
\mathcal{P}_S	32	16
\mathcal{P}_T	2	4
η_H	50%	100%

teristics in common. Both designs use the code polynomial in the 3GPP standard [12] with tail-biting method [27], and they support up to 4096 block size. The sliding window length is set to 32, and the iteration number is fixed at 8. These data widths and interleaver parameters are determined via fixed point simulation. By translating the decimal inter-block permutation parameters into binary expressions, the control signals for the multi-stage butterfly network can be derived. All multiplexers in the same stage are controlled by the same 1-bit signal for less storage requirement. Moreover, we apply the Max-Log MAP algorithm with 0.75 scaling factor for extrinsic information [22, 33, 34]. Due to their distinct methods and individual area constraints, other design factors of the two designs are different, including code rate, supportable block size, \mathcal{P}_S , \mathcal{P}_T , and η_H .

Design-I combines the parallel SISO decoder level and the parallel trellis stage level. This design is implemented with 130 nm technology, and it consists of 2.67M logic gates. To reduce the memory size for received codeword and I/O pin number, it is made into a

rate-1/2 code by puncturing. There are 32 radix-2² SISO decoders in this design, and each SISO decoder is responsible for one 128-bit sub-block. By forcing the most significant bits of inter-block permutation parameters to zero, it can support the following block sizes: 128, 256, 512, 1024, 2048, and 4096. This design utilizes the relocated radix-2 × 2 ACS units, and the clock rate is 265MHz according to the post-layout simulation. The operating efficiency is 50% because of the conventional processing schedule. In the SISO decoder with $\mathcal{P}_{\mathcal{T}} = 2$, each main component spends 16 cycles processing one window, so each half-iteration takes 8×16 cycles.

Design-II involves the hybrid of the SISO decoder level, the trellis stage level, and the pseudo turbo decoder level. This design consists of 2.66M gates count while implemented with 90 nm technology. There are 16 radix-2⁴ SISO decoders in this design, and each SISO decoder is responsible for one 256-bit sub-block. It can support the following block sizes: 256, 512, 1024, 2048, and 4096. All ACS units are modified to the radix-4 × 4 structure by two-stage technique. Due to the advanced technology, this design can operate at 250 MHz even without relocation technique. The interlaced schedule is applied so that the operating efficiency achieves 100%. The total memory modules for double codewords cost about 920k gates count. If the design uses original schedule for single codeword, the storage is 520k gates count. In the SISO decoder with $\mathcal{P}_{\mathcal{T}} = 4$, each main component spends 8 cycles processing one window, so each half-iteration takes 8×8 cycles.

Fig. 5.1 presents the corresponding BER performance of the two proposed designs in AWGN channel. For the 4096-bit block, the performance loss caused by puncture is about 0.7 dB when BER is 10^{-5} . From (5.1) and the parameters in TABLE 5.1, the throughput of the two designs is $2\mathcal{F}_{\mathcal{H}}$ and $4\mathcal{F}_{\mathcal{H}}$ respectively. After post-layout simulation, Design-I is expected to achieve 530 Mb/s, and Design-II is expected to achieve 1000 Mb/s. Fig. 5.2(a) shows the die photo of Design-I, and Fig. 5.2(b) shows the layout photo of Design-II. The parallel SISO decoders occupy the major area of both designs. A delay lock loop (DLL) circuit is used to generate internal clock source as four times the external frequency. However, we bypass the DLL mode of Design-I because of its malfunction.

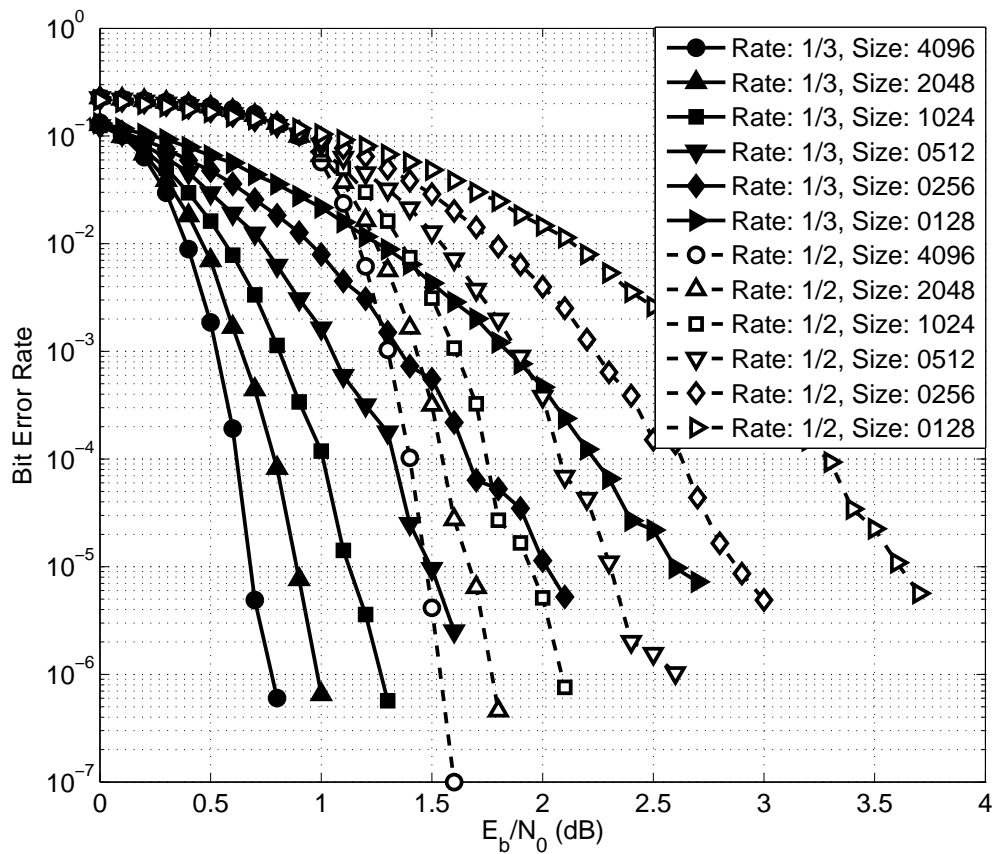
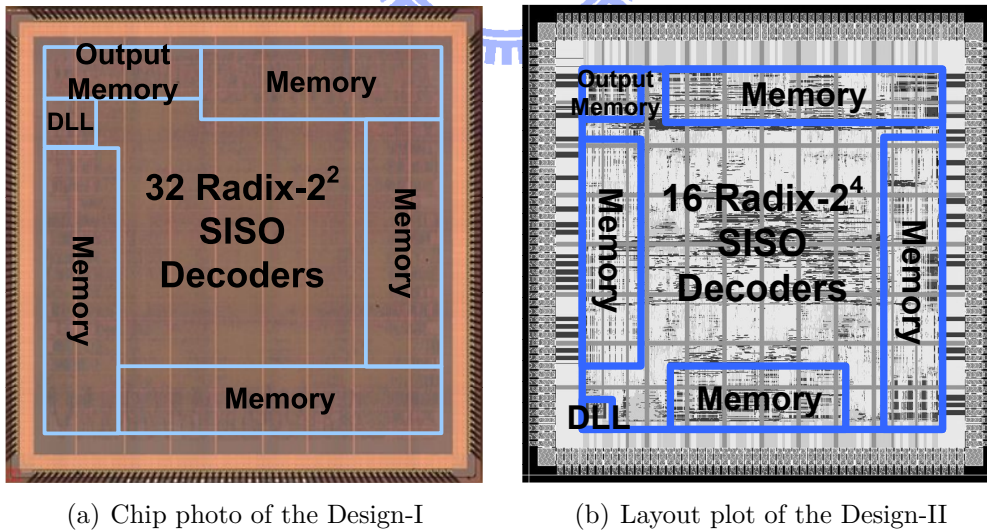


Figure 5.1: Performance of the proposed Design-I and Design-II.



(a) Chip photo of the Design-I

(b) Layout plot of the Design-II

Figure 5.2: Photo of the proposed turbo decoders using IBP interleaver

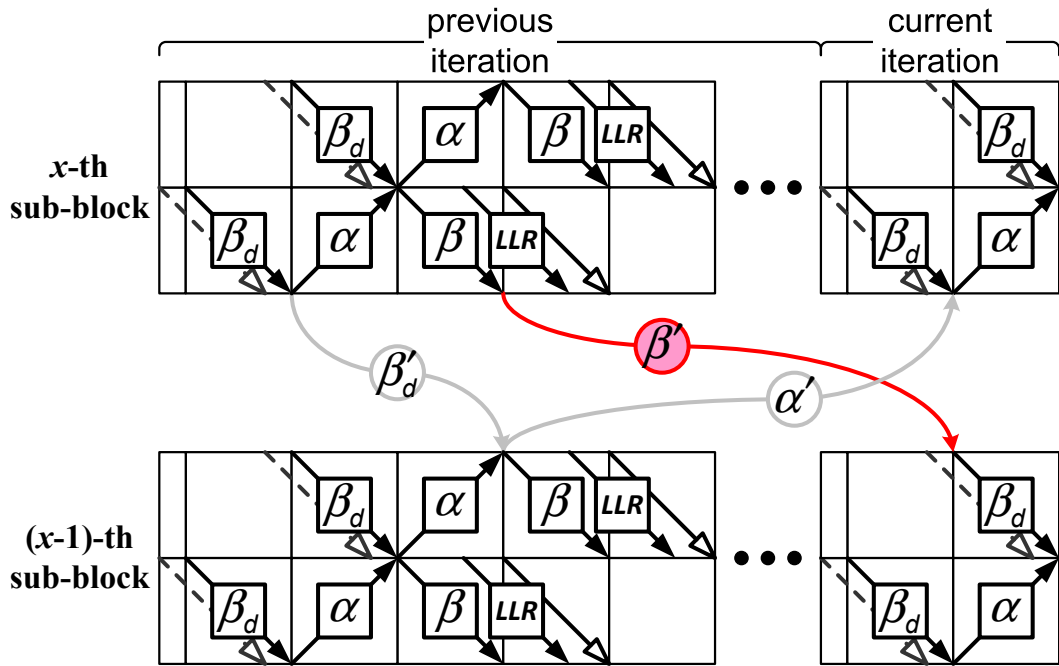
The measurement result indicates that the power consumption of Design-I is 275 mW with 1.32 V supply at 160 Mb/s, and the energy efficiency is 0.22 nJ/(bit·iteration).

From the simulation results of Design-II, its power is 1158 mW with 1.0 V supply at 1000 Mb/s, and the energy efficiency is 0.15 nJ/(bit·iteration).

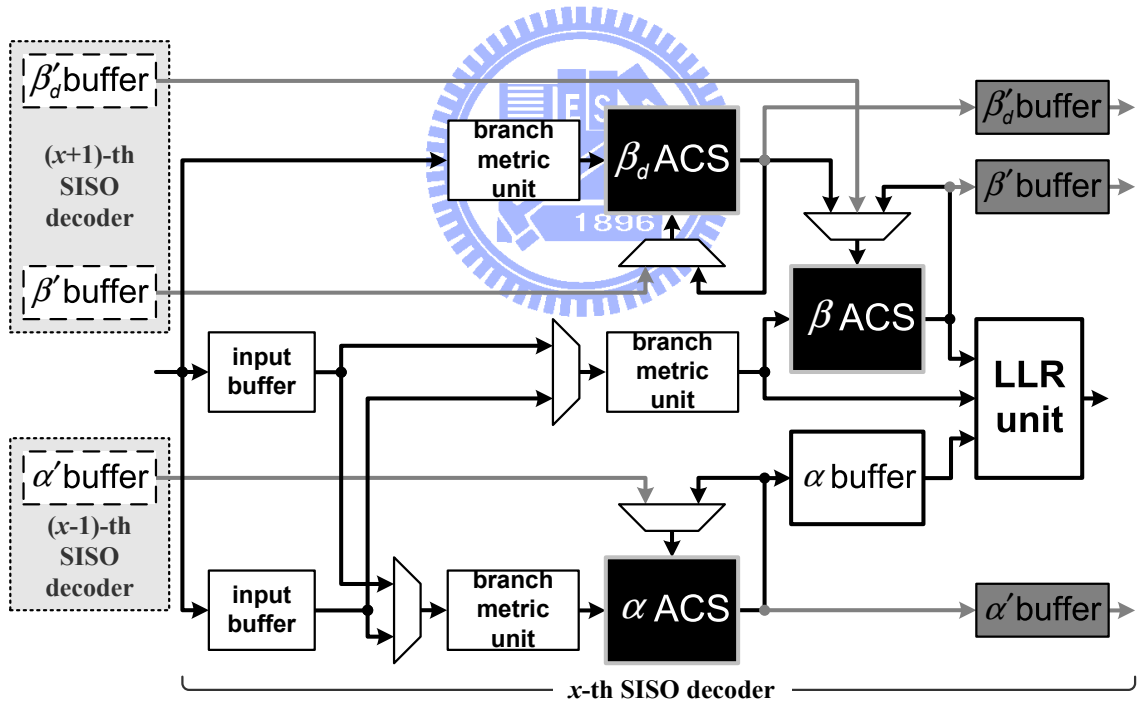
5.2 Reconfigurable design for 3GPP LTE system

In 3GPP LTE standard, there are 188 different sizes ranging from 40 to 6144, and each size has its respective interleaver parameters, f_1 and f_2 . Since 8 is the common factor of all block sizes, any codeword can be processed concurrently by 1, 2, 4, or 8 SISO decoders. We introduce some characteristics of this parallel turbo decoder and discuss its performance issue at first. The Max-Log-MAP algorithm is exploited [22], and only rate-1/3 code is considered. The window length is fixed at 16 for less area overhead and tolerable performance loss around 10^{-5} bit error rate. As sub-block size $M (= N/P_S)$ is not divisible by 16, each sub-block has $\lfloor M/16 \rfloor$ length-16 windows along with one smaller window. The quantized data include 6-bit received codewords, 9-bit metrics, 10-bit LLR, and 6-bit extrinsic information. A 0.75 scaling factor is applied for extrinsic information [33, 34]. Our design can execute all block sizes for at most 8 iterations, and we use fewer iterations for smaller blocks due to the similar performance as compared with further iterations.

The proposed design combines the dummy calculation with previous path metric to support various block sizes with high parallelism. Fig. 5.3(a) shows the processing schedule of two adjacent sub-blocks during two successive iterations, and some special initializations are imposed on the parallel architecture. The β'_d operation indicates that each SISO decoder will pass the boundary β_d to its backward SISO decoder. Therefore, the β_d of the first window in x -th sub-block can update the β of the last window in $(x - 1)$ -th sub-block in the same iteration. Similarly, the α' and β' operations refer to the transmission of path metrics between two iterations. The α' operation can avoid the latency for dummy calculation in every half-iteration. In each sub-block, the initial β_d at the last window will be the previous β from the neighboring SISO decoder, whereas



(a) Processing schedule of parallel sub-blocks



(b) Architecture of the x -th SISO decoder in parallel design

Figure 5.3: Modified SISO decoder for the 3GPP LTE turbo decoder

the initial β_d 's at other windows are zero. The β' operation are used in conjunction with the dummy β_d computation so that it can get more robust β initialization from a very

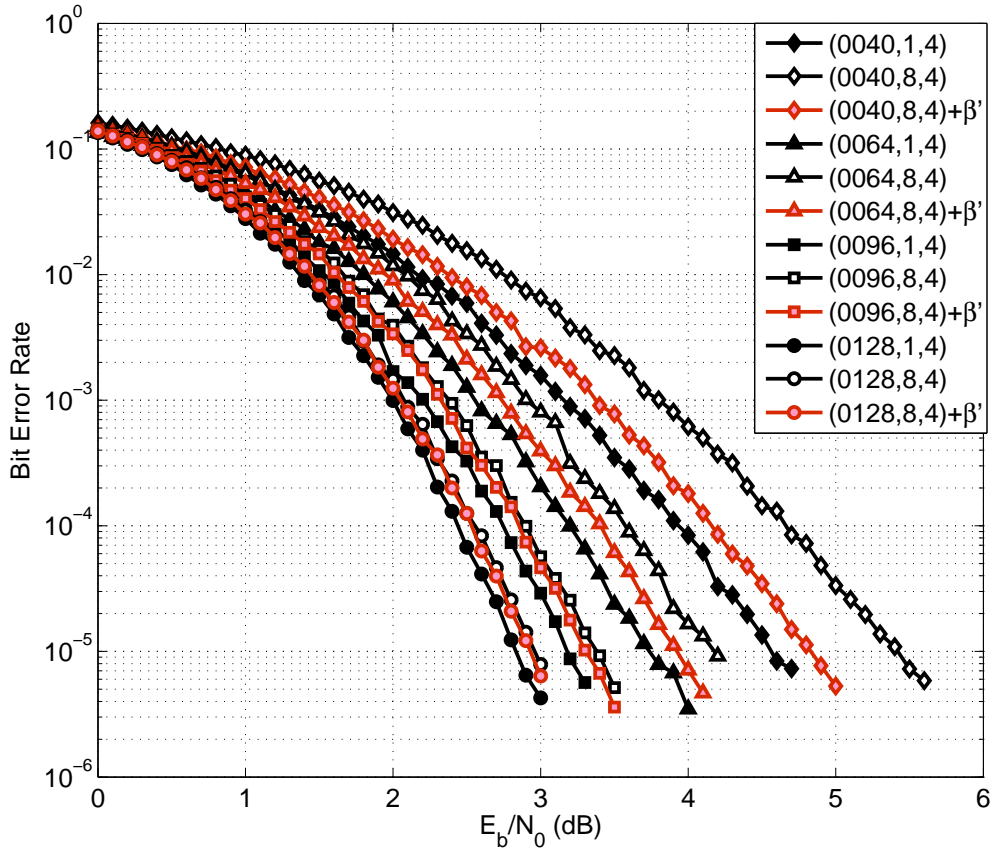


Figure 5.4: Performance of the proposed 3GPP LTE turbo decoder with $N \leq 128$

short trellis. Fig. 5.3(b) demonstrates the corresponding SISO decoder. Extra buffers and multiplexers for β'_d , β' , and α' are added to the conventional architecture in Fig. 2.15 (a) [44]. Compared to the SISO decoder in Fig. 2.16(a) [46], the previous β 's are fed into the β_d -ACS rather than β -ACS.

Fig. 5.4 and Fig. 5.5 present the fixed-point simulation results of small blocks with $\mathcal{P}_S = 1$ and $\mathcal{P}_S = 8$, where the legend format is {block size, parallelism, iteration}, and those legends with β' stands for the use of previous β . The modes with $\mathcal{P}_S = 8$ apply both α' and β'_d operations. When the parallel processing makes the whole sub-block or the last window of each sub-block too small, the shortened trellis structure lowers the reliability of path metrics. In these cases, the β' operation is used to compensate the initial β and improve the performance degradation significantly. As shown in Fig. 5.4, the loss of size-40 block at 10^{-5} error rate is reduced from 1.0 dB to 0.3 dB, while the

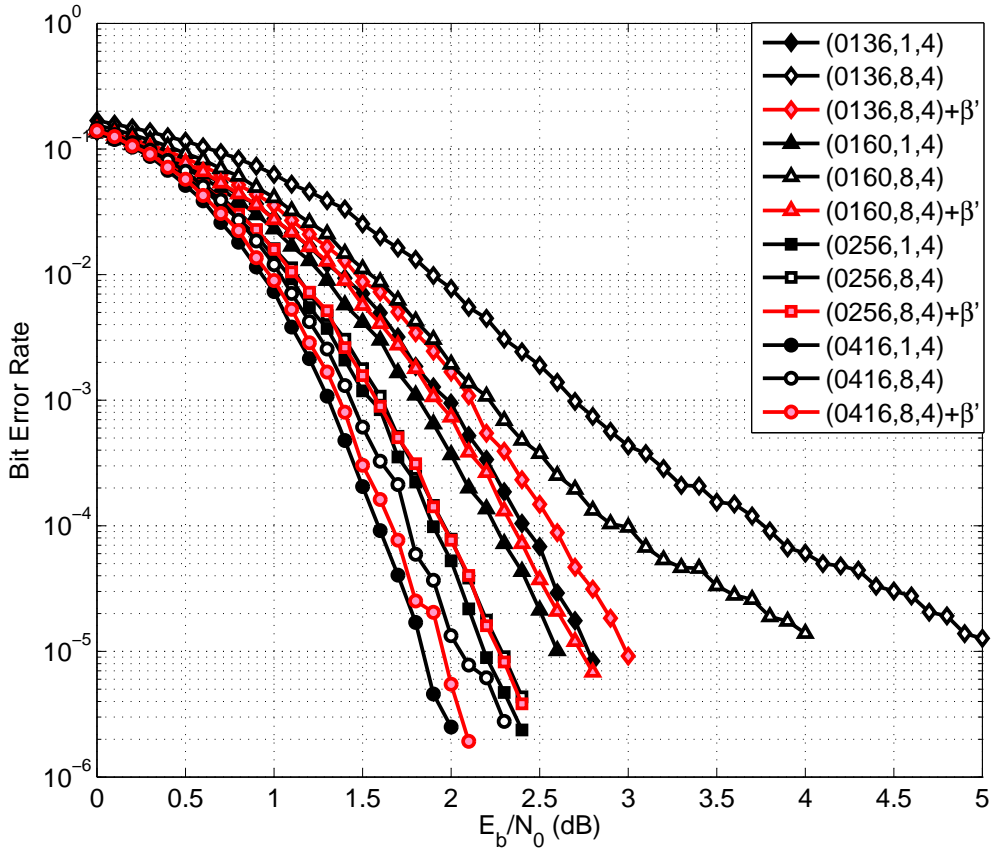


Figure 5.5: Performance of the proposed 3GPP LTE turbo decoder with small blocks

loss of size-64 block is reduced from 0.5 dB to 0.2 dB. In Fig. 5.5, the size-136, size-160, and size-416 blocks with β' achieve superior performance improvement in $\mathcal{P}_S = 8$. On the other hand, both initialization schemes can achieve similar performance for the modes with $16\mathcal{P}_S \mid N$ such as $N = 128$ and $N = 256$. Fig. 5.6 demonstrates the performance of large blocks. From the results whose window lengths are 16, the performance degrades slightly after using multiple SISO decoders. The loss in size-512 block is about 0.1 dB, and the losses in the others can be negligible. However, the error floor regions of size-4096 and size-6144 blocks appear before 10^{-6} error rate. Extending the window can enhance the performance of these blocks, but it would also introduce more area overhead.

The selection of processing schedule and window length are influenced by performance and hardware cost. When L is 16, one SISO decoder without β' needs 34.6k gates count. The utilization of β' would increase the equivalent gates count to 36.9k. It

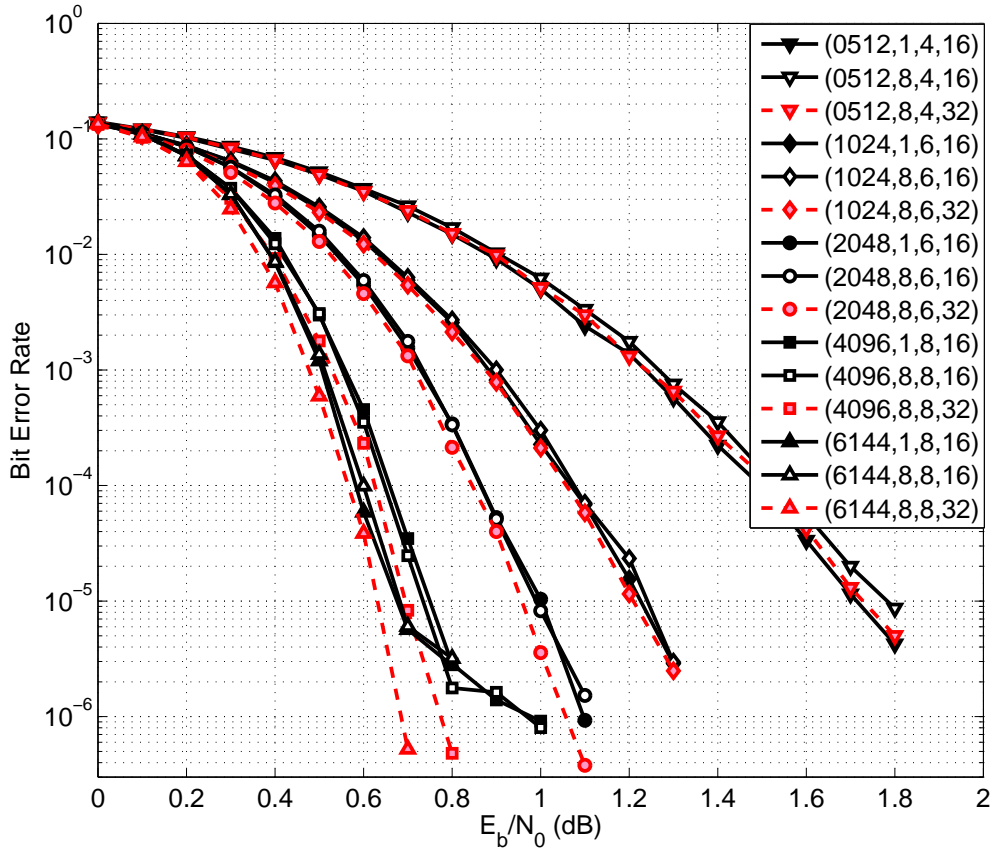


Figure 5.6: Performance of the proposed 3GPP LTE turbo decoder with large blocks

costs low overhead to guarantee the error correction capability of small blocks in the parallel architecture. If the window length L is extended to 32, the SISO decoder requires additional 10k gates count to store more temporary data. Since this growth is substantial, we have to make a trade-off between the area and performance.

The proposed design consists of 8 SISO decoders and 8 separate memory modules. The block sizes and interleaving parameters must satisfy $40 \leq N \leq 6144$, $8 \mid N$, $2 \nmid f_1$, and $2 \mid f_2$ due to available memory and network constraints, and it can support the 188 block sizes in 3GPP LTE standard. In addition to the configurable iteration number \mathcal{I} , the parallelism can be 1, 2, 4, or 8 at each block size. After determining the N , f_1 , f_2 , \mathcal{I} , and \mathcal{P}_S , this decoder would initialize the address generator and network controller within 16 cycles; then it starts to decoding received blocks. Our design is fabricated with 90 nm process and operated successfully at 275 MHz from measurement results. TABLE 5.2 lists

Table 5.2: Throughput of selected modes with 275 MHz frequency

Size (Iteration)	Throughput (Mb/s) / Operating Efficiency $\eta_{\mathcal{H}}$ (%)			
	$\mathcal{P}_{\mathcal{S}} = 1$	$\mathcal{P}_{\mathcal{S}} = 2$	$\mathcal{P}_{\mathcal{S}} = 4$	$\mathcal{P}_{\mathcal{S}} = 8$
40 (4)	16 / 46.5	21 / 30.3	31 / 22.7	47 / 17.2
128 (4)	25 / 73.6	40 / 58.2	56 / 41.0	71 / 25.8
256 (4)	29 / 84.8	51 / 73.6	80 / 58.2	113 / 41.0
512 (4)	32 / 91.8	58 / 84.8	101 / 73.6	160 / 58.2
1024 (6)	21 / 95.7	42 / 91.8	78 / 84.8	135 / 73.6
2048 (6)	22 / 97.8	44 / 95.7	84 / 91.8	155 / 84.8
4096 (8)	16 / 98.9	34 / 97.8	66 / 95.7	126 / 91.8
6144 (8)	17 / 99.3	34 / 98.5	67 / 97.1	130 / 94.4

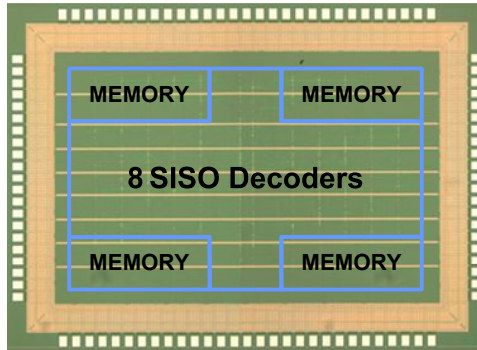


Figure 5.7: Photo of the proposed turbo decoder for 3GPP LTE application

the operating efficiency $\eta_{\mathcal{H}}$ and the throughput derived from (5.1) in various modes. For small blocks, there is a noticeable decline in $\eta_{\mathcal{H}}$, leading to less throughput improvement than large blocks in higher parallelism. When $\mathcal{P}_{\mathcal{S}}$ is 8, the blocks with $N \geq 256$ can achieve 100 Mb/s. By setting \mathcal{I} to 6, the proposed decoder can approximate such target at the expense of 0.1 dB performance loss at 10^{-5} error rate.

Fig. 5.7 shows the chip micro-photo, where the 2.10 mm^2 core area includes 0.634 mm^2 memory. The total gates count is 602k with 81.02% core utilization. Fig. 5.8 illustrates the measured power consumption of various sizes in four parallel modes. The power of size-4096 block is 111 mW, 128 mW, 162 mW, and 213 mW; as well as the power of size-6144 is 111 mW, 128 mW, 164 mW, and 219 mW for different parallel modes (i.e. 1, 2, 4, and 8, respectively). As $\mathcal{P}_{\mathcal{S}}$ increases from 1 to 8, both size-4096 and size-6144 blocks have around 7.6 times speedup while costing double power. The power growth is mainly

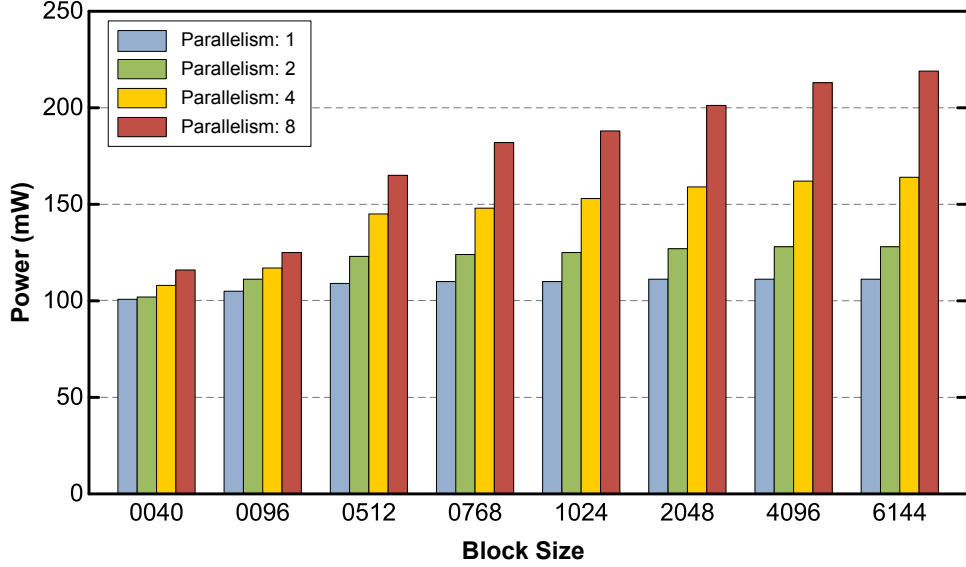


Figure 5.8: Power consumption from measurement with 1.0 V and 275 MHz.

caused by the increasing switching activity of more utilized SISO decoders, and there is certain common power dissipation in all modes. In addition, the $\eta_{\mathcal{H}}$, affected by block size deeply, is also an important factor to switching activity. When $\mathcal{P}_{\mathcal{S}}$ is fixed, more power is required in larger blocks, and the increment is in proportion to the change in $\eta_{\mathcal{H}}$.

Table 5.3: Comparison of different parallel designs for wireless application

	Proposed	[83]	[84]	[85]	[86]
Max. Block Size	6144	6144	480	6144	6144
Max. $\mathcal{P}_{\mathcal{S}}$	8	32	10	8	8
Max. $\mathcal{P}_{\mathcal{T}}$	1	2	2	2	2
Max. Iteration	8	6	4	8	5.5
Technology	90 nm	65 nm	130 nm	130 nm	130 nm
Supply Voltage (V)	1.0	N/A	1.2	1.2	1.2 V
Area (mm ²)	2.10	N/A	6.66 (3.20 [†])	10.7 (3.20 [†])	3.57
Frequency (MHz)	275	200	100	250	302
Throughput (Mb/s)	130	711	115	186	391
Power (mW)	219	N/A	197	N/A	789
Energy Efficiency (nJ/(bit·iteration))	0.21	N/A	0.43 (0.14 [‡])	0.61 (0.20 [‡])	0.37 (0.12 [‡])

[†] The normalized factor for area is 0.48 (= (90 nm/130 nm)²).

[‡] The normalized factor for energy efficiency is 0.33 (= (1.0 V/1.2 V)² × (90 nm/130 nm)²).

The throughput is 130 Mb/s while using 8 SISO decoders to process the size-6144 block for 8 iterations. The power consumption is 219 mW with 1 V supply in this mode, and the energy efficiency is 0.21 nJ/(bit·iteration). TABLE 5.3 lists the chip summary and the comparison with simulation results in [83, 84] and measurement results in [85, 86]. Except the decoder in [84], the others can support the 3GPP LTE standard. All these works are parallel turbo decoders with contention-free interleavers, and the last four designs utilize radix-4 structure. The results of each design are derived with its largest block size, iteration, and parallelism. The technology scaling of area and energy efficiency is given for reference.

5.3 Full-efficiency design using QPP interleaver

This design adopts the processing schedule and architecture in Fig. 2.16. Instead of β_d calculation, the previous boundary metrics are used to initial the path metric of each window. The actual hardware cost depends on the radix factor \mathcal{P}_T and window number κ . After exploiting the high-radix structure, the area of combinational circuits grows rapidly, but the boundary metric storage is unaffected. If the processed blocks are small, the hardware cost of the modified SISO decoder might be less than that of conventional design. This advantage will become noticeable in the high-radix SISO decoder. TABLE 5.4 gives the synthesis results of the SISO decoder with architectures in

Table 5.4: Area of main components in different radix- $2^{\mathcal{P}_T}$ SISO decoders

Architecture	Equivalent gates count ($L = 32, \kappa = 4, 5\text{-bit input, and } 8\text{-bit metric}$)			
	Fig. 2.15(a) (α, β, β_d)		Fig. 2.16(a) (α, β)	
	Radix- 2^1	Radix- 2^4	Radix- 2^1	Radix- 2^4
Branch metric units	0.25k×3	2.45k×3	0.25k×2	2.45k×2
ACS circuits	2.67k×3	9.32k×3	2.83k×2	9.97k×2
Input buffers	8.35k		4.41k	
Boundary storage	2.57k		7.48k	
Summation	19.7k	46.2k	18.1k	36.7k

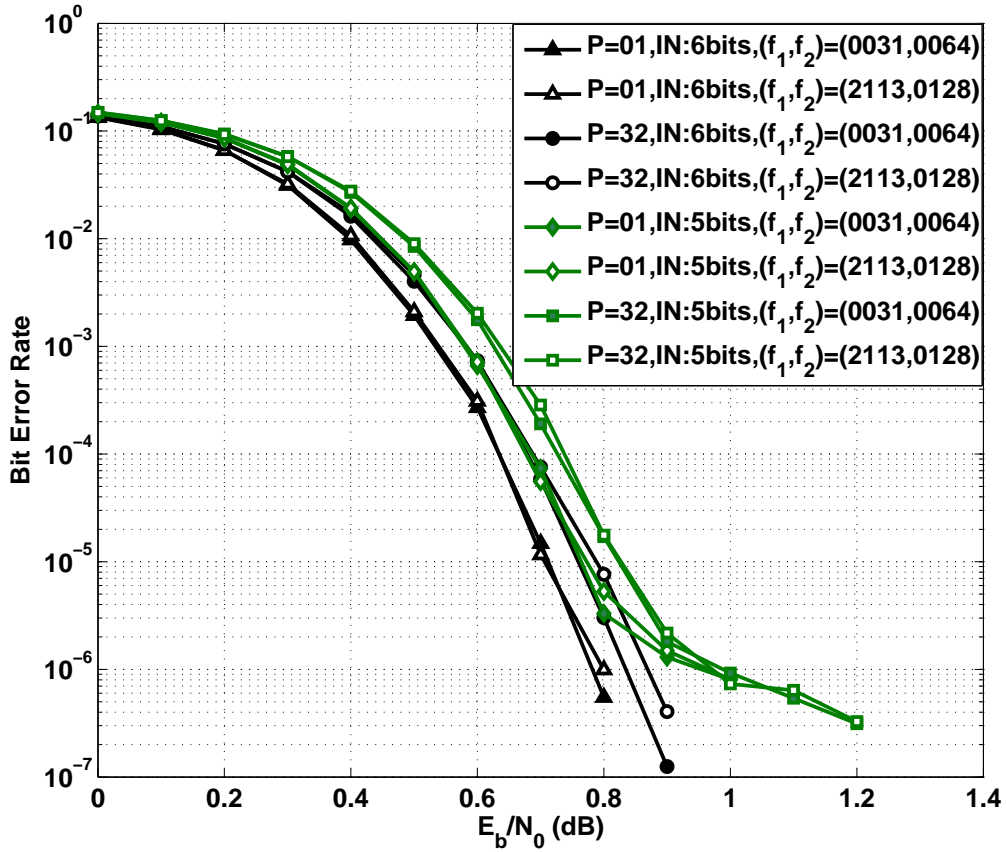


Figure 5.9: Performance of the proposed codes with various parameters

Fig. 2.15(a) and Fig. 2.16(a). Only the components greatly affected by different schedules are listed here. The conventional SISO decoder also requires some boundary storage for appropriate initialization in the parallel architecture. When the SISO decoder is designed for $N/\mathcal{P}_S = 128$ and $L = 32$, the circuits with the modified schedule save about 1.5k and 9.5k gates count in radix-2¹ and radix-2⁴ structures, respectively. Consequently, the parallel architecture using such radix-2⁴ SISO decoder for size-128 sub-block can benefit both operating efficiency and area overhead.

Fig. 5.9 shows the fixed-point simulation results of size-4096 blocks and length-32 windows with various combinations of (f_1, f_2) , parallelism, and data width. The tail-biting method in [27] is exploited in the constituent convolutional code with generator matrix (2.48). The data width of path metrics is 9 bits for 6-bit input and 8 bits for 5-bit input. In both cases, the last 3 bits of quantized input are used for the fraction

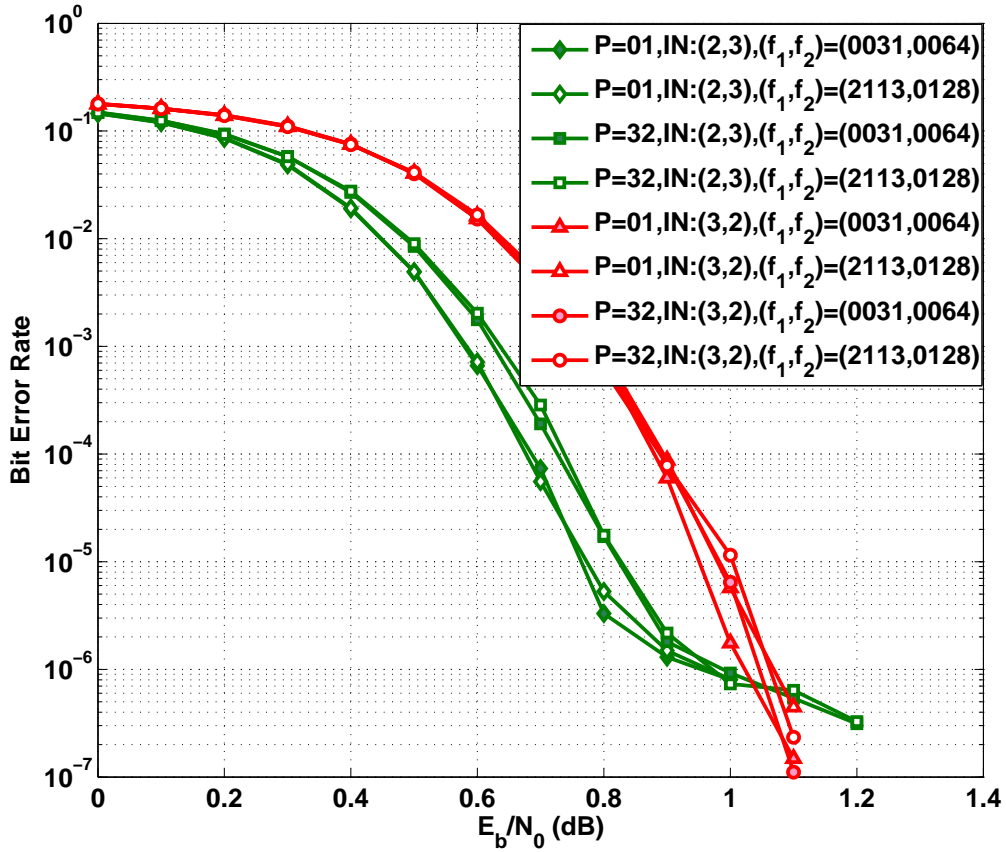


Figure 5.10: Performance with different formats of data width

part, while the others are for the integer part. Under the same parallelism and data width, the performance with $(f_1, f_2) = (2113, 128)$, which can meet Proposition 4.1, is similar to 3GPP LTE standard with $(f_1, f_2) = (31, 64)$ [15]. In contrast to the design with single SISO decoder, using 32 parallel SISO decoders will cause less than 0.1 dB performance loss. On the other hand, shorter data width leads to slight loss in low E_b/N_0 region, but it gives rise to error floor after the error rate reaching 10^{-6} . The reason is that the quantization of internal data fails to correctly represent the maximal or minimal value. One possible solution to the above problem is adjusting the expression of input data. Fig. 5.10 illustrates the results with different formats of data width, where the new format involves 2-bit fraction part and 3-bit integer part. Although the loss in precision makes performance degrade about 0.2 dB in low E_b/N_0 region, its extension of expression range prevents error floor occurring too early. If there is sufficient design area,

the employment of 6-bit input is preferred; otherwise, we choose another quantization with moderate performance and less overhead.

Our design with 32 radix-2⁴ SISO decoders utilizes barrel-shift network for lower routing effort [82] and *two-stage* technique for less area overhead [71]. The design can configure the (f_1, f_2) and choose the suitable schedule. If the (f_1, f_2) could satisfy Proposition 4.1 or Proposition 4.2, the turbo decoder can operate with either overlapping process ($\eta_{\mathcal{H}} = 100\%$) or normal process ($\eta_{\mathcal{H}} = 66.7\%$); otherwise, only normal process ($\eta_{\mathcal{H}} = 66.7\%$) is permitted. From synthesis results, the data with 6-bit codewords (9-bit metrics) requires 3304k gates count, while that with 5-bit codewords (8-bit metrics) requires 2787k. Considering the available area, the proposed design adopts the latter quantization. Fig. 5.11 shows the layout graph implemented with 90 nm technology. The post-layout simulation results indicate that the core area is 9.61 mm² with 85.75% utilization. Its equivalent gates count is 2833k. The maximal operating frequency is 175 MHz; so this design can reach 1.4 Gb/s with $\eta_{\mathcal{H}} = 100\%$ and 933 Mb/s with $\eta_{\mathcal{H}} = 66.7\%$ while executing for 8 iterations. The power consumption of these two modes is 1.356 W and 0.994 W respectively. TABLE 5.5 lists the design summary and compares parallel turbo decoders with QPP interleaver [82, 83, 85, 86]. The simulation results in [83] and the measurement results in [77, 85, 86] are given. We calculate the operating efficiency of these works via

$$\eta_{\mathcal{H}} = \frac{\text{Throughput} \times 2 \times \mathcal{I}}{\mathcal{P}_{\mathcal{H}} \times \mathcal{F}_{\mathcal{H}}}$$

Their area and energy efficiency are also normalized for reference. With specific parameters and overlapping half-iterations, the proposed design avoids the decreasing operating efficiency and achieves significant speedup in the parallel architecture.

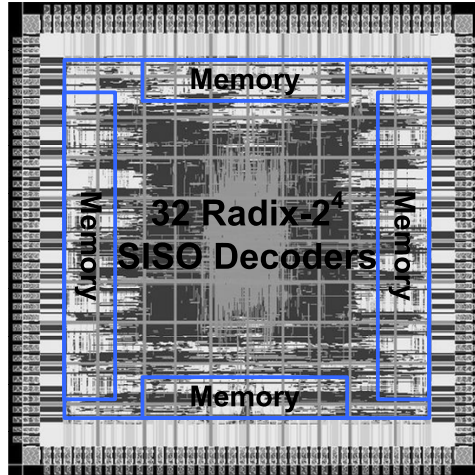


Figure 5.11: Layout graph of the proposed design with 100% efficiency.

Table 5.5: Comparison of different parallel turbo decoders using QPP interleaver

	Proposed	[83]	[82]	[85]	[86]
Technology	90 nm	65 nm	90 nm	130 nm	130 nm
Voltage (V)	0.9	N/A	1.0	1.2	1.2
Max. Block Size	4096	6144	6144	6144	6144
Max. \mathcal{P}_S	32	32	8	8	8
Max. \mathcal{P}_T	4	2	1	2	2
Max. Iteration	8	6	8	8	5.5
Operating Efficiency	100%	66.7%	94.4%	74.4%	89.0%
Area (mm ²)	9.61	N/A	2.10	10.7 (5.12 [‡])	3.57 (1.71 [‡])
Frequency (MHz)	175	200	275	250	302
Throughput (Mb/s)	1400	711	130	186	391
Power (mW)	1356	N/A	219	N/A	789
Energy Efficiency (nJ/(bit·iteration))	0.12	N/A	0.21 (0.17 [†])	0.61 (0.16 [‡])	0.37 (0.10 [‡])

[‡] The normalized factor for this area is 0.48 ($= (90 \text{ nm}/130 \text{ nm})^2$).

[†] The normalized factor for this energy metric is 0.81 ($= (0.9 \text{ V}/1.0 \text{ V})^2$).

[‡] The normalized factor for this energy metric is 0.27 ($= (90 \text{ nm}/130 \text{ nm})^2 \times (0.9 \text{ V}/1.2 \text{ V})^2$).

Chapter 6

Conclusion

6.1 Summary

The research on the turbo decoder design and implementation is reported in this dissertation. Among various design issues, our discussions focus on the throughput, complexity, and performance. Exploiting parallel architecture is a common method to get high throughput, and it is usually classified into three levels: parallel turbo decoder level, parallel SISO decoder level, and parallel trellis stage level. The throughput enhancement and extra circuit overhead of these parallel levels are quite different. Although the first level can be applied to any turbo decoder, it needs extra memories for multiple codewords and does not shorten the decoding latency. The other two levels have lower cost and less processing time for single codeword, yet they have difficulties in parallel data transmission and lower operating efficiency. Most practical designs choose parallel SISO decoder level for its benefits, and some of them also combines the parallel trellis stage level for more speedup. Our proposed turbo decoders are also based on such hybrid levels. However, it will encounter severe collision problems, complex data transmission, and considerable efficiency loss, especially in high parallelism. To overcome these difficulties, the contention-free interleavers is utilized to avoid collision during memory access. We further investigate multi-stage networks to connect multiple SISO decoders and multiple memory modules in the parallel turbo decoders using two different contention-free interleavers. For IBP interleaver, the interleaving rules based on butterfly network and double prime method are introduced. For QPP interleaver, a barrel-shift network is developed. Both networks can provide the necessary interconnections in their respective designs, and

they will take lower routing effort than the fully-connected network. The implementation of highly parallel turbo decoders becomes much simpler.

The decreasing operating efficiency is another drawback in the parallel architecture. When decoding small blocks or sub-blocks, the data dependency between constituent codes makes the executions of component circuits inefficient. We need to resolve such dependency and modify the processing schedule; then the functional units will have shorter inactive periods across two successive half-iterations. Our solution is using the inactive period of original schedule for processing independent data. The interlaced schedule is proposed for the design with arbitrary interleaver. Its decoding flow alternates between the half-iterations of two or more individual codewords. We can treat it as a variation of parallel turbo decoder level. Although the interlaced schedule requires more memory for additional codewords and might extend decoding time, the flexible method can compensate the low efficiency in many applications. If the turbo decoder adopts the QPP interleaver that can meet certain constraints, the overlapping half-iterations can be applied to it. Unlike the interlaced schedule, such decoding process deals with single codeword. With the constrained interleaving rules, the decoder can carry out partial processes of current half-iteration and partial processes of following half-iteration concurrently. In the schedule with overlapping half-iterations, we must utilize the modified SISO decoder that initialize the path metric by previous results. Moreover, the execution order of all sliding windows should be arranged. The second method has more restrictions, but it avoids extra overhead and remains outstanding performance.

Based on the proposed methodologies, several parallel turbo decoders are implemented. The first decoder chip, implemented with 130 nm technology, uses IBP interleaver and contains 32 radix-2² SISO decoders; it has 160 Mb/s and 0.22 nJ/(bit-iteration) after measurement. The second design also chooses IBP interleaver; it uses 16 radix-2⁴ SISO decoders and achieves 100% efficiency with interlaced half-iterations, leading to 1000 Mb/s and 0.15 nJ/(bit-iteration) in post-layout simulation with 90 nm technology. The third design employs QPP interleaver and supports the 3GPP LTE standard; it can allow

one, two, four, or eight SISO decoders to decode each block with configurable iteration. A robust path metric initialization is given to improve the performance loss in small blocks and high parallelism. After fabrication in the 90 nm process, the chip can achieve 130 Mb/s with 219 mW and 0.21 nJ/(bit·iteration) for the size-6144 block and eight iterations. The last design using QPP interleaver consists of 32 radix-2⁴ SISO decoders. It provides the processing mode with overlapping half-iterations and 100% efficiency when the parameters satisfy the constraints. Our post-layout simulation 90 nm technology indicates that the maximal throughput is 1.4 Gb/s with 1356 mW power consumption and 0.12 nJ/(bit·iteration) energy efficiency.

6.2 Future work

The throughput of a parallel turbo decoder is dominated by the parallelism, frequency, operating efficiency, and iteration number. There are still many design challenges with respect to these factors. In the parallel architecture, the component circuits are duplicated to process multiple data every cycle. The maximal parallelism might be restricted due to the available area. If we could use shorter data width for all quantized symbols without performance loss, higher parallelism is possible, and the critical path delay can be decreased. For operating efficiency, the schedule with overlapping half-iterations is preferred due to less overhead. As this method can support other interleavers than QPP interleaver, it can be exploited in more applications. In order to maintain the error-correcting capability, it always takes 8 or more iterations for large blocks. Using early stopping criterion can lower the average iteration. We would like to further shorten the maximal iteration even though the transmitted data suffer from large noise. As a consequence, more significant speedup will be feasible by optimizing the four factors.

References

- [1] C. E. Shannon, “A mathematical theory of communication,” *Bell Syst. Tech. J.*, vol. 27, pp. 379–428(Part I), Jul. 1948.
- [2] ———, “A mathematical theory of communication,” *Bell Syst. Tech. J.*, vol. 27, pp. 623–656(Part II), July 1948.
- [3] R. J. McEliece, *The theory of information and coding*, 2nd ed. Cambridge, UK: Cambridge University Press, 2004.
- [4] S. Lin and D. J. Costello, Jr., *Error control coding: fundamentals and applications*, 2nd ed. Englewood Cliffs, NJ: Pearson-Hall, 2004.
- [5] C. Berrou, A. Glavieux, and P. Thitimajshima, “Near Shannon limit error-correcting coding and decoding: turbo-codes,” in *IEEE Proc. Int. Conf. Communications*, May 1993, pp. 1064–1070.
- [6] D. J. C. MacKay and R. M. Neal, “Near Shannon limit performance of low density parity check codes,” *Electronics Letters*, vol. 33, no. 6, pp. 457–458, Mar. 1997.
- [7] B. Vucetic and J. Yuan, *Turbo codes: principles and applications*. Boston, MA: Kluwer Academic, 2000.
- [8] D. J. Costello, Jr. and G. D. Forney, Jr., “Channel coding: The road to channel capacity,” *Proc. IEEE*, vol. 95, no. 6, pp. 1150–1177, Jun. 2007.
- [9] E. Boutillon, C. Douillard, and G. Montorsi, “Iterative decoding of concatenated convolutional codes: Implementation issues,” *Proc. IEEE*, vol. 95, no. 6, pp. 1201–1227, Jun. 2007.
- [10] K. Gracie and M.-H. Hamon, “Turbo and turbo-like codes: Principles and applications in telecommunications,” *Proc. IEEE*, vol. 95, no. 6, pp. 1228–1254, Jun. 2007.
- [11] B. Vucetic, Y. Li, L. C. Pérez, and F. Jiang, “Recent advances in turbo code design and theory,” *Proc. IEEE*, vol. 95, no. 6, pp. 1323–1344, Jun. 2007.
- [12] *Technical Specification Group Radio (3GPP) Access Network; Multiplexing and channel coding (FDD)*, 3GPP Std. TS 25.212, Rev. 8.0.0, 2007.
- [13] *Physical Layer Standard for CDMA2000 Spread Spectrum Systems*, 3GPP2 Std. C.S0002-C, 2002.

- [14] *Local and Metropolitan Area Network Part 16: Air Interface for Fixed Broadband Wireless Access Systems*, IEEE Std. 802.16e/D11, 2005.
- [15] *Technical Specification Group Radio Access Network; Evolved Universal Terrestrial Radio Access; Multiplexing and channel coding (Release 8)*, 3GPP Std. TS 36.212, Rev. 8.5.0, Dec. 2008.
- [16] L. R. Bahl, J. Cocke, F. Jelinek, and J. Raviv, "Optimal decoding of linear codes for minimizing symbol error rate," *IEEE Trans. Inf. Theory*, vol. IT-20, pp. 284–287, Mar. 1974.
- [17] P. Elias, "Coding for noisy channels," *IRE Conv. Rec.*, vol. pt.4, pp. 37–47, 1955.
- [18] G. D. Forney, Jr., "Convolutional codes I: algebraic structure," *IEEE Trans. Inf. Theory*, vol. IT-16, pp. 720–738, Nov. 1970.
- [19] R. Johannesson and Z. X. Wan, "A linear algebra approach to minimum convolutional encoders," *IEEE Trans. Inf. Theory*, vol. 39, no. 4, pp. 37–47, Jul. 1993.
- [20] A. J. Viterbi, "Error bounds for convolutional codes and asymptotically optimum decoding algorithm," *IEEE Trans. Inf. Theory*, vol. IT-13, no. 2, pp. 260–269, Apr. 1967.
- [21] J. Hagenauer, E. Offer, and L. Papke, "Iterative decoding of binary block and convolutional codes," *IEEE Trans. Inf. Theory*, vol. 42, no. 2, pp. 429–445, Mar. 1996.
- [22] P. Robertson, E. Villebrun, and P. Hoeher, "A comparison of optimal and sub-optimal decoding algorithm," in *IEEE Proc. Int. Conf. Communications*, Jun. 1995, pp. 1009–1013.
- [23] J. A. Erfanian, S. Pasupathy, and G. Gulak, "Reduced complexity symbol detectors with parallel structures for ISI channels," *IEEE Trans. Commun.*, vol. 42, no. 2/3/4, pp. 1261–1271, Feb./Mar./Apr. 1994.
- [24] S. A. Barbulescu, "Iterative decoding of turbo codes and other concatenated codes," Ph.D. dissertation, Univ. South Australia, 1996.
- [25] A. J. Viterbi, "An intuitive justification and a simplified implementation of the MAP decoder for convolutional codes," *IEEE J. Sel. Areas Commun.*, vol. 16, no. 2, pp. 260–264, Feb. 1998.
- [26] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near optimum error correcting coding and decoding: turbo-codes," *IEEE Trans. Commun.*, vol. 44, no. 10, pp. 1261–1271, Oct. 1996.
- [27] C. Weiß, C. Bettstetter, S. Riedel, and D. J. Costello, "Turbo decoding with tail-biting trellises," in *IEEE Proc. URSI Int. Symp. Signals, Systems, and Electronics*, Oct. 1998, pp. 343–348.
- [28] S. Benedetto and G. Montorsi, "Unveiling turbo codes: Some results on parallel concatenated coding schemes," *IEEE Trans. Inf. Theory*, vol. 42, no. 2, pp. 409–428, Mar. 1996.

- [29] S. Benedetto, G. Montorsi, and D. Divsalar, “Concatenated convolutional codes with interleavers,” *IEEE Commun. Mag.*, vol. 41, no. 8, pp. 102–109, Aug. 2003.
- [30] H. R. Sadjadpour, N. J. A. Sloane, M. Salehi, and G. Nebe, “Interleaver design for turbo codes,” *IEEE J. Sel. Areas Commun.*, vol. 19, no. 5, pp. 831–837, May 2001.
- [31] 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.
- [32] A. Worm, P. Hoeher, and N. Wehn, “Turbo-decoding without SNR estimation,” *IEEE Commun. Lett.*, vol. 4, no. 6, pp. 193–195, Jun. 2000.
- [33] J. Vogt and A. Finger, “Improving the Max-Log-MAP turbo decoder,” *IET Electronics Letters*, vol. 36, no. 23, pp. 1937–1937, Nov. 2000.
- [34] P. H. Y. Wu and S. M. Pisuk, “Implementation of a low complexity, low power, integer-based turbo decoder,” in *IEEE Global Telecommunications Conf.*, Nov. 2001, pp. 946–951.
- [35] Y. Wu, B. D. Woener, and T. K. Blankenship, “Data width requirements in SISO decoding with modulo normalization,” *IEEE Trans. Commun.*, vol. 49, no. 11, pp. 1861–1868, Nov. 2001.
- [36] T. K. Blankenship and B. Classon, “Fixed-point performance of low-complexity turbo decoding algorithms,” in *IEEE Vehic. Tech. Conf.*, May 2001, pp. 1483–1487.
- [37] A. P. Hekstra, “An alternative to metric rescaling in Viterbi decoders,” *IEEE Trans. Commun.*, vol. 37, no. 11, pp. 1220–1222, Nov. 1989.
- [38] C. B. Shung, P. H. Siegel, G. Ungerboeck, and H. K. Thapar, “VLSI architectures for metric normalization in the Viterbi algorithm,” in *IEEE Proc. Int. Conf. Communications*, vol. 4, Atlanta, CA, Apr. 1990, pp. 1723–1728.
- [39] J. H. Han, A. T. Erdogan, and T. Arslan, “High speed Max-Log-MAP turbo SISO decoder implementation using branch metric normalization,” in *IEEE Computer Society Annual Symp. VLSI*, 2005, pp. 173–178.
- [40] I. Lee and J. L. Sonntag, “A new architecture for the fast Viterbi algorithm,” *IEEE Trans. Commun.*, vol. 51, no. 10, pp. 1624–1628, Oct. 2003.
- [41] R. Y. Shao, S. Lin, and P. C. Fossorier, “Two simple stopping criteria for turbo decoding,” *IEEE Trans. Commun.*, vol. 47, no. 8, pp. 1117–1120, Aug. 1999.
- [42] Y. Wu, B. D. Woener, and W. J. Ebel, “A simple stopping criteria for turbo decoding,” *IEEE Commun. Lett.*, vol. 4, no. 8, pp. 258–260, Aug. 2000.
- [43] G. Masera, G. Piccinini, M. R. Roch, and M. Zamboni, “VLSI architecture for turbo codes,” *IEEE Trans. VLSI Syst.*, vol. 7, no. 3, pp. 369–379, Sep. 1999.
- [44] G. Masera, M. Mazza, G. Piccinini, F. Viglione, and M. Zamboni, “Architectural strategies for low-power VLSI turbo decoders,” *IEEE Trans. VLSI Syst.*, vol. 10, no. 3, pp. 279–285, Jun. 2002.

- [45] S. Yoon and Y. Bar-Ness, "A parallel MAP algorithm for low latency turbo decoding," *IEEE Commun. Lett.*, vol. 6, no. 7, pp. 288–290, Jul. 2002.
- [46] Z. He, P. Fortier, and S. Roy, "Highly-parallel decoding architecture for convolutional turbo codes," *IEEE Trans. VLSI Syst.*, vol. 14, no. 10, pp. 1147–1151, Oct. 2006.
- [47] R. Dobkin, M. Peleg, and R. Ginosar, "Parallel VLSI architecture for MAP turbo decoder," in *IEEE Int. Symp. Personal, Indoor and Mobile Radio Communications*, Sep. 2002, pp. 15–18.
- [48] M. Bickerstaff, L. Davis, C. Thomas, D. Garrett, and C. Nicol, "A 24Mb/s radix-4 LogMAP turbo decoder for 3GPP-HSDPA mobile wireless," in *IEEE Int. Solid-State Circuit Conf.*, Feb. 2003, pp. 151–484.
- [49] C. Benkeser, A. Burg, T. Cupaiuolo, and Q. Huang, "A 58mw 1.22mm² HSDPA turbo decoder ASIC in 0.13 μ m cmos," in *IEEE Int. Solid-State Circuit Conf.*, Feb. 2008, pp. 264–266.
- [50] O. Muller, A. Baghdadi, and M. Jézéquel, "Exploring parallel processing levels for convolutional turbo decoding," in *2nd Information and Communication Technologies*, Apr. 2006, pp. 2353–2358.
- [51] Y.-X. Zheng and Y.-T. Su, "A new interleaver design and its application to turbo codes," in *Proc. IEEE Vehicular Technology Conf.*, vol. 3, Sep. 2002, pp. 1437–1441.
- [52] O. Y. Takeshita, "On maximum contention-free interleavers and permutation polynomials over integer rings," *IEEE Trans. Inf. Theory*, vol. 52, no. 3, pp. 1249–1253, Mar. 2006.
- [53] M. J. Thul, N. Wehn, and L. P. Rao, "Enabling high-speed turbo decoding through concurrent interleaving," in *IEEE Proc. Int. Symp. Circuits and Systems*, May 2002, pp. 26–29.
- [54] A. Tarable, S. Benedetto, and G. Montorsi, "Mapping interleaving laws to parallel turbo and LDPC decoder architecture," *IEEE Trans. Inf. Theory*, vol. 50, no. 9, pp. 2002–2009, Sep. 2004.
- [55] A. Giulietti, L. V. der Perre, and M. Strum, "Parallel turbo coding interleavers: Avoiding collisions in accesses to storage elements," *Elec. Lett.*, vol. 38, no. 5, pp. 232–234, Feb. 2002.
- [56] D. Gnaedig, E. Boutillon, M. Jezequel, V. C. Gaudet, and P. G. Gulak, "On multiple slice turbo code," in *Proc. 3rd Int. Symp. Turbo Codes and Related Topics*, Sep. 2003, pp. 343–346.
- [57] A. Abbasfar and K. Yao, "Interleaver design for high speed turbo decoders," in *IEEE Wireless Communications and Networking Conf.*, Mar. 2004, pp. 1611–1615.
- [58] L. Dinoi and S. Benedetto, "Variable-size interleaver design for parallel turbo decoder architectures," in *IEEE Global Telecommunications Conf.*, Nov. 2004, pp. 3108–3112.

- [59] Z. He, S. Roy, and P. Fortier, "High speed and low power design of parallel turbo decoder," in *IEEE Proc. Int. Symp. Circuits and Systems*, 2005, pp. 6018–6021.
- [60] G. Prescher, T. Gemmeke, and T. Noll, "A parameterizable low-power high-throughput turbo decoder," in *IEEE Int. Acoustics, Speech, and Signal Processing Conf.*, Mar. 2005, pp. V/25–V/28.
- [61] H. Moussa, O. Muller, A. Baghdadi, and M. Jezequel, "Butterfly and bene-based on-chip communication networks for multiprocessor turbo decoding," in *Design, Automation and Test in Europe Conference and Exhibition*, Apr. 2007, pp. 1–6.
- [62] R. Dobkin, M. Peleg, and R. Ginosar, "Parallel interleaver design and VLSI architecture for low-latency MAP turbo decoders," *IEEE Trans. VLSI Syst.*, vol. 13, no. 4, pp. 427–438, Apr. 2005.
- [63] D. Gnaedig, E. Boutillon, J. Tusch, and M. Jézéquel, "Towards an optimal parallel decoding of turbo codes," in *Proc. 4th Int Symp. Turbo Codes Related Topics*, Apr. 2006.
- [64] P. J. Black and T. H. Meng, "A 140 Mb/s, 32-state, radix-4 Viterbi decoder," *IEEE Commun. Lett.*, vol. 27, no. 12, pp. 1877–1885, Dec. 1992.
- [65] H. Dawid, G. Fettweis, and H. Meyr, "A CMOS IC for Gb/s Viterbi decoding: system design and VLSI implementation," *IEEE Trans. VLSI Syst.*, vol. 4, no. 1, pp. 17–31, Mar. 1996.
- [66] C. C. Lin, C. C. Wu, and C. Y. Lee, "A low power and high speed Viterbi decoder chip for WLAN applications," in *Proc. 29th Europe Solid State Circuits Conf.*, Sep. 2003, pp. 723–726.
- [67] M. Anders, S. Mathew, R. Krishnamurthy, and S. Borkar, "A 64-state 2GHz 500Mbps 40mW Viterbi accelerator in 90nm CMOS," in *Symop. VLSI Circuits Dig. Tech. Papers*, 2004, pp. 174–175.
- [68] S. W. Choi and S. S. Choi, "200Mbps Viterbi decoder for UWB," in *Int. Conf. Advanced Communication Tech.*, vol. 2, 2005, pp. 904–907.
- [69] C.-L. Chen, "High-speed Viterbi decoder based on the two-dimensional ACS structure," Master's thesis, National Chiao-Tung University, 2005.
- [70] C.-C. Lin, "Channel decoder design and implementation," Ph.D. dissertation, National Chiao-Tung University, 2006.
- [71] C.-H. Tang, C.-C. Wong, C.-L. Chen, C.-C. Lin, and H.-C. Chang, "A 952Mb/s Max-Log MAP decoder chip using radix-4×4 ACS architecture," in *IEEE Asian Solid-State Circuits Conf.*, Nov. 2006, pp. 79–82.
- [72] S. Crozier, J. Lodge, P. Guinand, and A. Hunt, "Performance of turbo codes with relative prime and golden interleaving strategies," in *6th Int. Mobile Satellite Conf.*, Jun. 1999, pp. 268–275.

- [73] C.-C. Wong, C.-H. Tang, M.-W. Lai, Y.-X. Zheng, C.-C. Lin, H.-C. Chang, C.-Y. Lee, and Y. T. Su, "A 0.22nJ/b/iter 0.13 μ m turbo decoder chip using inter-block permutation interleaver," in *IEEE Custom Integrated Circuits Conf.*, Sep. 2007, pp. 273–276.
- [74] J. Ryu and O. Y. Takeshita, "On quadratic inverse for quadratic permutation polynomials over integer rings," *IEEE Trans. Inf. Theory*, vol. 52, no. 3, pp. 1254–1260, Mar. 2006.
- [75] R. Asghar, D. Wu, J. Eilert, and D. Liu, "Memory conflict analysis and implementation of a re-configurable interleaver architecture supporting unified parallel turbo decoding," *Journal of Signal Processing Systems*, vol. 60, no. 1, pp. 15–29, Jul. 2010.
- [76] I. Ahmed and C. Vithanage, "Dynamic reconfiguration approach for high speed turbo decoding using circular rings," in *Proceedings of the 19th ACM Great Lakes symposium on VLSI*, May 2009, pp. 475–480.
- [77] C.-C. Wong, Y.-Y. Lee, and H.-C. Chang, "A 188-size 2.1mm² reconfigurable turbo decoder chip with parallel architecture for 3GPP LTE system," in *Symposium on VLSI Circuits*, Jun. 2009, pp. 288–289.
- [78] S. Dolinar and D. Divsalar, "Weight distribution of turbo codes using random and nonrandom permutations," Jet Propulsion Lab., TDA Progress Report 42-122, Aug. 1995.
- [79] S. Crozier, "New high-spread high-distance interleavers for turbo-codes," in *Proc. 20th Bienn. Symp. Communications*, May 2000, pp. 3–7.
- [80] S. Crozier, P. Guinand, and A. Hunt, "Estimating the minimum distance of turbo-codes using double and triple impulse methods," *IEEE Commun. Lett.*, vol. 9, no. 7, pp. 631–633, Jul. 2005.
- [81] C.-C. Wong, M.-W. Lai, C.-C. Lin, H.-C. Chang, and C.-Y. Lee, "Turbo decoder using contention-free interleaver and parallel architecture," *IEEE J. Solid-State Circuits*, vol. 45, no. 2, pp. 422–432, Feb. 2010.
- [82] C.-C. Wong and H.-C. Chang, "Reconfigurable turbo decoder with parallel architecture for 3GPP LTE system," *IEEE Trans. Circuits Syst. II*, vol. 57, no. 7, pp. 566–570, Jul. 2010.
- [83] Y. Sun, Y. Zhu, M. Goel, and J. R. Cavallaro, "Configurable and scalable high throughput turbo decoder architecture for multiple 4G wireless standard," in *IEEE Int. Conf. on Application-Specific Systems, Architectures and Processors*, Jul. 2008, pp. 209–214.
- [84] C.-H. Lin, C.-Y. Chen, A.-Y. Wu, and T.-H. Tsai, "Low-power memory reduced trace-back MAP decoding for double-binary convolutional turbo decoder," *IEEE Trans. Circuits Syst. I*, vol. 56, no. 5, pp. 1005–1016, May 2009.

- [85] J.-H. Kim and I.-C. Park, "A unified parallel radix-4 turbo decoder for mobile WiMAX and 3GPP-LTE," in *IEEE Custom Integrated Circuits Conference*, Sep. 2009, pp. 487–490.
- [86] C. Studer, C. Benkeser, S. Belfanti, and Q. Huang, "A 380Mb/s 3.57mm² 3GPP-LTE turbo decoder ASIC in 0.13 μ m CMOS," in *IEEE Int. Solid-State Circuit Conf.*, Feb. 2010, pp. 274–276.

