國立交通大學

電子工程學系電子研究所

博士論文

通道解碼器之設計與實作

Channel Decoder Design and Implementation

研究生：林建青

指導教授：李鎮宜
張錫嘉

中華民國 九十五 年 六 月

通道解碼器之設計與實作

Channel Decoder Design and Implementation

研究生： 林建青　　Student： Chien Ching Lin

指導教授： 李鎮宜　博士　　Advisor： Dr. Chen-Yi Lee

張錫嘉　博士　　Dr. Hsie-Chia Chang

國立交通大學

電子工程學系電子研究所

博士論文

A Dissertation

Submitted to Department of Electronics Engineering & Institute 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

July 2006

Hsinchu, Taiwan, Republic of China.

中華民國 九十五 年 六 月

通道解碼器之設計與實作

林建青

國立交通大學電子工程學系電子研究所

指導教授：李鎮宜教授、張錫嘉教授

摘要

本論文由演算法到架構設計與電路實現探討通道解碼器。依解碼方式不同可分成三個主要部分討論，分別是代數解碼，機率解碼以及重複解碼方式。

採用代數解碼的 Reed-Solomon code 用以研究其通用多系統的可能性。我們應用 Montgomery 乘法演算法於通用有限場乘法器，使得有限場算數運算器不受有限場的限制。利用這些通用型運算器與簡化結構即可建構出一可符合大部分系統規範之 Reed-Solomon 解碼器，且不需進行電路修正。經由電路實現可以發現達成此通用能力僅需要小於一倍複雜度的額外成本，電路量測結果則證實解碼速度足以滿足目前大部分應用的需求。

Viterbi 解碼演算法是一種針對 convolutional code 的最大可能性解碼方式，目前被廣泛應用於數位通訊系統。針對低功率設計，我們採行動態殘存路徑記憶體存取以及資料路徑變形。殘存路徑記憶體搭配所提出之路徑匹配與路徑預測方法可根據通道狀況動態調整截斷長度。運用一暫存記憶體，當路徑回歸至暫存路徑時，即可關閉大量記憶體讀取而降低功率消耗。另一方面吾人提出將加法-比較-選擇運算轉換成比較-選擇-加法，藉以降低運算量而達到降低電路成本與功率消耗。由電路量測結果得知這兩種技術可降低 Viterbi 解碼器的功率消耗達 30%~ 40%。針對高速 Viterbi 解碼器則提出二維加法-比較-選擇運算結構，在實際基數 16 的柵狀圖上實現 Viterbi 解碼器可達到 1Gb/s 的資料解碼速度。

再進一步研究重複解碼方式，以turbo code 與 low-density parity-check (LDPC) code為對象。針對 turbo 解碼器應用於行動通訊系統並具有大型打亂器之編碼，以簡單解碼器架構降低電路成本，並考量記憶體最佳化以降低功率消耗。並且在一整合型 turbo 與 Viterbi 解碼器晶片上得到更好的能量效率。對於 LDPC 解碼器則因其高平行度解碼演算法而進行高速應用之研究。由於不規則之對偶檢查矩陣以及大量運算元造成大量不規則資訊交換，我們提出一交換式記憶體以容納此大量資訊之傳遞。在電路實做上可達到5.92Gb/s 之資料解碼速度，面積則因大於 70%之晶片密度而顯的更有效益。

本論文在研究三種通道解碼方式以及其應用上的實做，並探討系統規範提出可實現之方案並進行分析。最後透過電路實做進行量測或評估，實驗結果則顯示如預期之正面結論。

CHANNEL DECODER DESIGN AND IMPLEMENTATION

by
Chien-Ching Lin,
Department of Electronics Engineering
National Chiao Tung University, 2006
Chen-Yi Lee and Hsie-Chia Chang, Adviser

This dissertation investigates the channel decoders from algorithms to architecture designs and circuit implementation. Three different decoding schemes are studied, including the algebraic, the probabilistic, and the iterative decoding algorithms.

The Reed-Solomon code based on the algebraic decoding is exploited to many system specifications. We apply the Montgomery multiplication algorithm to the universal finite field multiplier; as a result, the arithmetic units are capable of different finite field definitions. The Reed-Solomon decoder is constructed based on the proposed arithmetic operations and modified for less complexity. Hence the decoder can be applied to many systems without circuit modification. The chip implementation results show that the overhead due to the universality is no more than 100%. Moreover, the decoding speed from the measurement can meet most current or future applications.

The maximum-likelihood decoding, Viterbi decoding algorithm, for the convolutional code is widely used in many digital communications. The low power design techniques for the Viterbi decoder are proposed for the dynamically survivor memory access and the datapath transformation. The survivor memory unit with the path merging and the path prediction algorithms can adaptively adjust the truncation length according to the channel conditions. Combining a cache buffer, we can avoid many read operations in the memory, leading less power consumption. On the other hand, we also transform the add-compare-select (ACS) operation to compare-select-add (CSA) for less computations resulting in lower cost as well as lower power dissipation. The implementation results indicate about 30%∼40% power reduction is accomplished with the proposed architecture. The high speed and area efficient Viterbi decoder is also presented with the two-dimensional ACS structure. The decoder on the radix-16 trellis is implemented and shown to achieve over 1Gb/s data throughput.

We further conduct the research into the iterative decoding based turbo codes and low-density parity-check (LDPC) codes. The turbo decoder is considered in the mobile communication system with large interleaver size. The simple decoder architecture is utilized for cost consideration, and the memory is optimized for power consumption. The unified turbo and Viterbi decoder chip is also shown to achieve better energy efficiency. The LDPC decoder is designed for high speed applications for its highly parallelizable decoding algorithm. Because of the irregular parity check matrix and the large number of processing elements, the register exchange memory is introduced to accommodate the large message passing in the decoder. As a result, the circuit implementation leads to a high decoding speed, which is 5.92Gb/s, and area efficient decoder chip whose chip density is larger than 70%.

In this dissertation, the research includes different channel decoding schemes as well as their implementation for applications. Exploring the system requirements, we provide various design methods and analysis for the decoders. Finally, the circuits are realized for measurement or analysis, and the results reveal the positive consequence as expected.

誌　謝

這幾年的博士求學生涯是一段豐富的旅程，在許多師長提攜與朋友的協助之下，一路走來既沈穩且順利。非常感謝指導教授李鎮宜老師提供了最好的研究環境，他的諄諄教誨賦予我面對更多挑戰的信心。另外我要感謝亦師亦友的張錫嘉教授，由於他的啟蒙與指導，使我能夠在通道編碼領域有所發揮，並且針對許多關鍵點提供了很好的建議。

Si2 實驗室是一個充滿知識的寶庫，感謝實驗室同仁相助，我得以在此學習成長，吸收許多寶貴經驗。OCEAN 研究團隊，由小樹成長為頗具規模的團隊，我幸運地參與了它的成長。OCEAN 團隊一直是我持續努力的動力，每位成員，戰戰兢兢，努力不懈。因為你們才能有這一份論文，因為有你們 OCEAN 團隊持續發展，並創造了許多珍貴的成果。

誠摯地感謝我的家人，你們無悔的付出，讓我無後顧之憂地完成博士論文。最後謝謝口試委員的指教與寶貴意見。

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

A communication system conveys a information source to a destination through a channel. The channel can be microwave links, wireline cables, or storage mediums. Fig. 1.1 illustrates a traditional digital communication system. The information data is first processed by the source encoder for more compactness. The data is compressed by assigning short descriptions to the frequent symbols and longer descriptions to the less frequent symbols [1, 2]. The channel encoder then transforms the source encoded data into longer sequence where some redundant symbols are introduced, also termed parity symbols. Afterward, the modulator will convert the channel coded symbols to analog signals transmitted through the channel and corrupted by noise, distortion, and interference. In the receiver, the demodulator estimates the received symbols from the channel outputs and produces the continuous or quantized symbols. The received symbols may contain errors because the demodulator processes the channel outputs including noise. If so, the channel decoder will use the coded data with parity symbols to recover the whole transmitted information. An ideal channel decoder would

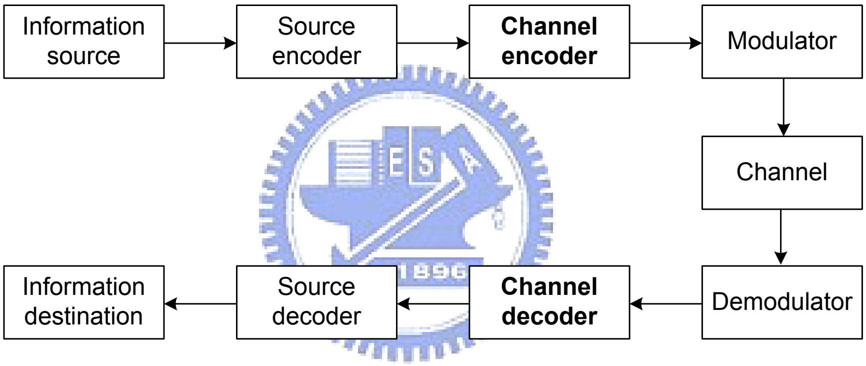


Figure 1.1: Block diagram of a digital communication system

correct all the errors and generate the data sequence matching the compressed information. Finally, the source decoder reproduces the information source according to the channel decoder outputs.

The development of the channel encoder and the decoder, referred to *channel coding* or *error control coding*, begins with the remarkable papers by C. E. Shannon in 1948 [3, 4]. Shannon indicated that any reliable transmission is achievable if the information rate R in bits per channel is less than the *channel capacity* C for the channel.

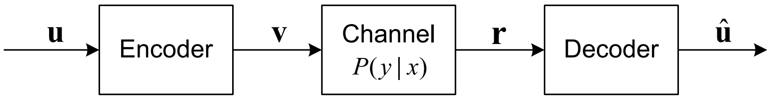


Figure 1.2: A communication channel

Fig. 1.2 presents a simplified communication system that focuses on the the channel coding. Assume two finite sets $\mathbf{X}$ and $\mathbf{Y}$ are the channel input and output alphabets, we can denote a discrete channel by the transition probability $P(y|x)$ for $x \in \mathbf{X}$ and $y \in \mathbf{Y}$. The coding scheme is first defined to be an (n, k) code with the encoding function

$$f : \mathbf{X}^k \mapsto \mathbf{X}^n \tag{1.1}$$

and the decoding function

$$g : \mathbf{Y}^n \mapsto \mathbf{X}^k. \tag{1.2}$$

The information sequence $\mathbf{u} \in \mathbf{X}^k$ is encoded to a codeword $\mathbf{v} \in \mathbf{X}^n$ with the rate $R = k/n$. For the discrete memoryless channel, we can transmit the codeword with the following probability

$$P(\mathbf{r}|\mathbf{v}) = \prod_{i=1}^{n} P(x = r_i|y = v_i), \tag{1.3}$$

where $r_i \in \mathbf{X}$ and $v_i \in \mathbf{Y}$. The channel decoder will estimate the information sequence $\hat{\mathbf{u}}$ that maximizes $P(\mathbf{v}|\mathbf{r})$. The decoding error occurs when $\hat{\mathbf{u}} \neq \mathbf{u}$, and the error probability is $P_e = P\{\hat{\mathbf{u}} \neq \mathbf{u}\}$. The *channel coding theorem* [3, 4] by Shannon states that for all rates

$R < C$, there exists a (n, k) code that can achieve $P_e \rightarrow 0$.

The first class of codes, *block codes*, has an algebraic coding structure [5]. The first block code, discovered by R. E. Hamming [6] in 1950, is a class of single error correction codes and termed Hamming codes. Another class of multiple random error correcting block code is the Reed-Muller code that first introduced by D. E. Muller in 1954 [7] and completed by I. S. Reed [8] for error correction and detection in communication and data storage systems. The breakthrough in 1960 was due to Bose and Ray-Chaudhuri [9], and Hocquenghem [10] who found a large class of multiple error correcting codes, referred to BCH codes. The important non-binary Reed-Solomon codes are proposed by I. S. Reed and G. Solomon in 1960 [11], and have been classified into non-binary BCH codes [12].

The second class is the *convolutional code* that has the probabilistic decoding features. The convolutional code was first introduced by P. Elias in 1955 [13], and the efficient sequential decoding method was then proposed by J. M. Wozencraft and B. Reiffen [14]. The Viterbi algorithm proposed by A. J. Viterbi in 1967 [15] is a much simple algorithm for implementation, leading to the widespread applications of the convolutional code. The optimal symbol by symbol detection algorithm that estimates the *a posteriori* probabilities of information symbols were provided by Bahl, Cocke, Jelinek, and Raviv in 1974 [16] and termed BCJR decoding algorithm. This algorithm also led to the invention of the turbo code in 1993 [17, 18]. In 1989, J. Hagenauer introduced the sub-optimal algorithm, soft-output Viterbi algorithm (SOVA) [19], that also calculates the *a posteriori* probability for each information symbol.

In 1993, there is a significant advance in channel coding because of the advent of turbo codes, discovered by C. Berrou, A. Glavieux and P. Thitimajshima [18]. The soft iterative decoding of turbo codes based on either the BCJR or the SOVA algorithms results in the performance near the Shannon limit and still has the reasonable complexity for practical implementation. These attractive features make the turbo code popular in the modern digital communication systems. The other capacity approaching code, low-density parity-check (LDPC) code, was first proposed by R. G. Gallager in 1962 [20, 21], but was nearly forgotten until its rediscovery in the late 1990s. The graphical representation for the LDPC code was presented by R. M. Tanner [22] in 1981. MacKay and Neal rediscovered the LDPC code and investigated its graph based iterative decoding algorithm [23,24]. It has been shown in [25] that long LDPC codes based on the belief propagation algorithm [26] can achieve an error performance very closing to the Shannon limit. Furthermore, the turbo decoding can also be interpreted as the the belief propagation algorithm on graphs. [27, 28]. The LDPC code has many benefits, including its good error performance, and more high speed communication systems have considered employing LDPC code to enhance performance.

Based on a specific decoding algorithm, the channel decoder implementation depends on the system requirements or the constraints; it also relays on the target circuit technology. Generally, the cost and the power consumption are always the optimization objectives. In the channel coding design, it should be a trade-off between the error performance and the design cost, or the power consumption. The decoder design starts from the algorithm and the performance analysis in the system level where the design parameters are determined according to the system specifications. The architecture design connects the decoding algorithm, the circuit technology, and the target application. A channel decoder with better quality would require a close link from the algorithm and the system to the circuit implementation. For different code types and system definitions, we will investigate the design methodology from coding basics to circuit implementation.

In the design of algebraic block codes, we consider the Reed-Solomon decoders for general purposes; they have been extensively applied to many applications, but often have different specifications, even in a system. Because of the relatively simple arithmetic units and decoding algorithm, the Reed-Solomon decoders include the high speed design approach [29–31], the the low complexity architecture [32–34], the low power design [35, 36], and the reconfigurable decoders [36–38]. The advance in VLSI technology facilitates the decoding speed and reduces the circuit cost as well as power dissipation. Therefore, the need for time to market motivates the design of universal RS decoders. The throughput and the overhead for universality will be the major consideration in our study.

For the convolutional code, the Viterbi decoding algorithm as well as its implementation will be covered, including the low power design and high speed structures. The Viterbi decoder in wireless communications would have the constraints in power consumption and cost, especially for the mobile devices. The low power techniques can be in the circuit level [39,40] and in the architecture level [41–44], including the algorithm modifications. For the Viterbi decoding algorithm, the adaptive approximations for low power are also reported in [45,46]. In our research, the low power Viterbi decoder is investigated in terms of the adaptive memory access and the low complexity datapath architecture. The present adaptive approach, without any performance loss, can remove most redundant memory access in the decoder [47].

High speed Viterbi decoders are also important because of the demand for high data transmission rate in wireless applications [48–50]. The Viterbi decoders in [51–53] break down the critical path delay by means of bit-level pipeline and accomplish high throughput with very high clock frequencies. Furthermore, the dynamic circuit techniques are also exploited to accelerate the critical path. The architectures using high radix trellis in [49, 50,54,55] achieve parallel processing that enhances the throughput. The four states Viterbi decoder based on sliding block approach that performs decoding concurrently in forward and backward directions is also reported in [56]. However, as the constraint length or the parallelism increases, the complexity of trellis based decoding grows rapidly for the parallel architectures. Hence we consider a more aggressive parallel architecture and explore the possible structure for implementation [57].

Furthermore, the soft iterative decoder for the turbo decoding is studied while considering the low power mobile communication system. The turbo decoder in [58] reports a design for the 3GPP system [59] whose interleaver size is 5114. The high throughput decoders [60–62] mainly focus on the parallelism and speed optimization of computational units. We refer to the 3GPP2 communication system [63] where the turbo code is defined to have a larger interleaver size 20730. The memory required to realize interleaver function will occupy a large circuit area and cause significant power consumption. For the mobile communication system, we concentrate on the minimization of memory area as well as power dissipation [64].

The LDPC decoder based on the message passing decoding is also exploited for multiple Gb/s communications. We will discuss the design issues of parallel architectures and circuit implementation difficulty. The parallel LDPC decoder will cause a large number of signals that convey messages between a large number of processing elements, leading to much complicated interconnections [65]. This will degrade the decoding throughput and enlarge the circuit area. In the dissertation, we conduct the research on the efficient memory management strategy for high speed parallel processing [66]. The low complexity architectures for the processing elements are also presented.

The Reed-Solomon coding is first introduced in Chapter 2, including the universal decoder architecture. Chapter 3 discusses the convolutional code, and the Viterbi decoding algorithm as well as the decoder architectures for the low power and the high speed considerations. The BCJR algorithm is also presented in the final part of Chapter 3. The soft iterative decoding algorithms and decoder designs for the turbo code and the LDPC code are given in Chapter 4. Finally, we conclude the dissertation in Chapter 6.

Chapter 2

Reed-Solomon Codes

Reed-Solomon (RS) codes were first constructed by Reed and Solomon in 1960 [11]. In general, RS codes can be categorized into the non-binary BCH codes [9, 10] whose symbols are taken from the field $GF(2^m)$. The block length of a (n, k) RS code is n symbols consisting of k message symbols in $GF(2^m)$. The minimum distance of $(n-k+1)$ leads to the correcting capability of $t = \lfloor \frac{n-k}{2} \rfloor$ random errors With the erasure information of ρ symbols, the number of correctable errors becomes $\lfloor \frac{n-k-\rho}{2} \rfloor$. Because of the effective error correction, the applications of RS codes include data storage systems and digital communications, either wireless or wireline systems.

Before the presentation of RS codes, the finite field and the BCH code are reviewed in sections 2.1 and 2.2. The following sections 2.3 and 2.4 describe the encoding and decoding procedures, and the decoder designs.

2.1 Finite Field

A field F is a set of at least two elements where two binary operations, addition "+" and multiplication "·", are defined to satisfy the following conditions.

1. F is an Abelian group under "+" with identity element denoted by 0.

2. The set of nonzero elements in F is an Abelian group under "·" with identity element denoted by 1.

3. The multiplication “$\cdot$” is distributive and therefore

$$a \cdot (b + c) = a \cdot b + a \cdot c \ \ \forall a, b, c \in F$$

The number of elements in a field, or the *order* of the field, can be infinite or finite. The well-known examples of infinite fields include the set of all real numbers and the set of all rational numbers. However, the interested fields are those with finite numbers of elements that are termed *finite field* or *Galois field* (GF).

For a prime number p, a finite field $GF(p)$ with p elements can be constructed based on the modulo p arithmetic [67] and has the elements $\{0, 1, \ldots, p-1\}$. Furthermore, for any positive integer m, the field $GF(p^m)$ which is referred to the extension field of $GF(p)$ can also be built with p^m elements , and p is called its *characteristic*. Actually, as proved in [67], the order of any finite field is a power of a prime. The finite field $GF(p^m)$ is regarded as a vector space over $GF(p)$ with the dimension m, and each element in $GF(p^m)$ can be represented by m tuples $\hat{a} = (a_0, a_1, \ldots, a_{m-1})$. In addition, the elements in $GF(p^m)$ can also be denoted by the polynomials over $GF(p)$ with the degrees less than or equal to $m-1$. Therefore, $\hat{a}$ is also expressed by $a_0 + a_1 x + \cdots + a_{m-1} x^{m-1}$. The addition is defined component-wise, and the multiplication should be based on the modulo $f(x)$ operation. Notice that $f(x)$ must be an *irreducible* m-th degree polynomial over $GF(p)$, meaning no polynomials over $GF(p)$ of degrees less than m but larger than 0 divide $f(x)$. Thus, for $\hat{a}, \hat{b}, \hat{c} \in GF(p^m)$, the addition and multiplication can be proceeded as follows.

$$\hat{c} = \hat{a} + \hat{b} = (a_0 + b_0) + (a_1 + b_1)x + \cdots + (a_{m-1} + b_{m-1})x^{m-1} \tag{2.1}$$

$$\hat{c} = \hat{a} \cdot \hat{b} = [(a_0 + a_1 x + \cdots + a_{m-1} x^{m-1})(b_0 + b_1 x + \cdots + b_{m-1} x^{m-1})] \mod f(x) \tag{2.2}$$

Consider an element β in $GF(p^m)$, the monic polynomial $m_\beta(x)$ of the least degree with coefficients in $GF(p)$ such that $m_\beta(\beta) = 0$ is referred to the *minimum polynomial* of β. The polynomial $m_\beta(x)$ is irreducible and has a degree less than or equal to m [68]. Because β is a zero of $m_\beta(x)$, the *conjugates* of β that are distinct elements $\beta, \beta^p, \beta^{p^2}, \ldots,$ are also zeros of $m_\beta(x)$ [67, 69]. The smallest integer k such that $\beta^{q^k} = \beta$ is the *degree* of β, and k is a divisor of m. Although $m_\beta(x)$ is irreducible over $GF(p)$, it can be factored over $GF(p^m)$

$$m_\beta(x) = (x - \beta)(x - \beta^p) \cdots (x - \beta^{p^{k-1}}), \tag{2.3}$$

where the degree is exactly k.

If $f(x)$ is any polynomial over $GF(p)$ satisfying $f(\beta) = 0$, it is easy to verify that $m_\beta(x)|f(x)$. Furthermore, since $GF(p^m)$ has $p^m - 1$ nonzero elements, β^i has at most $p^m - 1$ distinct values for all integers i, and therefore there exists an integer t, with $1 \leq t \leq p^m - 1$, that results in $\beta^{i+t} = \beta^i$ or $\beta^t = 1$. The smallest positive integer t such that $\beta^t = 1$ is termed the *order* of β. The nonzero element α in $GF(p^m)$ is said to be *primitive* if its order is $p^m - 1$. Thus, the powers of the primitive element α generate all the nonzero elements of $GF(p^m)$. That is

$$GF(p^m) = \{0, 1, \alpha, \alpha^2, \ldots, \alpha^{p^m-2}\}.$$

Hence every element in $GF(p^m)$ is a root of $x^{p^m} - x = 0$; alternatively, $(\alpha^i)^{p^m} = (\alpha^{p^m})^i = \alpha^i$ for $i = 0, 1, \ldots, p^m - 2$, and $0^{p^m} = 0$. The root number of $x^{p^m} - x = 0$ is at most p^m which is the order of $GF(p^m)$. It follows that

$$x^{p^m} - x = \prod_{\beta \in GF(p^m)} (x - \beta). \tag{2.4}$$

The definition of minimum polynomial also deduces that $m_\beta(x)|(x^{p^m} - x)$ for all β in $GF(p^m)$. On the other hand, the minimum polynomial of α is called the *primitive* polynomial $p(x)$, and the degree of $p(x)$ is m determined by the degree of α. Furthermore, due to the order of α, we found that $p(x)$ divides $x^{p^m-1} - 1$, but does not divide any $x^n - 1$ with $n < p^m - 1$. Note that not all irreducible polynomials are primitive. Nevertheless, it is convenient to use

Table 2.1: Representation of the elements in $GF(2^4)$

Power	Polynomial							4-tuple
0							0	0 0 0 0
1							1	0 0 0 1
α					α			0 0 1 0
α^2			α^2					0 1 0 0
α^3	α^3							1 0 0 0
α^4					α	$+$	1	0 0 1 1
α^5			α^2	$+$	α			0 1 1 0
α^6	α^3	$+$	α^2					1 1 0 0
α^7	α^3	$+$			α	$+$	1	1 0 1 1
α^8			α^2	$+$			1	0 1 0 1
α^9	α^3	$+$			α			1 0 1 0
α^{10}			α^2	$+$	α	$+$	1	0 1 1 1
α^{11}	α^3	$+$	α^2	$+$	α			1 1 1 0
α^{12}	α^3	$+$	α^2	$+$	α	$+$	1	1 1 1 0
α^{13}	α^3	$+$	α^2			$+$	1	1 1 0 1
α^{14}	α^3	$+$					1	1 0 0 1

primitive polynomials to construct finite fields.

Table 2.1 shows an example of $GF(2^4)$ that is an extension filed of $GF(2)$ and is built with the primitive polynomial $p(x) = x^4 + x + 1$. The primitive element α is a root of $p(x)$ and therefore $p(\alpha) = \alpha^4 + \alpha + 1 = 0$. The nonzero elements of $GF(2^4)$ comprise powers of α, or can be represented as polynomials with degrees ≤ 3 according to the equation $\alpha^4 = \alpha + 1$. Alternatively, the other useful representation for the elements in $GF(2^4)$ is 4-tuple where the four components are the coefficients in the polynomial representation. In Table 2.1, we can find that the power representation is useful for multiplication, while the polynomial or 4-tuple format is more practical for addition.

Additionally, with the primitive polynomial $p(x) = x^2 + x + 2$ over $GF(3)$, the elements of $GF(3^2)$ are also presented in Table 2.2. The primitive element α in $GF(3^2)$ satisfies $p(\alpha) = \alpha^2 + \alpha + 2 = 0$, meaning that $\alpha^2 = 2\alpha + 1$. According to these facts, the power, the polynomial, and the 2-tuple representations can be formed as shown in Table 2.2.

The most widely applied finite fields are the prime field $GF(2)$ and its extension $GF(2^m)$ for the binary arithmetic operations that are well suited for digital circuit design.

Table 2.2: Representation of the elements in $GF(3^2)$

Power	Polynomial	2-tuple
0	0	0 0
1	1	0 1
α	α	1 0
α^2	$2\alpha + 1$	2 1
α^3	$2\alpha + 2$	2 2
α^4	2	0 2
α^5	2α	2 0
α^6	$\alpha + 2$	1 2
α^7	$\alpha + 1$	1 1

2.2 BCH code

BCH code is a large class of multiple-error-correcting codes, a generalization of the Hamming codes; furthermore, and it is best expressed as cyclic code [70]. For any positive m and t, more than a primitive binary BCH code can be constructed to correct t errors while the block length is $2^m - 1$ with no more than mt redundant digits. The generator polynomial $g(x)$ is defined by the least common multiple (LCM) of the minimum polynomials over $GF(2)$ $m_1(x)$, $m_2(x)$, $\ldots$, and $m_{2t}(x)$ corresponding to α, α^2, ..., and α^{2t} where α is primitive in $GF(2^m)$.

$$g(x) = \mathrm{LCM}\{m_1(x), m_2(x), \cdots, m_{2t}(x)\} \tag{2.5}$$

Therefore, α^i for $i = 1, 2, \cdots, 2t$ as well as their conjugates are zeros of $g(x)$. Since $(\alpha^i)^{2^j}$ is a conjugate of α^i, they have the same minimum polynomial $m_i(x)$. As a result, $m_{i2^j}(x) = m_i(x)$ for each positive integer j and $1 \leq i2^j \leq 2t$; hence $g(x)$ will be reduced to

$$g(x) = \mathrm{LCM}\{m_1(x), m_3(x), m_5(x), \cdots, m_{2t-1}(x)\}, \tag{2.6}$$

and each $m_i(x)$ has degree at most m. Consequently, the degree of $g(x)$ is no more than mt and results in at most mt parity check digits for the code. If $c(x)$ is a codeword polynomial, it must be a multiple of $g(x)$, that is $g(x)|C(x)$. From the definition of $g(x)$, we can find that $c(\alpha^i) = 0$ for $1 \leq i \leq 2t$. The encoding of a message polynomial $u(x) = u_0 + u_1 x + u_2 x^2 + \cdots + u_{k-1} x^{k-1}$ is multiplying $u(x)$ by $g(x)$,

$$c(x) = u(x) \cdot g(x) \tag{2.7}$$

Accordingly, if $g(x) = g_0 + g_1 x + \cdots + g_r x^r$, the $k \times n$ generator matrix G is constructed as

$$G = \begin{bmatrix} g_0 & g_1 & g_2 & \cdots & g_r & 0 & 0 & 0 & \cdots & 0 \\ 0 & g_0 & g_1 & g_2 & \cdots & g_r & 0 & 0 & \cdots & 0 \\ 0 & 0 & g_0 & g_1 & \cdots & \cdots & g_r & 0 & \cdots & 0 \\ \vdots & & & & & & & & & \vdots \\ 0 & 0 & 0 & 0 & g_0 & g_1 & g_2 & \cdots & \cdots & g_r \end{bmatrix} \tag{2.8}$$

Obviously, this encoding procedure is not in systematic form. The systematic encoding proceeds as follows: multiplying $u(x)$ by x^{n-k}, dividing $x^{n-k}u(x)$ by $g(x)$ to obtain the remainder polynomial $r(x)$, and adding $r(x)$ to $x^{n-k}u(x)$. The systematic code $c(x)$ becomes

$$c(x) = x^{n-k}u(x) + r(x). \tag{2.9}$$

Since $x^{n-k}u(x) = q(x)g(x) + r(x)$ where $q(x)$ is the quotient polynomial, we find that

$$c(x) = x^{n-k}u(x) + r(x) = q(x)g(x)$$

which is a multiple of $g(x)$. Because α^i for $1 \leq i \leq 2t$ are roots of $c(x)$,

$$c(\alpha^i) = c_0 + c_1\alpha^i + c_2\alpha^{2i} + \cdots + c_{n-1}\alpha^{(n-1)i} = 0. \tag{2.10}$$

Based on (2.10), we can construct the parity check matrix

$$H = \begin{bmatrix} 1 & \alpha & \alpha^2 & \alpha^3 & \cdots & \alpha^{(n-1)} \\ 1 & \alpha^2 & (\alpha^2)^2 & (\alpha^2)^3 & \cdots & (\alpha^2)^{(n-1)} \\ \vdots & & & & & \vdots \\ 1 & \alpha^{2t} & (\alpha^{2t})^2 & (\alpha^{2t})^3 & \cdots & (\alpha^{2t})^{(n-1)} \end{bmatrix}, \tag{2.11}$$

and obtain the following result

$$\hat{c} \cdot H^T = 0,$$

where $\hat{c} = (c_0, c_1, \ldots, c_{n-1})$ is the codeword vector.

The codeword $c(x)$ is transmitted and corrupted by noise. In the recover, the received vector $r(x)$ will be

$$r(x) = c(x) + e(x),$$

and $e(x)$ is the error pattern. The syndrome of $r(x)$ is defined by the $2t$-tuple $\hat{S} = (S_1, S_2, S_3, \ldots, S_{2t})$ in which

$$S_i = r(\alpha^i) = c(\alpha^i) + e(\alpha^i) = e(\alpha^i), \text{ for } 1 \leq i \leq 2t \tag{2.12}$$

The result in (2.12) follows from the fact that α^i is a root of $c(x)$. Assume $e(x)$ has v $(\leq t)$ errors in the positions j_1, j_2, $\cdots$, j_v with $0 \leq j_1 < j_2 < \cdots < j_v \leq n-1$, then $e(x)$ can be expressed as

$$e(x) = x^{j_1} + x^{j_2} + \cdots + x^{j_v}. \tag{2.13}$$

According to (2.12) and (2.13), the syndrome equations with $i = 1 \sim 2t$ will be

$$\begin{aligned} S_i &= (\alpha^{j_1})^i + (\alpha^{j_2})^i + \cdots + (\alpha^{j_v})^i \qquad (2.14) \\ &= \beta_1^i + \beta_2^i + \cdots + \beta_v^i, \end{aligned}$$

if we define $\beta_x = \alpha^{j_x}$ for $x = 1, 2, \ldots, v$. Additionally, the *error location* polynomial is defined as

$$\begin{aligned} \sigma(x) &= (1-\beta_1 x)(1-\beta_2 x)\cdots(1-\beta_v x) \\ &= \sigma_0 + \sigma_1 x + \cdots + \sigma_v x^v \end{aligned} \tag{2.15}$$

The roots of $\sigma(x) = 0$ are β_1^{-1}, β_2^{-1}, $\cdots$, and β_v^{-1}; the coefficients of $\sigma(x)$ satisfy the following equations:

$$\begin{aligned} \sigma_0 &= 1 \\ \sigma_1 &= \beta_1 + \beta_2 + \cdots + \beta_v \\ \sigma_2 &= \beta_1\beta_2 + \beta_2\beta_3 + \cdots + \beta_{v-1}\beta_v \\ &\vdots \\ \sigma_v &= \beta_1\beta_2\cdots\beta_v \end{aligned} \tag{2.16}$$

Furthermore, the expression in (2.16) is related to the syndromes $S_1 \sim S_v$ be the following *Newton's identities* [70]:

$$\begin{aligned} S_1 + \sigma_1 &= 0 \\ S_2 + \sigma_1 S_1 + 2\sigma_2 &= 0 \\ S_3 + \sigma_1 S_2 + \sigma_2 S_1 + 3\sigma_2 &= 0 \\ &\vdots \\ S_v + \sigma_1 S_{v-1} + \cdots + \sigma_{v-1} S_1 + v\sigma_v &= 0 \\ &\vdots \\ S_{2t} + \sigma_1 S_{2t-1} + \sigma_2 S_{2t-2} + \cdots + \sigma_v S_{2t-v} &= 0 \end{aligned} \tag{2.17}$$

If the syndromes $S_1, S_2, \cdots, S_v$ are calculated according to (2.12), the coefficients of $\sigma(x)$ can be obtained from (2.17). The error location information $\beta_1, \beta_2, \cdots, \beta_v$ can then be found through solving the roots of $\sigma(x) = 0$. Since the number of elements in $GF(2^m)$ is finite, $\sigma(x) = 0$ is able to be efficiently solved by substituting each nonzero element of $GF(2^m)$ into the equation, a process known as *Chien search* [71].

2.3 Reed-Solomon code

A Reed-Solomon code [11] over $GF(q)$ is an important subclass of non-binary BCH codes and has a length of $n = q - 1$. From the definition of generator polynomial in (2.5), the $g(x)$ of a RS code that corrects t or fewer errors is described by the minimum polynomials of α, α^2, α^3, $\cdots$, and α^{2t}, and α is a primitive element in $GF(q)$. Besides, the minimum polynomial $m_i(x)$ over $GF(q)$ of each element α^i is $m_i(x) = x - \alpha^i$. Therefore, the generator polynomial of this code should be

$$g(x) = (x - \alpha)(x - \alpha^2) \cdots (x - \alpha^{2t}). \tag{2.18}$$

This polynomial always has the degree $2t$,and thus an (n, k) RS code satisfies $n - k = 2t$, the number of parity check digits. The minimum distance is shown to be $n - k + 1 = 2t + 1$ [5] that can correct at most t errors. Notice that $g(x)$ can also be characterized by the minimum polynomials of α^h, α^{h+1},$\cdots$, and α^{h+2t-1}. The express in (2.5) is generalized to

$$g(x) = (x - \alpha^h)(x - \alpha^{h+1}) \cdots (x - \alpha^{h+2t-1}). \tag{2.19}$$

The encoding of RS codes is similar to the binary BCH codes and can be either in the multiplication operation of (2.7) or in the systematic form of (2.9). Both encoding schemes make the codeword $c(x)$ be a multiple of $g(x)$; as a result, all roots of $g(x)$ are also zeros of $c(x)$, or $c(\alpha^{h+i}) = 0$ for $0 \leq i \leq 2t - 1$. The number of parity check symbols are $2t$, and the resulting codewords has the minimum distance of $2t+1$ that corresponds to correct at most t errors. Moreover, if there are ν erasures, referred to errors with , the RS code can correct at moat $\lfloor \frac{2t-\nu}{2} \rfloor$ errors.

The decoding of RS code is quite similar to binary BCH codes except that error values should be calculated as well. We also define the received data $r(x)$ as the codeword polynomial $c(x)$ corrupted by the error polynomial $e(x)$; that is $r(x) = c(x) + e(x)$. The syndromes will be

$$S_i = r(\alpha^i) = e(\alpha^i), \quad \text{for } 1 \leq i \leq 2t, \tag{2.20}$$

where α^{h+i-1} are all roots of the generator polynomial in (2.19). Since RS code is nonbinary, the error pattern $e(x)$ should be

$$e(x) = e_1 x^{j_1} + e_2 x^{j_2} + \cdots + e_v x^{j_v}, \tag{2.21}$$

indicating that there are v error values, e_1, e_2, ..., and e_v, in the locations j_1, j_2, ..., and j_v. Therefore, S_i in (2.20) can be expressed by

$$S_i = e_1\beta_1^i + e_2\beta_2^i + \cdots + e_v\beta_v^i, \tag{2.22}$$

and $\beta_x = \alpha^{j_x}$ for $x = 1, 2, \cdots, v$. Furthermore, we also define the syndrome polynomial $S(x)$ as [72]

$$\begin{aligned} S(x) &\triangleq \sum_{i=1}^{\infty} S_i x^{i-1} && (2.23) \\ &= \sum_{i=1}^{\infty} (\sum_{\kappa=1}^{v} e_\kappa \beta_\kappa^i) x^{i-1} \\ &= \sum_{\kappa=1}^{v} e_\kappa \beta_\kappa (\sum_{i=1}^{\infty} (\beta_\kappa x)^{i-1}) \\ &= \sum_{\kappa=1}^{v} \frac{e_\kappa \beta_\kappa}{1 - \beta_\kappa x} && (2.24) \end{aligned}$$

In (2.24), we apply the following equation

$$\frac{1}{1-\beta_\kappa x} = \sum_{i=1}^{\infty} (\beta_\kappa x)^{i-1}$$

The error location polynomial $\sigma(x)$ is also defined as shown in (2.15). The relation of $S(x)$ and $\sigma(x)$ can be constructed from

$$S(x)\sigma(x) = (\sum_{\kappa=1}^{v} \frac{e_\kappa \beta_\kappa}{1-\beta_\kappa x}) \times \prod_{i=1}^{v}(1-\beta_i x) = \sum_{\kappa=1}^{v} e_\kappa \beta_\kappa \prod_{i=1, i\neq\kappa}^{v} (1-\beta_i x) \triangleq \omega(x) \tag{2.25}$$

The result of $S(x)\sigma(x)$ is defined by the polynomial $\omega(x)$ with degree $v-1$. During decoding, only the coefficients $S_1 \sim S_{2t}$ in $S(x)$ are known, or we can obtain only $S(x) \mod x^{2t}$. Hence the *key equation* is defined as follows [72]:

$$S(x)\sigma(x) = \omega(x) \mod x^{2t} \tag{2.26}$$

The decoding problem is to solve $\sigma(x)$ and $\omega(x)$ from the syndrome information $S(x)$ according to the key equation (2.26). The error locations will be identified by applying Chien search [71] on $\sigma(x)$, and the error values can be calculated from $\omega(x)$ according to Forney's algorithm [73]:

$$e_\kappa = \frac{-\omega(\beta_\kappa^{-1})}{\sigma'(\beta_\kappa^{-1})}, \quad \text{for } 1 \leq \kappa \leq v. \tag{2.27}$$

Notice that $\sigma'(x)$ indicates the derivative of $\sigma(x)$ and therefore

$$\sigma'(x) = \frac{d\sigma(x)}{dx} = \sum_{\kappa=1}^{v} -\beta_\kappa \prod_{i=1, i\neq\kappa}^{v} (1-\beta_i x). \tag{2.28}$$

We note from (2.25) that the degree of $\omega(x)$ is $v-1$; as a result, in $S(x)\sigma(x)$, the coefficients of the terms with powers large than $v-1$ must be zero. The following Newton's identity can be derived [72]

$$\begin{bmatrix} S_1 & S_2 & \cdots & S_{v+1} \\ S_2 & S_3 & \cdots & S_{v+2} \\ S_3 & S_4 & \cdots & S_{v+3} \\ \vdots & \vdots & & \vdots \\ S_{2t-v} & S_{2t-v+1} & \cdots & S_{2t} \end{bmatrix} \begin{bmatrix} \sigma_v \\ \sigma_{v-1} \\ \vdots \\ \sigma_0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \tag{2.29}$$

It is inefficient to directly solve (2.29) for moderate or large v. The efficient iterative algorithm to solve key equation (2.26) was developed by Berlekamp [72, 74] and independently by Massey [75, 76] and is referred to Berlekamp-Massey algorithm. This algorithm is an approximation procedure for finding $\sigma(x)$ that satisfies (2.29) and has the smallest degree. The detailed procedure is given as follows.

- Initial conditions:

$$\begin{aligned} &\sigma^{(0)}(x) = 1, \quad \tau^{(0)}(x) = 1, \quad D_0 = 0 \\ &\Delta_0 = S_1, \qquad \delta = 1, \end{aligned}$$

- Iteration from $i = 1$ to $2t$:

$$\sigma^{(i)}(x) = \sigma^{(i-1)}(x) + \frac{\Delta_{i-1}}{\delta}\tau^{(i-1)}(x) \cdot x \tag{2.30}$$

$$\Delta_i = \sum_{j=0}^{t-1} \sigma_j^{(i)} S_{i+1-j} \tag{2.31}$$

If $\Delta_{i-1} = 0$ or $D_{i-1} \geq i - D_{i-1}$,

$$\begin{aligned} &\tau^{(i)}(x) = \tau^{(i-1)}(x) \cdot x, \\ &D_i = D_{i-1}. \end{aligned}$$

Otherwise,

$$\tau^{(i)}(x) = \sigma^{(i-1)}(x),$$
$$D_i = i - D_{i-1}, \qquad \delta = \Delta_{i-1}$$

After $2t$ iterations, we obtain the error location polynomial $\sigma(x) = \sigma^{(2t)}(x)$. In (2.30), the operation tends to make $\sigma^{(i)}$ satisfy the discrepancy $\Delta_{i-1} = 0$. The new discrepancy Δ_i in (2.31) continues to measure the correctness of dummy error location polynomial $\sigma^{(i)}(x)$ whose coefficients are $\sigma_j^{(i)}$. On the other hand, D_i is a control variable that indicates the degree of $\sigma^{(i)}(x)$ [5].

The evaluation of error value polynomial $\omega^{(i)}(x)$ can use the key equation [77]. We determine

$$\omega(x) = \omega_0 + \omega_1 x + \cdots + \omega_{v-1}x^{v-1}$$

after Berlekamp-Massey algorithm when $\sigma(x)$ is well defined. From the key equation (2.26), the coefficients of $\omega(x)$ will be calculated by

$$\omega_i = \sum_{j=0}^{i} \sigma_j S_{i+1-j}, \quad \text{for } 0 \leq i \leq v-1. \tag{2.32}$$

This equation is identical to the discrepancy computation in (2.31).

In addition to the Berlekamp-Massey algorithm, the Euclidean algorithm [78, 79] is the other efficient solution for solving key equation. The key equation (2.26) is first rewritten as

$$S(x)\sigma(x) + Q(x)x^{2t} = \omega(x). \tag{2.33}$$

Notice that $Q(x)$ is the quotient polynomial and $\omega(x)$ is the remainder polynomial while $S(x)\sigma(x)$ is divided by x^{2t}. Hence according to Euclidean algorithm, $\omega(x)$ can be acquired through the procedure that derives the greatest common divisor (GCD) polynomial of $S(x)$ and x^{2t}. The algorithm proceeds as follows.

- Initial conditions

$$\omega^{(-1)} = x^{2t}, \quad \omega^{(0)} = S(x)$$
$$\sigma^{(-1)}(x) = 0, \quad \sigma^{(0)}(x) = 1$$

- Iterations from $i = 1$:

$$\omega^{(i)}(x) \;=\; q^{(i)}(x)\omega^{(i-1)}(x) + \omega^{(i-2)}(x) \qquad (2.34)$$
$$\sigma^{(i)}(x) \;=\; q^{(i)}(x)\sigma^{(i-1)}(x) + \sigma^{(i-2)}(x) \qquad (2.35)$$

  The iteration is terminated when degree of $\sigma^{(i)}(x)$ is larger than that of $\omega^{(i)}(x)$. Otherwise, the iteration continues, and i is increased by one, or $i = i + 1$.

If the iteration is terminated, we set error location polynomial $\sigma(x) = \sigma^{(i)}(x)$ and error value polynomial $\omega(x) = \omega^{(i)}(x)$. The operation in (2.34) is polynomial division that divides $\omega^{(i-2)}(x)$ by $\omega^{(i-1)}(x)$; $q_i(x)$ and $\omega^{(i)}(x)$ correspond to quotient and remainder polynomials. The quotient $q_i(x)$ is then used in (2.35) for calculating new $\sigma^{(i)}(x)$. What should be noted is that the degree of $\omega^{(i)}(x)$ decreases with i whereas $\sigma^{(i)}(x)$ has an increasing degree. The stopping criterion comes from (2.25) where $\omega(x)$ has the degree at most $v - 1$ with the $\sigma(x)$ of degree v.

In summary, RS decoding starts from syndrome calculation (2.20) and then solves key equation (2.26) with either Berlekamp-Massey algorithm or Euclidean algorithm. Finally, we apply Chien search to error location polynomial $\sigma(x)$ for finding error locations; moreover, error values can be evaluated with Forney's algorithm (2.27).

The erasure and error decoding is quite similar to the above mentioned error only decoding procedure. If there are v errors and u erasures in the received vector $r(x)$, $2v + u \leq 2t$, the syndrome calculation is as (2.20), but the erasure symbols are replaced with arbitrary values, for example, zeros. The erasure location polynomial is also defined as

$$\lambda(x) = (1 - \alpha^{l_1}x)(1 - \alpha^{l_2}x)\cdots(1 - \alpha^{l_u}x) \qquad (2.36)$$

where $l_1, l_2, \ldots, l_u$ are erasure positions. The key equation should be modified by

$$S(x)\lambda(x)\sigma(x) = \omega(x) \mod x^{2t}. \tag{2.37}$$

The errata location polynomial $\Lambda(x) = \lambda(x)\sigma(x)$ identifies both error and erasure locations. We can denote the known polynomials $S(x)$ and $\lambda(x)$ with Forney syndrome polynomial [73]

$$\begin{aligned} T(x) &\triangleq S(x)\lambda(x) \mod x^{2t} \qquad (2.38) \\ &= T_1 + T_2 x + T_3 x^2 + \cdots + T_{2t} x^{2t-1}, \end{aligned}$$

and the key equation (2.37) will become

$$T(x)\sigma(x) = \omega(x) \mod x^{2t}. \tag{2.39}$$

Similar to (2.25), it can be deduced that $\omega(x)$ will have the maximum degree of $(v + u - 1)$ for the degree of $\lambda(x)\sigma(x)$ is $(v + u)$; as a result, the following equation is derived:

$$\begin{bmatrix} T_{u+1} & T_{u+2} & \cdots & T_{u+v+1} \\ T_{u+2} & T_{u+3} & \cdots & T_{u+v+2} \\ \vdots & \vdots & & \vdots \\ T_{2t-v} & T_{2t-v+1} & \cdots & T_{2t} \end{bmatrix} \begin{bmatrix} \sigma_v \\ \sigma_{v-1} \\ \vdots \\ \sigma_0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \tag{2.40}$$

Applying Berlekamp-Massey algorithm, we can compute $\sigma(x)$ iteratively [77] as follows.

- Initial conditions:

$$\sigma^{(u)}(x) = 1, \quad \tau^{(u)}(x) = 1, \quad D_u = 0$$

$$\Delta_u = T_{u+1}, \quad \delta = 1,$$

- Iteration from $i = (u+1)$ to $2t$:

$$\sigma^{(i)}(x) = \sigma^{(i-1)}(x) + \frac{\Delta_{i-1}}{\delta}\tau^{(i-1)}(x) \cdot x \tag{2.41}$$

$$\Delta_i = \sum_{j=0}^{t-1} \sigma_j^{(i)} T_{i+1-j} \tag{2.42}$$

If $\Delta_{i-1} = 0$ or $D_{i-1} \geq i - u - D_{i-1}$,

$$\tau^{(i)}(x) = \tau^{(i-1)}(x) \cdot x,$$
$$D_i = D_{i-1}.$$

Otherwise,

$$\tau^{(i)}(x) = \sigma^{(i-1)}(x),$$
$$D_i = i - u - D_{i-1}, \quad \delta = \Delta_{i-1}$$

When $i = 2t$, we obtain the error location polynomial $\sigma(x) = \sigma^{2t}(x)$ and error value polynomial $\omega(x)$ from (2.39).

On the other hand, Euclidean algorithm is also applicable for solving (2.39).

- Initial conditions

$$\omega^{(-1)} = x^{2t}, \quad \omega^{(0)} = T(x)$$
$$\sigma^{(-1)}(x) = 0, \quad \sigma^{(0)}(x) = 1$$

- Iterations from $i = 1$:

$$\omega^{(i)}(x) = q^{(i)}(x)\omega^{(i-1)}(x) + \omega^{(i-2)}(x) \tag{2.43}$$

$$\sigma^{(i)}(x) = q^{(i)}(x)\sigma^{(i-1)}(x) + \sigma^{(i-2)}(x) \tag{2.44}$$

The iteration is terminated when degree of $\sigma^{(i)}(x)$ plus u is larger than that of $\omega^{(i)}(x)$. Otherwise, the iteration continues, and i is increased by one, or $i = i + 1$.

The error location and error value polynomials are accomplished with $\sigma(x) = \sigma^{(i)}(x)$ and

$\omega(x) = \omega^{(i)}(x)$ after the termination of Euclidean algorithm.

The roots of $\sigma(x) = 0$ determining error locations are found by Chien search. Furthermore, according to Forney's algorithm, the error values should be

$$e_\kappa = \frac{-\omega(\beta_\kappa^{-1})}{\Lambda'(\beta_\kappa^{-1})} = \frac{-\omega(\alpha^{-j_\kappa})}{\Lambda'(\alpha^{-j_\kappa})}, \quad \text{for } 1 \leq \kappa \leq \upsilon, \tag{2.45}$$

and the erasure values are

$$\hat{e}_\rho = \frac{-\omega(\alpha^{-l_\rho})}{\Lambda'(\alpha^{-l_\rho})}, \quad \text{for } 1 \leq \rho \leq u. \tag{2.46}$$

In error and erasure decoding algorithm, the major difference is the Forney syndrome polynomial $T(x)$ in (2.38) and key equation (2.39). Additionally, there are corresponding modifications in both Berlekamp-Massey and Euclidean algorithms for the known erasure location polynomial $\lambda(x)$ and $T(x)$.

In addition to the presented decoding approach, RS codes can also be decoded in the frequency domain [80]. The time domain algorithm that eliminate syndrome calculations are also studied in [5], [81], and [82]. Although time domain approaches dispense with syndromes, the computation in solving key equation may require more computing complexity.

2.4 Design of Reed-Solomon decoder

RS decoders usually consist of a syndrome calculator, a key equation solver, a Chein search module, and a error value evaluator. Moreover, a Forney syndrome calculator and a polynomial multiplier are also included for erasure decoding. Fig. 2.1 illustrates the RS decoding flow with error and erasure. The syndrome calculator generates $2t$ syndromes from the received vector $r(x)$. If there are erasure symbols, the Forney syndrome calculator evaluates $T(x)$ based on (2.38) and $\lambda(x)$ based on (2.36). According to $S(x)$ or $T(x)$, the key equation solver delivers $\sigma(x)$ and $\omega(x)$ with either Berlekamp-Massey or Euclidean algorithm.

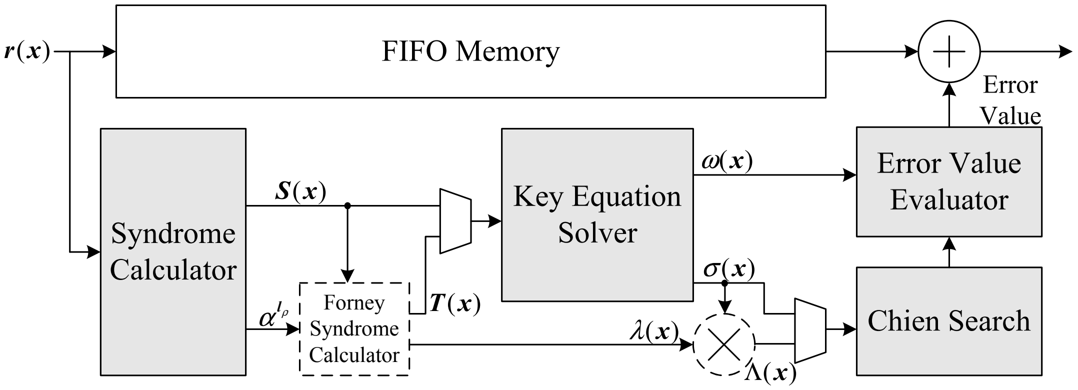


Figure 2.1: RS error and erasure decoding flow chart

Additionally, the errata location polynomial $\Lambda(x)$ needs to be calculated when $\lambda(x)$ exists. The error locations are found by Chien search block; besides, error value evaluator computes error values using (2.27) or both error and erasure values using (2.45) and (2.46). The first-in and first-out (FIFO) memory stores the received vector $r(x)$ that has to be corrected after decoding procedure.

The high-speed RS decoders are investigated in [29, 31, 83]. The low complexity approaches using inversionless Berlekamp-Massey algorithm are also presented in [32,33,84,85]. Furthermore, the efficient decomposed architecture for key equation solver is also reported in [31, 33, 77]. However, most designs are application specific solutions without configurability or programmability. A universal RS codec that can manipulate different code rates and block lengths has to support various finite fields. The difficulty for the universal RS codec is the dedicated hardware providing finite field operations for variable field degree m and irreducible polynomials, or primitive polynomials. Hence the software approach is proposed in [37] by using programmable digital signal processor (DSP) with optimized datapaths.

In the follow-up sections, a universal finite field datapath for $GF(2^m)$ is presented for both dedicated hardware and processor arithmetic unit [86]. As shown in (2.1), finite field addition (FFA) over $GF(2^m)$ is a component-wise operation over $GF(2)$ ans is simple to implement with different degree m. Nevertheless, in (2.2), finite field multiplication (FFM) is much complex due to the polynomial modulo operation. The present universal FFM will efficiently accommodate different irreducible polynomials and eliminate the effect of the field degree m.

2.4.1 Universal finite field multiplier

The universal FFM is based on the Montgomery multiplication algorithm [87] that improves the modular multiplications. With polynomial representation, multiplication of $\hat{a}$ and $\hat{b}$ in $GF(p^m)$ can be expressed as (2.2), which is a modular multiplication. Note that $\hat{c}$ is also an element of $GF(p^m)$ and $f(x)$ is an irreducible polynomial over $GF(p)$ with degree m. The Montgomery product is defined as

$$\hat{c}_M = \hat{a} \cdot \hat{b} \cdot \mu^* \mod f(x), \tag{2.47}$$

where $\mu^* = x^{-m} \mod f(x)$ is a constant element in $GF(p^m)$, and $\mu^* \cdot \mu = 1 \mod f(x)$ while $\mu = x^m$. Furthermore, we can find that $f(x)$ and μ are relatively prime because $f(x)$ is irreducible; as a result, there exists a polynomial $f^*(x)$ that satisfies the following property

$$\mu \cdot \mu^* + f(x) \cdot f^*(x) = 1. \tag{2.48}$$

With (2.48), the Montgomery product in (2.47) can be determined by

$$\hat{q} = \hat{a} \cdot \hat{b} \cdot f^*(x) \mod \mu \tag{2.49}$$

$$\hat{c}_M = (\hat{a} \cdot \hat{b} + \hat{q} \cdot f(x))/u. \tag{2.50}$$

The polynomial $f^*(x)$ can be obtained from (2.48) by using Euclidean algorithm [67]. The modular and division operations in (2.49) and (2.50) becomes much simple as compared with the modulo $f(x)$ operation in (2.48) because of $\mu = x^m$. The computation can be further partitioned into a series of simpler operations for less complexity. We use $\mu^* = x^{-m}$

mod $f(x)$ and polynomial representation of $\hat{a} = a_0 + a_1 x + \cdots + a^{m-1}x^{m-1}$ to decompose (2.47):

$$\hat{c}_M = [a_{m-1}\hat{b}x^{-1} \mod f(x)] + [a_{m-2}\hat{b}x^{-2} \mod f(x)] + \cdots [a_0\hat{b}x^{-m} \mod f(x)]. \quad (2.51)$$

This equation can be rearranged to the following iterative form:

$$\hat{c}_M = [a_{m-1}\hat{b} + [a_{m-2}\hat{b} + \cdots [a_0\hat{b}x^{-1} \mod f(x)] \cdots]x^{-1} \mod f(x)]x^{-1} \mod f(x). \quad (2.52)$$

Similar to (2.49) and (2.50), we can use the Montgomery product $(a_i\hat{b}x^{-1} \mod f(x))$ and rewrite (2.52) as

- Initial conditions

$$\hat{A}^{(0)} = 0$$

- Iterations from $i = 0$ to $m - 1$

$$\begin{aligned} \hat{q} &= [(\hat{A}^{(i)} + a_i\hat{b})f^*(x)] \mod x & (2.53) \\ \hat{A}^{(i+1)} &= (\hat{A}^{(i)} + a_i\hat{b} + \hat{q}f(x))/x & (2.54) \end{aligned}$$

After m iterations, we will obtain $\hat{c}_M = A_m$. Notice that $f^*(x)$ in (2.53) is the multiplicative inverse of $f(x)$ under modulo x operation, which is $f(x)f^*(x) = 1 \mod x$, or $f^*(x) = f^{-1}(x)$ mod x.

In $GF(2^m)$, the elements are often represented in binary digits, and the coefficients in the polynomial representation are either zero or one. Hence $f^*(x) = f^{-1}(x) \mod x$ is always one for $f(x)$ is irreducible, and the term $f^*(x)$ in (2.53) can be eliminated. The result $\hat{q}$ of (2.53) is the constant term of $(A_i + a_i\hat{b})$. The iteration number varies with the field degree m. Therefore, we define a constant integer d such that $m \leq d$ and let $u^* = x^{-d} \mod f(x)$ in (2.47). The corresponding iterative computation will become as follows:

- Initial conditions

$$\hat{A}^{(0)} = 0$$

- Iterations from $i = 0$ to $d - 1$

$$\begin{aligned} a_i &= 0, \quad \text{for} \quad i \geq m & (2.55) \\ \hat{T} &= \hat{A}^{(i)} + a_i \hat{b} & (2.56) \\ \hat{A}^{(i+1)} &= (\hat{T} + t_0 f(x))/x & (2.57) \end{aligned}$$

The final result is

$$\hat{c}_M = \hat{A}^{(d)} = \hat{a} \cdot \hat{b} \cdot x^{-d} \mod f(x) \tag{2.58}$$

Note that we force $a_i = 0$ when $i \geq m$ to ensure correct operation, and t_0 in (2.54) is the constant term of $\hat{T}$. For any $m \leq d$ and irreducible polynomial $f(x)$, we can compute the Montgomery product (2.58) with d iterations of (2.55)~(2.54) without modular operations. We also find that $\mu^* = x^{-d} \mod f(x)$ is a constant element in a given $GF(2^m)$. However, for the normal FFM, there is still a factor μ^* involved in the product $\hat{c}_M$. In order to remove this factor, we apply another Montgomery multiplication to correct the product $\hat{c}_M$

$$\hat{c} = \hat{c}_M \cdot \hat{\delta} \cdot x^{-d} \mod f(x) \tag{2.59}$$

and let $\hat{\delta} = x^{2d} \mod f(x)$; as a result, the normal FFM product $\hat{c}$ is obtained. In many applications, it is unnecessary to perform the product correction each time we apply Montgomery multiplication because d is a constant, and the correction is needed only after a series of multiplications. We will show this approach in the decoder design.

Fig. 2.2 illustrates an example of Montgomery multiplier structure for $d = 4$. This multiplier has three inputs $\hat{a}$, $\hat{b}$, and $f(x)$, represented by

$$\begin{aligned}\hat{a} &= a_3x^3 + a_2x^2 + a_1x + a_0\\ \hat{b} &= b_3x^3 + b_2x^2 + b_1x + b_0\\ f(x) &= f_4x^4 + f_3x^3 + f_2x^2 + f_1x + f_0.\end{aligned}$$

As derived in (2.58), the result $\hat{c}_M$ will be

$$\hat{A}^{(4)} = A_3^{(4)}x^3 + A_2^{(4)}x^2 + A_1^{(4)}x + A_0^{(4)}.$$

Note that f_0 always equals to one that can be eliminated for simplicity. The multiplier in Fig. 2.2 can operate over any $GF(2^m)$ with $m \leq 4$.

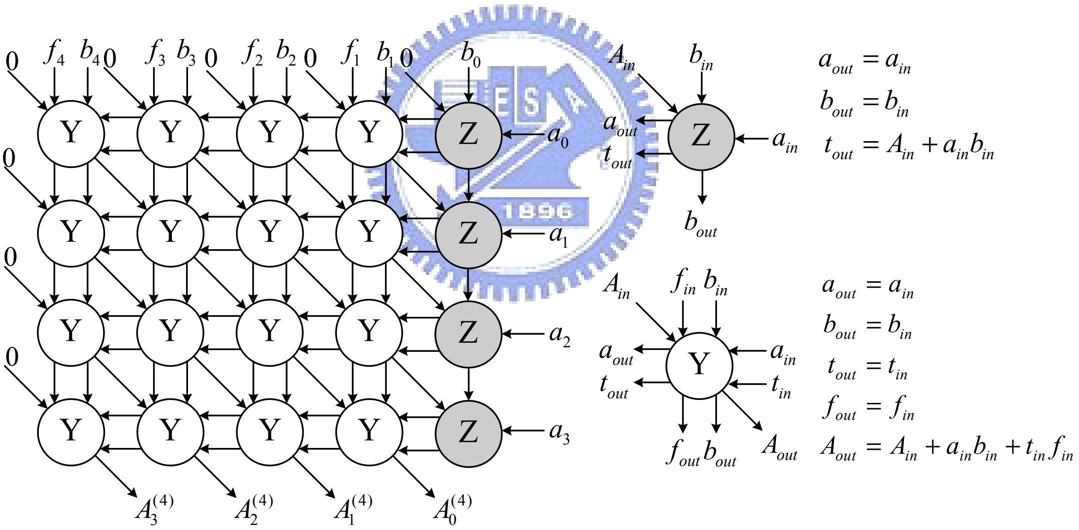


Figure 2.2: Montgomery multiplier structure for $GF(2^m)$ and $m \leq 4$

2.4.2 Universal finite field divider

The finite field division

$$\hat{c} = \hat{b}/\hat{a} = \hat{b} \cdot \hat{a}^{-1} \tag{2.60}$$

can be proceeded with two steps where the first one is finding the multiplicative inverse $\hat{a}^{-1}$ for $\hat{a}$, and the second one is multiplying $\hat{b}$ by $\hat{a}^{-1}$. The process in finding $\hat{a}^{-1}$ is referred to finite field inversion (FFI). There are possible solutions for FFI: table look-up approach and Fermat's algorithm [88]. The former constructs a $\hat{a}^{-1}$ table indexed by $\hat{a}$; additional memory is required for storing the table, but the operation is as simple as memory access. The latter uses the Fermat's identity

$$\hat{a}^{-1} = \hat{a}^{2^m-2} = \prod_{i=1}^{m-1} \hat{a}^{2^i} \tag{2.61}$$

which can be completely implemented with multipliers. However, with Montgomery multiplication, we will achieve the following result

$$\begin{aligned} \hat{a}_M^{-1} &= \hat{a}^{-1} \cdot \hat{\delta}^* & (2.62) \\ \hat{\delta}^* &= x^{-(2^m-2)d} \mod f(x) = x^d \mod f(x) & (2.63) \end{aligned}$$

In (2.63), we also use the Fermat's identity $x^{-1} = x^{2^m-2} \mod f(x)$. Hence the finite field division in (2.60) can be rewritten as

$$\hat{c} = \hat{b} \cdot (\hat{a}^{-1} \cdot x^d) \cdot x^{-d} \mod f(x) = \hat{b} \cdot \hat{a}_M^{-1} \cdot x^{-d} \mod f(x) \tag{2.64}$$

which is also a Montgomery multiplication. It is unnecessary to adjust the quotient $\hat{c}$ with any factor.

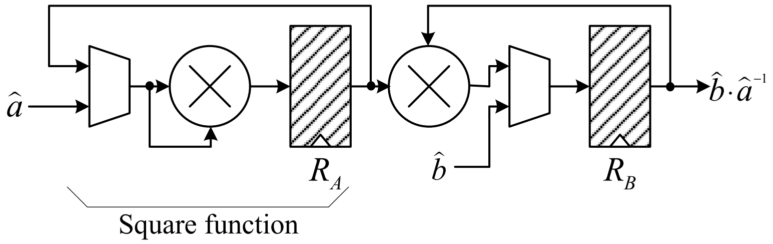


Figure 2.3: Montgomery division structure for $GF(2^m)$

The finite field divider using Fermat's algorithm is shown in Fig. 2.3. Initially, register R_B is loaded with $\hat{b}$. The multiplier on the left side performs the finite field square function generating a sequence of $\hat{a}^2$, $\hat{a}^4$,…, and $\hat{a}^{2^{m-1}}$; the other multiplier serially multiplies $\hat{b}$ by this sequence. After $m-1$ cycles, the result $\hat{b}\cdot\hat{a}^{-1}$ is obtained without correction factor.

2.4.3 Syndrome Calculator

The syndrome calculator computes $2t$ syndromes according to (2.20) which can be expressed as

$$\begin{aligned} S_i &= \sum_{j=0}^{n-1} r_j \alpha^{i\cdot j} && (2.65)\\ &= (\cdots((r_{n-1}\alpha^i + r_{n-2})\alpha^i + r_{n-3})\alpha^i + \cdots)\alpha^i + r_0 && (2.66)\end{aligned}$$

According to (2.66), we can construct a syndrome calculator for S_i as shown in Fig. 2.4, consisting of a register, a finite field adder, and a constant finite field multiplier which multiplies by α^i. The received data are serially inputted from r_{n-1} to r_0. After r_0 is calculated, the final result S_i will be in the register. The constant FFM is simpler than the standard FFM in terms of complexity and critical path timing. Nevertheless, if we want to design a universal syndrome calculator, the constant FFM should be replaced with Montgomery multiplication where the multiplier is α^{i+d}. Notice that α^{i+d} varies with the irreducible polynomial $f(x)$, limiting the simplication of Montgomery multiplication. Hence the modified syndrome computation is proposed [38] to reduce the Montgomery multiplication. We first

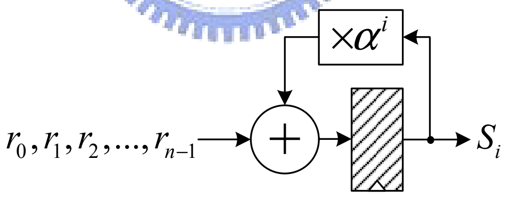


Figure 2.4: Syndrome calculator for S_i

rewrite (2.65) as follows:

$$S_i = \sum_{j=0}^{n-1} r_j \alpha^{i\cdot j} = \sum_{j=0}^{n-1} r_j \alpha^{d\cdot j + (i-d)\cdot j} = \sum_{j=0}^{n-1} (r_j \alpha^{d\cdot j}) \alpha^{(i-d)\cdot j} \tag{2.67}$$

The new received symbol r'_j is defined to be $r_j \alpha^{d\cdot j}$, and (2.67) can also be expand into iterative form:

$$S_i = (\cdots (r'_{n-1}\alpha^{i-d} + r'_{n-2})\alpha^{i-d} + \cdots r'_1)\alpha^{i-d} + r'_0. \tag{2.68}$$

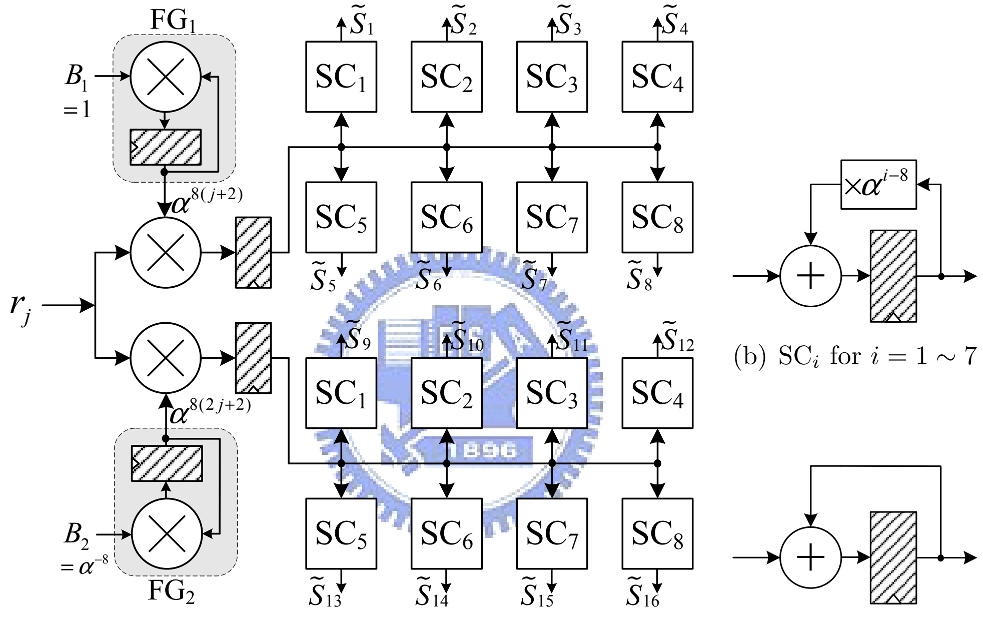
Therefore, because of the term α^{i-d}, we can use Montgomery multiplication with constant multiplier α^i if $i < d$. Since d is the maximum value of m, $\hat{b} = \alpha^i$ with $i < d$ in (2.58) can be represented as a constant, regardless of different $f(x)$. Furthermore, when $i = d$, $\alpha^{i-d} = \alpha^0$ becomes one and the constant multiplier can also be eliminated. The corresponding Montgomery multiplier will be simplified due to the constant $\hat{b} = \alpha^i$. On the other had, once i is larger or equal than d, we can modified (2.67) to achieve the similar results; that is

$$S_i = \begin{cases} \sum_{j=0}^{n-1} (r_j \alpha^{d\cdot j}) \alpha^{(i-d)\cdot j}, & \text{for} \quad 0 < i \le d \\ \sum_{j=0}^{n-1} (r_j \alpha^{2d\cdot j}) \alpha^{((i-d)-d)\cdot j}, & \text{for} \quad d < i \le 2d \\ \sum_{j=0}^{n-1} (r_j \alpha^{3d\cdot j}) \alpha^{((i-2d)-d)\cdot j}, & \text{for} \quad 2d < i \le 3d \\ \vdots & \vdots \\ \sum_{j=0}^{n-1} (r_j \alpha^{\pi d\cdot j}) \alpha^{((i-(\pi-1)d)-d)\cdot j}, & \text{for} \quad (\pi-1)d < i \le \pi d \end{cases} \tag{2.69}$$

We can find that $(i - (\pi - 1)d)$ is still between 0 and d as $(\pi - 1)d < i \le \pi d$, and therefore a group of constant multipliers can be defined for α^i with $0 < i < d$. It is also convenient to deliver $\tilde{S}_i = \alpha^d S_i$ which will facilitate the key equation solver implementation. A syndrome calculator with $d = 8$ and $t \le 8$ is presented in Fig. 2.5. There are at most 16 syndromes that should be computed from $\tilde{S}_i = \alpha^8 r(\alpha^i)$ for $i = 1 \sim 16$. For $d = 8$, we construct eight syndrome cells, $\mathrm{SC}_1 \sim \mathrm{SC}_8$, as shown in Fig. 2.5; besides, SC_8 is a special cell without constant multipliers. Based on (2.69), we can express $\tilde{S}_i$ as follows:

$$\tilde{S}_i = \alpha^8 S_i = \begin{cases} \sum_{j=0}^{n-1}(r_j\alpha^{8(j+2)}\alpha^{-8})\alpha^{(i-8)\cdot j}, & \text{for } \ 0 < i \leq 8 \\ \sum_{j=0}^{n-1}(r_j\alpha^{8(2j+2)}\alpha^{-8})\alpha^{((i-8)-8)\cdot j}, & \text{for } \ 8 < i \leq 16. \end{cases} \tag{2.70}$$

As $0 < i \leq 8$ and $8 < i \leq 16$, the received symbol r_j should respectively multiply by factors $\alpha^{8(j+1)}$ and $\alpha^{8(2j+1)}$, or $\alpha^{8(j+2)}$ and $\alpha^{8(2j+2)}$ in Montgomery multiplication. In Fig. 2.5, two factor generators, FG_1 and FG_2, are allocated to produce the scaling factors with Montgomery multipliers. The register in FG_1 initially contains $\alpha^{8(n+1)}$ and serially multiplies by $B_1 = 1$; as a result, $\alpha^{8(j+2)}$ in which j counts from $n-1$ to 0 will be generated. Similarly, to create $\alpha^{8(2j+2)}$ for $j = (n-1) \sim 0$ in FG_2, the register sequentially multiplies by $B_2 = \alpha^{-8}$ with initial value $\alpha^{8\cdot 2n}$. We then multiply r_j and the factors from FG_1 and FG_2 with Montgomery multipliers.



(b) SC$_i$ for $i = 1 \sim 7$

(a) Syndrome calculator

(c) SC$_8$

Figure 2.5: Syndrome calculator with $d = 8$ and $t \leq 8$

Although the syndrome calculator in Fig. 2.5 supports only $t \leq 8$, it can be extended to handle $t \leq 16$ syndrome calculation which has at most 32 syndromes $S_i = r(\alpha^i)$ with

$i = 1 \sim 32$. The first 16 syndromes $\tilde{S}_1 \sim \tilde{S}_{16}$ can be computed by using the configuration in Fig. 2.5. The calculation of other syndromes needs some modifications for FG_1 and FG_2. We first represent $\tilde{S}_{17} \sim \tilde{S}_{32}$ in the following form:

$$\tilde{S}_i = \alpha^8 S_i = \begin{cases} \sum_{j=0}^{n-1} (r_j \alpha^{8(3j+2)} \alpha^{-8}) \alpha^{(i-16)\cdot j}, & \text{for} \;\; 16 < i \leq 24 \\ \sum_{j=0}^{n-1} (r_j \alpha^{8(4j+2)} \alpha^{-8}) \alpha^{((i-24)-8)\cdot j}, & \text{for} \;\; 24 < i \leq 32. \end{cases} \tag{2.71}$$

In (2.71), the constant multiplications remain the same as compared to (2.70). The difference is the factors by which r_j multiplies; consequently, as j counts from $(n-1)$ to 0, FG_1 and FG_2 will generate $\alpha^{8(3j+2)}$ and $\alpha^{8(4j+2)}$. The initial value becomes $\alpha^{8(3n-1)}$, and the B_1 input α^{-16} in FG_1. On the other hand, we configure FG_2 with the initial value $\alpha^{8(4n-2)}$ and the input $B_2 = \alpha^{-24}$. Because there are 16 computation cells in Fig. 2.5, it will require double time to complete 32 syndrome calculations in contrast to $t \leq 8$ case.

Generally, according to (2.69), the architecture (see Fig. 2.5) is possible to be extended to more syndromes, but the computation time also increases. The present approach provides the feasibility for different syndrome calculations; nevertheless, the trade-off between the number of syndrome cells and the the computation time should depend on system specifications and requirements.

2.4.4 Key Equation Solver

The algorithm in solving key equation (2.26) or (2.37) can be either Berlekamp-Massey algorithm or Euclidean algorithm. As indicated in section 2.3, Berlekamp-Massey algorithm has the fixed iteration number $2t$, which is much regular for different RS codes and therefore is selected for universal RS decoder. For error and erasure decoding, we combine the Forney syndrome calculator and the polynomial multiplier for $\Lambda(x)$ in Fig. 2.1 with the key equation solver. Th inversionless algorithm is also applied [32,33,84,85] to avoid finite field division, leading to less complexity. The inversionless Berlekamp-Massey algorithm is presented as follows:

- Initial conditions:

$$\begin{aligned}&\sigma^{(u)}(x) = 1, \quad \tau^{(u)}(x) = 1, \quad D_u = 0\\ &\Delta_u = T_{u+1}, \quad \delta = 1,\end{aligned}$$

- Iteration from $i = (u+1)$ to $2t$:

$$\sigma^{(i)}(x) \;=\; \delta\sigma^{(i-1)}(x) + \Delta_{i-1}\tau^{(i-1)}(x)\cdot x \tag{2.72}$$

$$\Delta_i \;=\; \sum_{j=0}^{t-1}\sigma_j^{(i)}T_{i+1-j} \tag{2.73}$$

If $\Delta_{i-1} = 0$ or $D_{i-1} \geq i - u - D_{i-1}$,

$$\begin{aligned}&\tau^{(i)}(x) = \tau^{(i-1)}(x)\cdot x,\\ &D_i = D_{i-1}.\end{aligned}$$

Otherwise, the erasure information $\alpha^{l_1} \sim \alpha^{l_u}$

$$\begin{aligned}&\tau^{(i)}(x) = \sigma^{(i-1)}(x),\\ &D_i = i - u - D_{i-1}, \quad \delta = \Delta_{i-1}\end{aligned}$$

The final result $\sigma^{(2t)}(x)$ will be $(\eta\cdot\sigma(x))$ that includes a element $\eta \in GF(2^m)$. Note that η will be ineffective for the roots of $\sigma(x) = 0$ and the error evaluation in (2.45) and (2.46) because error evaluation polynomial will also be $(\eta\cdot\omega(x))$ [77].

Before we start to solve key equation, the erasure information $\alpha^{l_1} \sim \alpha^{l_u}$ in (2.36) should be generated. Similarly, we generate $\alpha^{l_1+d} \sim \alpha^{l_u+d}$ for the simplification in key equation solver. The erasure information generator is illustrated in Fig. 2.6 with a constant α^{-1} multiplier, where we let $\alpha^{-1} = \alpha^{(d-1)-d}$ and assign the constant multiplier input to be α^{d-1}. As what we had discussed previously, α^{d-1} can be represented as a constant element. The register initially contains $\alpha^{(n-1)+d}$ and serially multiplies by α^{-1}, corresponding to the received sequence r_j.

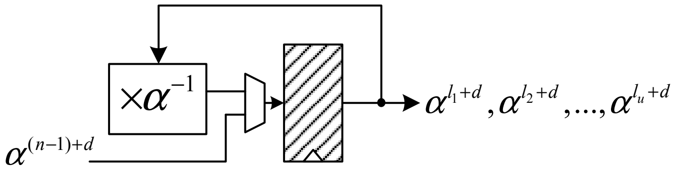


Figure 2.6: Erasure information generator

The contents of the register are captured whenever the erasure appears in the received data. After receiving all r_j, we obtain the the erasure information $\alpha^{l_1+d} \sim \alpha^{l_u+d}$.

With the syndrome polynomial

$$\tilde{S}(x) = \sum_{i=1}^{2t} \tilde{S}_i x^{i-1} = \sum_{i=1}^{2t} \alpha^d S_i x^{i-1}, \tag{2.74}$$

the key equation can be solved as follows:

- Initial conditions:

$$\sigma^{(0)}(x) = \tilde{S}(x), \quad \tau^{(0)}(x) = \tilde{S}(x)$$
$$\Delta_0 = \alpha^{l_1+d}, \qquad \delta = \alpha^d$$

- Iteration from $i = 1$ to u:

$$\sigma^{(i)}(x) = \delta\sigma^{(i-1)}(x) + \Delta_{i-1}\tau^{(i-1)}(x) \cdot x \pmod{x^{2t}} \tag{2.75}$$

$$\Delta_i = \alpha^{l_{i+1}+d}, \qquad \tau^{(i)}(x) = \sigma^{(i)}(x)$$

- When $i = u$, we set the following conditions for further computation.

$$\tilde{T}(x) = \sum_{i=1}^{2t} \tilde{T}_i x^{i-1} = \sigma^{(u)}(x), \quad \Delta_u = \tilde{T}_{u+1},$$
$$\sigma^{(u)}(x) = \alpha^d, \quad \tau^{(u)}(x) = \alpha^d, \quad D_u = 0$$

- Iteration from $i = (u+1)$ to $2t$:

$$\sigma^{(i)}(x) = \delta\sigma^{(i-1)}(x) + \Delta_{i-1}\tau^{(i-1)}(x) \cdot x \tag{2.76}$$

$$\Delta_i = \sum_{j=0}^{t-1} \sigma_j^{(i)} \tilde{T}_{i+1-j} \tag{2.77}$$

If $\Delta_{i-1} = 0$ or $D_{i-1} \geq i - u - D_{i-1}$,

$$\tau^{(i)}(x) = \tau^{(i-1)}(x) \cdot x,$$
$$D_i = D_{i-1}.$$

Otherwise,

$$\tau^{(i)}(x) = \sigma^{(i-1)}(x),$$
$$D_i = i - u - D_{i-1}, \quad \delta = \Delta_{i-1}$$

When all $2t$ iterations are finished, we will attain the error location polynomial

$$\tilde{\sigma}(x) = \sigma^{(2t)}(x) = \eta'\sigma(x) = \sum_{i=0}^{v} \tilde{\sigma}_i x^i \tag{2.78}$$

with with $\eta' = \alpha^d\eta$. The iteration from $i = 1$ to u intends to estimate the Forney syndrome polynomial $\tilde{T}(x) = \alpha^d T(x)$; the inversionless Berlekamp-Massey algorithm proceeds from $i = (u+1)$ to $2t$. Since we apply the Montgomery multiplication to all FFM computations, each quantity will contains an additional factor α^d. Subsequently, we can establish the error value polynomial

$$\tilde{\omega}(x) = \eta'\omega(x) = \sum_{i=0}^{u+v-1} \tilde{\omega}_i x^i \tag{2.79}$$

by the following property

$$\tilde{\omega}_i = \sum_{j=0}^{i} \tilde{\sigma}_j \tilde{T}_{i+1-j}, \tag{2.80}$$

which is quite similar to the discrepancy evaluation in (2.77). On the other hand, we can derive the errata location polynomial $\Lambda(x)$ with the process of $\tilde{T}(x)$ construction. The procedure is described as follows:

- Initial conditions:

$$\sigma^{(2t)}(x) = \tilde{\sigma}(x), \quad \tau^{(2t)}(x) = \tilde{\sigma}(x)$$
$$\Delta_{2t} = \alpha^{l_1+d}, \qquad \delta = \alpha^d$$

- Iteration from $i = 2t + 1$ to $2t + u$:

$$\sigma^{(i)}(x) = \delta\sigma^{(i-1)}(x) + \Delta_{i-1}\tau^{(i-1)}(x) \cdot x \tag{2.81}$$

$$\Delta_i = \alpha^{l_{i+1}+d}, \qquad \tau^{(i)}(x) = \sigma^{(i)}$$

The errata polynomial will be $\tilde{\Lambda}(x) = \sigma^{(2t+u)}(x) = \eta'\Lambda(x)$. Notice that the uniformity of (2.75), (2.76), and (2.81) lead to the simple key equation solver architecture. Moreover, the calculations of discrepancy and error value polynomial calculations can use identical computational structures.

The above mentioned algorithm requires the computations of Forney syndrome polynomial and errata location polynomial; nevertheless, those computations can be combined with Berlekamp-Massey algorithm as reported in [89]. The modified flow is as follows:

- Initial conditions:

$$\sigma^{(0)}(x) = \alpha^d, \quad \tau^{(0)}(x) = \alpha^d$$
$$\Delta_0 = \alpha^{l_1+d}, \qquad \delta = \alpha^d$$

- Iteration from $i = 1$ to u:

$$\sigma^{(i)}(x) = \delta\sigma^{(i-1)}(x) + \Delta_{i-1}\tau^{(i-1)}(x) \cdot x \tag{2.82}$$

$$\Delta_i = \alpha^{l_{i+1}+d}, \qquad \tau^{(i)}(x) = \sigma^{(i)}(x)$$

- When $i = u$, we initialize Berlekamp-Massey algorithm with $\sigma^{(u)}(x) = \alpha^d \lambda(x)$.

$$\tau^{(u)}(x) = \sigma^{(u)}(x), \quad \Delta_u = \tilde{S}_{u+1}, \quad D_u = 0$$

- Iteration from $i = (u+1)$ to $2t$:

$$\begin{aligned} \sigma^{(i)}(x) &= \delta\sigma^{(i-1)}(x) + \Delta_{i-1}\tau^{(i-1)}(x) \cdot x && (2.83) \\ \Delta_i &= \sum_{j=0}^{t-1} \sigma_j^{(i)} \tilde{S}_{i+1-j} && (2.84) \end{aligned}$$

If $\Delta_{i-1} = 0$ or $D_{i-1} \geq i - u - D_{i-1}$,

$$\begin{aligned} &\tau^{(i)}(x) = \tau^{(i-1)}(x) \cdot x, \\ &D_i = D_{i-1}. \end{aligned}$$

Otherwise,

$$\begin{aligned} &\tau^{(i)}(x) = \sigma^{(i-1)}(x), \\ &D_i = i - u - D_{i-1}, \quad \delta = \Delta_{i-1} \end{aligned}$$

The errata location polynomial will be finally obtained as

$$\tilde{\Lambda}(x) = \sigma^{(2t)}(x) = \sum_{j=0}^{u+v} \tilde{\Lambda}_j x^j. \qquad (2.85)$$

Additionally, the coefficients of error value polynomial can be derived according to the key equation (2.37). That is

$$\tilde{\omega}_i = \sum_{j=0}^{i} \tilde{\Lambda}_j \tilde{S}_{i+1-j}, \quad \text{for} \;\; i = 0 \sim u + v - 1. \qquad (2.86)$$

The operations from $i = 1$ to u are erasure location polynomial expansion that recursively computes $\alpha^d\lambda(x)$ by (2.82). The Berlekamp-Massey algorithm, which is from $i = u + 1$

to $2t$, starts with both $\sigma^{(u)}(x)$ and $\tau^{(u)}(x)$ equal to $\alpha^d\lambda(x)$, and therefore the final result will be $(\eta'\sigma(x)\lambda(x)) = \tilde{\Lambda}(x)$. Furthermore, to achieve less decoding latency, the polynomial expansion in (2.82) can work in parallel with syndrome calculator because it is independent of the syndromes $\tilde{S}_i$.

Based on the decomposed architecture in [77], the key equation solver is demonstrated in Fig. 2.7. Only three Montgomery multipliers are required due to the serial operations, leading to the modest complexity. There are two buffer memory, buffer-σ and buffer-τ, for storing $\sigma^{(i-1)}(x)$ and $\tau^{(i-1)}(x)$. We configure this architecture for erasure location polynomial, Berlekamp-Massey algorithm, and error value polynomial. When $i = 1 \sim u$, Fig. 2.7 is in polynomial expansion mode (2.82) with $\Delta_i = \alpha^{l_{i+1}+d}$ and $\delta = \alpha^d$, and the result $\sigma^{(i)}(x)$ is then saved to both buffer-σ and buffer-τ. After u iterations, we have $\alpha^d\lambda(x)$ in both buffers which are ready for the next Berlekamp-Massey algorithm. As the syndrome polynomial $\tilde{S}(x)$ is available, the solver will perform (2.83) and (2.84) from $i = u + 1$ to $2t$ and finally generate $\tilde{\Lambda}(x)$ (see (2.85)) in buffer-σ. The error value polynomial can also obtained with Fig. 2.7 according to (2.86). We let $\Delta_{i-1} = 0$ and $\delta = \alpha^d$; consequently, the coefficient $\tilde{\Lambda}_j$ from buffer-σ will multiply by $\tilde{S}_{i+1-j}$, and the product will be accumulated to be $\tilde{\omega}_i$.

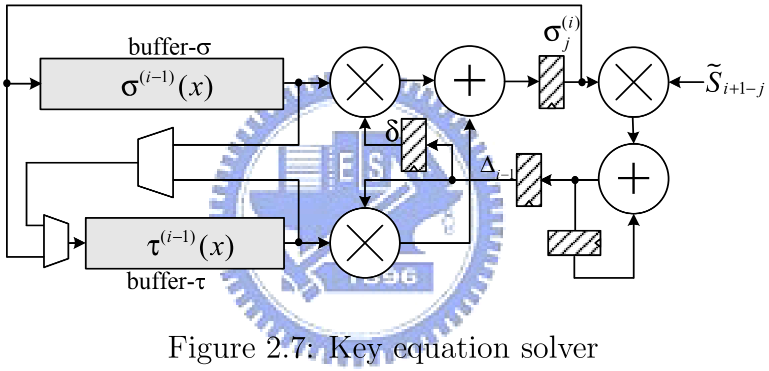


Figure 2.7: Key equation solver

2.4.5 Chein Search

The key equation solver will provide errata location polynomial $\tilde{\Lambda}(x)$ for Chien search operation that repeatedly check $\tilde{\Lambda}(x) = 0$ when $x = \alpha^0, \alpha^{-1}, \ldots, \alpha^{-(n-1)}$. We can represent

Chien search as the following equation

$$\tilde{\Lambda}(\alpha^{-i}) = \sum_{j=0}^{u+v} \tilde{\Lambda}_j \alpha^{-i\cdot j}, \tag{2.87}$$

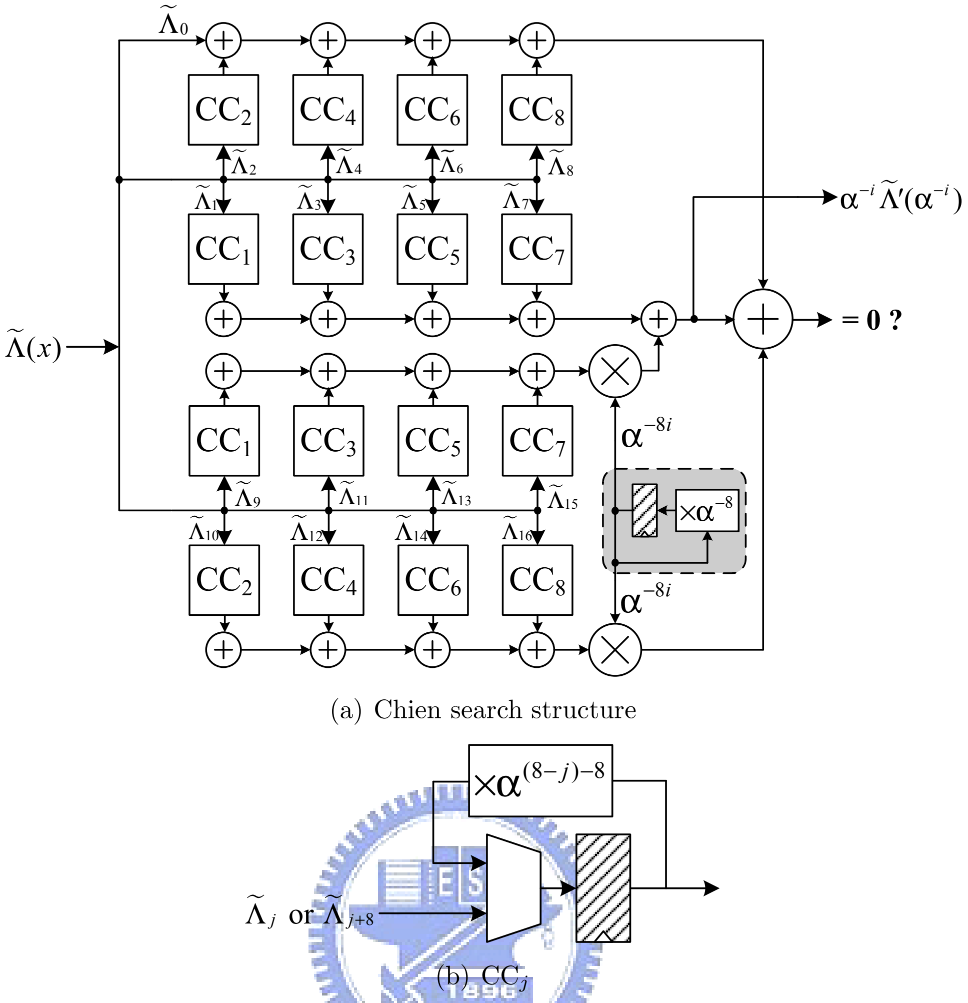
which is similar the syndrome calculation (2.65). In order to reduce the multiplier complexity, (2.87) is also transformed to

$$\begin{aligned} \tilde{\Lambda}(\alpha^{-i}) &= \sum_{j=0}^{u+v} \tilde{\Lambda}_j (\alpha^{(d-j)-d})^i, & (2.88) \\ &= \tilde{\Lambda}_0 + \sum_{\pi=0}^{\pi_m} (\alpha^{-\pi d})^i \cdot \sum_{j=1}^{d} \tilde{\Lambda}_{\pi d+j} (\alpha^{(d-j)-d})^i, & (2.89) \end{aligned}$$

where $\pi_m = \lfloor \frac{u+v}{d} \rfloor$, and $\tilde{\Lambda}_{\pi d+j} = 0$ when $\pi d + j > u + v$. We divide the coefficients of $\tilde{\Lambda}(x)$ in (2.89) into $(\pi_m + 1)$ groups and let $\tilde{\Lambda}_{\pi d+j}$ multiply by $\alpha^{((d-j)-d)\cdot i}$ with $1 \leq j \leq d$. As a result, α^{d-j} can be represented as a constant input for Montgomery multiplier because $0 \leq d - j \leq d - 1$. With $d = 8$ and $t \leq 16$ including erasures, the Chien search structure is presented in Fig. 2.8. Since the maximum degree of $\tilde{\Lambda}(x)$ is 16, this architecture should have 16 Chien search cells (CC). The j-th Chien search cell (CC$_j$) use a constant multiplier in which one of the inputs is forced to α^{8-j}. Based on (2.89), there is a factor generator producing α^{-8i} for $i = 1 \sim (n - 1)$. Additionally, the values $\tilde{\Lambda}'(\alpha^{-j_\kappa})$ in (2.45) and $\tilde{\Lambda}'(\alpha^{-j_\rho})$ in (2.46) are determined when $\tilde{\Lambda}(\alpha^{-j_\kappa})$ or $\tilde{\Lambda}(\alpha^{-j_\rho})$ equals to zero [77]. However, the outputs of Fig. 2.8 are $\tilde{\Lambda}'(\alpha^{-j_\kappa}) \cdot \alpha^{-j_\kappa}$ or $\tilde{\Lambda}'(\alpha^{-j_\rho}) \cdot \alpha^{-j_\rho}$ because of the following conditions:

$$\tilde{\Lambda}'(x) = \sum_{j=0}^{d_\Lambda} \tilde{\Lambda}_{2j+1} x^{2j} = x^{-1} \cdot \sum_{j=0}^{d_\Lambda} \tilde{\Lambda}_{2j+1} x^{2j+1} \triangleq x^{-1} \cdot \tilde{\Lambda}_{odd}(x) \tag{2.90}$$

in which $d_\Lambda = \frac{u+v}{2} - 1$ as $(u + v)$ is even, or $d_\Lambda = \frac{u+v-1}{2}$ as $(u + v)$ is odd. We also define a polynomial $\tilde{\Lambda}_{odd}(x)$ to be $\tilde{\Lambda}(x)$ with zero coefficients in the even degree terms; that is, $\tilde{\Lambda}_{2j} = 0$. Consequently, the structure in Fig. 2.8 can also generate $\tilde{\Lambda}_{odd}(\alpha^{-i}) = \alpha^{-i}\tilde{\Lambda}'(\alpha^{-i})$



(a) Chien search structure

(b) CC_j

Figure 2.8: Chien search module with $d = 8$ and $t \leq 16$

during Chien search operation. When Chien search identifies the error or erasure locations, the results $\alpha^{-i}\tilde{\Lambda}'(\alpha^{-i})$ will be delivered for error or erasure value evaluation.

2.4.6 Error Value Evaluator

In the error and erasure value computations with (2.45) and (2.46), not only the inversions of $\tilde{\Lambda}'(\alpha^{-j_\kappa})$ and $\tilde{\Lambda}'(\alpha^{-j_\rho})$, but also $\tilde{\omega}(\alpha^{-j_\kappa})$ and $\tilde{\omega}(\alpha^{-j_\rho})$ need to be determined. In order to comply the data from Chien search, we modify the error and erasure values to be

$$e_\kappa = \frac{-\alpha^{-j_\kappa}\tilde{\omega}(\alpha^{-j_\kappa})}{\alpha^{-j_\kappa}\tilde{\Lambda}'(\alpha^{-j_\kappa})} \tag{2.91}$$

$$e_\rho = \frac{-\alpha^{-j_\rho}\tilde{\omega}(\alpha^{-j_\rho})}{\alpha^{-j_\rho}\tilde{\Lambda}'(\alpha^{-j_\rho})}, \tag{2.92}$$

meaning that $x \cdot \tilde{\omega}(x)$ should be used in both (2.91) and (2.92). Fig. 2.9 shows the corresponding architecture for both error and erasure values. The cell CC_j is identical to the j-th Chien search cell in Fig. 2.8, and the evaluation of $\alpha^{-i}\tilde{\omega}(\alpha^{-i})$ is based on the equation:

$$\alpha^{-i}\tilde{\omega}(\alpha^{-i}) = \sum_{j=1}^{u+v} \tilde{\omega}_{j-1}(\alpha^{(d-j))-d})^i = \sum_{\pi=0}^{\pi_m} (\alpha^{-\pi d})^i \cdot \sum_{j=1}^{d} \tilde{\omega}_{\pi d+j-1}(\alpha^{(d-j)-d})^i. \tag{2.93}$$

For $d = 8$ and $t \leq 16$, the computation of (2.93) can be divided into two groups corresponding to $\pi = 0$ and $\pi = 1$. In the $\pi = 1$ group, we need an additional α^{-8i} generator to adjust the result. Finally, the divider performs finite field division to complete (2.91) and (2.92). The divider can be either a look-up table combined with a multiplier or the structure in Fig. 2.3 using Fermat's identity (2.61). The look-up table requires multiple contents to satisfy various $GF(2^m)$; accordingly, we use a dynamic look-up table updated on-the-fly for different finite field definitions. The inversion table is presented in Fig. 2.10. A random access memory (RAM) with the size of $2^d \times d$ is implemented to store the inversion contents that create by α^{-i+d} generator. We let each α^{-i+d} be indexed by α^i to compensate the α^{-d} factor in Montgomery multiplication completing the division operation. The inversion contents can be generated during the syndrome calculation. The registers in α^{-i+d} and α^i generators are initialized with the constants α^d and $\alpha^0 = 1$. As i counts from 0 to $n-1$, each value α^{-i+d} will be written to the address α^i. Notice that α^{-i+d} generator uses a constant Montgomery multiplier, however, the constant α multiplier in α^i generator is a direct implementation without Montgomery algorithm because α^{1+d} is unable to be a constant multiplier input. As compared with the divider in Fig. 2.3, the look-up table based architecture provides much

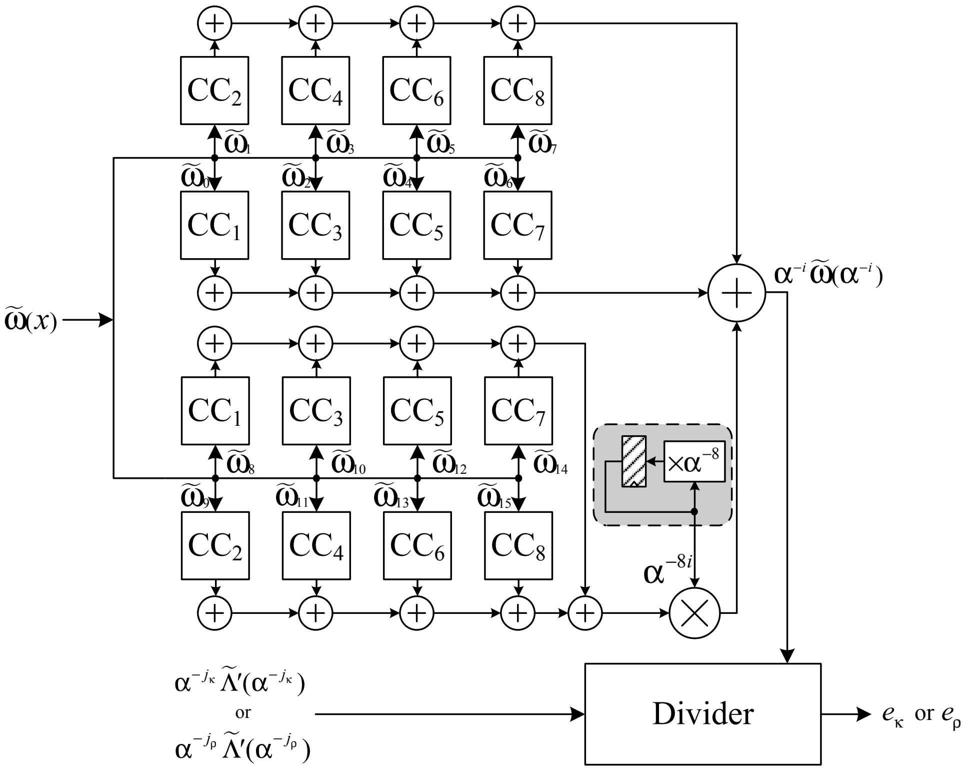
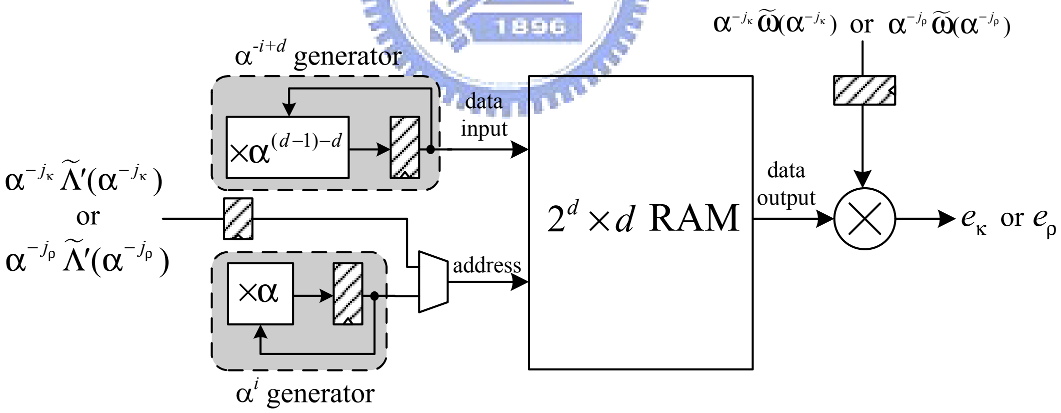


Figure 2.9: Error value evaluator with $d = 8$ and $t \leq 16$

faster computation although a memory is required.



α^{-i+d} generator
$\alpha^{-j_\kappa}\widetilde{\omega}(\alpha^{-j_\kappa})$ or $\alpha^{-j_\rho}\widetilde{\omega}(\alpha^{-j_\rho})$
$\times\alpha^{(d-1)-d}$
data input
$2^d \times d$ RAM
data output
$\alpha^{-j_\kappa}\widetilde{\Lambda}'(\alpha^{-j_\kappa})$
or
$\alpha^{-j_\rho}\widetilde{\Lambda}'(\alpha^{-j_\rho})$
e_κ or e_ρ
$\times\alpha$
address
α^i generator

Figure 2.10: Finite field divider

Chapter 3

Convolutional codes

Convolutional codes first proposed by Elias [13] in 1955 have been widely exploited in communication systems to provide a superior error correction capacity. Compared with block codes, convolutional encoder contains memory, or the codeword symbols depends not only on the current information symbols, but also on some previous information symbols. Therefore, an (n, k, m) convolutional code can be defined as follows: the information stream or block is divided to frames of k symbols, and current information frame as well as m previous ones are encoded into codeword frame of length n. The error correction performance is determined by the code rate $R = k/n$ and the memory order m.

The decoding of convolutional codes includes sequential decoding [14], threshold decoding [90], and Viterbi algorithm [15]. Moreover, the maximum *a posteriori* (MAP) decoding algorithm [16] and the soft output Viterbi algorithm (SOVA) [19] are also utilized for soft iterative decoding. Sequential decoding is the first practical decoding algorithm whose complexity is independent of the memory order m. The Fano algorithm [91] and the stake algorithm [92, 93] are also of this type. Considering simple implementation, the threshold decoding [94], or majority-logic decoding, is proposed in [90], however, the performance is sub-optimum. The Viterbi algorithm is a maximum likelihood (ML) decoding that optimally minimizes the block error probability. Because of the parallel computation, the decoder using Viterbi algorithm is quite suitable for high speed applications. Hence, the requirement for high speed data transmission has conducted many researches in algorithm transformations and VLSI implementation. The MAP algorithm, or referred to BCJR algorithm, and the sub-optimum SOVA are symbol by symbol detection algorithm that minimizes the symbol error probability. They are much popular due to the iterative decoding of the parallel concatenated convolutional codes (PCCC), named turbo code [17,18].

This section will focus on the Viterbi decoding algorithm as well as the architecture design after the introduction of convolutional codes. The MAP algorithm and the SOVA are also presented finally.

3.1 Convolutional codes and encoders

An (n, k) convolution code over $F = GF(q)$ can be defined as a k-dimensional subspace of n-dimensional vector space $F^n((D))$ [94,95]. The set $F((D))$ of Laurent series in D over F is the set of sequences $a(D) = \sum_{i=r}^{\infty} a_i D^i$ in which $a_i \in F$, and r is an arbitrary integer. Furthermore, we can express the information stream as the sequence

$$\mathbf{u}(D) = \sum_{i=r}^{\infty} u_i D^i \tag{3.1}$$

where $u_i = (u_i^{(1)}, u_i^{(2)}, \ldots, u_i^{(k)})$ is i-th k-tuple information frame, and therefore $\mathbf{u}(D) \in F^k((D))$. The codeword stream is also the sequence $\mathbf{v}(D)$ in $F^n((D))$ and can be represented with the n-tuple codeword frame $v_i = (v_i^{(1)}, v_i^{(2)}, \ldots, v_i^{(n)})$:

$$\mathbf{v}(D) = \sum_{i=r}^{\infty} v_i D^i \tag{3.2}$$

The (n, k) convolutional encoder can then have the following definition [95,96]:

Definition 3.1. An encoder is a $k \times n$ convolutional encoder over F if the mapping $F^k((D)) \mapsto F^n((D))$ realized by the encoder can be represented by $\mathbf{v}(D) = \mathbf{u}(D)G(D)$ where $G(D)$ is a $k \times n$ matrix of rank k with entries in the subset $F(D)$ of $F((D))$.

We represent the set of finite degree polynomials in D over F by $F[D]$; moreover, $F(D)$ denotes the set of rational functions $a(D)/b(D)$ with both $a(D)$ and $b(D)$ in $F[D]$. Clearly,

$F[D]$ and $F(D)$ are subsets of $F((D))$. The generator matrix $G(D)$, also termed *encoder*, in the definition 3.1 results in the codewords of length n and coding rate $R = k/n$. Generally, $G(D)$ is often expressed as a generator matrix:

$$G(D) = \begin{bmatrix} g_1^{(1)}(D) & g_1^{(2)}(D) & \cdots & g_1^{(n)}(D) \\ g_2^{(1)}(D) & g_2^{(2)}(D) & \cdots & g_2^{(n)}(D) \\ \vdots & \vdots & \ddots & \vdots \\ g_k^{(1)}(D) & g_k^{(2)}(D) & \cdots & g_k^{(n)}(D) \end{bmatrix}. \tag{3.3}$$

where $g_i^{(j)}(D) \in F(D)$ for all $i = 1, 2, \ldots, k$ and $j = 1, 2, \cdots, n$. We express the rational function as $g_i^{(j)}(D) = a_i^{(j)}(D)/b_i^{(j)}(D)$ with the polynomials $a_i^{(j)}(D)$ and $b_i^{(j)}(D)$. Accordingly, the memory order m is the maximum degree among all $a_i^{(j)}(D)$ in $G(D)$.

$$m = \max_{1\leq i\leq k, 1\leq j\leq n} \deg[a_i^{(j)}(D)] \tag{3.4}$$

In the graphical representation, we can use the *controller canonical form* for the linear system with the input $u(D) \in F((D))$:

$$v(D) = u(D)a(D)/b(D), \tag{3.5}$$

where $a(D), b(D) \in F[D]$ and

$$a(D) = \sum_{i=0}^{m} a_i D^i$$
$$b(D) = 1 + \sum_{i=1}^{m} b_i D^i$$

As shown in Fig. 3.1, the delay elements denoted by D construct a single input shift register. The m delay elements are memory units that store the previous m shift register inputs. We can find that $a(D)/b(D)$ is a rational transfer function for the input $u(D)$, and the output

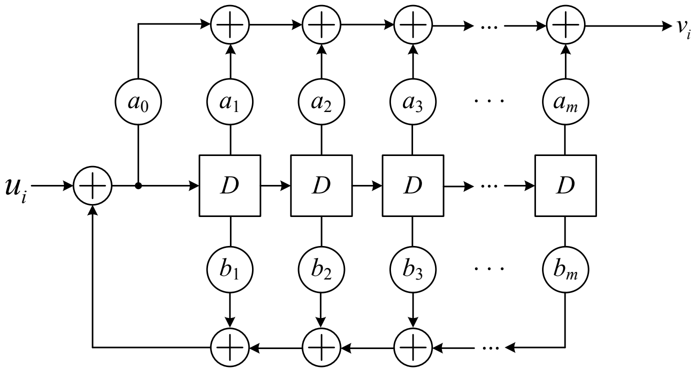


Figure 3.1: The controller canonical form of (3.5)

$v(D)$ is also in $F((D))$. Alternatively, if (3.5) is rewritten as

$$v(D) = u(D)a(D) + v(D)(\sum_{i=1}^{m} b_i D^i), \tag{3.6}$$

the *observer canonical form* of the system is illustrated in Fig. 3.2, which is another realization of Fig. 3.1 for the same transfer function. The generator matrix in (3.3) with the

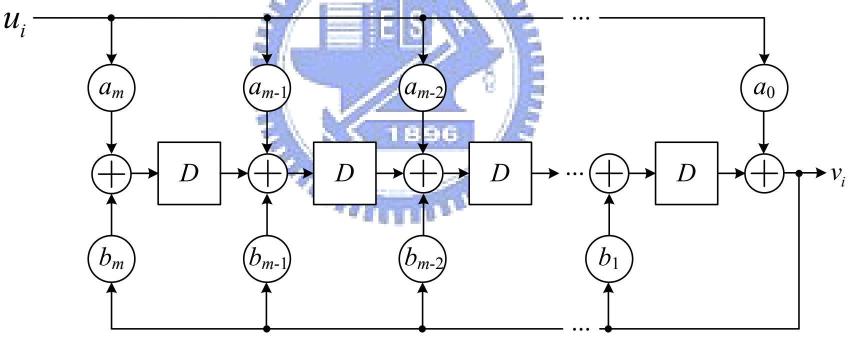


Figure 3.2: The observer canonical form of (3.6)

entries in $F(D)$ is referred to a *rational transfer function matrix*.

The above mentioned encoder (3.3) leads to the *constraint length* definitions [94] listed below:

1. The constraint length for i-th information sequence is the maximum degree within i-th row of $G(D)$.

$$\nu_i = \max_{1 \le j \le n} \deg[a_i^{(j)}(D)] \tag{3.7}$$

Note that $a_i^{(j)}(D)$ is the numerator of the rational function $g_i^{(j)}(D)$.

2. The overall constraint length is the summation of (3.7) for all information sequence.

$$\nu = \sum_{i=1}^{k} \nu_i \tag{3.8}$$

3. The input constraint length is the number of information symbols that affect each codeword frame.

$$K = \nu + k \tag{3.9}$$

4. The output constraint length is the number of codeword symbols that related to a single information frame.

$$n_m = n(m+1) \tag{3.10}$$

The required number of memory elements in the controller canonical form (see Fig. 3.1) is equal to the overall constraint length ν. Notice that many literatures define a single constraint length for an encoder with one of the above four statements. For example, in [97] the overall constraint length ν in (3.8) is defined to be the constraint length of a convolutional encoder.

There are infinite number of generator matrices that produce the same set of codewords. Thus we can define the equivalence of two encoders as:

Definition 3.2 (Forney [98])**.** Two encoders $G(D)$ and $G'(D)$ are equivalent if they generate the same code.

Furthermore, two equivalent encoders $G(D)$ and $G'(D)$ can be related by $G(D) = T(D)G'(D)$ if and only if $T(D)$ is a $k \times k$ nonsingular matrix over $F(D)$ [96]. For two codeword sets $\{\mathbf{v}(D)\}$ and $\{\mathbf{v}'(D)\}$ corresponding to $G(D)$ and $G'(D)$, or

$$\begin{aligned}\mathbf{v}(D) &= \mathbf{u}(D)G(D) = \mathbf{u}(D)T(D)G'(D) = \mathbf{u}'(D)G'(D)\\ \mathbf{v}'(D) &= \mathbf{u}(D)G'(D),\end{aligned}$$

we can find that $\{\mathbf{u}'(D) = \mathbf{u}(D)T(D)\}$ and $\{\mathbf{u}(D)\}$ are identical sets because $T(D)$ is invertible. It can also be verified that $T(D)$ is nonsingular if $G(D)$ and $G'(D)$ are equivalent.

The encoder in definition 3.1 is required to perform one-to-one mapping for correctly decoding. Therefore, these exists a $G^{-1}(D)$ such that

$$\mathbf{v}(D)G^{-1}(D) = \mathbf{u}(D)G(D)G^{-1}(D) = \mathbf{u}(D)$$

for all $\mathbf{u}(D)$. We use the equation $G(D)G^{-1}(D) = I_k$ where I_k is the $k \times k$ identity matrix, and $G(D)^{-1}$ is the right inverse of $G(D)$. The basic encoder is defined as follows:

Definition 3.3 (Forney [98])**.** A generator matrix $G(D)$ is *basic* if it is polynomial and has a polynomial right inverse.

It follows that every rational encoder is equivalent to a basic convolutional encoder [96, 98]. Generally, basic encoders are not unique, and therefore, two basic encoders $G(D)$ and $G'(D)$ are equivalent if and only if $G'(D) = T(D)G(D)$ where $T(D)$ is a $k \times k$ polynomial matrix with determinant 1 [95, 96]. Because $T(D)$ is nonsingular, $G'(D) = T(D)G(D)$ indicates the equivalence of $G'(D)$ and $G(D)$. Conversely, if $G(D)$ and $G'(D)$ are equivalent, there exist a $k \times k$ matrix $T(D)$ over $F(D)$ such that $G'(D) = T(D)G(D)$, and $T(D)^{-1}$ exists. Furthermore, since $G(D)$ is basic and thus has a polynomial right inverse $G^{-1}(D)$, we can find that $T(D) = G'(D)G^{-1}(D)$ is also polynomial; besides, $T(D)^{-1}$ should be polynomial. As a result, $G'(D) = T(D)T^{-1}(D)G'(D)$ and $T(D)T^{-1}(D) = I_k$. Since both $T(D)$ and $T^{-1}(D)$ are polynomials, $T(D)$ must have determinant 1, or $\det(T(D)) = 1$.

Among all equivalent basic encoders, there is one that requires a minimal number of memory elements. Accordingly, such encoder is defined by

Definition 3.4 (Johannesson [96]). A *minimal basic* encoding matrix is a basic generator matrix whose overall constraint length ν is minimal over all equivalent basic encoding matrices.

Let $G(D)$ be a basic encoder, and it can be decomposed into three parts:

$$G(D) = G_0(D) + \begin{bmatrix} D^{\nu_1} & & & \\ & D^{\nu_2} & & \\ & & \ddots & \\ & & & D^{\nu_k} \end{bmatrix} \cdot \bar{G}_h \tag{3.11}$$

Notice that $\bar{G}_h$ is a $k \times n$ matrix over F and has nonzero entries in the positions (i, j) where $\deg g_i^j(D) = \nu_i$. As a result, $G_0(D)$ contains the enteritis in $G(D)$ where the highest degree terms in each row are removed. For $G(D)$ with the overall constraint length ν, we can obtain the following equivalent statements [96, 97]:

1. $G(D)$ is a minimal basic encoder.

2. The maximum degree of all $k \times k$ sub-determinants of $G(D)$ equals to ν.

3. $\bar{G}_h$ is of full rank.

According to (3.11), the second and the third statements are equivalent. The proof about the equivalence of the first and the second statements can be found in [96] and [97]. The above three statements characterize the minimal basic encoder in definition 3.4 and provide the rules for constructing a minimal basic encoder. Moreover, a minimal basic encoder for a convolutional code is unnecessarily unique, and every encoding matrix is equivalent to a minimal basic encoding matrix [98].

The *physical state* of a rational encoder $G(D)$ at some time instance is defined to be the contents of the memory elements in its realization, which can be either the controller or the observer canonical form. On the other hand, the *abstract state* is the output sequence at time 0 and later if the input sequence $\mathbf{u}(D)$ occurs up to time -1 and becomes all zero thereafter. Hence, the abstract state depends only on $G(D)$ instead of its realization [96]. Different abstract states must correspond to different output sequences and different physical states. However, different physical states may correspond to the same abstract state [95]. Let P be the projection operator that forces the sequences to all zero for non-negative time instants, and Q be the projection operator that truncates the sequences at negative time instants. For example, assume $r < 0$,

$$\mathbf{u}(D)P = \sum_{i=r}^{-1} u_i D^i \tag{3.12}$$

$$\mathbf{u}(D)Q = \sum_{i=0}^{\infty} u_i D^i. \tag{3.13}$$

Therefore, the abstract state $\mathbf{v}_S(D)$ can be formally written by

$$\mathbf{v}_S(D) = \mathbf{u}(D)PG(D)Q. \tag{3.14}$$

Accordingly, the minimal basic encoder in definition 3.4 will have the following properties [96, 97, 99]:

1. If $G(D)$ is minimal basic and

   $$\mathbf{u}(D) = \sum_{i=-m}^{n} u_i D^i = (u_i^{(i)}, u_i^{(i)}, \ldots, u_i^{(k)})D^i, \tag{3.15}$$

   $\mathbf{u}(D)G(D)Q = 0$ will deduce that $\mathbf{u}(D) = 0$.

2. The abstract state number of a minimal basic encoder equals the physical state number.

3. The abstract state number of an encoder is always larger than or equal to that of an equivalent minimal basic encoder.

The first property can be verified with (3.11), and

$$\mathbf{v}(D) = \mathbf{u}(D)G_0(D) + \mathbf{u}(D)\begin{bmatrix} D^{\nu_1} & & & \\ & D^{\nu_2} & & \\ & & \ddots & \\ & & & D^{\nu_k} \end{bmatrix}\bar{G}_h \tag{3.16}$$

must have zero coefficients v_i for $i \geq 0$. Without loss of generality, we may assume that

$$m = \nu_1 = \nu_2 = \cdots = \nu_l > \nu_{l+1} \geq \cdots \geq \nu_k \tag{3.17}$$

Consequently, the coefficient v_{m+n} of D^{m+n} in $\mathbf{v}(D)$ will be

$$(u_n^{(0)}, u_n^{(1)}, \ldots, u_n^{(l)}, 0, \ldots, 0)\bar{G}_h = 0. \tag{3.18}$$

Since $\bar{G}_h$ has full rank, we can obtain $u_n^{(0)} = u_n^{(1)} = \cdots = u_n^{(l)} = 0$. The coefficient v_i for $i < m + n$ can also be examined in such method, and we will find that $\mathbf{u}(D) = 0$. For example, if $(m-1) = \nu_{l+1} > \nu_{l+2}$, the coefficient $\mathbf{v}_{m+n-1}$ becomes

$$(u_{n-1}^{(0)}, u_{n-1}^{(1)}, \ldots, u_{n-1}^{(l)}, u_n^{(l+1)}, 0, \ldots, 0)\bar{G}_h = 0, \tag{3.19}$$

and therefore $u_n^{(l+1)} = u_{n-1}^{(0)} = u_{n-1}^{(1)} = \cdots = u_{n-1}^{(l)} = 0$.

The second property of a minimal basic encoder $G(D)$ ensures the minimum realization of its controller canonical form. Because the basic encoder is polynomial, the following input sequence

$$\mathbf{u}(D) = (\sum_{i=-1}^{-\nu_1} u_i^{(1)} D^i, \sum_{i=-1}^{-\nu_2} u_i^{(2)} D^i, \ldots, \sum_{i=-1}^{-\nu_k} u_i^{(k)} D^i) \tag{3.20}$$

will be the physical state in the controller canonical form at time instant 0. According to (3.14), the abstract state of $G(D)$ can be written as

$$\mathbf{v}_S(D) = \mathbf{u}(D)G(D)Q. \tag{3.21}$$

If there are two different physical states $\mathbf{u}_1(D)$ and $\mathbf{u}_2(D)$ in the form of (3.20) correspond to the the same abstract state, we will have

$$\mathbf{v}_S(D) = \mathbf{u}_1(D)G(D)Q = \mathbf{u}_2(D)G(D)Q, \tag{3.22}$$

or

$$(\mathbf{u}_1(D) - \mathbf{u}_2(D))G(D)Q = 0. \tag{3.23}$$

Directly from the first property, we can conclude that $(\mathbf{u}_1(D) - \mathbf{u}_2(D))$ must be zero, resulting in $\mathbf{u}_1(D) = \mathbf{u}_2(D)$. Hence the abstract state number should equal to the physical state number for a minimal basic encoder.

The third property shows the minimum of a minimal basic encoder. We consider two equivalent encoders $G(D)$ and $G'(D)$ whose abstract states are $\mathbf{v}_S(D)$ and $\mathbf{v}'_S(D)$ respectively. From the abstract state definition, $\mathbf{v}_S(D)$ can be obtained

$$\begin{aligned}\mathbf{v}_S(D) &= \mathbf{u}(D)PG(D)Q\\ &= \mathbf{u}(D)PT(D)G'(D)Q\\ &= \mathbf{u}(D)PT(D)(P+Q)G'(D)Q\\ &= \mathbf{u}(D)PT(D)PG'(D)Q + \mathbf{u}(D)PT(D)QG'(D)Q. \end{aligned}\tag{3.24}$$

Note that

$$\mathbf{v}'_S(D) = [\mathbf{u}(D)PT(D)P]G'(D)Q \tag{3.25}$$

is an abstract state of $G'(D)$; moreover,

$$\mathbf{v}'(D) = [\mathbf{u}(D)PT(D)Q]G'(D)Q = [\mathbf{u}(D)PT(D)Q]G'(D) \tag{3.26}$$

is a codeword generated by $G'(D)$ and the input sequence $\mathbf{u}(D)PT(D)Q$. Therefore, the abstract state of $G(D)$ can be expressed by the sum of an abstract state and a codeword corresponding to $G'(D)$. That is

$$\mathbf{v}_S(D) = \mathbf{v}'_S(D) + \mathbf{v}'(D) \tag{3.27}$$

If $G'(D)$ is minimal basic, the relation in (3.27) is unique. This uniqueness can be verified by the contrary assumption:

$$\mathbf{v}_S(D) = \mathbf{v}'_{S1}(D) + \mathbf{v}'_1(D) = \mathbf{v}'_{S2}(D) + \mathbf{v}'_2(D) \tag{3.28}$$

where $\mathbf{v}'_{S1}(D)$ and $\mathbf{v}'_{S1}(D)$ are different abstract states of $G'(D)$; $\mathbf{v}'_1(D)$ and $\mathbf{v}'_2(D)$ are codewords by $G'(D)$. Rearranging (3.28), we have

$$\mathbf{v}''_S(D) = \mathbf{v}'_{S1}(D) - \mathbf{v}'_{S2}(D) = \mathbf{v}'_2(D) - \mathbf{v}'_1(D) = \mathbf{v}''(D). \tag{3.29}$$

Notice that $\mathbf{v}''_S(D)$ is also an abstract state, and $\mathbf{v}''(D)$ is a codeword. Hence they should be

$$\mathbf{v}''_S(D) = \mathbf{u}''_S(D)PG'(D)Q \tag{3.30}$$

$$\mathbf{v}''(D) = \mathbf{u}''(D)G'(D)Q. \tag{3.31}$$

The input sequence $\mathbf{u}''_S(D)$ has the degree $\geq -m$, which is sufficient to produce an abstract state. The operator Q in (3.31) comes from (3.26), where we can also find that $\mathbf{u}''(D)$ must be polynomial. Furthermore, the combination of (3.29), (3.30), and (3.31) leads to

$$\mathbf{u}_S''(D)PG'(D)Q - \mathbf{u}''(D)G'(D)Q = [\mathbf{u}_S''(D)P - \mathbf{u}''(D)]G'(D)Q = 0. \tag{3.32}$$

Based on the first property, we will obtain

$$\mathbf{u}_S''(D)P - \mathbf{u}''(D) = 0, \tag{3.33}$$

and therefore $\mathbf{u}_S''(D)P = \mathbf{u}''(D) = 0$ since $\mathbf{u}''(D)$ is polynomial. As a result, $\mathbf{v}_{S1}'(D) = \mathbf{v}_{S2}'(D)$ and $\mathbf{v}_2'(D) = \mathbf{v}_1'(D)$, resulting in the uniqueness of (3.27) if $G'(D)$ is minimal basic. In other words, for any abstract state $\mathbf{v}_S'(D)$ of $G'(D)$, there exists an abstract state $\mathbf{v}_S(D)$ of $G(D)$ such that (3.27) holds; in addition, $\mathbf{v}_S'(D)$ can be a sum of an abstract state and a codeword of $G(D)$. Therefore, the map $\mathbf{v}_S(D) \mapsto \mathbf{v}'(D)$ is surjective, and the abstract state number of $G(D)$ must be larger than or equal to that of the equivalent minimal basic encoder $G'(D)$.

An encoder with the minimal abstract state number is of interest because the minimality also effects the complexity of encoders and decoders. The more general *minimal* encoder is then defined as follows:

Definition 3.5. [Johannesson [96]] An encoder $G(D)$ is minimal if its abstract state number is minimal over all equivalent encoders.

Clearly, a minimal basic encoder is a minimal encoder. Moreover, for an encoder $G(D)$ whose equivalent minimal basic encoder is $G_{mb}(D)$, the following statements are equivalent [96, 98]:

1. $G(D)$ is a minimal encoder.

2. The abstract state number of $G(D)$ equals to that of $G_{mb}(D)$.

3. For $G(D)$, only the abstract state of zero can be a codeword.

4. $G(D)$ has a polynomial right inverse in D and a polynomial right inverse in D^{-1}.

The second statement comes directly form the definition 3.5 and the second property of a minimal basic encoder. As mentioned above, the map $\mathbf{v}_S(D) \mapsto \mathbf{v}_{Smb}(D)$ from the abstract state of $G(D)$ to that of $G_{mb}(D)$ has been shown to be surjective. From the second statement, we can find the unique relation that

$$\mathbf{v}_{Smb}(D) = \mathbf{v}_S(D) + \mathbf{v}(D) \tag{3.34}$$

with $\mathbf{v}(D)$ being a codeword. With the similar derivation from (3.29) to (3.33), it can be correspondingly verified that only zero abstract state of $G(D)$ can be a codeword. Hence the map $\mathbf{v}_S(D) \mapsto \mathbf{v}_{Smb}(D)$ should be bijective from the second and the third statements. The equivalence about the fourth statement is detailed in [96]. Notice that the minimal encoder is defined to be the one that can be realized with a minimum number of memory units, and the realization may be in neither controller canonical nor observer canonical form. Therefore, as indicated in [96], some encoders are minimal, but are not minimal basic.

If we consider the 2×3 encoder over $GF(2)$

$$G(D) = \begin{bmatrix} 1+D+D^2 & D^2 & 1 \\ 1+D & 1 & D \end{bmatrix}, \tag{3.35}$$

the information stream

$$\mathbf{u}(D) = (1,0) + (1,1)D + (0,0)D^2 + (0,1)D^3 = (1+D, D+D^3) \tag{3.36}$$

can be encoded to be

$$\begin{aligned} \mathbf{v}(D) &= \mathbf{u}(D)G(D) \qquad (3.37) \\ &= (1+D^2+D^4, D+D^2, 1+D+D^2+D^4) \qquad (3.38) \end{aligned}$$

Furthermore, in Fig. 3.3, we illustrate the encoder (3.35) in controller canonical form with three delay elements. This is a rate $R = 2/3$ encoder with memory order $m = 2$, meaning that each codeword frame v_i will depend on u_i as well as two previous frames u_{i-1} and u_{i-2}. Therefore, we also refer Fig. 3.3 to be the $(3, 2, 2)$ convolutional encoder. With (3.7)∼(3.9),

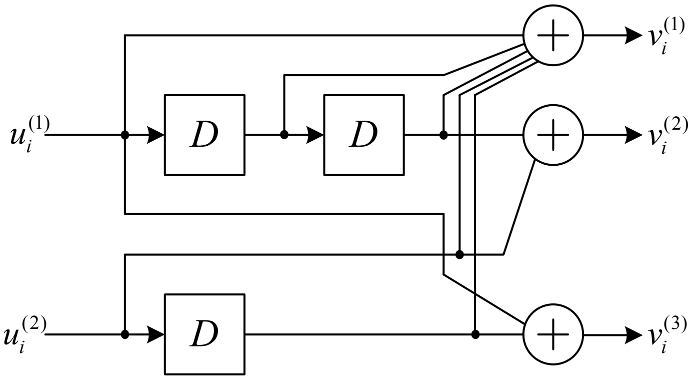


Figure 3.3: A $(3, 2)$ convolutional encoder with memory order $m = 2$

the constraint lengths of the encoder $G(D)$ in (3.35) are $\nu_1 = 2$, $\nu_2 = 1$, $\nu = 3$, $n_m = 9$, and $K = 5$. Moreover, we can easily check that $G(D)$ is a minimal basic encoder having eight (2^3) physical states or abstract states, and the realization in Fig. 3.3 is minimal.

Alternatively, the convolutional encoder with a polynomial generator matrix can be expressed by the analytical representation [94, 100]. We first denote the i-th information sequence with $u^{(i)} = (u_0^{(i)}, u_1^{(i)}, u_2^{(i)}, \ldots)$ and the j-th codeword sequence with $v^{(j)} = (v_0^{(j)}, v_1^{(j)}, v_2^{(j)}, \ldots)$. For $j = 1, 2, \ldots, n$, the encoding function will be written as the discrete convolution of $u^{(j)}$ and the generator sequence $g_i^{(j)}$; that is

$$v^{(j)} = \sum_{i=1}^{k} u^{(i)} * g_i^{(j)}, \tag{3.39}$$

and $g_i^{(j)} = (g_{i,0}^{(j)}, g_{i,1}^{(j)}, \ldots, g_{i,m}^{(j)})$. This convolution operation in (3.39) can also be in matrix multiplication form. The information stream should be $\mathbf{u} = (u_0, u_1, u_2, \ldots)$ with $u_i =$

$(u_i^{(1)}, u_i^{(2)}, \ldots, u_i^{(k)})$. On the other hand, the generator matrix $\bar{G}$ becomes semi-infinite:

$$\bar{G} = \begin{bmatrix} G_0 & G_1 & G_2 & \cdots & G_m & & & \\ & G_0 & G_1 & \cdots & G_{m-1} & G_m & & \\ & & G_0 & \cdots & G_{m-2} & G_{m-1} & G_m & \\ & & & \ddots & & & & \ddots \end{bmatrix}, \tag{3.40}$$

and the $k \times n$ sub-matrix is

$$G_x = \begin{bmatrix} g_{1,x}^{(1)} & g_{1,x}^{(2)} & \cdots & g_{1,x}^{(n)} \\ g_{2,x}^{(1)} & g_{2,x}^{(2)} & \cdots & g_{2,x}^{(n)} \\ \vdots & \vdots & \ddots & \vdots \\ g_{k,x}^{(1)} & g_{k,x}^{(2)} & \cdots & g_{k,x}^{(n)} \end{bmatrix}, \quad \text{for } x = 0, 1, \ldots, m\ . \tag{3.41}$$

Hence, the convolutional encoding of $\mathbf{u}$ is given by

$$\mathbf{v} = \mathbf{u} \cdot \bar{G} \tag{3.42}$$

Note that the codeword $\mathbf{v} = (v_0, v_1, v_2, \ldots)$ is a linear combination of the rows in $\bar{G}$. With the analytical representation, we can write the the following generator matrix for the information stream in 3.36 and the (3,2,2) encoder in Fig. 3.3:

$$\bar{G} = \begin{bmatrix} 101 & 100 & 110 & & & \\ 100 & 101 & 000 & & & \\ & 101 & 100 & 110 & & \\ & 100 & 101 & 000 & & \\ & & 101 & 100 & 110 & \\ & & 100 & 101 & 000 & \\ & & & 101 & 100 & 110 \\ & & & 100 & 101 & 000 \end{bmatrix}. \tag{3.43}$$

According to (3.42), the information $\mathbf{u} = (10, 11, 00, 01)$ will be encoded to be

$$\mathbf{v} = (10, 11, 00, 01) \cdot \bar{G} = (101, 011, 111, 000, 101, 000),$$

which is equivalent to (3.38).

Convolution encoders can be in either systematic form or non-systematic form. In a systematic encoder, the information sequence is a part of the codeword sequence; therefore, the generator matrix can be in the following form:

$$G_s(D) = [\ I(D) \quad G'(D)\ ]. \tag{3.44}$$

Notice that $I(D)$ is a $k \times k$ identity matrix, and $G'(D)$ is a $k \times (n-k)$ matrix

$$G'(D) = \begin{bmatrix} g_1^{(k+1)}(D) & g_1^{(k+2)}(D) & \cdots & g_1^{(n)}(D) \\ g_2^{(k+1)}(D) & g_2^{(k+2)}(D) & \cdots & g_2^{(n)}(D) \\ \vdots & \vdots & \ddots & \vdots \\ g_k^{(k+1)}(D) & g_k^{(k+2)}(D) & \cdots & g_k^{(n)}(D) \end{bmatrix}. \tag{3.45}$$

The encoder in (3.44) will generate the codeword frame where the first k symbols are information frame, and the remaining $(n-k)$ symbols are called the parity-check sequences. The encoders without systematic features are termed non-systematic. The systematic form of (3.44) has the following right inverse

$$G_s^{-1}(D) = \begin{bmatrix} I(D) \\ \bar{0} \end{bmatrix} \tag{3.46}$$

where $\bar{0}$ is an $(n-k) \times k$ zero matrix. We will observe that $G_s^{-1}(D)$ is polynomial in D and D^{-1}, and thus it is a minimal encoder. Therefore, all systematic encoders are minimal encoders [98].

Form [101], we know that every encoder is equivalent to a systematic rational encoder. Consequently, the systematic encoder $G_s(D)$ can be obtained from an equivalent non-systematic encoder $G(D)$. We first find a $k \times k$ submatrix $T(D)$ of $G(D)$ and compute

$$G_s(D) = T^{-1}(D)G(D). \tag{3.47}$$

For instance, the equivalent systematic encoder $G_s(D)$ of the encoder in (3.35) can be computed by letting

$$T(D) = \begin{bmatrix} 1+D+D^2 & D^2 \\ 1+D & 1 \end{bmatrix}. \tag{3.48}$$

As a result,

$$\begin{aligned} G_s(D) &= T^{-1}(D)G(D) \\ &= \frac{1}{1+D+D^3}\begin{bmatrix} 1 & D^2 \\ 1+D & 1+D+D^2 \end{bmatrix}\begin{bmatrix} 1+D+D^2 & D^2 & 1 \\ 1+D & 1 & D \end{bmatrix} \\ &= \begin{bmatrix} 1 & 0 & \frac{1+D^3}{1+D+D^3} \\ 0 & 1 & \frac{1+D^2+D^3}{1+D+D^3} \end{bmatrix} \end{aligned} \tag{3.49}$$

The corresponding observer canonical realization of (3.49) is illustrated in Fig. 3.4. The memory unit number is three which is equal to that of Fig. 3.3, resulting in the minimal realization of (3.49).

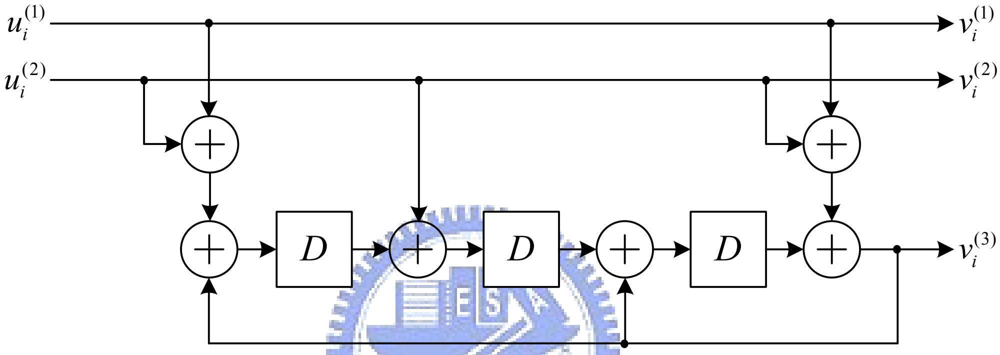


Figure 3.4: A (3, 2) systematic convolutional encoder in observer canonical form

There is a encoder $G(D)$ that generates finite weight codewords $v(D)$ for the infinite weight input sequence $\mathbf{u}(D)$ [98, 102]. Such encoder is referred to *catastrophic* because a finite number of errors in $\mathbf{v}(D)$ may cause an infinite number of errors in $\mathbf{u}(D)$ [97]. We consider the (2,1,2) encoder

$$G(D) = [\ 1+D \quad 1+D^2\ ],$$

and the infinite weight data stream

$$\mathbf{u}(D) = \frac{1}{1+D} = \sum_{i=0}^{\infty} D^i.$$

After encoding, the codeword sequence becomes $\mathbf{v}(D) = (1, 1 + D)$, which has a Hamming weight of 3. A small number of errors may cause the decoder to estimate an all zero codeword, leading to infinite decoding errors. According to [98], a non-catastrophic encoder should have a right pseudo-inverse $\tilde{G}^{-1}(D)$ without feedback, or the polynomial right inverse $\tilde{G}^{-1}(D)$ such that

$$G(D)\tilde{G}^{-1}(D) = D^l I(D) \tag{3.50}$$

for some $l \geq 0$, and $k \times k$ identity matrix $I(D)$. Consequently, all minimal encoders are non-catastrophic. The sufficient condition [102] for an $(n, 1)$ convolutional encoder

$$G(D) = \begin{bmatrix} g_1^{(1)}(D) & g_1^{(2)}(D) & \cdots & g_1^{(n)}(D) \end{bmatrix} \tag{3.51}$$

having right pseudo-inverse is the greatest common divisor (gcd) of the entries equals D^l:

$$\gcd[g_1^{(1)}(D), g_1^{(2)}(D), \cdots, g_1^{(n)}(D)] = D^l. \tag{3.52}$$

Moreover, for an (n, k) encoder in (3.3), the sufficient condition becomes

$$\gcd[\Delta_i(D), i = 1, 2, \cdots, \binom{n}{k}] = D^l, \tag{3.53}$$

where $\Delta_i(D)$ for $i = 1, 2, \cdots, \binom{n}{k}$ denote the determinants of distinct $k \times k$ submatrices of $G(D)$.

The behavior of an encoder can be described graphically. All possible physical states, or directly termed states, as well as the input data frame that may cause the state transitions can be represented by a *state diagram*. Accordingly, a state diagram consists of all states

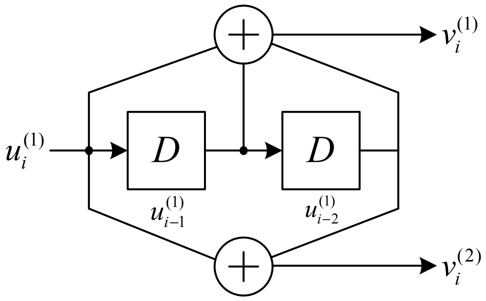


Figure 3.5: The (2,1,2) convolutional encoder

and branches that indicate transitions caused by the input data. Generally, each state has 2^k incoming branches and 2^k outgoing branches. We use the (2,1,2) encoder over $GF(2)$

$$G(D) = [\ 1 + D + D^2 \quad 1 + D^2\ ] \tag{3.54}$$

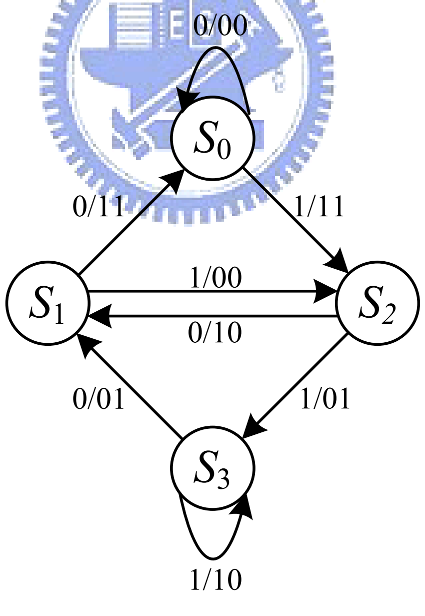


Figure 3.6: The state diagram of the (2,1,2) convolutional encoder

as an example. The controller canonical realization is illustrated in Fig. 3.5, and the corresponding state diagram is shown in Fig. 3.6. The state S_j is the memory unit contents $(u_{i-1}^{(1)}, u_{i-2}^{(1)})$ in Fig. 3.5. Since the overall constraint length $\nu = 2$ results in total $2^2 = 4$ states, we have $S_0 = (0,0)$, $S_1 = (0,1)$, $S_2 = (1,0)$, and $S_3 = (1,1)$. The branches labeled with

$u_i^{(1)}/v_i^{(1)}v_i^{(2)}$ show the state transitions according to the input $u_i^{(1)}$, and one codeword frame $(v_i^{(1)}, v_i^{(2)})$ will be generated within each state transition. Therefore, the encoding operation is a series of state transitions form a known state, for example S_0.

The other useful graphical representation is the *tree diagram* where states at different time instants will correspond to different nodes. Starting at a known root node (or state), the tree expands one of 2^k branch candidates based on the input data. The example in Fig. 3.6 is also illustrated in the tree diagram Fig. 3.7. Each node has two possible outgoing branches, the upper branch relates to the input $u_i^{(1)} = 0$, and the lower one to $u_i^{(1)} = 1$. The output codeword $v_i^{(1)}v_i^{(2)}$ is also marked on the branches. The tree expands from left to right as the input $u_i^{(1)}$ enters the encoder, and each path corresponds to a distinct input sequence. We can also find that though the possible node number increases while the tree expands, there are at most four states, $S_0 \sim S_3$, in the diagram.

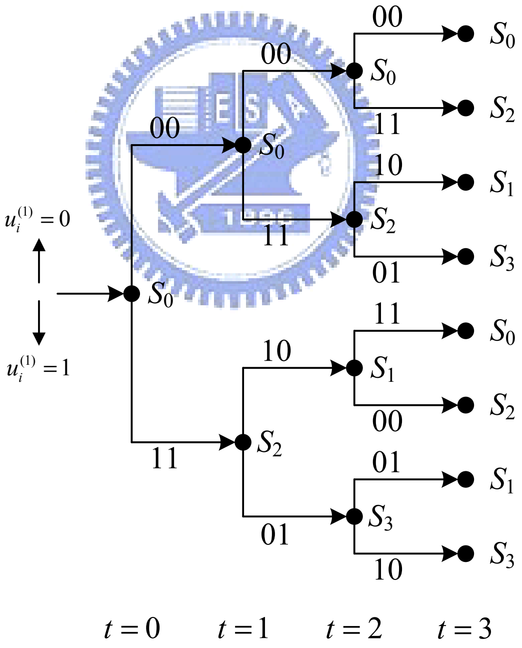


Figure 3.7: Tree diagram of the (2,1,2) convolutional encoder

The third graphical representation is the *trellis diagram*. We first note from the tree diagram in Fig. 3.7 that the four possible states appear at the time instant $t = 2$, and the eight nodes at $t = 3$ can be represented by the four states. Hence we can reduce the tree diagram to the trellis diagram with four states at each time instant, avoiding the exponentially increasing branches. For a minimal basic (n, k) encoder over $GF(2)$ with overall constraint length ν, there are 2^{ν} states at each time instant, and each state has 2^k incoming and 2^k outgoing branches. Fig. 3.8 shows the trellis diagram associated with the tree diagram in Fig. 3.7; the solid branches indicate the input $u_i^{(1)} = 1$, and the dash branches $u_i^{(1)} = 0$. The labels on the branches also reveal the input and the output $u_i^{(1)}/v_i^{(1)}v_i^{(2)}$.

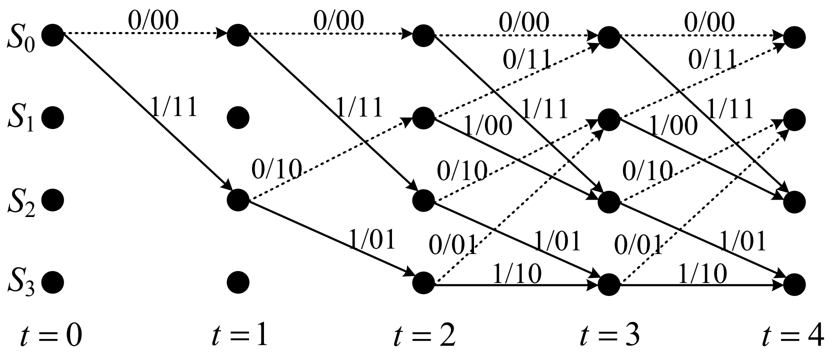


Figure 3.8: Trellis diagram of the (2,1,2) convolutional encoder

The distance property of the convolutional code can be described with weight enumerating function (WEF) [103], or the transfer function of a signal flow graph [94, 100]. If we assume the encoding starts from S_0 and terminates at S_0 with arbitrary lengths, the state diagram in Fig. 3.6 can be modified to be the signal flow graph in Fig. 3.9. To avoid ambiguity, S_0' denotes the terminated state S_0 at some time instant. We first consider the input-output weight enumerating function (IOWEF)

$$A(W, D, L) = \sum_{w,d,l} A_{w,d,l} W^w D^d L^l \tag{3.55}$$

where $A_{w,d,l}$ is the number of weight d codewords encoded from weight w information sequences whose length is l branches. Therefore, we label each branch in Fig. 3.9 with a gain $W^w D^d L^l$ according to the input and output weight information in Fig. 3.6. Note that Fig. 3.9 is a signal flow graph with source node S_0 and sink node S_0'. Since $A(W, D, L)$ can be regarded as the transfer function of the signal flow graph, (3.55) is obtained through

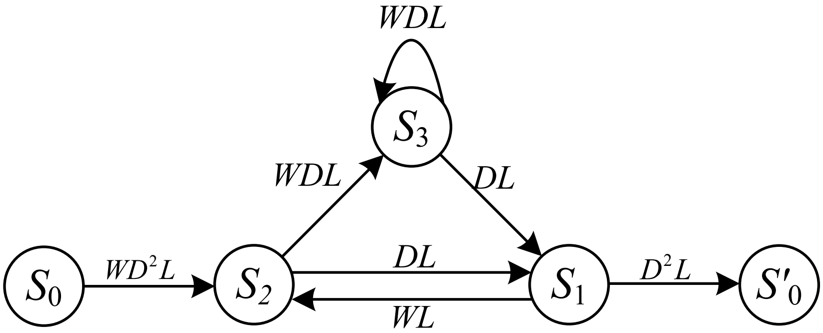


Figure 3.9: State diagram with weighted branches of the (2,1,2) convolutional encoder

solving the following equations:

$$\Psi_{S_2} = WD^2L \cdot \Psi_{S_0} + WL \cdot \Psi_{S_1} \tag{3.56}$$

$$\Psi_{S_3} = WDL \cdot \Psi_{S_2} + WDL \cdot \Psi_{S_3} \tag{3.57}$$

$$\Psi_{S_1} = DL \cdot \Psi_{S_2} + DL \cdot \Psi_{S_3} \tag{3.58}$$

$$\Psi_{S_0'} = D^2L \cdot \Psi_{S_1} \tag{3.59}$$

Finally, we achieve the rational function

$$A(W, D, L) = \frac{\Psi_{S_0'}}{\Psi_{S_0}} = \frac{WD^5L^3}{1 - WDL(1 + L)}. \tag{3.60}$$

Generally, the function (3.60) can be expanded to be

$$A(W, D, L) = WD^5L^3 \sum_{i=0}^{\infty} [WDL(1 + L)]^i \tag{3.61}$$

$$= WD^5L^3 + W^2D^6L^4(1 + L) + W^3D^7L^5(1 + L)^2 + \dots \tag{3.62}$$

For the encoder (3.54), we can learn from (3.61) that there is a weight five codeword associated with the input sequence of weight one and length three. A simple IOWEF will eliminate L in (3.55), and therefore

$$\begin{aligned} A(W,D) &= \sum_{w,d} A_{w,d} W^w D^d = A(W,D,L)|_{L=1} && (3.63) \\ &= \frac{WD^5}{1-2WD} && (3.64) \end{aligned}$$

The value $A_{w,d}$ counts the number of weight d codewords generated from weight w input sequences, and $A_{w,d} = \sum_l A_{w,d,l}$. Furthermore, the WEF is just

$$\begin{aligned} A(D) &= \sum_{d} A_d D^d = A(W,X,L)|_{W=L=1} && (3.65) \\ &= \frac{D^5}{1-2D} = D^5 + 2D^6 + 4D^7 + \ldots && (3.66) \end{aligned}$$

that enumerates the codewords of all possible weights, and $A_d = \sum_{w,l} A_{w,d,l}$. Form (3.66), the code contains a non-zero codeword with the minimum weight five, and thus the free distance (d_{free}) of the code is five.

3.2 Viterbi algorithm

The Viterbi algorithm, proposed in 1967 [15], is a maximum likelihood (ML) decoding technique for convolutional codes [104]. Assuming that the codeword $\mathbf{v}$ is transmitted through a discrete memoryless channel, the received sequence $\mathbf{r}$ is observed from the channel output. The ML decoder will find a codeword $\hat{\mathbf{v}}$, the estimation of $\mathbf{v}$, for which the *a posteriori* probability $P(\hat{\mathbf{v}}|\mathbf{r})$ is maximum. With Bayes' rule, the probability can be written as

$$P(\hat{\mathbf{v}}|\mathbf{r}) = \frac{P(\mathbf{r}|\hat{\mathbf{v}})P(\hat{\mathbf{v}})}{P(\mathbf{r})}, \qquad (3.67)$$

and the maximization of $P(\hat{\mathbf{v}}|\mathbf{r})$ is equivalent to maximizing $P(\mathbf{r}|\hat{\mathbf{v}})P(\hat{\mathbf{v}})$. Note that each $\hat{\mathbf{v}}$ corresponds to a distinct state sequence of length N denoted by $(x_0, x_1, \ldots, x_N)$, assuming

x_0 and x_N are known. The sequence, also a path in the trellis diagram, is a finite-state discrete-time Markov process [104]. Therefore, the probability of being in state x_{t+1} at time $t+1$, given all states up to time t, depends only on the state x_t at time t. That is

$$P(x_{t+1}|x_0, x_1, \ldots, x_t) = P(x_{t+1}|x_t), \tag{3.68}$$

and

$$\begin{aligned} P(\hat{\mathbf{v}}) &= P(x_0, x_1, x_2, \ldots, x_N) \\ &= \prod_{t=0}^{N-1} P(x_{t+1}|x_t, \ldots, x_0) \\ &= \prod_{t=0}^{N-1} P(x_{t+1}|x_t) \end{aligned} \tag{3.69}$$

For a discrete memoryless channel, we can also write

$$P(\mathbf{r}|\hat{\mathbf{v}}) = \prod_{t=0}^{N-1} P(r_t|\hat{v}_t), \tag{3.70}$$

and the decoder will maximize the probability

$$P(\mathbf{r}, \hat{\mathbf{v}}) = P(\mathbf{r}|\hat{\mathbf{v}})P(\hat{\mathbf{v}}) = \prod_{t=0}^{N-1} P(r_t|\hat{v}_t)P(x_{t+1}|x_t) \tag{3.71}$$

For convenience, we assign a metric

$$\Gamma = -\log P(\mathbf{r}, \hat{\mathbf{v}}) = \sum_{t=0}^{N-1} -\log P(r_t|\hat{v}_t) - \log P(x_{t+1}|x_t) \tag{3.72}$$

to the path; thus, the decoder need to find a path such that Γ is minimum. Note that $P(x_{t+1}|x_t)$ depends on the t-th encoder input u_t and is zero if there is no branch between x_{t+1} and x_t. If the data sequence $\mathbf{u}$ is an equally probable source, $P(x_{t+1}|x_t)$ is a constant equal to $1/2^k$ for a binary (n, k, m) convolutional code. Consequently, we can reduce the metric to

$$\Gamma = \sum_{t=0}^{N-1} - \log P(r_t|\hat{v}_t) \tag{3.73}$$

We first consider the path $(x_0, x_1, \ldots, x_t)$ terminating in x_t at time t and its path metric

$$\Gamma(x_t) = \sum_{i=0}^{t-1} - \log P(r_i|\hat{v}_i). \tag{3.74}$$

Although there are many possible paths that terminate in x_{t+1}, the one with the smallest path metric is of interest and is denoted by $\hat{\mathbf{x}}(x_{t+1})$, the *survivor* corresponding to the state x_{t+1}. The set of all physical states is defined to be $\mathbf{S}$. The Viterbi decoding algorithm is proceeded as follows:

- Initialization:

$$\begin{aligned}
&t = 0 \\
&\hat{\mathbf{x}}(x_0) = x_0, \qquad\qquad \Gamma(x_0) = 0 \\
&\hat{\mathbf{x}}(\chi) \text{ is arbitrary}, \quad \Gamma(\chi) = \infty \text{ for } \chi \in \mathbf{S} \text{ and } \chi \neq x_0
\end{aligned}$$

- Iterations until $t = N$:

  For each x_{t+1} in $\mathbf{S}$, we compute

$$\Gamma(x_{t+1}) = \min_{x_t}(\Gamma(x_t) + \gamma(x_{t+1}, x_t)), \tag{3.75}$$

  and

$$\gamma(x_{t+1}, x_t) = - \log P(r_t|\hat{v}(x_{t+1}, x_t)). \tag{3.76}$$

  Notice that $\hat{v}(x_{t+1}, x_t)$ is the codeword sequence that corresponds to the branch between x_{t+1} and x_t. Among the paths entering x_{t+1}, only the one with the minimum metric is stored to be $\hat{\mathbf{x}}(x_{t+1})$, and the others are discarded; moreover, the path metric

$\Gamma(x_{t+1})$ is saved for the next iteration. If $t = N$, the operation is completed; otherwise, t is increased by one to resume the next iteration.

Finally, we can obtain the survivor $\hat{\mathbf{x}}(x_N)$ as well as the estimated data sequence $\hat{\mathbf{u}}$ on the survivor. Some decoding examples can be found in [104], [94], and [105]. The term $\gamma(x_{t+1}, x_t)$ in (3.76) is called branch metric. If the n-tuple codeword is

$$\hat{v}_t = \hat{v}(x_{t+1}, x_t) = (\hat{v}_t^{(1)}, \hat{v}_t^{(2)}, \ldots, \hat{v}_t^{(n)}),$$

and the received sequence is also n-tuple

$$r_t = (r_t^{(1)}, r_t^{(2)}, \ldots, r_t^{(n)}),$$

we can rewrite the branch metric as

$$\gamma(x_{t+1}, x_t) = -\sum_{i=1}^{n} \log P(r_t^{(i)} | \hat{v}_t^{(i)}). \tag{3.77}$$

For a code over $GF(2)$ and a binary symmetric channel (BSC) with transition probability $p < 0.5$, the branch metric will be

$$\gamma(x_{t+1}, x_t) = d(r_t, \hat{v}_t) \log \frac{1-p}{p} + n \log \frac{1}{1-p}, \tag{3.78}$$

where $d(r_t, \hat{v}_t)$ is the Hamming distance between r_t and $\hat{v}_t$. Additionally, since $n \log \frac{1}{1-p}$ is constant and $\log \frac{1-p}{p} > 0$, the branch metric in (3.78) can be reduced to

$$\gamma(x_{t+1}, x_t) = d(r_t, \hat{v}_t) \tag{3.79}$$

without any effect on finding the least metric path in (3.75). On the other hand, if the code is transmitted over an AWGN channel with BPSK signals, the probability is expressed

$$P(r_t^{(i)}|\hat{v}_t^{(i)}) = \frac{1}{\sqrt{2\pi\sigma^2}}\, e^{-\frac{(r_t^{(i)}-\hat{v}_t^{(i)})^2}{2\sigma^2}}. \tag{3.80}$$

Notice that $\hat{v}_t^{(i)}$ has been mapped with $0 \mapsto -1$ and $1 \mapsto +1$, and $2\sigma^2 = N_0/E_s$ in which N_0 is the one sided power spectra density of noise, and E_s is the energy per signal. Moreover, E_s/N_0 is often termed the signal to noise ratio (SNR). As a result, the branch metric becomes

$$\gamma(x_{t+1}, x_t) = \frac{n}{2}\ln(2\pi\sigma^2) + \frac{1}{2\sigma^2}\sum_{i=1}^{n}(r_t^{(i)} - \hat{v}_t^{(i)})^2 \tag{3.81}$$

that can also be simplified to

$$\gamma(x_{t+1}, x_t) = \sum_{i=1}^{n}(r_t^{(i)} - \hat{v}_t^{(i)})^2 = \sum_{i=1}^{n}[(r_t^{(i)})^2 - 2r_t^{(i)}\hat{v}_t^{(i)} + (\hat{v}_t^{(i)})^2] \tag{3.82}$$

for n and σ^2 are constant. Notice that $\sum(r_t^{(i)})^2$ is the same for all survivors, and $(\hat{v}_t^{(i)})^2$ is constant in BPSK modulation. Therefore, the metric is further reduced to

$$\gamma(x_{t+1}, x_t) = -\sum_{i=1}^{n} r_t^{(i)}\hat{v}_t^{(i)} \tag{3.83}$$

which is the negative inner product between the received r_t and the codeword $\hat{v}_t$.

Based on the Viterbi decoding algorithm, the decoding error probability can be evaluated [94,100,106]. We first assume an all zero data sequence $\mathbf{u}$ over $GF(2)$ is encoded ($\mathbf{v} = 0$) and transmitted through a binary symmetric channel. Any 1s in the decoded sequence $\hat{\mathbf{u}}$ are decoding errors. In the trellis diagram Fig. 3.8, for instance, the correct state sequences are all S_0. If some errors occur, the decoder will trace the path that diverges from the correct one. We consider the first event error that an incorrect path first diverges from the correct path at time t and remerges to it after some time instants. Assuming the incorrect path has codewords of weight d, the first event error probability is

$$P_d = \begin{cases} \displaystyle\sum_{e=(d+1)/2}^{d} \binom{d}{e} p^e (1-p)^{d-e}, & \text{for odd } d \\ \displaystyle\frac{1}{2}\binom{d}{d/2} p^{d/2}(1-p)^{d/2} + \sum_{e=d/2+1}^{d} \binom{d}{e} p^e (1-p)^{d-e}, & \text{for even } d. \end{cases} \tag{3.84}$$

Based on (3.65), the first error event probability caused by all incorrect paths at time t is overbounded by

$$P_f(E) < \sum_{d=d_{free}}^{\infty} A_d P_d. \tag{3.85}$$

Notice that the error event probabilities at any time instants must be (3.85) because of the independence of t. There may be many error events after the first error event. As shown in Fig. 3.10, the first error event cause the decoded path to be $\mathbf{v}_1$ instead of the correct $\mathbf{v}$ at time t_1. Moreover, the decoder eliminates v_1 at time t_2 due to the second error event, the the survivor becomes v_2. As a result, we have the following path metrics for $\mathbf{v}$, $\mathbf{v}_2$, and $\mathbf{v}_2$ at time t_2:

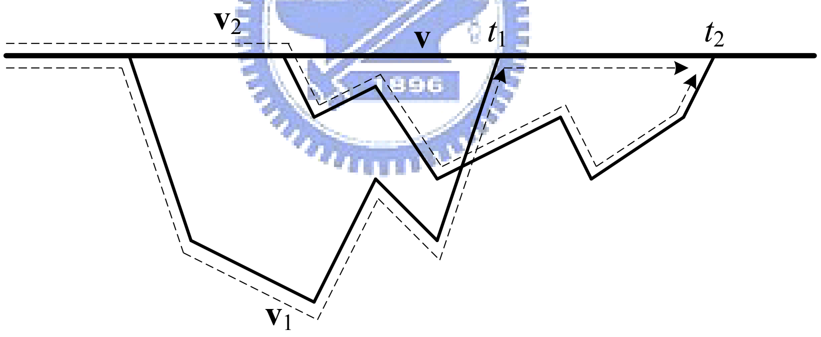


Figure 3.10: Illustration of error events in the trellis diagram

$$\Gamma(\mathbf{v}) \geq \Gamma(\mathbf{v_1}) \geq \Gamma(\mathbf{v_2}) \tag{3.86}$$

We can find that if the path selection at time t_2 is between $\mathbf{v}$ and $\mathbf{v_2}$, the survivor will also be $\mathbf{v_2}$. Hence the error event probability is still bounded by (3.85), and we can conclude that the error event probability at any time instant is

$$P(E) < \sum_{d=d_{free}}^{\infty} A_d P_d. \tag{3.87}$$

The probability P_d in (3.84) can be upper bounded by

$$\begin{aligned} P_d &< \binom{d}{\lceil d/2 \rceil} p^{d/2}(1-p)^{d/2} + \sum_{e=\lceil d/2 \rceil+1}^{d} \binom{d}{e} p^e (1-p)^{d-e} \\ &< \binom{d}{\lceil d/2 \rceil} p^{d/2}(1-p)^{d/2} + \sum_{e=\lceil d/2 \rceil+1}^{d} \binom{d}{e} p^{d/2} (1-p)^{d/2} \\ &< (p^{d/2}(1-p)^{d/2}) \sum_{e=0}^{d} \binom{d}{e} \\ &= 2^d p^{d/2}(1-p)^{d/2} \end{aligned} \tag{3.88}$$

Consequently, we can upper bound $P(E)$ in (3.87) with the WEF in (3.65); that is

$$P(E) < \sum_{d=d_{free}}^{\infty} A_d (2\sqrt{p(1-p)})^d = A(D)|_{D=2\sqrt{p(1-p)}}. \tag{3.89}$$

If p is small, the small degree terms will dominate the bound, and we can approximate (3.89) as

$$P(E) \approx A_{d_{free}} (2\sqrt{p(1-p)})^{d_{free}} = A_{d_{free}} (2\sqrt{p})^{d_{free}} \tag{3.90}$$

Furthermore, the bit error probability $P_b(E)$ for the source sequence $\mathbf{u}$ can be upper bounded by

$$P_b(E) < \frac{1}{k}(wA_{w,d})P_d, \tag{3.91}$$

where w and $A_{w,d}$ are defined in IOWEF (3.55), and k is the information bit number per branch; thus, $wA_{w,d}$ is the total number of non-zero information bits on all weight d paths.

Similarly, based on (3.88) and (3.55), we can further bound $P_b(E)$ as

$$P_b(E) < \frac{1}{k}\frac{\partial A(W,D)}{\partial W}|_{D=2\sqrt{p(1-p)},W=1} \tag{3.92}$$

In the AWGN channel with binary inputs and continuous outputs, the error probability can be derived similarly according to the above mentioned approach [100,106]. The all zero sequence is assumed to be transmitted with BPSK modulation, where 1 is mapped to $+1$, and 0 to -1. The correct path $\mathbf{v}$ is a codeword of all $-1s$. As shown in Fig. 3.10, if the error event $\mathbf{v}_1$ containing d +1 codeword symbols merges $\mathbf{v}$ at time t_1, the path metric of $\mathbf{v}_1$ must be smaller, and therefore

$$\sum_{e=1}^{d}(r^{(e)} - (-1))^2 \geq \sum_{e=1}^{d}(r^{(e)} - (+1))^2, \tag{3.93}$$

where $r^{(e)}$ denote the received symbols corresponding to which $\mathbf{v}_1$ has $+1$ codeword symbols. Moreover, we can write

$$\sum_{e=1}^{d}[(r^{(e)} - (-1))^2 - (r^{(e)} - (+1))^2] = 4\sum_{e=1}^{d} r^{(e)} \geq 0, \tag{3.94}$$

and the event error probability becomes

$$P_d = Pr\{\xi = \sum_{e=1}^{d} r^{(e)} \geq 0\}. \tag{3.95}$$

We further note that $r^{(e)}$ are independent Gaussian random variables with mean -1 and variance $\sigma^2 = N_0/2E_s$; as a result, ξ is also Gaussian with mean $-d$ and variance $d\sigma^2$.

Hence, the probability in (3.95) will be

$$\begin{aligned} P_d &= \int_0^{\infty} \frac{1}{\sqrt{2\pi d\sigma^2}}\, e^{-\frac{(\xi-(-d))^2}{2d\sigma^2}} d\xi \\ &= \int_{d/\sqrt{d\sigma^2}}^{\infty} \frac{1}{\sqrt{2\pi}}\, e^{-\frac{x^2}{2}} dx = Q(\sqrt{\frac{d}{\sigma^2}}) = Q(\sqrt{\frac{2dE_s}{N_0}}). \end{aligned} \tag{3.96}$$

The Gaussian error integral

$$Q(x) \triangleq \frac{1}{\sqrt{2\pi}} \int_x^{\infty} e^{-\frac{x^2}{2}} dx = \frac{1}{2}\mathrm{erfc}(\frac{x}{\sqrt{2}}), \tag{3.97}$$

and the complementary error function (erfc) is defined in [107]. According to (3.87), the event error probability for the AWGN channel can be represented by

$$P(E) < \sum_{d=d_{free}}^{\infty} A_d Q(\sqrt{\frac{2dE_s}{N_0}}). \tag{3.98}$$

With the following bound [108, 109],

$$Q(x) \leq \frac{1}{2} e^{-\frac{x^2}{2}} < e^{-\frac{x^2}{2}}, \tag{3.99}$$

we will have

$$P(E) < A(D)|_{D=e^{-E_s/N_0}} \tag{3.100}$$

as well as the bit error probability

$$P_b(E) < \frac{1}{k}\frac{\partial A(W,D)}{\partial W}|_{D=e^{-E_s/N_0}, W=1} \tag{3.101}$$

The upper bounds of (3.100) and (3.101) are derived from the weaker bound in (3.99). The tighter versions can be found in [106] and [100]. Moreover, the more accurate approximations for $Q(x)$ are discussed in [109], [110], and [111].

The same ensemble average error bound for time-varying convolutional codes is shown in theorem 3.1, assuming the maximum likelihood decoding. The time-varying convolutional codes are counterparts of fixed, or time-invariant, convolutional codes in which the generator polynomials are invariant over different time instants. Consequently, in time-varying convolutional codes, the generator matrix (see (3.40)) may have different sum-matrices G_x at distinct rows, leading to the following encoder:

$$\bar{G} = \begin{bmatrix} G_0^{(0)} & G_1^{(0)} & G_2^{(0)} & \cdots & G_m^{(0)} & & & \\ & G_0^{(1)} & G_1^{(1)} & \cdots & G_{m-1}^{(1)} & G_m^{(1)} & & \\ & & G_0^{(2)} & \cdots & G_{m-2}^{(2)} & G_{m-1}^{(2)} & G_m^{(2)} & \\ & & & \ddots & & & & \ddots \end{bmatrix}. \tag{3.102}$$

The sub-matrices $G_x^{(t)}$ and $G_x^{(t')}$ for $t \neq t', x = 0 \sim m$ may be unequal. The convolutional channel coding theorem for binary codes is described as follows [100, 106].

Theorem 3.1 (Viterbi [100])**.** *For any discrete input memoryless channel with capacity C, there exists a time-varying convolutional code of constraint length K, rate k/n bits per channel symbol, and arbitrary block length, whose bit error probability P_b, resulting from maximum likelihood decoding, is bounded by*

$$P_b < (2^k - 1)\frac{2^{-KkE_c(R)/R}}{(1 - 2^{-\epsilon kE_c(R)/R})^2}, \quad \textit{for small } \epsilon > 0 \tag{3.103}$$

where

$$E_c(R) = R_0 = \max_{\mathbf{p}} E_0(1, \mathbf{p}) \quad \textit{for } 0 \leq R \leq R_0(1 - \epsilon) \tag{3.104}$$

and

$$E_c(R) = \max_{\mathbf{p}} E_0(\rho, \mathbf{p}) \quad 0 \leq \rho \leq 1 \tag{3.105}$$

for $R = (1 - \epsilon)\max_{\mathbf{p}} \frac{E_0(\rho, \mathbf{p})}{\rho}$, and $R_0(1 - \epsilon) \leq R \leq C(1 - \epsilon)$.

The Gallager function [112, 113] is defined as follows:

$$E_0(\rho, \mathbf{p}) \triangleq -\ln \sum_y \left[\sum_x p(x) p(y|x)^{\frac{1}{1+\rho}} \right]^{1+\rho}. \tag{3.106}$$

For the set of all possible channel input alphabets $\mathbf{X}$, the arbitrary set $\mathbf{p} = \{p(x)|x \in \mathbf{X}\}$ satisfies $p(x) \geq 0, \forall x \in \mathbf{X}$ and $\sum_x p(x) = 1$. The transition probability $p(y|x)$ for $y \in \mathbf{Y}$ and $x \in \mathbf{X}$ indicates a discrete memoryless channel, and $\mathbf{Y}$ denotes the channel output alphabets. The code rate R in theorem 3.1 is nats per channel symbol; that is, $R = k \ln 2/n$.

3.2.1 Path truncation

As was indicated in the Viterbi decoding algorithm, the paths, or survivors, terminated at each state should be stored up to the last received codeword, meaning that the entire received sequence are analyzed before any decoding output. In real applications, the information sequence length N may be very large that cause massive storage requirement. Due to the practical storage constraint, the survivor for each state should be truncated to a finite length as shown in Fig. 3.11. The corresponding trellis diagram with state number $M = 2^\nu$ is truncated to finite time instants T, and there are M paths terminating at time $t + T$. With the truncation length of T, the decoder is required to output data on the branch at depth t according to the path metrics at time $t + T$ [100, 114]. If all surviving paths have a

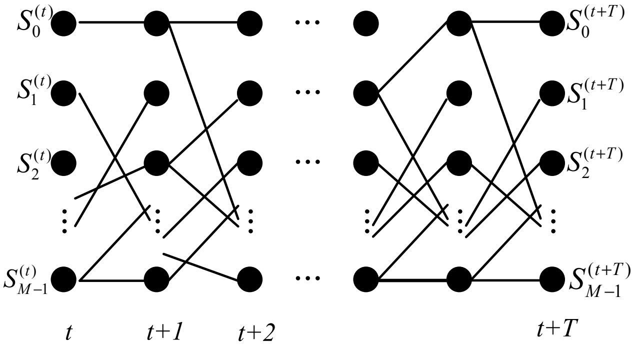


Figure 3.11: Trellis diagram truncated to T instants

common node at time t, the unique branch is chosen. Otherwise the branch corresponding to the best metric value at time $t+T$ will be selected. This truncation technique can result in an additional error if an incorrect path diverges from the correct path at depth t, and remains unmerged from it before time $t+T$. Therefore, T must be larger enough such that the truncation error is comparable to or less than the maximum-likelihood decoding [115].

We also assume an all zero information sequence over $GF(2)$ is encoded and transmitted through a memoryless channel. In the truncated trellis diagram of length T, the truncation error will be caused by the incorrect paths that diverge from the correct path before time t and extend to $S_i \neq S_0$ at time $t+T$ without going through S_0. Therefore, the WEF for those incorrect paths connecting S_0 and S_i can be defined as

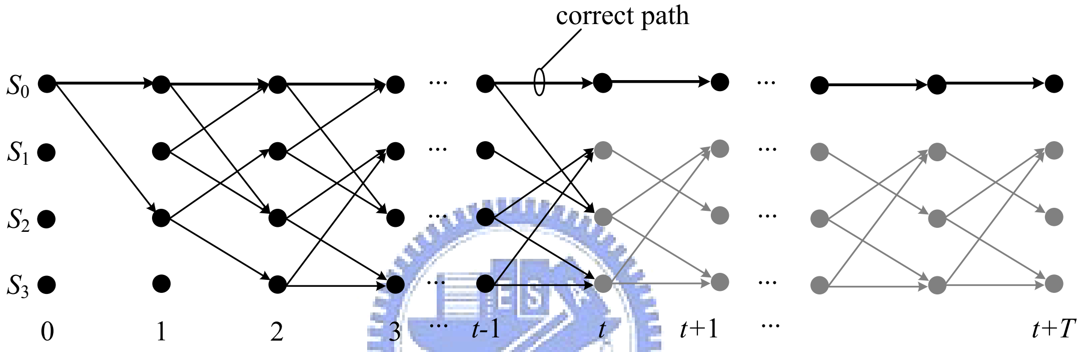


Figure 3.12: Incorrect paths with lengths larger than T

$$A^T_{S_0,S_i}(D,L) = \sum_{d,l>T} A_{d,l,i} D^d L^l \tag{3.107}$$

that enumerates the weights for all the paths between S_0 and S_i of lengths at least $T+1$ [115]. Notice that (3.107) can be derived from the transfer function of S_0 and S_i in the signal flow graph. For example, considering the state diagram in Fig. 3.9, we first obtain

$$\begin{aligned} A_{S_0,S_1}(D,L) &= \frac{\Psi_{S_1}}{\Psi_{S_0}}|_{W=1} = \frac{D^3L^2}{1-DL(1+L)} \\ &= D^3L^2 + D^4L^3(1+L) + D^5L^4(1+L)^2 + \cdots, \end{aligned} \tag{3.108}$$

and $A^T_{S_0,S_1}(D,L)$ will be derived by discarding the terms with the degrees of L being less than or equal to T in (3.108). As a result, the error probability arising from truncation should be

$$P(T) < \sum_{i=1}^{M-1} A^T_{S_0,S_i}(D,L)|_{D=D_0,L=1}, \tag{3.109}$$

where $D_0 = 2\sqrt{p(1-p)}$ for the BSC, and $D_0 = e^{-E_s/N_0}$ for the AWGN channel. Note that $P(T)$ is the probability that the decoder makes an error decoding at time t. From (3.109) and (3.101), the overall bit error probability including truncation error is upper bounded by

$$P_b(E,T) < \frac{1}{k}\frac{\partial A(W,D)}{\partial W} + \sum_{i=1}^{M-1} A^T_{S_0,S_i}(D,L)|_{D=D_0,W=1,L=1}. \tag{3.110}$$

The determination of T should make the second term in (3.110) become eliminable. For small channel noise (small p or large E_s/N_0), the upper bound of bit error probability is dominated by the least degree term according to (3.90). In (3.110), let d_T denote the smallest degree of D in the second term, the minimum T (T_{min})is required to achieve $d_T > d_{free}$ because the smallest degree is d_{free} for the first term. Continuing the example in Fig. 3.9, we can generate (3.107) for $S_i = S_1$, S_2, and S_3. Since $d_{free} = 5$, the value d_T should be at least six, and therefore T is larger than seven ($T_{min} = 7$).

Alternatively, in [100], the ensemble average probability $\bar{P}(T)$ corresponds to an error path which is unmerged from the correct path for exactly T time instants, and there are no more than 2^{kT} such error paths for a binary code. As a result, the error probability can be bounded by

$$\begin{aligned} \overline{P(T)} &< 2^{kT\rho}e^{-TnE_0(\rho,\mathbf{p})} \\ &= 2^{-kT(E_0(\rho,\mathbf{p})-\rho R))/R} \quad \text{for } 0 < \rho \leq 1 \\ &= 2^{-kTE_b(R)/R} \end{aligned} \tag{3.111}$$

where R is in nats per symbol, and

$$E_b(R) = \max_{\rho,\mathbf{p}}[E_0(\rho, \mathbf{p}) - \rho R] \tag{3.112}$$

according to [112]. Furthermore, we can deduce that

$$E_b(R) = \max_{\mathbf{p}} E_0(1, \mathbf{p}) - R \quad \text{for } 0 \leq R < \left(\max_{\mathbf{p}} \frac{\partial E(\rho, \mathbf{p})}{\partial \rho}\bigg|_{\rho=1} \right), \tag{3.113}$$

and

$$E_b(R) = \max_{\mathbf{p}}[E_0(\rho, \mathbf{p}) - \rho \frac{\partial E_0(\rho, \mathbf{p})}{\partial \rho}], \quad \text{for } 0 < \rho \leq 1 \tag{3.114}$$

when

$$R = \max_{\mathbf{p}} \frac{\partial E_0(\rho, \mathbf{p})}{\partial \rho}, \quad \text{and} \quad \left(\max_{\mathbf{p}} \frac{\partial E(\rho, \mathbf{p})}{\partial \rho}\bigg|_{\rho=1} \right) \leq R < C.$$

Note that C is the channel capacity [1, 113]. As a result, from the exponents of (3.103) and (3.111), the criterion of truncation length T selection is shown to be $TE_b(R) > KE_c(R)$, or

$$\frac{T}{K} > \frac{E_c(R)}{E(R)}, \tag{3.115}$$

where K is the input constraint length in (3.9), and $R = k \ln 2/n$ is the code rate.

The more intuitive formulation of (3.115) can be derived with the very noisy channel, which is assumed to be discrete and memoryless [100]. The conditional probability of receiving $y \in \mathbf{Y}$, given that $x \in \mathbf{X}$ is transmitted, is expressed by

$$p(y|x) = p(y)(1 + \epsilon_{xy}), \tag{3.116}$$

where $|\epsilon_{xy}| \ll 1$ for all x and y, and $\sum_{y \in \mathbf{Y}} p(y)\varepsilon_{xy} = 0$ for all x. The condition in (3.115) can then be reduced to

$$\frac{T}{K} \geq \begin{cases} \frac{1}{1-2R/C} & 0 \leq R < \frac{C}{4} \\ \frac{1}{2(1-\sqrt{R/C})^2} & \frac{C}{4} \leq R \leq \frac{C}{2} \\ \frac{1+\sqrt{R/C}}{1-\sqrt{R/C}} & \frac{C}{2} < R < C. \end{cases} \tag{3.117}$$

For different coding rate R, the truncation length ranges from K to infinite as R approaches channel capacity C. Of different R/C ratios, Fig. 3.13 shows the valid truncation length in gray region. In real applications, truncation length T must vary with respect to R and C.

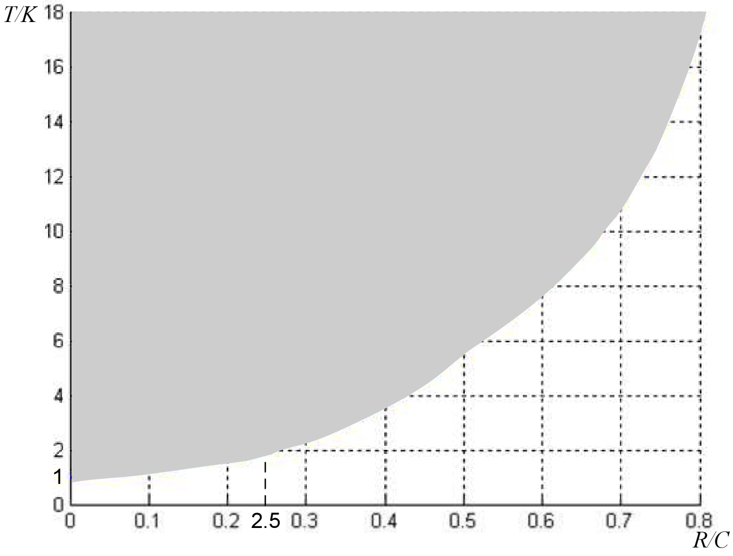


Figure 3.13: Truncation length versus coding rate.

From Fig. 3.13, we can find that the truncation length increases with the ratio R/C. This trend also appears in the results of (3.110). For larger channel noise, the degree d_T must be enlarged to reduce the truncation error, and we have to increase T_min accordingly. The T_{min} is based on the best path metric decoding. The research in [116] further indicates that the truncation length about $2T_{min}$ can lead to negligible truncation error if the decoder select any survivor at time $t+2T_{min}$ without considering the path metrics. This approach, termed *fixed state decoding*, eliminates the search for the best path metric among all survivors, however, more storage is required.

3.2.2 Punctured convolutional codes

The rate k/n code can be encoded by a $k \times n$ generator matrix; alternatively, it can also be obtained by periodically deleting some codeword symbols form rate $1/n$ codewords [117]. The complexity of Viterbi decoding algorithm is proportional to the number of branch entering to each state, and the branch number increases exponentially with k. In a punctured convolutional code, the decoder complexity is comparable with the original rate $1/n$ code, referred to the *mother code*. A single mother code can be used to generated different rates through puncturing. We define an $n \times p_t$ puncturing matrix $\mathbf{P}$ with elements in $GF(2)$, and the zero elements in $\mathbf{P}$ will indicate which codeword symbols within the puncturing period p_t should be deleted. For example, in the matrix

$$\mathbf{P} = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \tag{3.118}$$

for a rate 1/2 mother code whose codeword sequence is represented by

$$\mathbf{u}(D) = \sum_{i=0}^{\infty} \left(\begin{bmatrix} u_{2i}^{(1)} \\ u_{2i}^{(2)} \end{bmatrix} D^{2i} + \begin{bmatrix} u_{2i+1}^{(1)} \\ u_{2i+1}^{(2)} \end{bmatrix} D^{2i+1} \right), \tag{3.119}$$

the codeword symbol $u_{2i+1}^{(1)}$ is deleted before transmission, resulting in the rate 2/3 punctured code. More puncturing matrices for rate 1/2 codes can be found in [118].

The Viterbi decoding algorithm for punctured codes operates quite similar to that for decoding the rate $1/n$ mother code, except that the removed symbols are not considered in the branch metric calculation. Hence we can use one decoder to decode various punctured codes from the same mother code.

The rate $(n-1)/n$ punctured encoder can also be characterized with the equivalent ordinary $(n-1) \times n$ encoder without puncturing [119]. Therefore, it is possible to discuss the minimal and the catastrophic properties of a punctured encoder [120, 121]

3.3 Viterbi decoder architecture

The Viterbi decoder consists of four main units [122]: branch metric unit (BMU), add-compare-select unit (ACSU), path metric (PM) memory, and survivor memory unit (SMU). As illustrated in Fig. 3.14, BMU calculates the branch metrics from the input data based on either (3.79) or (3.82). The ACSU recursively accumulates branch metrics (BM) as path metrics stored in the PM memory, and makes decisions to select the most likely state sequence. The add-compare-select (ACS) operation is formulated in (3.75). Finally, the SMU traces the decisions to extract this sequence in the survivor memory that keeps all survivors terminating at each state.

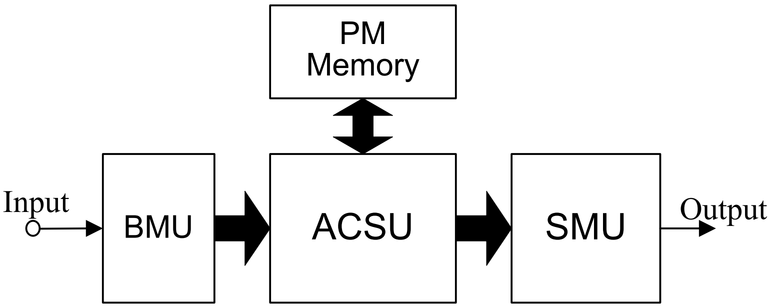


Figure 3.14: Block diagram of Viterbi decoder

The nonlinear and recursive nature of ACSU limits the maximum achievable throughput rate. Furthermore, as the overall constraint length ν rises, the large number (2^{ν}) of ACS operations are required to determine 2^{ν} survivors. The hardware complexity increases exponentially, and so does the power consumption, leading to many researches on the the optimization for ACSU.

On the other hand, the SMU is also an area and power consuming blocks in Viterbi decoders. There are two mainly solutions for the SMU: the register-exchange and the memory traceback architectures [123–126]. As compared with the register-exchange approach, the traceback based SMU has a limited memory bandwidth in nature, and thus limits the decoding speed. However, the traceback approach with memory is more area efficient for large constraint lengths; it is also considerably more power efficient without data movement in the memory.

Each component in the Viterbi decoder of Fig. 3.14 will be addressed in the following sections.

3.3.1 Branch metric unit

The BMU evaluates the metric, or distance, between the received samples and the codewords on branches. In the *hard decision* decoding scheme, the channel is assumed to be BSC where the received signals have been decided to be the alphabets during transmission. In binary cases, the received symbols are either one or zero. Therefore, the branch metric (BM) is the Hamming distance between the received data and the codewords as expressed in (3.78).

Alternatively, the *soft decision* decoding scheme can be applied to improve the decoding performance [94, 127]. The BM evaluation is shown in (3.82). Due to the finite precision in practical implementation, it is necessary to quantize the channel symbols, but this will cause additional quantization noise. Such noise may increase the required SNR (E_s/N_0) to achieve a specific bit error rate (BER). The hardware complexity increases linearly with the quantization bit number in the demodulated symbols; therefore, the objective is to find the sufficient quantization levels that minimize the effect of quantization loss on the decoding performance.

For a ϱ bits quantizer, we consider the uniform quantization because nonuniform ones can achieve only slight improvement when $\varrho \geq 3$ [128]. The stepsize Δ is also defined to be the spacing between any two quantized values. For the received BPSK signal $r_t^{(i)}$, Fig. 3.15 illustrates a $\rho = 3$ example, and $r_{t,q}^{(i)}$ is the quantization results between $-2^{\rho-1}$ and $2^{\rho-1}-1$. We can write the quantization function $\Phi(r_t^{(i)})$ as

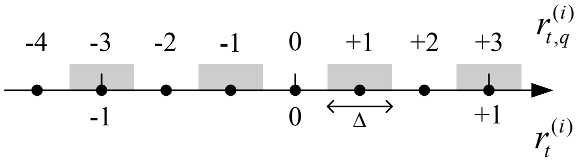
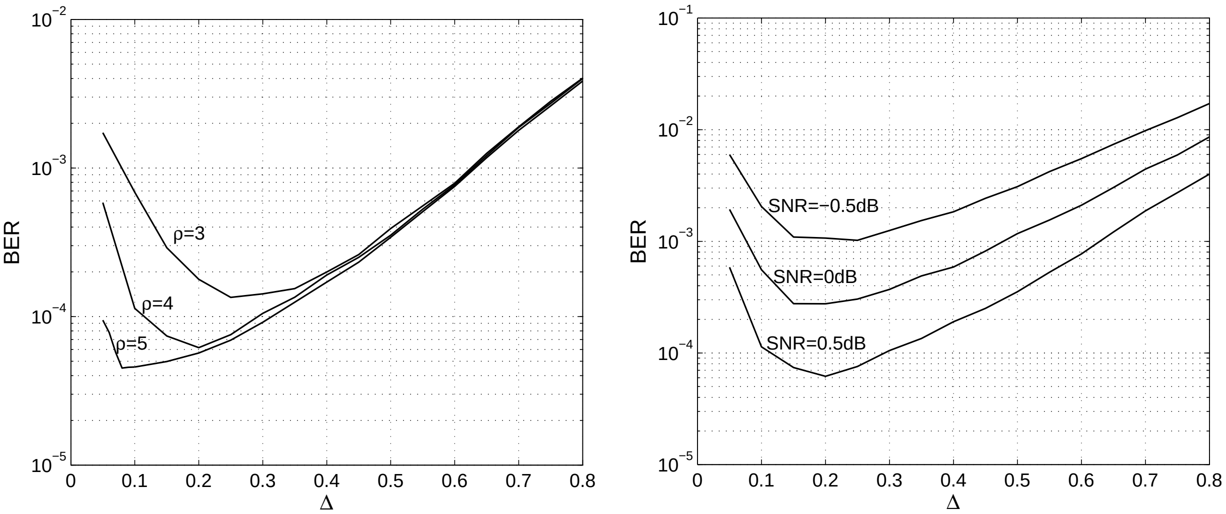


Figure 3.15: Block diagram of Viterbi decoder

$$r_{t,q}^{(i)} = \Phi(r_t^{(i)}) = \begin{cases} 2^{\rho-1}-1 & \frac{r_t^{(i)}}{\Delta} \geq (2^{\rho-1}-1.5) \\ \lfloor \frac{r_t^{(i)}}{\Delta} + 0.5 \rfloor & (-2^{\rho-1}-0.5) < \frac{r_t^{(i)}}{\Delta} < (2^{\rho-1}-1.5) \\ -2^{\rho-1} & \frac{r_t^{(i)}}{\Delta} \leq (-2^{\rho-1}-0.5). \end{cases} \quad (3.120)$$

The quantization bits ρ and the stepsize Δ vary with modulation types and channel conditions. Moreover, for each ρ, there is a optimal Δ that minimize the BER. We consider the (2,1,6) convolutional encoder



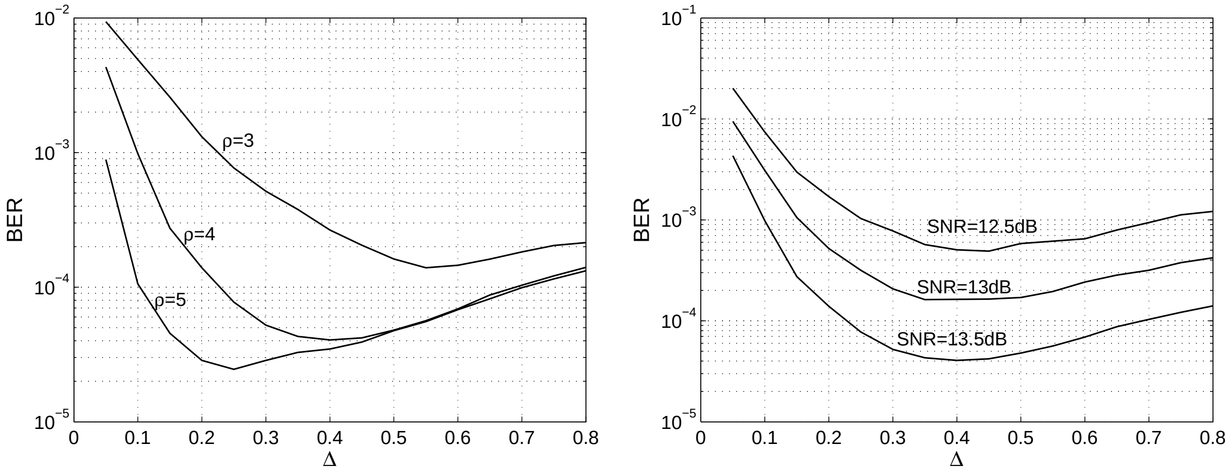
(a) SNR=0.5dB

(b) $\rho = 4$

Figure 3.16: Different quantization schemes in BPSK modulation and their BER performance

$$G(D) = [\ 1 + D^2 + D^3 + D^5 + D^6 \quad 1 + D + D^2 + D^3 + D^6\ ] \quad (3.121)$$

in the IEEE 802.11a wireless LAN (WLAN) system [129]. With the BPSK modulation, the performance figures in AWGN channel are shown in Fig. 3.16, where the input codewords are quantized to be different bit number and step sizes. In Fig. 3.16(b) with the fixed SNR=0.5dB, the step size Δ significantly affects the bit error rate, and the performance degrades rapidly for smaller Δ. The optimal step size that minimizes BER decreases as ρ increases. However, it may be better to apply the Δ larger than the optimal value to avoid serious performance degradation [128]. If ρ is determined to be four, Fig. 3.16 also shows that the optimal Δ is almost independent of the channel SNRs; such property avoids the necessity of dynamically adjusting Δ according channel conditions, which are quite difficult to be estimated in real applications. Additionally, the results in the 64 quadrature amplitude modulation (64-QAM) is presented in Fig. 3.17. The quantization is slightly different from Fig. 3.15 because of the amplitude modulation. Referring to the 64-QAM constellation in [129], we use the following quantizer scheme to demap the first three bits:



(a) SNR=13.5dB

(b) $\rho = 4$

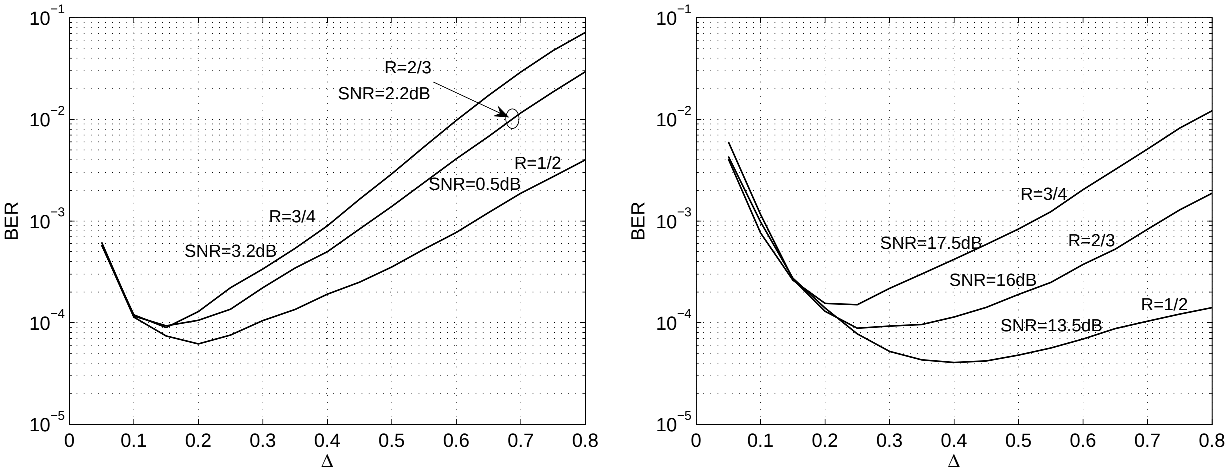
Figure 3.17: Different quantization schemes in 64-QAM and their BER performance

$$b_0 = \Phi(I) \tag{3.122}$$

$$b_1 = \Phi(4 - |I|) \tag{3.123}$$

$$b_2 = \Phi(2 - |4 - |I||) \tag{3.124}$$

where I is the in-phase component (carrier) in the received signal. The other three bits can also be obtained from (3.122)$\sim$ (3.124) with I being replaced by the quadrature component Q. The performance in Fig. 3.17(a) indicates a significant improvement from $\rho = 3$ to $\rho = 4$, meaning that more resolution is requited as compared to the BPSK modulation.



(a) BPSK

(b) 64-QAM

Figure 3.18: Rate effect on the step size in BPSK modulation and 64-QAM

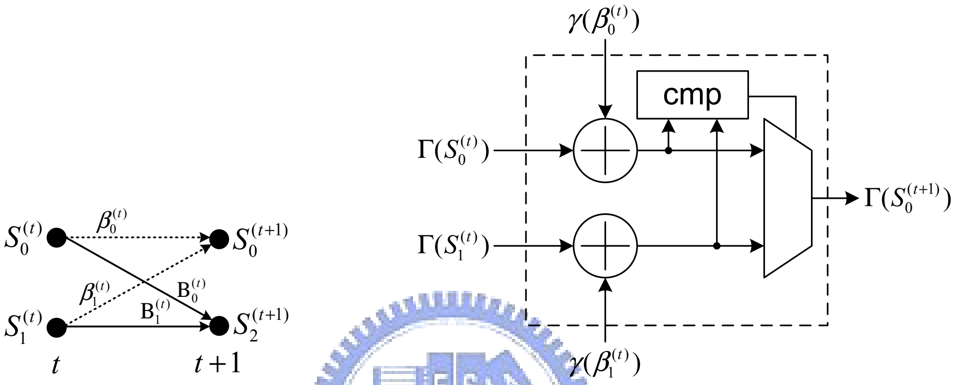
Applying different puncturing matrices [129] to the encoder (3.121), we obtain the performance figures in Fig. 3.18. Notice that the optimal step size decreases with the increasing rate especially in 64-QAM. in Fig. 3.18(b), the lower rate convolutional code with more redundant information seems much flexible over a wide range of step sizes, whereas the punctured codes with rates $R = 3/4$ and $R = 2/3$ become sensitive to the step size variation.

3.3.2 Add-compare-select unit

The ACSU is the major arithmetic unit in the Viterbi decoder, and it also dominates the computational complexity. The most common solution to develop a high throughput Viterbi decoder is fully parallel approach where ACS units are assigned to each state. Nevertheless, the throughput is also limited by the recursive operation. Fig. 3.19(a) shows a subset of the trellis diagram in Fig. 3.8, where $S_i^{(t)}$ denotes state S_i at time t, and $\beta_i^{(t)}$ represents the branch connecting $S_i^{(t)}$ and $S_0^{(t+1)}$. We can construct the ACS unit for state $S_0^{(t+1)}$ in Fig. 3.19(b) according to the following operation:

$$\Gamma(S_0^{(t+1)}) = \min_{i=0,1}[\Gamma(S_i^{(t)}) + \gamma(\beta_i^{(t)})]. \tag{3.125}$$

The branch metric $\gamma(\beta_i^{(t)})$ can be based on either (3.83) for soft decision decoding or (3.79) for hard decision decoding. The two-way comparator (cmp) finds the best path metric associated with the survivor.



(a) trellis diagram (b) ACS unit

Figure 3.19: ACS unit structure and the corresponding trellis diagram

Moreover, the critical path delay and the cost of the ACSU is also determined by the word length of fixed-point path metric Γ. With a finite word length, metric normalization is necessary to rescale Γ for the accumulation in (3.125) may cause overflow. Note that different normalization schemes will lead to different word lengths. Among various normalization approaches, the modulo normalization can simplify the circuit implementation [130, 131] since it exploits the nature of two's complement arithmetic and dispenses with extra control circuits. From the discussion of path truncation, we know that all survivors at time $t+T$ would very likely originate from the same state at time t for sufficient large truncation length T; otherwise, there would be a significant truncation error. After input symbol quantization,

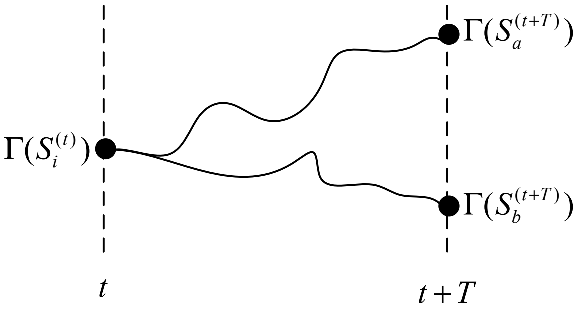


Figure 3.20: Illustration of two survivors at time $t + T$

we can bounded the branch metric by

$$|\gamma| \leq nB. \tag{3.126}$$

The value of B depends on the quantization scheme and the branch metric evaluation. For instance, B should be $2^{\rho-1}$ if the input symbol is quantized to ρ bits, and γ is calculated with (3.83). Therefore, as shown in Fig. 3.20, the difference of any two path metrics $\Gamma(S_a^{(t+T)})$ and $\Gamma(S_b^{(t+T)})$ at time $t + T$ will be bounded by

$$|\Gamma(S_a^{(t+T)}) - \Gamma(S_b^{(t+T)})| \leq (TnB + \Gamma(S_i^{(t)})) - (-TnB + \Gamma(S_i^{(t)})) = 2TnB. \tag{3.127}$$

Additionally, the compare operation in (3.125) can be written as the path metric difference:

$$\begin{aligned}\Delta\Gamma &= [\Gamma(S_1^{(t)}) + \gamma(\beta_1^{(t)})] - [\Gamma(S_0^{(t)}) + \gamma(\beta_0^{(t)})] \\ &= [\Gamma(S_1^{(t)}) - \Gamma(S_0^{(t)})] + [\gamma(\beta_1^{(t)}) - \gamma(\beta_0^{(t)})] \\ &\leq (2TnB) + 2nB = (2T + 1)nB, \end{aligned} \tag{3.128}$$

and the sign of $\Delta\Gamma$ will be the compare result. In two's complement arithmetic with c bits, the addition is defined by modulo 2^c, and $\Delta\Gamma$ is reduced to

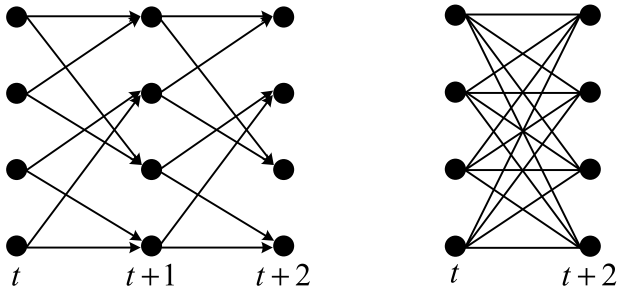
$$\Delta\Gamma \mod 2^c. \tag{3.129}$$

The reduced metric difference in (3.129) equals to the true $\Delta\Gamma$ if the following condition is satisfied [130]:

$$(2T+1)nB < 2^{c-1}. \tag{3.130}$$

Hence the bit number c required to represent the path metrics depends on the truncation length T and the branch metric bound nB. In real applications, the number c is lower than the bound in (3.130) because the equality in (3.126) holds occasionally. In the modulo normalization scheme, the comparator has to be implemented with two's complement sub tractors instead of comparators. The modified comparison rule for efficient implementation is reported in [131].

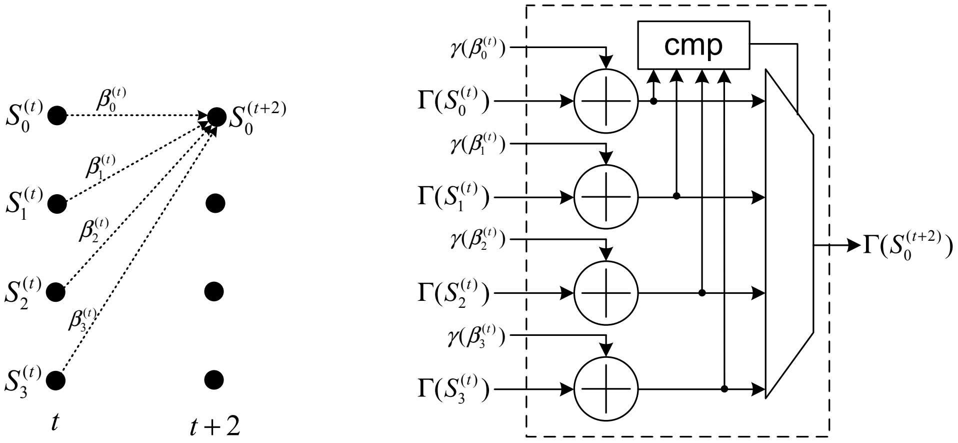
The decoders using high radix trellis in [42, 49, 50, 54, 55] achieve high-speed with τ steps of lookahead where the throughput will be enhanced by τ. Nevertheless, the ideal speedup is difficult to achieve due to the exponentially increasing number of branches compared within each ACS unit, restricting τ to be at most two in most designs. Fig. 3.21 illustrates a $\tau = 2$ example from the encoder (3.54); the radix-4 trellis diagram in Fig. 3.21(b) is obtained form the original radix-2 diagram in Fig. 3.21(a) that the branches between t and $t+2$ are merged. There are four entering branches in each state, and thus the ACS unit has to perform the comparison among four path metrics, which is double as compared with radix-2 trellis diagram. The decoding throughput is also doubled because each branch in Fig. 3.21(b) contains two information symbols. Accordingly, the radix-4 ACS structure is also illustrates



(a) radix-2 (b) radix-4

Figure 3.21: Radix-2 and radix-4 trellis diagrams

in Fig. 3.22. The number of adders becomes four, and there are four path metrics that should be compared. Generally, the radix-4 ACS unit can achieve a better compromise between cost and throughput [54]. In a radix-8 trellis diagram with eight states, for example, the ACS unit requires eight adders which is four times as many as the radix-2 ACS unit, but the decoding speed can only be enhanced up to three times.



(a) trellis diagram

(b) ACS unit

Figure 3.22: ACS unit structure and the corresponding radix-4 trellis diagram

The Viterbi decoders in [51–53] break down the critical path delay by means of bit-level pipeline and accomplish high throughput with very high clock frequencies. Furthermore, the dynamic circuit techniques are also exploited to reduce the critical path. The four states Viterbi decoder based on sliding block approach that performs decoding concurrently in forward and backward directions is also reported in [56]. However, as the constraint length increases, the complexity grows rapidly because of the highly parallel ACSU as well as large skew buffers on signal wires.

For different applications and design constraints, the implementation approach ranges from fully parallel computing array to sharing the computational resources through multiplexing. As for the WLAN category, the fully parallel approach is preferred because the demanded data rate may reach decades of Mb/s or even hundreds of Mb/s. A further improvement is the parallel architecture [132] where path metrics and decisions are calculated concurrently with an expense of carry-save-adders. The transformations of ACS unit in [133], [134], and [43] result in a compare-select-add (CSA) structure, leading to lower computational complexity. $S_{x,y}^{t-1}$ connects to S_x^t through $\beta_{x,y}^{t-1}$, and $S_{x,y,z}^{t-2}$ attaches to $S_{x,y}^{t-1}$ via $\beta_{x,y,z}^{t-2}$. The recursion in (3.137) is an ACS operation shown in Fig. 3.19(b) that iteratively updates the path metrics in each time instance. It is the serial operations within ACSU that However, the longer critical path delay will result in a slower decoding speed. A modified CSA (MCSA) is presented to improve the data-path delay in original CSA architectures while preserving lower complexity [47].

Referring to the trellis diagram in Fig. 3.19(a), we can obtain the comparison results for $S_0^{(t+1)}$ and $S_2^{(t+1)}$:

$$D(S_0^{(t+1)}) = \mathrm{sgn}[\Gamma(S_0^{(t)}) - \Gamma(S_1^{(t)}) + \gamma(\beta_0^{(t)}) - \gamma(\beta_1^{(t)})] = \mathrm{sgn}[\Delta\Gamma_{0,1}^{(t)} + \Delta\gamma_\beta^{(t)}] \tag{3.131}$$

$$D(S_2^{(t+1)}) = \mathrm{sgn}[\Gamma(S_0^{(t)}) - \Gamma(S_1^{(t)}) + \gamma(\mathrm{B}_0^{(t)}) - \gamma(\mathrm{B}_1^{(t)})] = \mathrm{sgn}[\Delta\Gamma_{0,1}^{(t)} + \Delta\gamma_\mathrm{B}^{(t)}], \tag{3.132}$$

where

$$\Delta\Gamma_{0,1}^{(t)} = \Gamma(S_0^{(t)}) - \Gamma(S_1^{(t)}) \tag{3.133}$$

$$\Delta\gamma_\beta^{(t)} = \gamma(\beta_0^{(t)}) - \gamma(\beta_1^{(t)}) \tag{3.134}$$

$$\Delta\gamma_\mathrm{B}^{(t)} = \gamma(\mathrm{B}_0^{(t)}) - \gamma(\mathrm{B}_1^{(t)}). \tag{3.135}$$

Generally, $D(S_i^{(t+1)})$ is referred to the *decision* of state S_i at time $t+1$. For the encoder in (3.54), we will find that

$$\Delta\gamma_\beta^{(t)} = -\Delta\gamma_\mathrm{B}^{(t)} \triangleq \Delta\gamma \tag{3.136}$$

The identical term $\Delta\gamma$ indicates that the computations of $D(S_0^{(t+1)})$ and $D(S_0^{(t+1)})$ can share the same arithmetic unit. The reconstructed CSA unit is shown in Fig. 3.23. Notice that the $\Delta\Gamma$ calculation can be shared by many CSA units. The resource sharing of $\Delta\Gamma_{0,1}^{(t)}$ for $D(S_0^{(t+1)})$ and $D(S_0^{(t+1)})$ reduces one subtractor as compared to the conventional ACS.

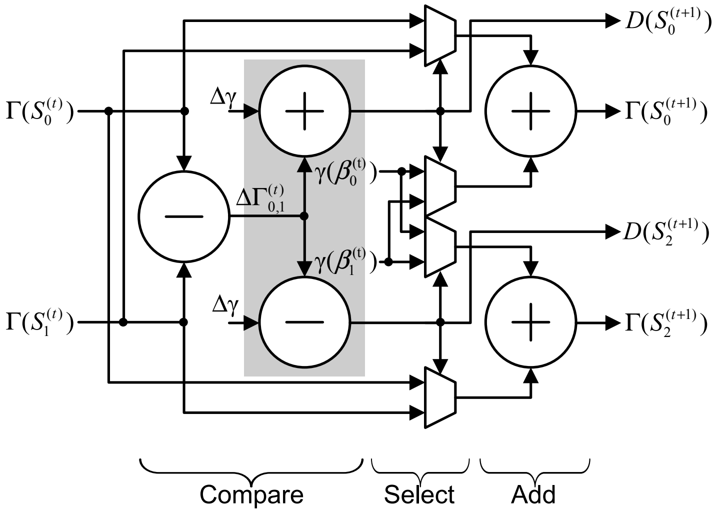


Figure 3.23: The CSA architecture

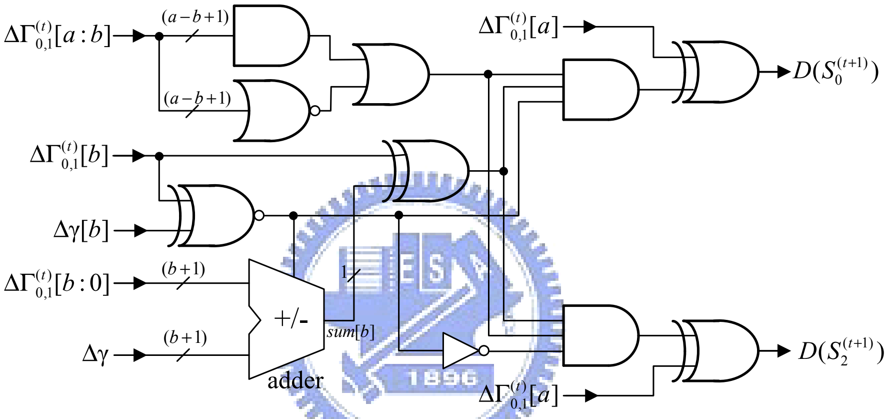

$\Delta\Gamma_{0,1}^{(t)}[a:b]$
$(a-b+1)$
$(a-b+1)$
$\Delta\Gamma_{0,1}^{(t)}[a]$
$D(S_0^{(t+1)})$
$\Delta\Gamma_{0,1}^{(t)}[b]$
$\Delta\gamma[b]$
$\Delta\Gamma_{0,1}^{(t)}[b:0]$
$(b+1)$
1
+/-
$sum[b]$
$\Delta\gamma$
$(b+1)$
adder
$\Delta\Gamma_{0,1}^{(t)}[a]$
$D(S_2^{(t+1)})$

Figure 3.24: The modified comparator unit

Furthermore, a modified comparator unit is proposed in Fig. 3.24 to enhance the adder and the subtractor in the compare operation of Fig. 3.23, assuming $\Delta\Gamma_{0,1}^{(t)}$ is $a+1$ bits, $\Delta\gamma$ is $b+1$ bits, and $a > b$. In (3.131) and (3.132), the signs of $\Delta\Gamma(S_{0,1}^{(t)}) + \Delta\gamma$ and $\Delta\Gamma_{0,1}^{(t)} - \Delta\gamma$ are expected results and can be obtained by comparing the magnitudes of $\Delta\Gamma_{0,1}^{(t)}$ and $\Delta\gamma$, and by checking their difference in signs. The improvement in Fig. 3.24 comes from $a > b$, and therefore only the $b+1$ bits adder as well as the carry propagation circuit are required, resulting in the lookahead architecture that speeds up the compare operations. As a result, the circuit in Fig. 3.24 implements both (3.131) and (3.132) with a lower critical path delay and lower complexity.

Table 3.1: Comparison of different types of 64-state ACSU

	Conventional ACS	Parallel ACS [132]	CSA	MCSA
Gate count [1]	17.3k	32k	19.6k	14.7k
Critical path report [1]	6.21ns	5.75ns	8.28ns	8.06ns
Power dissipation [2]	28mW	31mW	43mW	24mW

[1] Timing constraints: 8.5ns in the worst operating condition.
[2] Simulated with 100MHz frequency and 1.8V supply.

Based on the 0.18-μm standard cell library [135], Table 3.1 summarizes a design example where the state number is 64 and the word lengths are set to be $a = 12$ and $b = 6$. Considering the gate level design, we investigate the gate count and the power dissipation of different ACSUs, including the parallel ACS structure in [132]. In the modified CSA (MCSA) scheme, the area reduction from the conventional ACS structure is about 15%, and an average 14.3% power saving is observed through simulation.

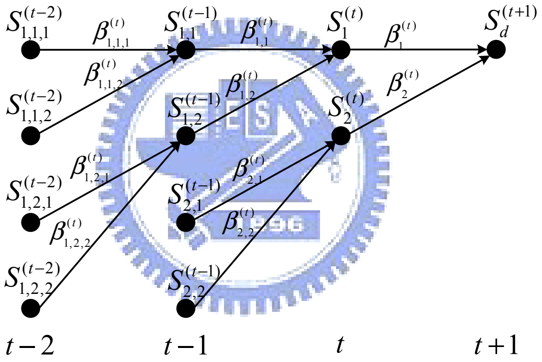
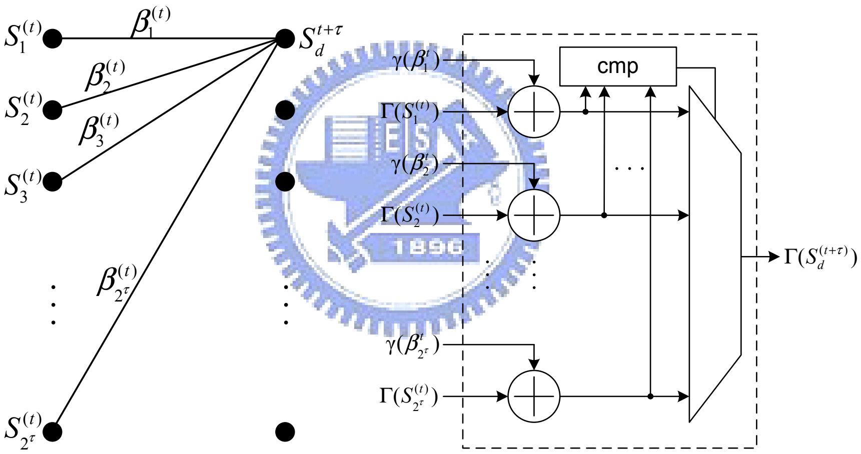


Figure 3.25: Trellis representation for the recursive path metric calculation

For high-speed applications [136,137], the data rate is required to achieve hundreds MB/s or even 1 Gb/s, and thus the ACSU may need to have more parallelism. Therefore, we present the ACS unit for radix-2^{τ} trellis with $\tau > 2$ [57]. an highly parallel ACS design [57]. Since path metrics are the recursive ACS operations, from Fig. 3.25 can the metric $\Gamma(S_d^{(t+1)})$ be iteratively obtained with

$$
\begin{aligned}
\Gamma(S_d^{(t+1)}) &= \min_{x=1,2}[\Gamma(S_x^{(t)}) + \gamma(\beta_x^{(t)})] \\
\Gamma(S_x^{(t)}) &= \min_{y=1,2}[\Gamma(S_{x,y}^{(t-1)}) + \gamma(\beta_{x,y}^{(t-1)})] \qquad (3.137) \\
\Gamma(S_{x,y}^{(t-1)}) &= \min_{z=1,2}[\Gamma(S_{x,y,z}^{(t-2)}) + \gamma(\beta_{x,y,z}^{(t-2)})] \\
&\ \vdots
\end{aligned}
$$

Note that $S_x^{(t)}$ connects to arbitrary state $S_d^{(t+1)}$ at time $t+1$ through $\beta_x^{(t)}$, $S_{x,y}^{(t-1)}$ to $S_x^{(t)}$ through $\beta_{x,y}^{(t-1)}$, and $S_{x,y,z}^{(t-2)}$ to $S_{x,y}^{(t-1)}$ through $\beta_{x,y,z}^{(t-2)}$. The recursion in (3.137) is the ACS operation shown in Fig. 3.19(b) that iteratively updates the path metrics at each time instance. It is the serial operations within ACSU that causes the critical path bottleneck.



(a) Radix-2^τ trellis diagram (b) Radix-2^τ ACS unit

Figure 3.26: The radix-2^τ ACS operation

With τ steps of lookahead, the trellis structure becomes radix-2^τ in Fig. 3.26(a), and the $\Gamma(S_d^{(t+\tau)})$ at time instance $t+\tau$ can be expressed by

$$
\Gamma(S_d^{(t+\tau)}) = \min_{x\in\mathbf{C}}[\Gamma(S_x^{(t)}) + \gamma(\beta_x^{(t)})], \qquad (3.138)
$$

and $\mathbf{C} = \{1, 2, \ldots, 2^\tau\}$ is the set of indexes indicating $S_x^{(t)}$ connects to $S_d^{(t+\tau)}$ through $\beta_x^{(t)}$.

The equivalent radix-2^τ ACS unit in Fig. 3.26(b) achieves τ times speedup as compared to the radix-2 ACSU in Fig. 3.19(b). Nevertheless, the number of branches in (3.138)The radix-2^M PM updating and the corresponding ACS structure will be $2^{\tau-1}$ times of that in radix-2 trellis, leading to the exponentially increasing complexity. The high radix approach that accelerates Viterbi decoding can also cause large critical path delay due to exponentially increasing branches. As shown in Fig. 3.26, the adders can be proceeded simultaneously, but the speed of the comparator will be degraded as the number of branches increases. Therefore, the comparator should be optimized to acquire the corresponding enhancement contributed by the high radix trellis. The retiming technique is then presented to speedup the ACS operations.

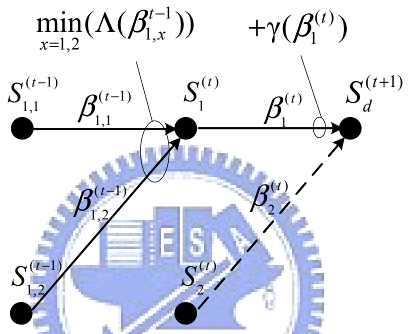


Figure 3.27: The trellis diagram after retiming

The retiming approacher tends to parallelize the computations within ACS unit. The pre-path metric (pre-PM), denoted by Λ, is first defined by

$$\Lambda(\beta_x^{(t)}) \triangleq \Gamma(S_x^{(t)}) + \gamma(\beta_x^{(t)}), \tag{3.139}$$

From the recursion in (3.137), we know that $\Gamma(S_x^{(t)})$ is function of the information coming from $\beta_{x,1}^{(t-1)}$ and $\beta_{x,2}^{(t-1)}$. Therefore, (3.139) can be rewritten as

$$\Lambda(\beta_x^{(t)}) \triangleq \min_{y=1,2}[\Lambda(\beta_{x,y}^{(t-1)})] + \gamma(\beta_x^{(t)}),\ \forall \beta_{x,y}^{(t-1)} \text{ connect to } S_x^{(t)} \tag{3.140}$$

$$= \min_{y=1,2}[\Lambda(\beta_{x,y}^{(t-1)}) + \gamma(\beta_x^{(t)})], \tag{3.141}$$

leading to a new recursion for Λ. Notice that (3.140) contains a compare-select (CS) function for $\Lambda(\beta_{x,y}^{(t-1)})$ and an addition of $\gamma(\beta_x^{(t)})$. Fig. 3.27 illustrates the operation (3.140) in the trellis diagram when $x = 1$. The final addition is independent of the compare function and can be performed concurrently with the compare function, resulting in better critical path delay. Since the recursion has been changed from $\Gamma(S_d^{(t+1)})$ to $\Lambda(\beta_1^{(t)})$ and $\Lambda(\beta_2^t)$, the number of adders and multiplexers is doubled in contrast to the original ACS unit in Fig. 3.19(b).

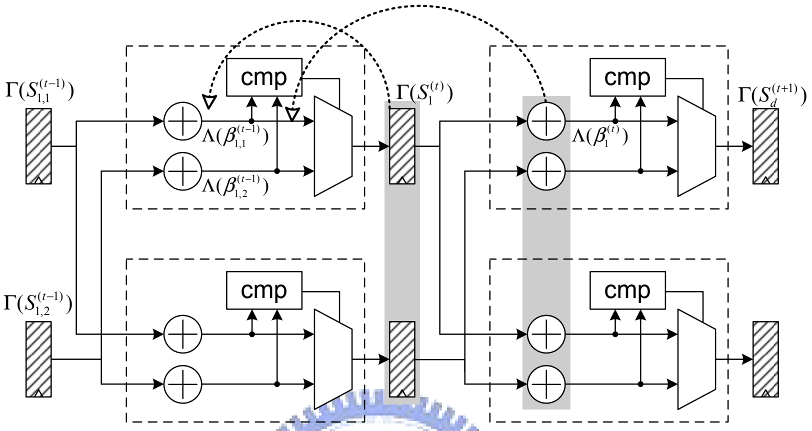

$\Gamma(S_{1,1}^{(t-1)})$
$\Gamma(S_{1,2}^{(t-1)})$
cmp
cmp
$\Lambda(\beta_{1,1}^{(t-1)})$
$\Lambda(\beta_{1,2}^{(t-1)})$
$\Gamma(S_1^{(t)})$
cmp
cmp
$\Lambda(\beta_1^{(t)})$
$\Gamma(S_d^{(t+1)})$

Figure 3.28: The retiming among different time instances for a two states trellis diagram

The transformation is a retiming of registers storing path metrics and adders among different time instances. Fig. 3.28 demonstrates the retiming procedure for a two state trellis diagram. The registers at t are moved to the branches between t and $t-1$ to keep the pre-path metric Λ, and the registers number becomes double. Furthermore, the adders are also relocated to be in parallel with the compare operations. The result after retiming is presented in Fig. 3.29 in which the registers, the adders, and the multiplexers are double as many as the structure before retiming. Actually, the architecture in Fig. 3.29 is identical to the double state approach presented in [138–140].

Based on the retiming technique, the critical path due to exponentially increasing branches in high-radix ($\tau > 2$) ACS unit can be improved. The comparator of a radix-2^τ ACS unit is to search for the minimum pre-path metrics among 2^τ candidates. One of the solutions to simplify the searching algorithm is the decomposition of the candidates that need to be

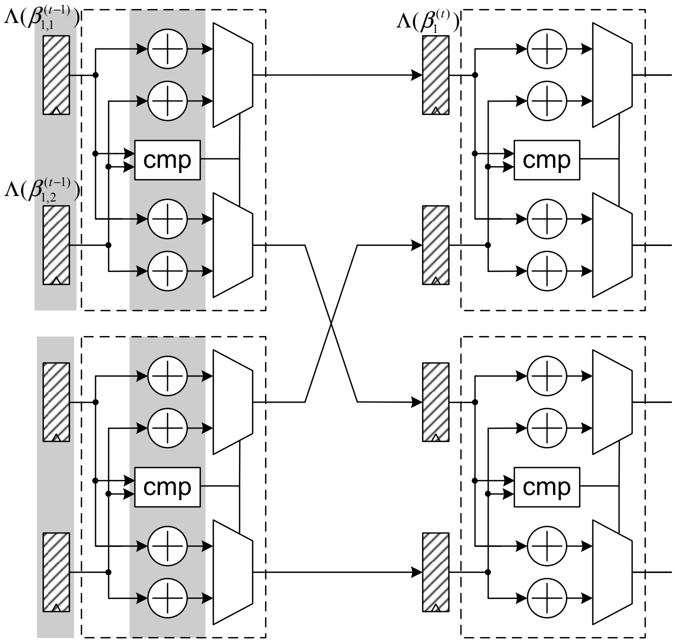

$\Lambda(\beta_{1,1}^{(t-1)})$
$\Lambda(\beta_{1,2}^{(t-1)})$
$\Lambda(\beta_{1}^{(t)})$
cmp
cmp
cmp
cmp

Figure 3.29: After retiming

compared. The ACS operation in (3.138) can be re-written as

$$\Gamma(S_d^{(t+\tau)}) = \min_{x\in \mathrm{X}_q}[\min_{y\in \mathrm{Y}_p}[\Gamma(S_{x,y}^{(t)}) + \gamma(\beta_{x,y}^{(t)})]], \qquad (3.142)$$

where $\mathrm{X}_q = \{1, 2, \ldots, 2^q\}$, $\mathrm{Y}_p = \{1, 2, \ldots, 2^p\}$, and $\tau = p + q$. The minimum function is decomposed into two levels, and the 2^τ candidates are partitioned into 2^q subsets. The first level is 2^p-way CS (CS-2^p) operations that finds the minimum within each subset containing 2^p candidates. Similarly, with a 2^q-way CS (CS-2^q) function, the outputs from the first level are compared consecutively to produce the final result. Fig. 3.30 demonstrates the architecture of a radix-2^τ ACS unit. The critical path in Fig. 3.30 will be the adder as well as the two levels of comparator and multiplexer. We also define $\Lambda(\beta_x^{(t)})$ as the result in the first level comparison (see Fig.3.30); consequently,

$$\Lambda(\beta_x^{(t)}) = \min_{y\in \mathrm{Y}_p}[\Gamma(S_{x,y}^{(t)}) + \gamma(\beta_{x,y}^{(t)})]. \qquad (3.143)$$

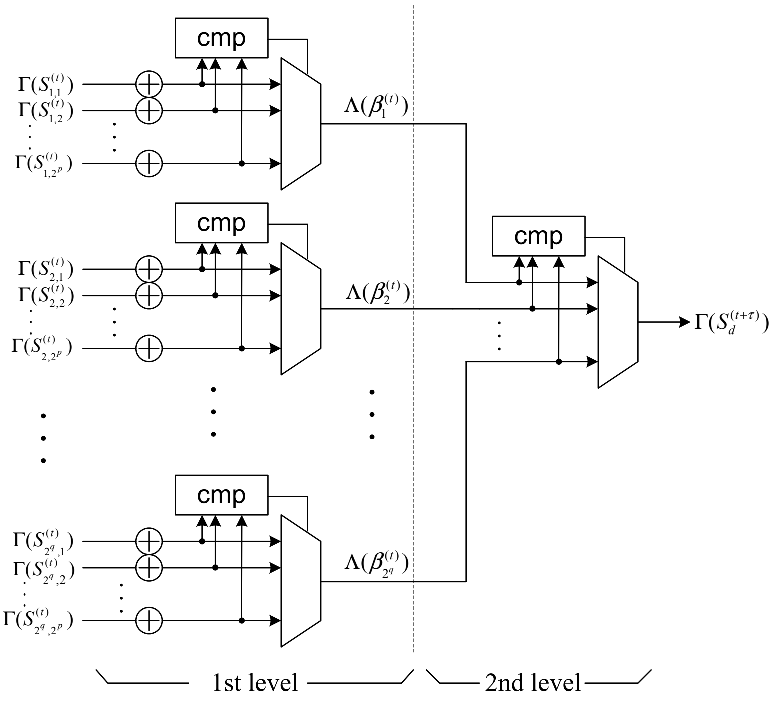


Figure 3.30: Radix-2^τ ACS unit with two levels of compare-select functions

The original ACS recursion in (3.142) can then be transformed to

$$\Gamma(S_d^{t+\tau}) = \min_{x\in \mathrm{X}_q}[\Lambda(\beta_x^{(t)})]. \tag{3.144}$$

Substituting $\Lambda(\beta_{x,y,z}^{(t-\tau)})$ for $\Gamma(S_{x,y}^{(t)})$ in (3.143), we can deduce the recursion of $\Lambda(\beta_x^{(t)})$:

$$\Lambda(\beta_x^{(t)}) = \min_{y\in \mathrm{Y}_p}[\min_{z\in \mathrm{X}_q}[\Lambda(\beta_{x,y,z}^{(t-\tau)})] + \gamma(\beta_{x,y}^{(t)})], \tag{3.145}$$

where $\beta_{x,y,z}^{(t-\tau)}$ is the z-th branch entering $S_{x,y}^{(t)}$.

Fig. 3.31 shows the corresponding radix-2^τ trellis diagram for (3.145). With two level computations, the first CS-2^q finds the minimum $\Lambda(\beta_{x,y,z}^{t-\tau})$ for all $z \in \mathrm{X}_q$, and the second ACS-2^p completes the remaining ACS operation in (3.145). Note that $\gamma(\beta_{x,y}^{(t)})$ is constant for different z, the first CS-2^q and the additions in ACS-2^p can be proceeded simultaneously, achieving less datapath delay.

Fig. 3.32 shows the retiming process (RT-1) for radix-2^τ ACS unit according to (3.145)

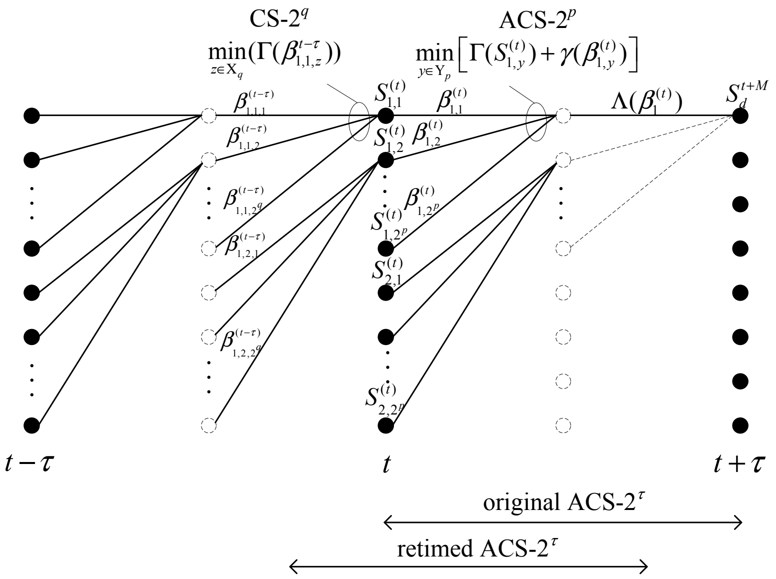

CS-2^q
$\min_{z\in X_q}(\Gamma(\beta_{1,1,z}^{t-\tau}))$
ACS-2^p
$\min_{y\in Y_p}\left[\Gamma(S_{1,y}^{(t)})+\gamma(\beta_{1,y}^{(t)})\right]$
$\beta_{1,1,1}^{(t-\tau)}$
$\beta_{1,1,2}^{(t-\tau)}$
$\beta_{1,1,2^q}^{(t-\tau)}$
$\beta_{1,2,1}^{(t-\tau)}$
$\beta_{1,2,2^q}^{(t-\tau)}$
$S_{1,1}^{(t)}$
$S_{1,2}^{(t)}$
$S_{1,2^p}^{(t)}$
$S_{2,1}^{(t)}$
$S_{2,2^p}^{(t)}$
$\beta_{1,1}^{(t)}$
$\beta_{1,2}^{(t)}$
$\beta_{1,2^p}^{(t)}$
$\Lambda(\beta_1^{(t)})$
S_d^{t+M}
$t-\tau$
t
$t+\tau$
original ACS-2^τ
retimed ACS-2^τ

Figure 3.31: Radix-2^τ trellis diagram with retiming

and Fig. 3.31. In Fig. 3.32, the registers are moved to the branch $\beta_{x,y,z}$ to store $\Lambda(\beta_{x,y,z}^{(t-\tau)})$ for $z \in \mathrm{X}_q$, and the register number becomes 2^q times. Furthermore, the adders are changed to the inputs of 2^q-to-1 multiplexer, and their amount also increases $2^q - 1$ times. The number of multiplexers in the second level operation should be 2^τ times because each state has 2^τ leaving branches. Fig. 3.33 shows the structure of the retimed radix-2^τ ACS unit where the comparisons in the first level coincide with the additions.

The datapath delay is shown in Fig. 3.34 where T_{CS-2^p}, T_{CS-2^q}, and T_{ACS-2^p} correspond to the delay times of the CS-2^p, the CS-2^q, and the ACS-2^p operations. The longest delay path is reduced to the two levels of CS operations, assuming comparators have larger delay time than adders. Consequently, the processing speed can be enhanced through retiming the radix-2^τ ACS unit.

Although the datapath delay can be reduced through the retiming approach, the exponentially increasing complexity of high radix Viterbi decoders causes the difficulty in VLSI implementation. The number of branch metrics ($2^{\tau n}$) generated by the BMU also increases exponentially. Therefore, we introduce a radix-$2^p \times 2^q$ structure that achieves the throughput equivalent to radix-2^τ approach where $\tau = p + q$ and $p, q > 0$. With the retiming

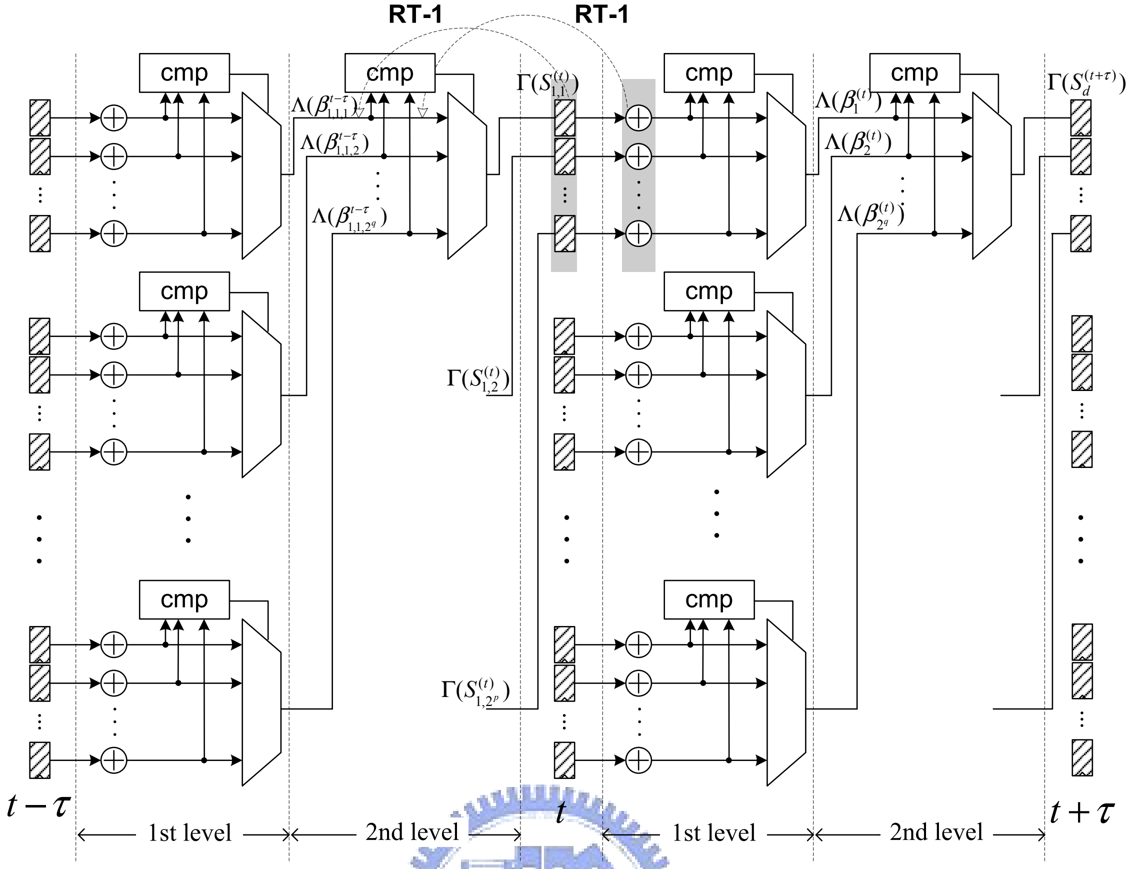


Figure 3.32: Retiming of the radix-2^τ ACS structure

technique, the radix-$2^p \times 2^q$ structure can achieve more area efficiency than the radix-2^τ architecture [57].

Fig. 3.35 shows the radix-$2^p \times 2^q$ ACSU consisting of two levels of consecutive radix-2^p and radix-2^q ACS units. The path metrics at time $t + p$ are obtained from the first level ACS units and directly passed to the second level that computes $\Gamma(S_d^{(t+\tau)})$. The successive ACS operations result in an equivalent radix-2^τ ACS operation; that is,

$$\begin{aligned}\Gamma(S_d^{(t+\tau)}) &= \min_{x\in \mathrm{X}_q}[\Gamma(S_x^{t+p}) + \gamma(\beta_x^{(t+p)})] \\ &= \min_{x\in \mathrm{X}_q}[\min_{y\in \mathrm{Y}_p}[\Gamma(S_{x,y}^{(t)}) + \gamma(\beta_{x,y}^{(t)})] + \gamma(\beta_x^{(t+p)})] \qquad (3.146)\end{aligned}$$

As compared with the radix-2^τ ACS unit, the radix-$2^p \times 2^q$ ACS unit in Fig. 3.36, termed

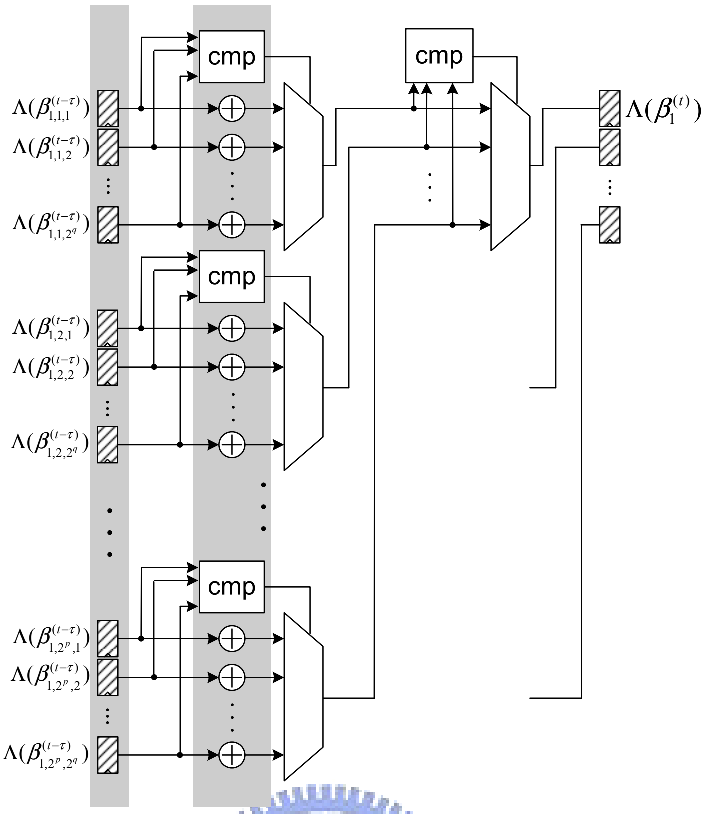
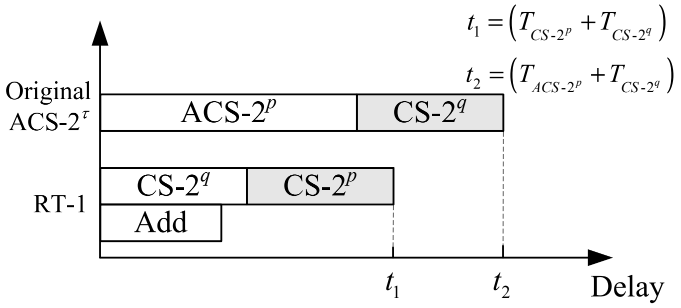

cmp
cmp
cmp
cmp
$\Lambda(\beta^{(t-\tau)}_{1,1,1})$
$\Lambda(\beta^{(t-\tau)}_{1,1,2})$
$\Lambda(\beta^{(t-\tau)}_{1,1,2^q})$
$\Lambda(\beta^{(t-\tau)}_{1,2,1})$
$\Lambda(\beta^{(t-\tau)}_{1,2,2})$
$\Lambda(\beta^{(t-\tau)}_{1,2,2^q})$
$\Lambda(\beta^{(t-\tau)}_{1,2^p,1})$
$\Lambda(\beta^{(t-\tau)}_{1,2^p,2})$
$\Lambda(\beta^{(t-\tau)}_{1,2^p,2^q})$
$\Lambda(\beta^{(t)}_{1})$

Figure 3.33: The radix-2^τ ACS unit after retiming

two-dimensional (2-D) structure, requires only smaller radix-2^p ACS (ACS-2^p) units and radix-2^q ACS (ACS-2^q) units. The exponentially increasing hardware cost is decomposed into smaller ACS units; as a result, the Viterbi decoder based on radix-$2^p \times 2^q$ architecture is more area efficient than that based on the radix-2^τ trellis. Nevertheless, the critical path of the 2-D structure is through two levels of ACS units, inducing one more adder delay as compared with the radix-2^τ ACS unit in Fig. 3.30. We will improve the 2-D ACS unit to achieve higher speed with acceptable cost through the retiming approach. There are two possible solutions based on (3.146).

The first solution comes from the independence of $\gamma(\beta_x^{t+p})$ and the function $\min_{y \in \mathrm{Y}_p}[\Gamma(S_{x,y}^{(t)})+\gamma(\beta_{x,y}^{(t)})]$. We can rewrite (3.146) as

$$\Gamma(S_d^{(t+\tau)}) = \min_{x \in \mathrm{X}_q}\left[\min_{y \in \mathrm{Y}_p}[\Gamma(S_{x,y}^{(t)}) + \gamma(\beta_{x,y}^{(t)}) + \gamma(\beta_x^{(t+p)})]\right]. \tag{3.147}$$



$t_1 = (T_{CS\text{-}2^p} + T_{CS\text{-}2^q})$
$t_2 = (T_{ACS\text{-}2^p} + T_{CS\text{-}2^q})$
Original ACS-2^τ
ACS-2^p
CS-2^q
RT-1
CS-2^q
CS-2^p
Add
t_1
t_2
Delay

Figure 3.34: Comparison of critical path delay for the original and the retimed ACS units

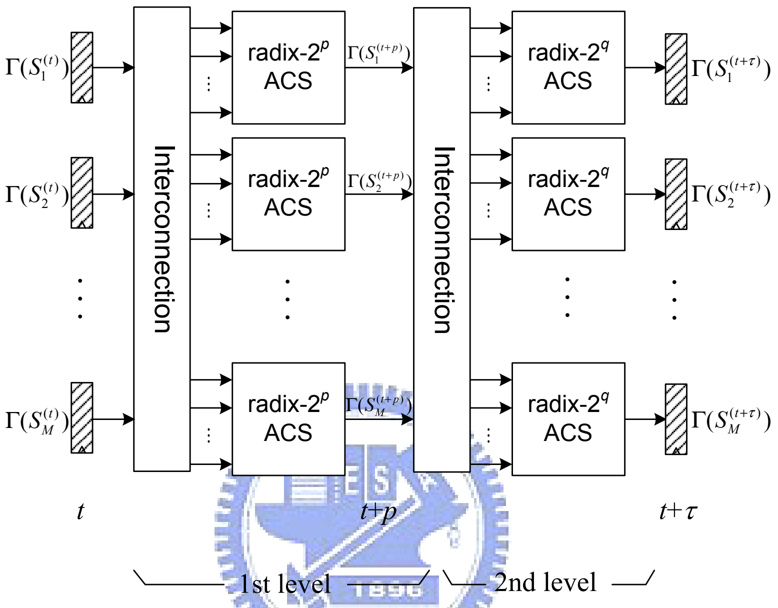


Figure 3.35: The structure of ACS-$2^p \times 2^q$ unit

The branch metric $\gamma(\beta_x^{t+p})$ is moved to the multiplexer inputs in the first level of Fig. 3.36. This retiming procedure (RT-2) as shown in Fig. 3.37 results in a ACS unit in Fig. 3.39(a). Note that the critical path is almost the same as the radix-2^τ ACS unit in Fig. 3.30. The overhead to attain this acceleration is $2^p - 1$ times more adders and multiplexers in the first level.

The other retiming technique for the radix-$2^p \times 2^q$ ACS unit first define $\Lambda(\beta_x^{(t+p)})$ to be

$$\Lambda(\beta_x^{(t+p)}) = \min_{y \in \Upsilon_p} [\Gamma(S_{x,y}^{(t)}) + \gamma(\beta_{x,y}^{(t)})] + \gamma(\beta_x^{(t+p)}), \tag{3.148}$$

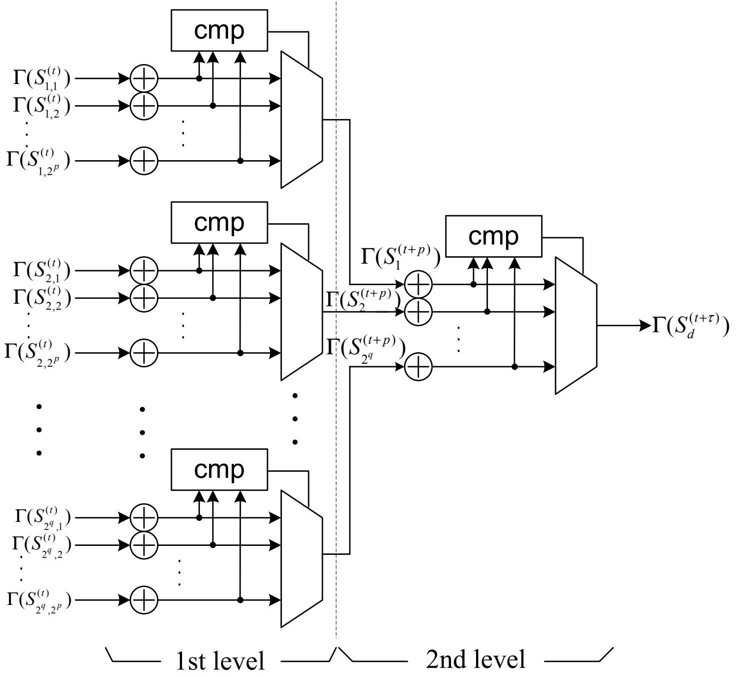


Figure 3.36: ACS-$2^p \times 2^q$ unit

and thus (3.146) will become

$$\Gamma(S_d^{(t+\tau)}) = \min_{x \in \mathrm{X}_q} [\Lambda(\beta_x^{(t+p)})]. \tag{3.149}$$

Accordingly, $\Gamma(S_{x,y}^{(t)})$ can be extended as a function of $\Lambda(\beta_{x,y,z}^{(t-q)})$ where $\beta_{x,y,z}^{(t-q)}$ is the incoming branch of state $S_{x,y}^{(t)}$, and we can rewrite (3.148) to be

$$\begin{aligned} \Lambda(\beta_x^{(t+p)}) &= \min_{y \in \mathrm{Y}_p} \left[\min_{z \in \mathrm{X}_q} [\Lambda(\beta_{x,y,z}^{(t-q)})] + \gamma(\beta_{x,y}^{(t)}) \right] + \gamma(\beta_x^{(t+p)}) \qquad (3.150) \\ &= \min_{y \in \mathrm{Y}_p} \left[\min_{z \in \mathrm{X}_q} [\Lambda(\beta_{x,y,z}^{(t-q)}) + \gamma(\beta_{x,y}^{(t)})] + \gamma(\beta_x^{(t+p)}) \right]. \qquad (3.151) \end{aligned}$$

Fig. 3.38 illustrates the corresponding operation on the radix-$2^p \times 2^q$ trellis for (3.150) with $x = 1$. The computation contains CS-2^q operations, ACS-2^p calculations, and a final addition (Add). Note that the additions can also be retimed for less datapath delay as shown in (3.151); however, the addition numbers for $\gamma(\beta_{x,y}^{(t)})$ and $\gamma(\beta_x^{(t+p)})$ will be increased respectively by 2^q and 2^p times.

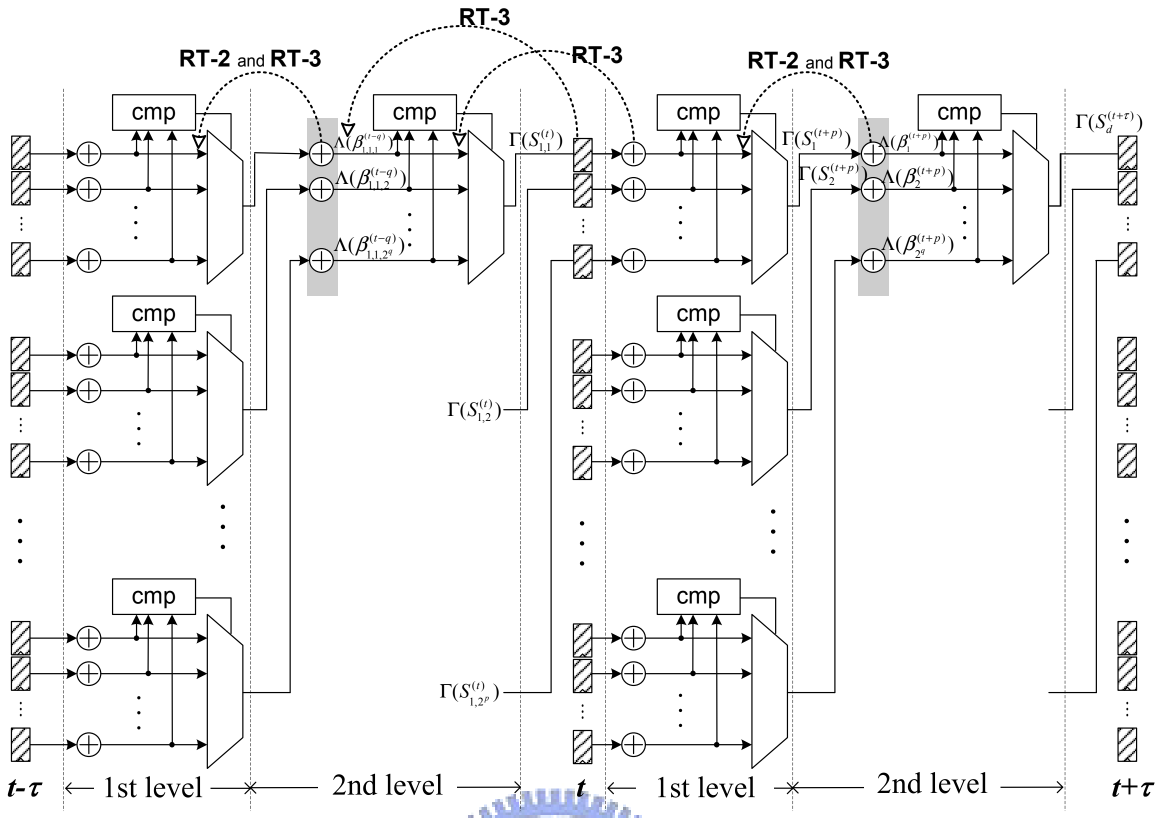


Figure 3.37: Retiming of the radix-$2^p \times 2^q$ ACS unit

We demonstrate this retiming procedure (RT-3) in Fig. 3.37 where both registers and adders are reallocated; the the final architecture is shown in Fig. 3.39(b). The registers in Fig. 3.39(b) stores the results $\Lambda(\beta_x^{(t+p)})$ on branches, and therefore the registers become 2^q times as many as the original ACS-$2^p \times 2^q$. In the radix-$2^p \times 2^q$ trellis, each multiplexer in the first level connects to 2^q adders, and the multiplexer in the second level connects to 2^p adders. As a result, there will require 2^q times 2^p-to-1 multiplexers and 2^p times 2^q-to-1 multiplexers after retiming.

The optimization methods as mentioned above tend to break the critical path through parallelizing the serial operations; thus, more hardware resources should be allocated. Instead of storing path metrics, the radix-2^τ and radix-$2^p \times 2^q$ ACS units with RT-1 and RT-3 save the pre-path metrics on branches, leading to more memory requirement. Moreover, to achieve parallel processing, the retiming of adders in RT-1, RT-2, and RT-3 causes more

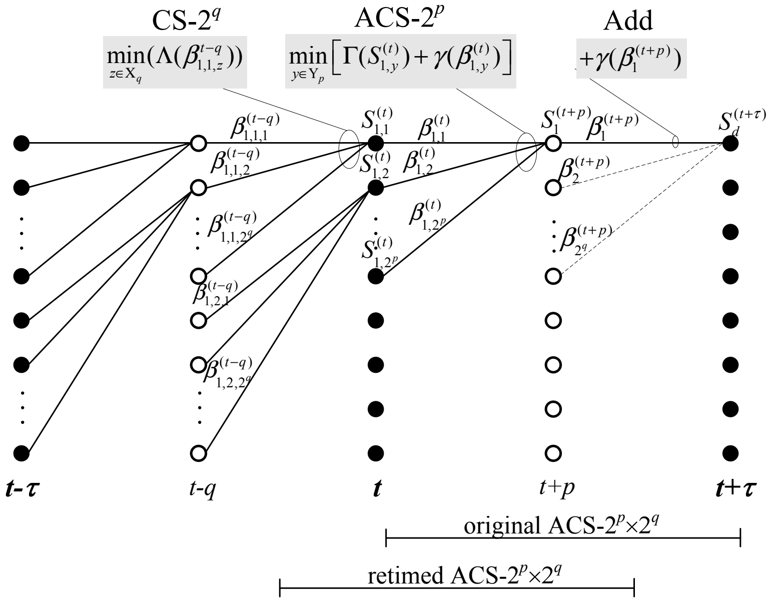
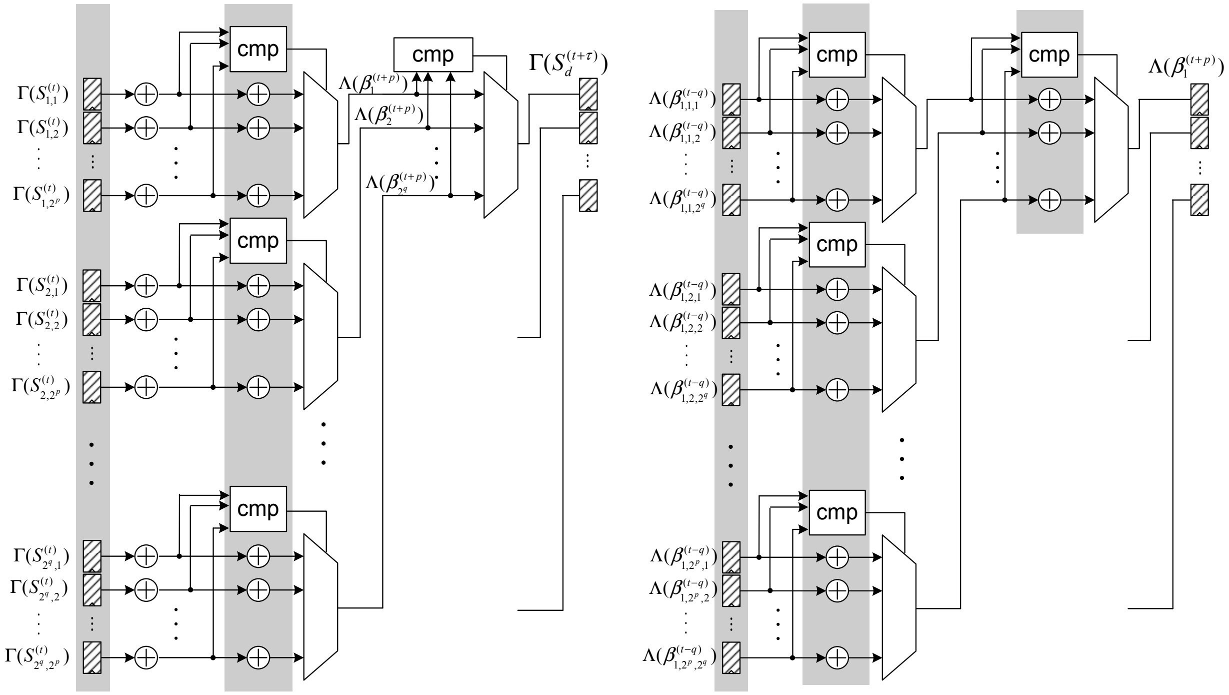


Figure 3.38: Radix-$2^p \times 2^q$ trellis

adders and multiplexers.

Fig. 3.40 compares the datapath delays of different ACS configurations. The delay times of CS-2^p, CS-2^q, ACS-2^p, and ACS-2^q functions are defined to be T_{CS-2^p}, T_{CS-2^q}, T_{ACS-2^p}, and T_{ACS-2^q}. Additionally, we also assume that the compare-select operation has larger delays, T_{CS-2^p} and T_{CS-2^q}, than those of additions (Add). What should be noted is that the major enhancement is the elimination of datapath delay of additions on the critical path. Both ACS-2^τ with RT-1 and ACS-$2^p \times 2^q$ with RT-3 can achieve the lowest delay time $T_{CS-2^p} + T_{CS-2^q}$ because of the parallel additions and comparisons. Furthermore, Fig. 3.40 also shows that ACS-$2^p \times 2^q$ with RT-2 can acquire a comparable performance to the high radix ACS-2^τ structure.

We summarize the complexity of different ACS architectures, mentioned above, in Table 3.2. The cost of ACS-$2^p \times 2^q$ is smaller than that of ACS-2^τ because $2^\tau \geq 2^p + 2^q$ for $\tau = p + q$. The minimum adder number required in the ACS-$2^p \times 2^q$ can be achieved when $p = \lceil \frac{\tau}{2} \rceil$ and $q = \tau - \lceil \frac{\tau}{2} \rceil$. Considering the ACS-$2^\tau$ with RT1 and the ACS-$2^p \times 2^q$ with RT-3, the adder number in the former is larger than that in the latter while $q > 1$. Moreover,


cmp
$\Gamma(S_d^{(t+\tau)})$
$\Lambda(\beta_1^{(t+p)})$
$\Lambda(\beta_2^{(t+p)})$
$\Lambda(\beta_{2^q}^{(t+p)})$
$\Gamma(S_{1,1}^{(t)})$
$\Gamma(S_{1,2}^{(t)})$
$\Gamma(S_{1,2^p}^{(t)})$
$\Gamma(S_{2,1}^{(t)})$
$\Gamma(S_{2,2}^{(t)})$
$\Gamma(S_{2,2^p}^{(t)})$
$\Gamma(S_{2^q,1}^{(t)})$
$\Gamma(S_{2^q,2}^{(t)})$
$\Gamma(S_{2^q,2^p}^{(t)})$
$\Lambda(\beta_{1,1,1}^{(t-q)})$
$\Lambda(\beta_{1,1,2}^{(t-q)})$
$\Lambda(\beta_{1,1,2^q}^{(t-q)})$
$\Lambda(\beta_{1,2,1}^{(t-q)})$
$\Lambda(\beta_{1,2,2}^{(t-q)})$
$\Lambda(\beta_{1,2,2^q}^{(t-q)})$
$\Lambda(\beta_{1,2^p,1}^{(t-q)})$
$\Lambda(\beta_{1,2^p,2}^{(t-q)})$
$\Lambda(\beta_{1,2^p,2^q}^{(t-q)})$
$\Lambda(\beta_1^{(t+p)})$

(a) ACS unit after RT-2

(b) ACS unit after RT-3

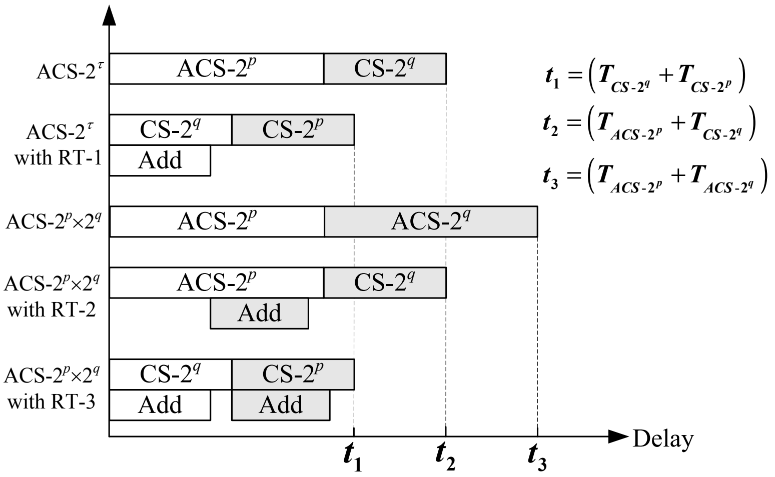
Figure 3.39: 2-D ACS unit after retiming

the ACS-2^τ with RT-1 has 2^q times as many 2^p-way comparators as the ACS-$2^p \times 2^q$ with RT-3. The original ACS-2^τ structure has the delay time similar to ACS-$2^p \times 2^q$ with RT-2, but has $(2^q - 1)$ times more 2^p-way comparators, which are considerably more complex than adders. According to the the summary in Table 3.2, the 2-D ACS-$2^p \times 2^q$ structure is more cost efficient with retiming for the high speed requirement.

Table 3.2: Comparison of complexity among different ACS configurations

	registers	adders	2^p-way comparator	2^q-way comparator	2^p to 1 multiplexer	2^q to 1 multiplexer
ACS-2^τ	M	$2^\tau \cdot M$	$2^q \cdot M$	M	$2^q \cdot M$	M
ACS-$2^p \times 2^q$	M	$(2^p + 2^q) \cdot M$	M	M	M	M
ACS-2^τ (RT-1)	$2^q \cdot M$	$2^q \cdot 2^\tau \cdot M$	$2^q \cdot M$	M	$2^q \cdot M$	$2^M \cdot M$
ACS-$2^p \times 2^q$ (RT-2)	M	$(2^p + 2^p \cdot 2^q) \cdot M$	M	M	$2^q \cdot M$	M
ACS-$2^p \times 2^q$ (RT-3)	$2^q \cdot M$	$(2^q \cdot 2^p + 2^p \cdot 2^q) \cdot M$	M	M	$2^q \cdot M$	$2^p \cdot M$

[1] The number of states is $M = 2^\nu$.
[2] $\tau = p + q$.



$$t_1 = (T_{CS\text{-}2^q} + T_{CS\text{-}2^p})$$
$$t_2 = (T_{ACS\text{-}2^p} + T_{CS\text{-}2^q})$$
$$t_3 = (T_{ACS\text{-}2^p} + T_{ACS\text{-}2^q})$$

Figure 3.40: Comparison of critical path delay for original and retimed ACS units

3.3.3 Survivor memory unit

The survivor memory stores the path history, or the survivors, within the truncation length T. For different management strategies, the survivor memory unit can be categorized into the *register exchange* (RE) architecture and the *traceback* (TB) architecture [123–126].

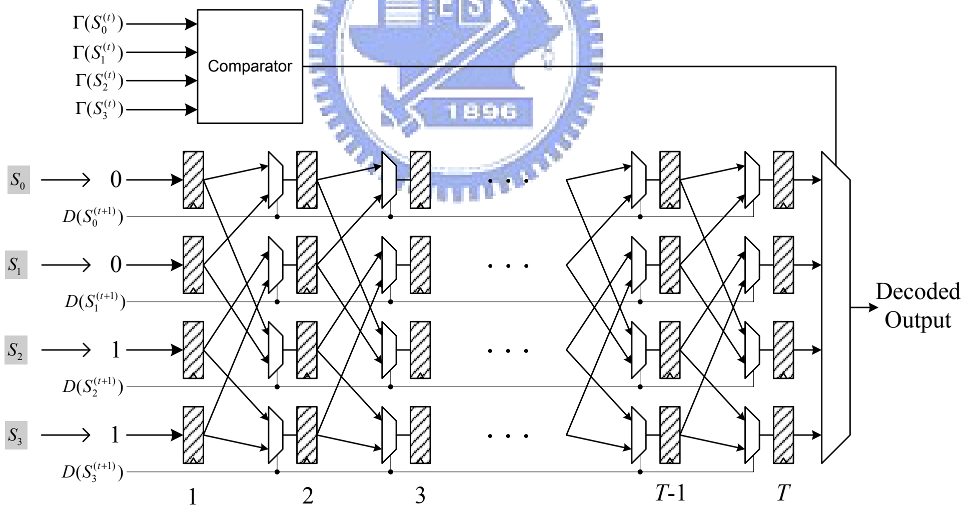


Figure 3.41: The SMU based on the register exchange approach

The register exchange is a direct implementation of the Viterbi decoding algorithm over the trellis diagram. Furthermore, any survivor, or the state sequence, also corresponds to a distinct branch sequence or information sequence. For example, the incoming branches of

S_0 in Fig. 3.8 are generated by the information bit “0”. To save the survivor memory, we keep the branch information rather than the entire state sequence. Fig. 3.41 illustrates a design example for the trellis diagram Fig. 3.8. The decisions $D(S_i^{(t+1)})$ for $i = 0 \sim 3$ come form the ACS units, and the path metrics $\Gamma(S_i^{(t)})$ are stored in the PM memory. There are 2^k decisions at each state for an (n, k) code. The survivors are shifted in the register array from the left to the right. In the leftmost size, the register inputs are “0”, “0”, “1”, and “1”, corresponding to the information symbols on the entering branches of S_0, S_1, S_2, and S_3. The output data will be obtained through finding the survivor with the best path metric. For a non-punctured code, the truncation length T is often chosen to be $4\nu \sim 5\nu$ [100] where ν is the overall constraint length. On the other hand, if the truncation length T is extended to 10ν [116], the output data can be selected form any register in the final stage with slight performance loss.

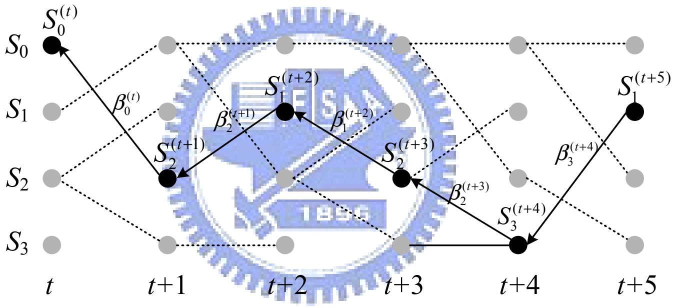


Figure 3.42: The traceback operation

The traceback approach proceeds after all survivors are stored in the memory. Fig. 3.42 is an example that the decoder tends to decode the information at time t according to the best metric $\Gamma(S_1^{(t+5)})$ at time $t + 5$. The traceback operation starts form S_1 at time $t + 5$ and sequentially reads the branches to trace the survivor form time $t + 4$ to t. Finally, we can find the state S_0 at time t and the information symbol associated with the branch $\beta_0^{(t)}$. Since only the branch is necessary to pinpoint the previous state, the memory will contain the branch information. Note that the decisions are used to identify the branches entering each state. In Fig. 3.43, for example, the upper branch coming into each state is marked with “0”, and the other is “1”. Hence the decisions will provide the branch information that should be stored into survivor memory.

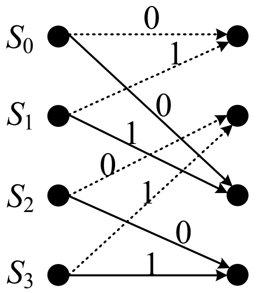
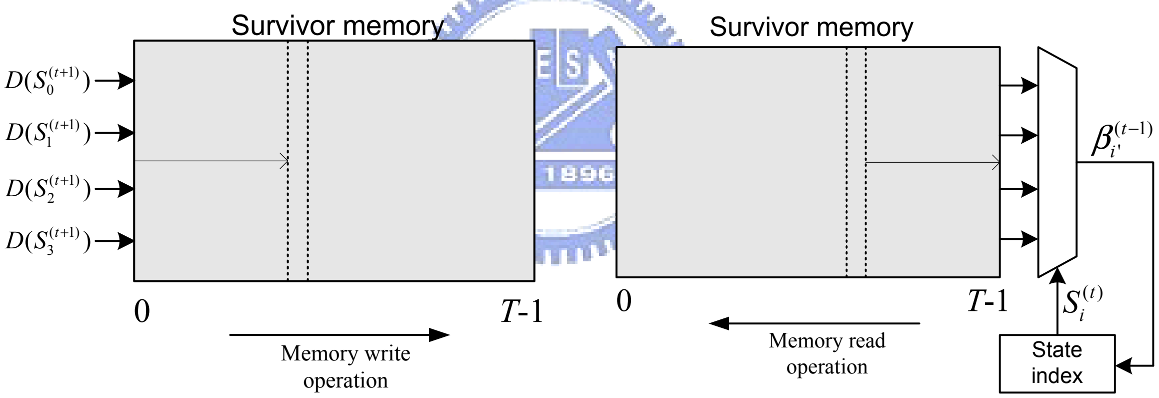


Figure 3.43: The branches are labeled with either 0 or 1

According to the trellis diagram Fig. 3.8, the SMU using traceback architecture is shown in Fig. 3.44. Fig. 3.44(a) is the memory update procedure in which the decisions are written to the survivor memory from the address 0 up to $T-1$, corresponding to the T time instances of t_1+1 to t_1+T. Fig. 3.44(b) is the traceback operation starting from the state $S_i^{(t_1+T)}$ with the best path metric at time t_1+T. The state $S_i^{(t_1+T)}$ is first used to index the branches read form the address $T-1$; the previous state $S_{i'}^{(t_1+T-1)}$ will be obtained from $S_i^{(t_1+T)}$ and its incoming branch $\beta_{i'}^{(t_1+T-1)}$. The state $S_i^{(t_1+T)}$ as well as the branch $\beta_{i'}^{(t_1+T-1)}$ is termed the *pointer* for the state $S_{i'}^{(t_1+T-1)}$. Referring to the example in Fig. 3.42, we will find that the branch $\beta_3^{(t+4)}$, labeled with 1, will be indexed by $S_1^{(t+5)}$ and is utilized to identify $S_3^{(t+4)}$.



(a) Write operation

(b) Read operation

Figure 3.44: The SMU based on the traceback approach

Since $S_1 = 01$ in the binary representation, we could get $S_3 = 11$ through left shifting S_1 by one bit and concatenating 1 to the least significant bit (LSB); that is,

$$S_3 = 2S_1 + 1 \mod 4.$$

As the branches are serially read form address $T-1$ to 0, we can subsequently find the state at time t_1 and decode the corresponding information symbol.

In the register-exchange or the traceback based architecture, the memory for a binary (n, k) code with the overall constraint length ν is required to store $k \times T \times 2^{\nu}$ bits and can be organized to T words where each word stores 2^{ν} decisions. As compared with the register-exchange approach, the traceback operation avoids the data movement within the survivor memory, and therefore has less dynamic power consumption. Nevertheless, the implementation of a high speed traceback unit would be more difficult because of the limited data bandwidth in the embedded memory.

The traceback algorithm, or k-pointer algorithm, for the survivor memory management has been proposed in [124] and [126]. It divides the memory into banks and accesses them concurrently to achieve the demanded data bandwidth. In the k-pointer algorithm, three operations are defined [125, 126]:

1. Writing new data (WR): The decisions form ACSU are written into the survivor memory. The writing address increases as the ACS operations proceed to the next time instance.

2. Traceback read (TB): This operation sequentially reads the branch from the survivor memory to perform the traceback operation as shown in Fig. 3.44(b). However, only the final state after all the branches are traced will be found without outputting any information symbols. Ensuring all the survivors have converged, this state is treated as the start point for the next traceback operation.

3. Decode read (DC): This operation is also a traceback read except that the information symbols will be decoded and outputted. Hence the decode read begins with the state determined by the previous traceback read operation. The output information will be in the reverse order because of the traceback operation.

Accordingly, the SMU based on the 3-pointer even algorithm can then be illustrated in Fig. 3.45. The three traceback pointers consist of two TB operations and one DC producing the decoded symbols. The SMU contains six banks, and each has $T/2$ words. Before any TB or DC operation, the memory banks need to be filled with decisions from the ACSU. The first TB operation starts in the bank 2, and the subsequent TB operation proceeds in the bank 1, leading to the traceback of T time instances. As a result, we can initiate the DC operation in the bank 0 because the beginning state is considered to be reliable after the previous traceback. Finally, a first in and last out (FILO) buffer of length $T/2$ symbols can be applied to reverse the output sequence. Note that the decoding throughput of Fig. 3.45 is similar to the register-exchange based SMU; however, the decoding latency of the former is $3T$ which is three times as much as the later, assuming one memory address is read within

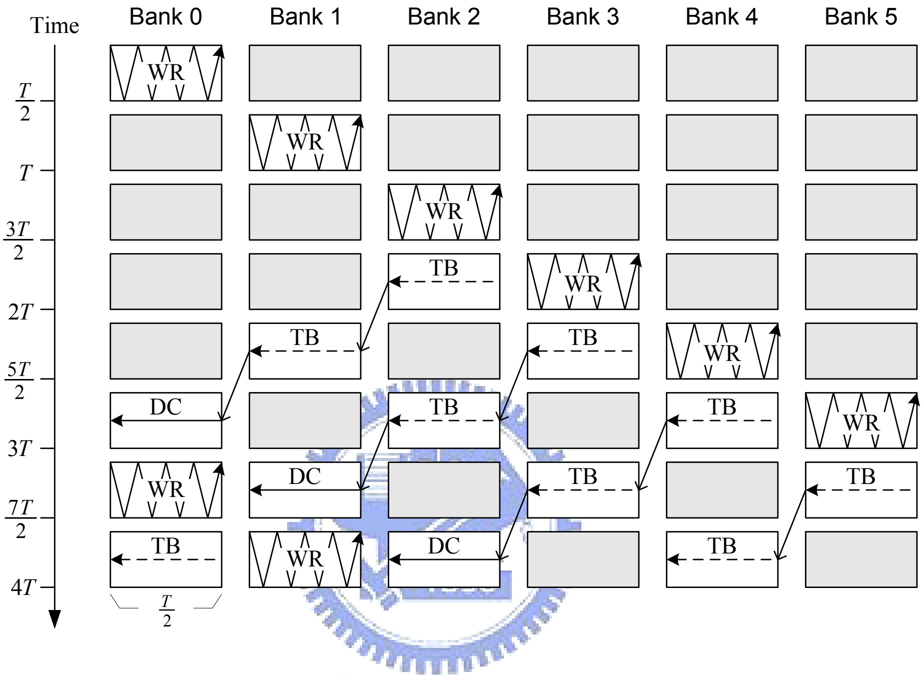


Figure 3.45: The SMU using 3-pointer even algorithm

one time instance.

Alternatively, the 3-pointer odd algorithm [126] combines the WR and the DC operations into the same memory bank. As the DC reads out one data, the WR operation will write new decisions to the address that is just read. Therefore, to achieve the corresponding throughout that the 3-pointer even algorithm can have, each memory bank requires one port for the data reading (DC), and the other port for the simultaneous data writing (WR). Moreover, the 3-pointer odd algorithm reduces the memory bank number to five. In general, the k-pointer even algorithm demands $2k$ memory banks having $\frac{T}{k-1}$ words in each bank. In the k-pointer odd algorithm, we require $2k-1$ memory banks with $\frac{T}{k-1}$ words in each. For higher k, we can accomplish more parallelism in the SMU, but requite more memory banks. In the trellis diagram, Fig. 3.46 also summarizes the operations in the SMU based on the the k-pointer algorithms. Because of the demand for high speed data transmission, both methods would cause a large amount of memory access operations as well as large power consumption.

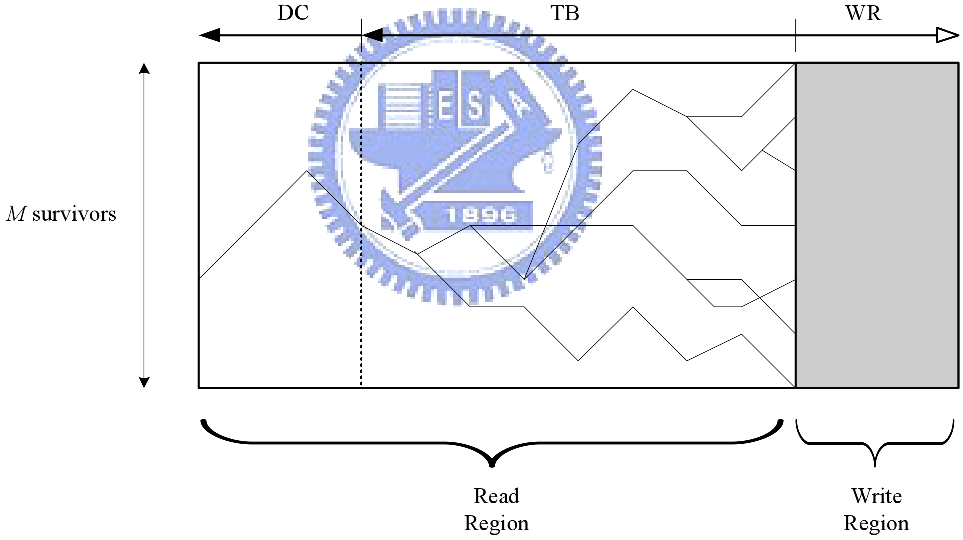


Figure 3.46: SMU operations over the trellis diagram

As shown in (3.117), the truncation length determining the size of the survivor memory and also the decoding latency, is a function of coding rates and channel capacity. The traced

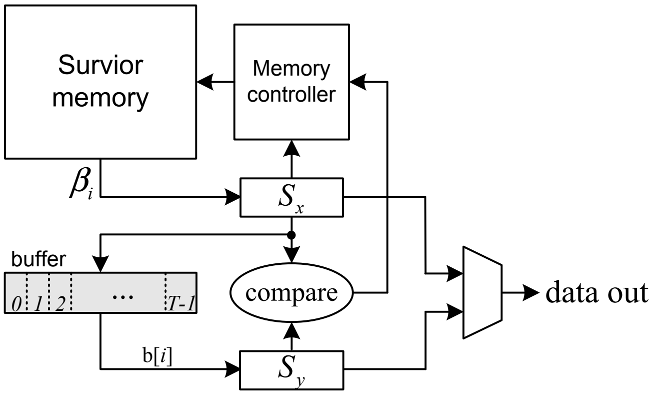


Figure 3.47: The proposed memory management of SMU

path will remerge to the correct path within the truncation length with a high probability. In order to preserve the performance, the truncation length is conventionally set to the maximum. Nevertheless, this will lead to many redundant operations during the traceback for the operating condition is not always the worst. It is inappropriate to design the Viterbi decoder works in the worst cases.

For a fixed T, SMU will trace the same path that had been traced recently as the path remerges to the correct one. This implies that the SMU tends to reuse the data which have been used in previous traceback operations. Considering the data locality in survivor memory, a dynamic traceback mechanism can be implemented to reduce power consumption caused by a great number of memory access [42, 47].

According to the cache based SMU design in Fig. 3.47, the modified traceback algorithm, path merging algorithm, is summarized in Fig. 3.48 The survivor memory initially contains M survivors. Since the correct path is hard to be exactly known in the receiver system, the buffer contents will be the path which is last traced. In the traceback operation, S_x is recursively updated according to the previous S_x and the branch β; simultaneously, it is compared with S_y obtained form the buffer. While S_x is different from S_y, the buffer contents should be revised to the new branch β_i. On the contrary, when the traced path merges to the previous one, SMU can stop further traceback operations because the buffer has contained the same survivor.

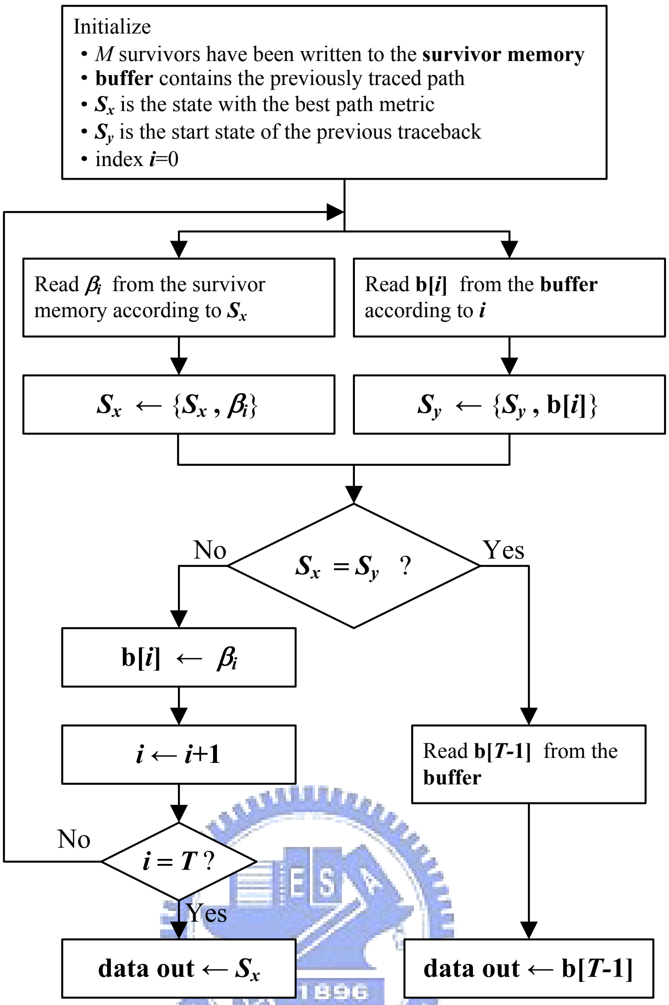

Initialize
• M survivors have been written to the **survivor memory**
• **buffer** contains the previously traced path
• S_x is the state with the best path metric
• S_y is the start state of the previous traceback
• index i=0
Read β_i from the survivor memory according to S_x
Read b[i] from the buffer according to i
$S_x \leftarrow \{S_x, \beta_i\}$
$S_y \leftarrow \{S_y, b[i]\}$
$S_x = S_y$?
No
Yes
b[i] $\leftarrow \beta_i$
$i \leftarrow i+1$
No
$i = T$?
Yes
Read b[T-1] from the buffer
data out $\leftarrow S_x$
data out $\leftarrow$ b[T-1]

Figure 3.48: The path merging algorithm

Simulation results, shown in Table 3.3, reveal the effects of the modified traceback algorithm, assuming the QPSK modulation over the AWGN channel. These results represent the length required by the traced path to merge to the last one, and over 94% paths will merge after tracing 3 time instances in different channel conditions. Therefore, many redundant operations can be eliminated through the proposed SMU design.

The path merging algorithm can be successfully applied to the traceback operations for the buffer contents are iteratively updated. However, in the WR operation where new survivors from ACSU are written into memory, there is nothing to update the buffer, and nothing can be read from it during the TB operation. Therefore, the path prediction algo-

Table 3.3: Distribution of path convergence

SNR †(dB)	2	3	4	5
1 time instance	90.82%	95.16%	97.58%	98.91%
2 time instances	93.33%	97.02%	98.81%	99.60%
3 time instances	94.03%	97.47%	99.05%	99.70%

† QPSK modulation and AWGN channel

rithm accompanied with the WR is proposed and shown in Fig. 3.49 where WR is assumed to process τ time instances. While ACSU proceeds each new time instance in the trellis diagram, the predicted state with the minimum path metric is also found and verified that a valid transition exists. Once the sequence of states encounters an invalid transition, the prediction process should be stopped to avoid improper path merging during TB operation. In terms of different SNRs, the simulation results in Fig. 3.50 represent the prediction accuracy, which is defined as the percentage of total predictable states in the WR operation.

Combined with the path merging algorithm, we conclude the buffer updating procedure in Fig. 3.51. The predicted state sequence is stored during the WR operation while new state sequence is rewritten during the TB operation until the traced path merges to that in the buffer. Even if incorrect states are predicted, they will be corrected in the TB operation, leading to no performance loss with the path prediction algorithm.

3.4 The MAP algorithm

The maximum *a posteriori* probability (MAP) decoding algorithm for linear codes is developed by Bahl, Cocke, Jelinek, and Raviv in 1974 [16] and is termed *BCJR algorithm*, the optimal symbol-by-symbol detection algorithm that minimizes the symbol error probability. As compared to the Viterbi algorithm that minimize the codeword error probability (see (3.67)), the BCJR algorithm estimates the *a posteriori* probabilities (APP) of the states as well as their transitions from the received sequence $\mathbf{r}$ over a discrete memoryless channel.

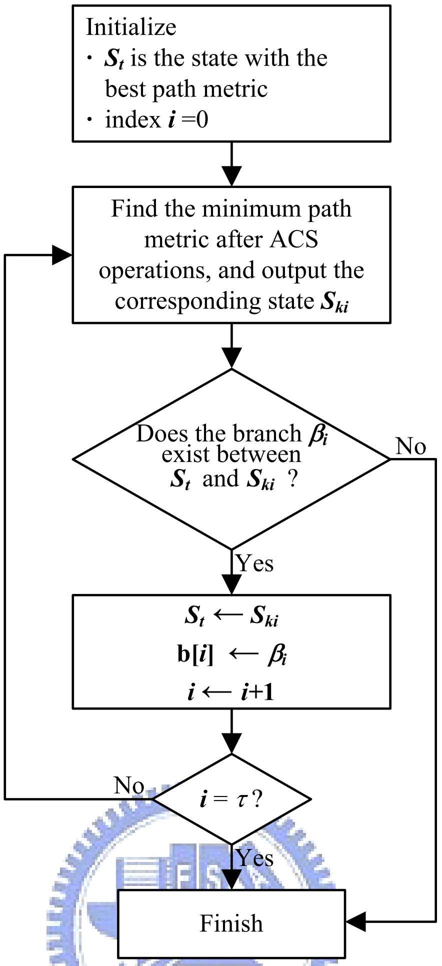
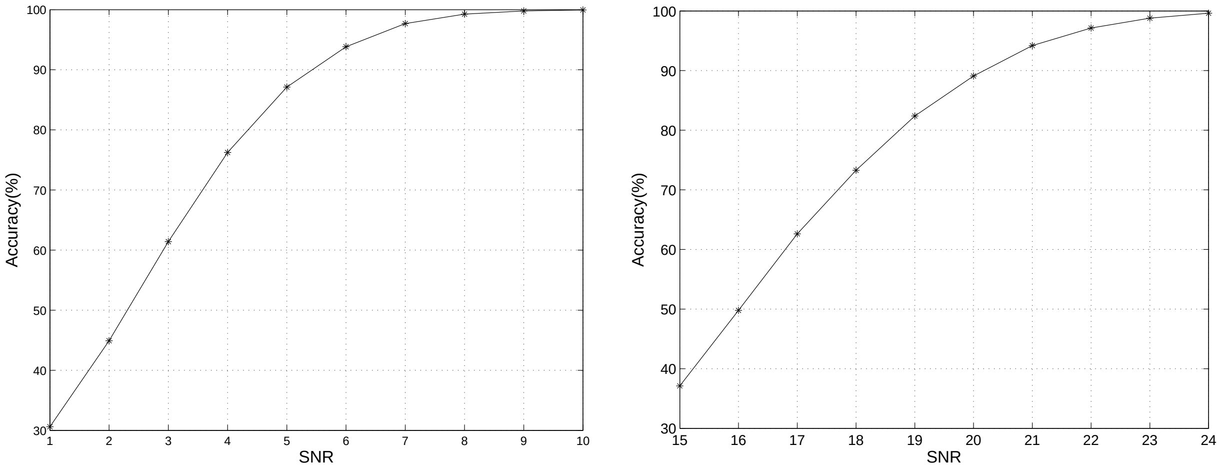

Initialize
· S_t is the state with the best path metric
· index i =0
Find the minimum path metric after ACS operations, and output the corresponding state S_{ki}
Does the branch β_i exist between S_t and S_{ki} ?
No
Yes
$S_t \leftarrow S_{ki}$
$\mathbf{b}[i] \leftarrow \beta_i$
$i \leftarrow i+1$
No
$i = \tau$?
Yes
Finish

Figure 3.49: The path prediction algorithm

Therefore, for any state transition from $S_{m'}^t$ at time t to $S_m^{(t+1)}$ at time $t+1$, we can estimate the joint probability

$$\begin{aligned}
\Pr\{S_{m'}^{(t)}, S_m^{(t+1)}, \mathbf{r}\} &= \Pr\{S_{m'}^{(t)}, S_m^{(t+1)}, \mathbf{r}_0^{t-1}, r_t, \mathbf{r}_{t+1}^{N-1}\} \\
&= \Pr\{\mathbf{r}_{t+1}^{N-1} | S_{m'}^{(t)}, S_m^{(t+1)}, \mathbf{r}_0^{t-1}, r_t\} \\
&\quad \times \Pr\{S_m^{(t+1)}, r_t | S_{m'}^{(t)}, \mathbf{r}_0^{t-1}\} \qquad (3.152) \\
&\quad \times \Pr\{S_{m'}^{(t)}, \mathbf{r}_0^{t-1}\} \\
&= \Pr\{\mathbf{r}_{t+1}^{N-1} | S_m^{(t+1)}\} \Pr\{S_m^{(t+1)}, r_t | S_{m'}^{(t)}\} \Pr\{S_{m'}^{(t)}, \mathbf{r}_0^{t-1}\}
\end{aligned}$$



(a)

(b)

Figure 3.50: Prediction accuracy in AWGN channel. (a) QPSK and $R = 1/2$. (b) 64-QAM and $R = 3/4$

Notice that $\mathbf{r}_0^{t-1}$ denotes the received sequence from time 0 to $t-1$, and $\mathbf{r}_{t+1}^{N-1}$ the received sequence from time $t+1$ to $N-1$. The second equation of (3.152) comes form Bayes' rule, and the third equation is due to the Markov process in the state transitions. We further define three functions:

$$\alpha(S_{m'}^{(t)}) = \Pr\{S_{m'}^{(t)}, \mathbf{r}_0^{t-1}\} \tag{3.153}$$

$$\gamma(S_{m'}^{(t)}, S_m^{(t+1)}) = \Pr\{S_m^{(t+1)}, r_t | S_{m'}^{(t)}\} \tag{3.154}$$

$$\beta(S_m^{(t+1)}) = \Pr\{\mathbf{r}_{t+1}^{N-1} | S_m^{(t+1)}\}, \tag{3.155}$$

and thus (3.152) can be rewritten as

$$\Pr\{S_{m'}^{(t)}, S_m^{(t+1)}, \mathbf{r}\} = \alpha(S_{m'}^{(t)})\gamma(S_{m'}^{(t)}, S_m^{(t+1)})\beta(S_m^{(t+1)}) \tag{3.156}$$

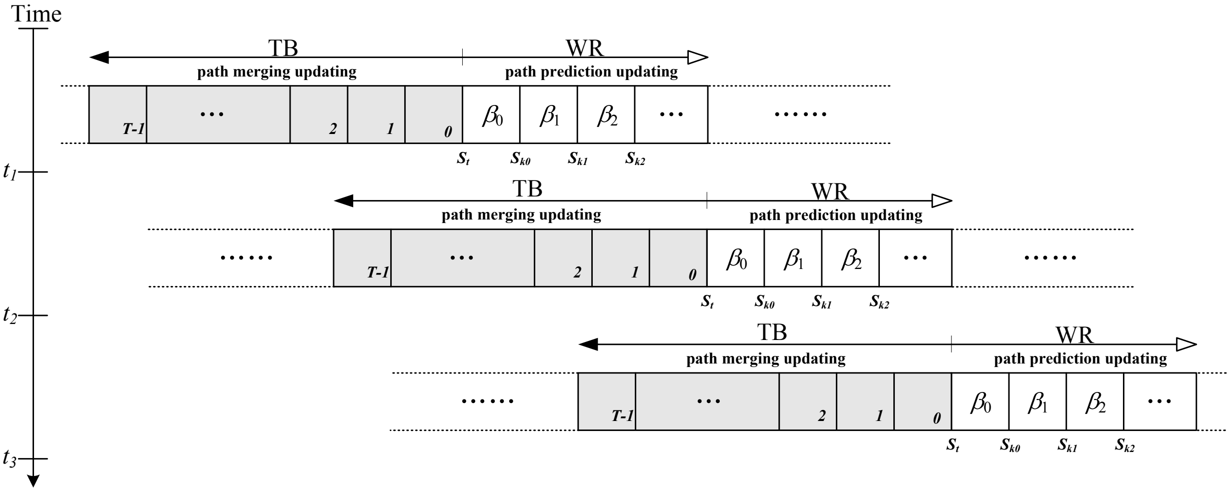


Figure 3.51: The buffer updating operations

On the other hand, we extend the definition of (3.153) to be

$$\begin{aligned}\alpha(S_m^{(t+1)}) &= \Pr\{S_m^{(t+1)}, \mathbf{r}_0^t\} \\ &= \sum_{S_{m'}^{(t)} \in \mathbf{S}} \Pr\{S_{m'}^{(t)}, S_m^{(t+1)}, \mathbf{r}_0^t\} \\ &= \sum_{S_{m'}^{(t)} \in \mathbf{S}} \Pr\{S_m^{(t+1)}, r_t, |S_{m'}^{(t)}, \mathbf{r}_0^{t-1}\} \Pr\{S_{m'}^{(t)}, \mathbf{r}_0^{t-1}\} \\ &= \sum_{S_{m'}^{(t)} \in \mathbf{S}} \Pr\{S_m^{(t+1)}, r_t, |S_{m'}^{(t)}\} \Pr\{S_{m'}^{(t)}, \mathbf{r}_0^{t-1}\} \\ &= \sum_{S_{m'}^{(t)} \in \mathbf{S}} \gamma(S_{m'}^{(t)}, S_m^{(t+1)}) \alpha(S_{m'}^{(t)}),\end{aligned} \tag{3.157}$$

where $\mathbf{S}$ is the set of all states. Similarly, we have

$$\begin{aligned}
\beta(S_{m'}^{(t)}) &= \sum_{S_m^{(t+1)}\in\mathbf{S}} \Pr\{S_m^{(t+1)}, \mathbf{r}_t^{N-1}|S_{m'}^{(t)}\} \\
&= \sum_{S_m^{(t+1)}\in\mathbf{S}} \Pr\{S_m^{(t+1)}, r_t, \mathbf{r}_{t+1}^{N-1}, S_{m'}^{(t)}\}/\Pr\{S_{m'}^{(t)}\} \\
&= \sum_{S_m^{(t+1)}\in\mathbf{S}} \Pr\{\mathbf{r}_{t+1}^{N-1}|S_m^{(t+1)}, r_t, S_{m'}^{(t)}\}\Pr\{S_m^{(t+1)}, r_t|S_{m'}^{(t)}\} \\
&= \sum_{S_m^{(t+1)}\in\mathbf{S}} \Pr\{\mathbf{r}_{t+1}^{N-1}|S_m^{(t+1)}\}\Pr\{S_m^{(t+1)}, r_t|S_{m'}^{(t)}\} \\
&= \sum_{S_m^{(t+1)}\in\mathbf{S}} \beta(S_m^{(t+1)})\gamma(S_{m'}^{(t)}, S_m^{(t+1)}),
\end{aligned} \tag{3.158}$$

and consequently the *forward metric* α in (3.153) and the *backward metric* β in (3.155) will be computed recursively with (3.157) and (3.158). If the encoder starts from $S_0^{(0)}$ and terminates at $S_0^{(N)}$, the following initial conditions are satisfied:

$$\begin{aligned}
\alpha(S_0^{(0)}) &= 1, \quad \alpha(S_x^{(0)}) = 0 \quad \text{for } S_x^{(0)} \in \mathbf{S}\backslash S_0 \\
\beta(S_0^{(N)}) &= 1, \quad \beta(S_x^{(N)}) = 0 \quad \text{for } S_x^{(N)} \in \mathbf{S}\backslash S_0
\end{aligned} \tag{3.159}$$

Finally, the *branch metric* in (3.154) can also be

$$\begin{aligned}
\gamma(S_{m'}^{(t)}, S_m^{(t+1)}) &= \frac{\Pr\{S_m^{(t+1)}, S_{m'}^{(t)}, r_t\}}{\Pr\{S_{m'}^{(t)}\}} \\
&= \frac{\Pr\{S_m^{(t+1)}, S_{m'}^{(t)}\}}{\Pr\{S_{m'}^{(t)}\}} \times \frac{\Pr\{S_m^{(t+1)}, S_{m'}^{(t)}, r_t\}}{\Pr\{S_m^{(t+1)}, S_{m'}^{(t)}\}} \\
&= \Pr\{S_m^{(t+1)}|S_{m'}^{(t)}\}\Pr\{r_t|S_m^{(t+1)}, S_{m'}^{(t)}\} \\
&= P(u_t)P(r_t|\hat{v}_t),
\end{aligned} \tag{3.160}$$

where u_t is the encoder input that causes the transition $S_{m'}^{(t)} \rightarrow S_m^{(t+1)}$, and $\hat{v}_t$ is the corresponding codeword. With the branch metric in (3.160), we can derive α and β for each state at different time instances; as a result, the joint probability (3.152) is available for all $S_{m'}^{(t)}, S_m^{(t+1)} \in \mathbf{S}$ and $t = 0 \sim N$.

The *a posteriori* the information for the symbol u_t is defined to be the log-likelihood ratio (LLR)

$$L(u_t) \triangleq \ln \frac{\Pr\{u_t = +1|\mathbf{r}\}}{\Pr\{u_t = -1|\mathbf{r}\}}, \tag{3.161}$$

and the decoder can decide the output

$$\hat{u}_t = \begin{cases} +1 & \text{if } L(u_t) \geq 0 \\ -1 & \text{if } L(u_t) < 0 \end{cases} \tag{3.162}$$

and the corresponding soft information $L(u_t)$. Combining with (3.152), we can express the APP of u_t as

$$\begin{aligned} L(u_t) &= \ln \frac{\sum_{(m',m)\in\mathbf{B}_t^{+1}} \Pr\{S_{m'}^{(t)}, S_m^{(t+1)}|\mathbf{r}\}}{\sum_{(m',m)\in\mathbf{B}_t^{-1}} \Pr\{S_{m'}^{(t)}, S_m^{(t+1)}|\mathbf{r}\}} \\ &= \ln \frac{\sum_{(m',m)\in\mathbf{B}_t^{+1}} \Pr\{S_{m'}^{(t)}, S_m^{(t+1)}, \mathbf{r}\}}{\sum_{(m',m)\in\mathbf{B}_t^{-1}} \Pr\{S_{m'}^{(t)}, S_m^{(t+1)}, \mathbf{r}\}} \end{aligned} \tag{3.163}$$

Among all possible state transitions $S_{m'}^{(t)} \to S_m^{(t+1)}$, $\mathbf{B}_t^{+1}$ is the set of all (m', m) that indicate the transitions are caused by $u_t = +1$, and $\mathbf{B}_t^{-1}$, the set of (m', m), denotes the transitions are due to $u_t = -1$. The MAP decoding algorithm not only decodes $\hat{u}_t$, but also estimates the APP for each u_t

$$L(u_t) = \ln \frac{\sum_{(m',m)\in\mathbf{B}_t^{+1}} \alpha(S_{m'}^{(t)})\gamma(S_{m'}^{(t)}, S_m^{(t+1)})\beta(S_m^{(t+1)})}{\sum_{(m',m)\in\mathbf{B}_t^{-1}} \alpha(S_{m'}^{(t)})\gamma(S_{m'}^{(t)}, S_m^{(t+1)})\beta(S_m^{(t+1)})} \tag{3.164}$$

Alternatively, the MAP algorithm can be defined in the logarithmic domain for computational efficiency. We first transfer the metrics to the logarithmic domain; that is

$$\bar{\gamma}(S_{m'}^{(t)}, S_m^{(t+1)}) = \ln \gamma(S_{m'}^{(t)}, S_m^{(t+1)}), \tag{3.165}$$

$$\bar{\alpha}(S_{m'}^{(t+1)}) = \ln \alpha(S_{m'}^{(t+1)}) = \ln \sum_{S_{m'}^{(t)} \in \mathbf{S}} e^{\bar{\gamma}(S_{m'}^{(t)}, S_m^{(t+1)}) + \bar{\alpha}(S_{m'}^{(t)})}, \tag{3.166}$$

and

$$\bar{\beta}(S_m^{(t)}) = \ln \beta(S_m^{(t)}) = \ln \sum_{S_m^{(t+1)} \in \mathbf{S}} e^{\bar{\beta}(S_m^{(t+1)}) + \bar{\gamma}(S_{m'}^{(t)}, S_m^{(t+1)})}. \tag{3.167}$$

Finally, the APP information in (3.164) will become

$$\begin{aligned} L(u_t) = \ln & \left[\sum_{(m',m) \in \mathbf{B}_t^{+1}} e^{\bar{\alpha}(S_{m'}^{(t)}) + \bar{\gamma}(S_{m'}^{(t)}, S_m^{(t+1)}) + \bar{\beta}(S_m^{(t+1)})} \right] \\ - \ln & \left[\sum_{(m',m) \in \mathbf{B}_t^{-1}} e^{\bar{\alpha}(S_{m'}^{(t)}) + \bar{\gamma}(S_{m'}^{(t)}, S_m^{(t+1)}) + \bar{\beta}(S_m^{(t+1)})} \right] \end{aligned} \tag{3.168}$$

Considering the following Jacobian function [141]

$$\ln(e^{x_1} + e^{x_2}) \triangleq \max{}^*(e^{x_1}, e^{x_2}) = \max(x_1, x_2) + \ln(1 + e^{-|x_1 - x_2|}), \tag{3.169}$$

and its extension

$$\ln(e^{x_1} + e^{x_2} + \cdots + e^{x_b}) \triangleq \max{}^*(e^{x_1}, e^{x_2}, \ldots, e^{x_b}), \tag{3.170}$$

$$= \max{}^*(\cdots \max{}^*(\max{}^*(x_1, x_2), x_3) \cdots, x_b), \tag{3.171}$$

we can write (3.166) and (3.167) in more simple forms:

$$\bar{\alpha}(S_{m'}^{(ti+1)}) = \max{}^*{}_{S_{m'}^{(t)} \in \mathbf{S}} [\bar{\gamma}(S_{m'}^{(t)}, S_m^{(t+1)}) + \bar{\alpha}(S_{m'}^{(t)})] \tag{3.172}$$

$$\bar{\beta}(S_m^{(t)}) = \max{}^*{}_{S_m^{(t+1)} \in \mathbf{S}} [\bar{\beta}(S_m^{(t+1)}) + \bar{\gamma}(S_{m'}^{(t)}, S_m^{(t+1)})], \tag{3.173}$$

and therefore,

$$\begin{aligned} L(u_t) = {} & {\max}^*_{(m',m)\in\mathbf{B}_t^{+1}}[\bar{\alpha}(S_{m'}^{(t)}) + \bar{\gamma}(S_{m'}^{(t)}, S_m^{(t+1)}) + \bar{\beta}(S_m^{(t+1)})] \\ & - {\max}^*_{(m',m)\in\mathbf{B}_t^{-1}}[\bar{\alpha}(S_{m'}^{(t)}) + \bar{\gamma}(S_{m'}^{(t)}, S_m^{(t+1)}) + \bar{\beta}(S_m^{(t+1)})]. \end{aligned} \tag{3.174}$$

The initial conditions become

$$\begin{aligned} \bar{\alpha}(S_0^{(0)}) = 0, \quad \bar{\alpha}(S_x^{(0)}) = -\infty \quad \text{for } S_x^{(0)} \in \mathbf{S}\backslash S_0 \\ \bar{\beta}(S_0^{(N)}) = 0, \quad \bar{\beta}(S_x^{(N)}) = -\infty \quad \text{for } S_x^{(N)} \in \mathbf{S}\backslash S_0 \end{aligned} \tag{3.175}$$

Note that (3.169) can be implemented with a max function as well as a lookup table for the term $\ln(1 + e^{-|x_1 - x_2|})$, leading to the much simple hardware design. Furthermore, according to the recursion in (3.171), the evaluations in (3.172), (3.173), and (3.174) will also benefit from the simplification of (3.169). The MAP decoding algorithm based on (3.172), (3.173), and (3.174) is termed Log-MAP algorithm [142, 143].

If the *a priori* information is represented by

$$L_a(u_t) \triangleq \ln \frac{P(u_t = +1)}{P(u_t = -1)}, \tag{3.176}$$

the *a priori* probability will be

$$P(u_t = \pm 1) = \frac{e^{\pm L_a(u_t)}}{1 + e^{\pm L_a(u_t)}} = \left[\frac{e^{-L_a(u_t)/2}}{1 + e^{-L_a(u_t)}}\right] e^{u_t L_a(u_t)/2} = A_t e^{u_t L_a(u_t)/2} \tag{3.177}$$

where A_t is independent of u_t. Furthermore, according to (3.80), the probability $P(r_t|\hat{v}_t)$ in the AWGN channel with $2\sigma^2 = N_0/E_s$ is

$$\begin{aligned} P(r_t|\hat{v}_t) &= \left(\frac{1}{\sqrt{2\pi\sigma^2}}\right)^n e^{-\frac{\sum_{i=1}^n (r_t^{(i)}-\hat{v}_t^{(i)})^2}{2\sigma^2}} \\ &= \left[\left(\frac{1}{\sqrt{2\pi\sigma^2}}\right)^n e^{-\frac{\sum_{i=1}^n [(r_t^{(i)})^2+(\hat{v}_t^{(i)})^2]}{2\sigma^2}}\right] e^{\frac{\sum_{i=1}^n r_t^{(i)}\cdot\hat{v}_t^{(i)}}{\sigma^2}} \\ &= B_t e^{\sum_{i=1}^n L_c\cdot r_t^{(i)}\cdot\hat{v}_t^{(i)}/2}. \end{aligned} \tag{3.178}$$

Notice that B_t is constant within each time instance since $r_t^{(i)}$ is the same for all branches and $\hat{v}_t^{(i)} = \pm 1$. Additionally, the channel reliability value L_c is defined to be $\frac{4E_s}{N_0}$ for the AWGN channel [144]. As a result, we will find that A_t and B_t will be canceled out in the LLR of either (3.164) or (3.174). The branch metric can then be expressed with both A_t and B_t being dropped;

$$\bar{\gamma}(S_{m'}^{(t)}, S_m^{(t+1)}) = \ln P(u_t)P(r_t|\hat{v}_t) = \frac{1}{2}(u_t L_a(u_t) + \sum_{i=1}^{n} L_c \cdot r_t^{(i)} \cdot \hat{v}_t^{(i)}) \tag{3.179}$$

The further simplification is achieved through discarding $\ln(1 + e^{-|x_1-x_2|})$ in (3.169). Consequently, we have the following approximations:

$$\max{}^*(e^{x_1}, e^{x_2}) \approx \max(x_1, x_2) \tag{3.180}$$

$$\max{}^*(e^{x_1}, e^{x_2}, \ldots, e^{x_n}) \approx \max_{i=1\sim n}(x_i). \tag{3.181}$$

Applying (3.180) and (3.181), we can reduce the Log-MAP algorithm to the Max-Log-MAP algorithm that contains only the additions and the max functions. However, the performance would degrade because of the information loss in (3.180) and (3.181).

In Fig. 3.52, the trellis diagram for the encoder (3.54), we demonstrate the MAP decoding in the the logarithmic domain. The dashed branches are generated by $u_t = -1$, and the

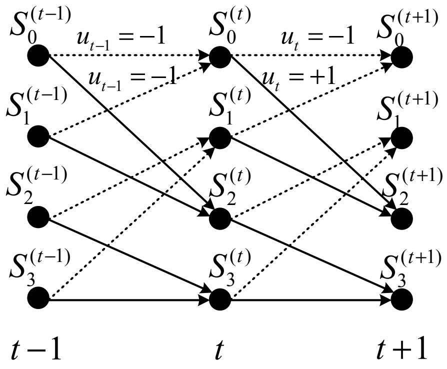


Figure 3.52: Trellis diagram for the MAP decoding algorithm

solid branches corresponds to $u_t = +1$. Accordingly, we know that

$$\mathbf{B}_t^{+1} = \{(0,2),(1,2),(2,3),(3,3)\} \tag{3.182}$$

$$\mathbf{B}_t^{-1} = \{(0,0),(1,0),(2,1),(3,1)\}; \tag{3.183}$$

the forward metric and the backward metric for $S_0^{(t)}$ should be

$$\bar{\alpha}(S_0^{(t)}) = \max{}^*[\bar{\alpha}(S_0^{(t-1)}) + \bar{\gamma}(S_0^{(t-1)}, S_0^{(t)}), \bar{\alpha}(S_1^{(t-1)}) + \bar{\gamma}(S_1^{(t-1)}, S_0^{(t)})] \tag{3.184}$$

$$\bar{\beta}(S_0^{(t)}) = \max{}^*[\bar{\beta}(S_0^{(t+1)}) + \bar{\gamma}(S_0^{(t+1)}, S_0^{(t)}), \bar{\beta}(S_2^{(t+1)}) + \bar{\gamma}(S_2^{(t+1)}, S_0^{(t)})]. \tag{3.185}$$

Generally, in the Max-Log-MAP decoding where the max* is simplified to the max function, both (3.184) and (3.185) are ACS operations that is similar to the Viterbi decoding algorithm. Furthermore, according to (3.80), the branch metric in the AWGN channel with $L_c = \frac{4E_s}{N_0}$ is

$$\bar{\gamma}(S_{m'}^{(t)}, S_m^{(t+1)}) = \frac{1}{2}(u_t L_a(u_t) + r_t^{(1)}\hat{v}_t^{(1)} + r_t^{(2)}\hat{v}_t^{(2)}), \tag{3.186}$$

and $L_a(u_t) = 0$ if the information bits are assumed to be equally likely, or $P(u_t = +1) = P(u_t = -1)$. We can evaluate the APP of u_t with (3.174) after evaluate $\bar{\alpha}$ for each state at time t, $\bar{\beta}$ for each state at time $t+1$, and all branch metrics between t and $t+1$.

In the MAP decoding algorithm,the whole codeword sequence as well as all the metrics

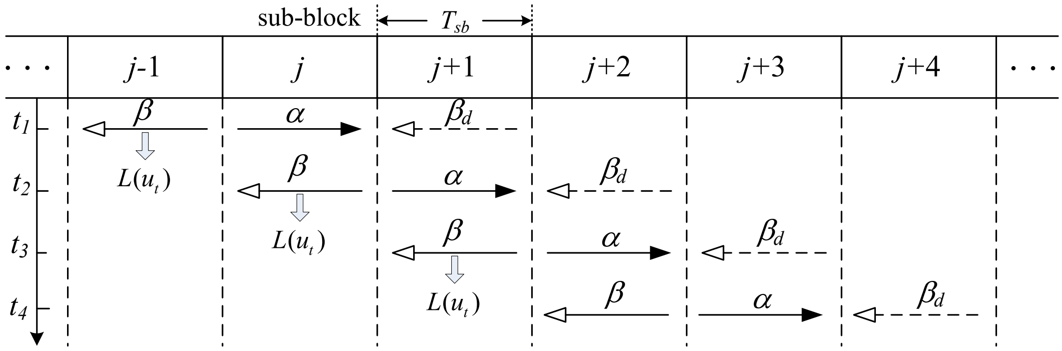


Figure 3.53: The windowed MAP algorithm

should be kept to calculate all $L(u_t)$ with $t = 1 \sim N$. It is impractical to implement a MAP decoder with large N. To reduce the memory requirement, the sliding window algorithm [145, 146] is applied to avoid storing the metrics corresponding to the entire codeword sequence. In Fig. 3.53, the codeword stream is divided into $\lceil N/T_{sb} \rceil$ sub-blocks of length T_{sb}, and the dummy backward recursion β_d is employed to establish the initial conditions for the true backward recursion β. Note that β_d is an operation that is similar to the traceback in the Viterbi algorithm. Although the initial condition for the β_d recursion is unknown except the last sub-block, we set the equally likely conditions for β_d within the $(j+1)$-th sub-block

$$\beta_d(S_m^{((j+1)\cdot T_{sb})}) = \frac{1}{M}, \quad \text{for all } S_m^{(j\omega)} \in \mathbf{S}. \tag{3.187}$$

After the β_d process of T_{sb} time instances, the initial metrics $\beta(S_m^{(j\cdot T_{sb})})$ in the j-th sub-block are available for the β recursion. During the $(j+1)$-th β_d operation, the forward α recursion proceeds concurrently in the j-th sub-block, and all the metric values are stored in the memory. In the backward β recursion of the j-th sub-block, we can calculated the $L(u_t)$ value with the α metrics in the memory, the β metrics in computing, and the corresponding branches metrics in the j-the sub-block. The length T_{sb} which is set to be 6ν is sufficient to ensure the reliable initialization for the β recursion [146].

Chapter 4

Soft Iterative Decoding

4.1 Message passing algorithm

The soft iterative decoding algorithm relies on the *message passing* or *belief propagation* algorithm [26, 27]. We consider the following conditional probability

$$P(x = a|C) \tag{4.1}$$

which is the *a posteriori* probability of x based on the knowledge of the constraint C. According to the Bayes' theorem, we can express (4.1) as

$$P(x = a|C) = \frac{P(C|x = a)P(x = a)}{P(C)}. \tag{4.2}$$

The term $P(x = a)$ is the *a priori* probability and is also referred to the *instrinsic* probability for x [147], denoted by $P_{int}(x = a)$. On the other hand, $P(C|x = a)$ is termed the *extrinsic* probability with respect to C. The extrinsic probability, defined by

$$P_{ext}(x = a) = (\sum_{a' \in \mathbf{A}} P(C|x = a'))^{-1} P(C|x = a) = \rho_e P(C|x = a) \tag{4.3}$$

provides a new information for x according to the constraint C, assuming a takes values from the alphabet set $\mathbf{A}$. Consequently, the *a posteriori* probability in (4.2) can be written as

$$P_{post}(x = a) = P(x = a|C) = \rho_p P_{ext}(x = a) P_{int}(x = a), \tag{4.4}$$

where $\rho_p = (\rho_e P(C))^{-1}$. If $\mathbf{A} = GF(2)$, the log-likelihood ratio representation for (4.4) will be

$$L_{post}(x) = \ln \frac{P_{post}(x=1)}{P_{post}(x=0)} = \ln \frac{P_{ext}(x=1)}{P_{ext}(x=0)} + \ln \frac{P_{int}(x=1)}{P_{int}(x=0)} = L_{ext}(x) + L_{int}(x). \tag{4.5}$$

In the graph representation, we use an undirected graph, referred to the *normal graph* [147,148], in which the constraints are denoted by vertices (nodes), state variables for message passing are denoted by ordinary edges, and symbol variables are denoted by left edges (half edges). Fig. 4.1 shows an example with three vertices; the edges connecting two vertices are ordinary edges, and the edges connecting only one vertex are left edges. Fig. 4.2 illustrates a graph consisting of a single vertex and d edges, which are all left edges. There

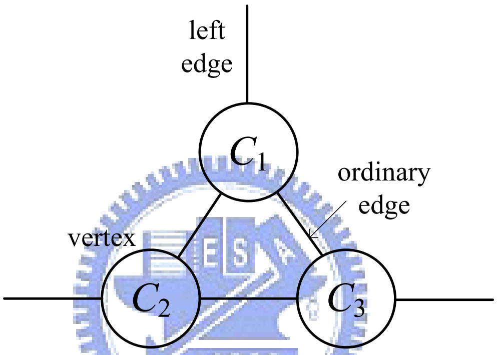


Figure 4.1: An example of normal graph

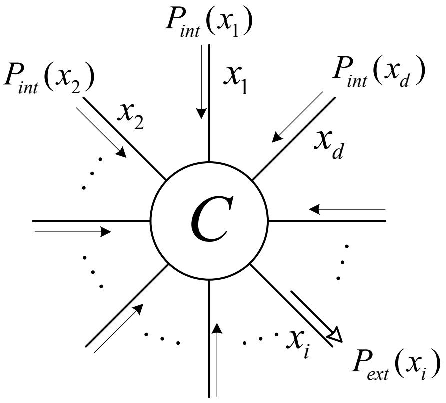


Figure 4.2: Graph representation of the extrinsic and the instrinsic probabilities

are d symbols, $x_1, x_2, ...,$ and x_d, that correspond to the constraint C. We define a set $\mathbf{S}_C$ which is a subspace of the d-dimensional vector space $\mathbf{A}^d$ ($\mathbf{S}_C \subset \mathbf{A}^d$), and any d-tuple $\mathbf{x} = (x_1, x_2, \ldots, x_d) \in \mathbf{S}_C$ will satisfy the constraint C. Assume each edge has the instrinsic probability $P_{int}(x_j)$ associated with the symbol x_j for $j = 1 \sim d$, the *a posteriori* probability of a symbol x_i with respect to C will be the combination of the intrinsic $P_{int}(x_i)$ and the extrinsic $P_{ext}(x_i)$. Therefore, we have to evaluate $P_{ext}(x_i)$ based on the constraint C and the other instrinsic probabilities $P_{int}(x_j)$ with $j \neq i$. The extrinsic probability is

$$\begin{aligned}
P_{ext}(x_i) &= \rho_c P(C|x_i) \\
&= \rho_c \sum_{\substack{x_j, \forall j \neq i \\ \mathbf{x} \in \mathbf{S}_C}} P(x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_d) \\
&= \rho_c \sum_{\substack{x_j, \forall j \neq i \\ \mathbf{x} \in \mathbf{S}_C}} \prod_{\substack{j=1 \\ j \neq i}}^{d} P_{int}(x_j)
\end{aligned} \tag{4.6}$$

where we assume the symbol variables $x_1, x_2, \ldots, x_d$ are independent, and ρ_c is a normalization constant.

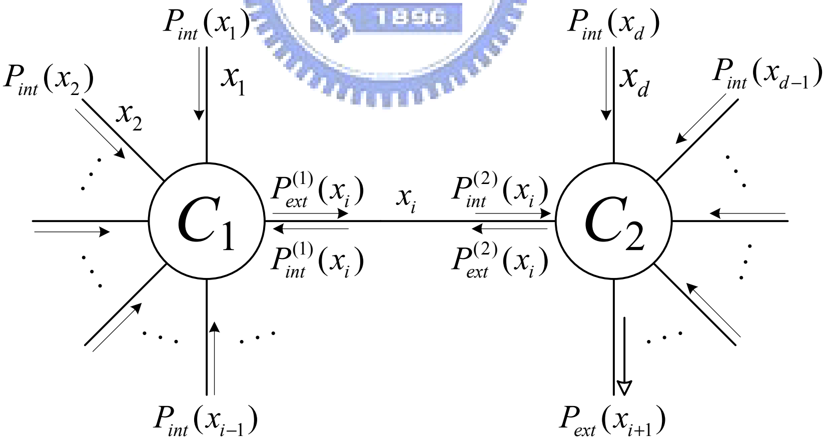


Figure 4.3: Graph representation of the message passing between two vertices

Additionally, we consider the graph with two vertices (constraints), C_1 and C_2, as shown in Fig. 4.3. The constraint C_1 has $i-1$ left edges and one ordinary edge, corresponding to the symbols $x_1 \sim x_{i-1}$ and x_i. On the other hand, $x_i \sim x_d$ are constrained by C_2 where only x_i is on the ordinary edges. We also define two constraint sets $\mathbf{S}_{C_1}$ and $\mathbf{S}_{C_2}$ such that any $\mathbf{x}_1 = (x_1, x_2, \ldots, x_i) \in \mathbf{S}_{C_1}$ and $\mathbf{x}_2 = (x_i, x_{i+1}, \ldots, x_d) \in \mathbf{S}_{C_2}$ will respectively satisfy C_1 and C_2. Similar to the single vertex graph in Fig. 4.2, we want to estimate the extrinsic probability for the left edge based on both C_1 and C_2. As shown in Fig. 4.3, the symbol x_{i+1} is first considered. If we only consider the constraint C_2, the extrinsic probability can be written as

$$P_{ext}(x_{i+1}) = \rho_2 P(C_2|x_{i+1}) = \sum_{\substack{\mathbf{x}_2 \backslash x_{i+1} \\ \mathbf{x}_2 \in \mathbf{S}_{C_2}}} P_{int}^{(2)}(x_i) \prod_{j=i+2}^{d} P_{int}(x_j) \tag{4.7}$$

according to the results in (4.6). However, the instrinsic probability $P_{int}^{(2)}(x_i)$ for C_2 is on the ordinary edge that is unable to be acquired form the inputs. In the following, we use both C_1 and C_2 to evaluate

$$P_{ext}(x_{i+1}) = \rho_C P(C_1, C_2|x_{i+1}). \tag{4.8}$$

The above conditional probability is rewritten as

$$\begin{aligned} P(C_1, C_2|x_{i+1}) &= \sum_{\substack{\mathbf{x}_2 \backslash x_{i+1} \\ \mathbf{x}_2 \in \mathbf{S}_{C_2}}} P(C_1, C_2, x_i, x_{i+2}, \ldots, x_d|x_{i+1}) \\ &= \sum_{\substack{\mathbf{x}_2 \backslash x_{i+1} \\ \mathbf{x}_2 \in \mathbf{S}_{C_2}}} P(C_2|C_1, \mathbf{x}_2) P(C_1, x_i, x_{i+2}, \ldots, x_d|x_{i+1}) \\ &= \sum_{\substack{\mathbf{x}_2 \backslash x_{i+1} \\ \mathbf{x}_2 \in \mathbf{S}_{C_2}}} P(C_1, x_i, x_{i+2}, \ldots, x_d|x_{i+1}), \end{aligned} \tag{4.9}$$

where the third equality comes from the fact that C_1, x_i, and C_2 in Fig. 4.3 from a Markov chain [147], and

$$P(C_1, C_2|x_i) = P(C_1|x_i) P(C_2|x_i). \tag{4.10}$$

Therefore, we can write

$$P(C_2|C_1,\mathbf{x}_2) = P(C_2|\mathbf{x}_2) = 1, \quad \text{for } \mathbf{x}_2 \in S_{C_2} \tag{4.11}$$

because $\mathbf{x}_2$ contains x_i. Continuing form (4.9), we can further derive the following factorization:

$$\begin{aligned} P(C_1, x_i, x_{i+2}, \ldots, x_d|x_{i+1}) &= P(C_1|\mathbf{x}_2)P(x_i, x_{i+2}, \ldots, x_d|x_{i+1}) \\ &= P(C_1|x_i)P(x_i)P(x_{i+2})\cdots P(x_d) \\ &= (\rho_1)^{-1}P_{ext}^{(1)}(x_i)P_{int}(x_i)\prod_{j=i+2}^{d} P_{int}(x_j). \end{aligned} \tag{4.12}$$

Notice that

$$P_{ext}^{(1)}(x_i) = \rho_1 P(C_1|x_i) \tag{4.13}$$

is the extrinsic probability of x_i with respect to C_1, $P_{int}(x_j)$ are instrinsic probabilities for the left edges connecting C_2, and $P_{int}(x_i)$ is the instrinsic probability for the ordinary edge variable x_i. Since the ordinary edge connects C_1 and C_2 without external input, the probability $P_{int}(x_i)$ can be initialized to be a constant; that is, $P_{int}(x_i) = \frac{1}{|\mathbf{A}|}$ for $x_i \in \mathbf{A}$. After the above derivation, the extrinsic probability in (4.9) can be expressed as

$$P_{ext}(x_{i+1}) = \rho_C' \sum_{\substack{\mathbf{x}_2 \backslash x_{i+1} \\ \mathbf{x}_2 \in \mathbf{S}_{C_2}}} P_{ext}^{(1)}(x_i) \prod_{j=i+2}^{d} P_{int}(x_j), \tag{4.14}$$

and $\rho_C' = \rho_C/(\rho_1|\mathbf{A}|)$. Referring to (4.8) , we can find if the extrinsic probability $P_{ext}^{(1)}(x_i)$ from C_1 is available, and

$$P_{int}^{(2)}(x_i) = P_{ext}^{(1)}(x_i), \tag{4.15}$$

only the constraint C_2 is necessary for estimating $P_{ext}(x_{i+1})$. Correspondingly, $P_{ext}(x_j)$ for $j = (i+2) \sim d$ can also be calculated. For $P_{ext}(x_l)$ with $l = 1 \sim (i-1)$, the extrinsic probability $P_{ext}^{(2)}(x_i)$ with respect to C_2 should be first computed, and the instrinsic probability for C_1 is set to be

$$P_{int}^{(1)}(x_i) = P_{ext}^{(2)}(x_i). \tag{4.16}$$

The process of (4.15) or (4.16) is the message passing between vertices C_1 and C_2. With the message passing algorithm, the problem of solving both C_1 and C_2 is decomposed into solving the single vertex graph, which is much simpler than the two vertices case. The message passed on the edge x_i can be represented by

$$\mu_{C_1 \to C_2}(x_i) = P_{ext}^{(1)}(x_i) = \rho_1 \sum_{\substack{\mathbf{x}_1 \backslash x_i \\ \mathbf{x}_1 \in \mathbf{S}_{C_1}}} \prod_{j=1}^{i-1} P_{int}(x_j) \tag{4.17}$$

$$\mu_{C_2 \to C_1}(x_i) = P_{ext}^{(2)}(x_i) = \rho_2 \sum_{\substack{\mathbf{x}_2 \backslash x_i \\ \mathbf{x}_2 \in \mathbf{S}_{C_2}}} \prod_{j=i+1}^{d} P_{int}(x_j) \tag{4.18}$$

Moreover, the operation in calculating $\mu_{C_1 \to C_2}(x_i)$ or $\mu_{C_2 \to C_1}(x_i)$ is the sum of products, and thus the message passing algorithm is also termed the *sum-product* algorithm [149].

In a graph with vertices, C_0, C_1,$\cdots$, and C_d, the vertex C_0 has d ordinary edges that respectively connect to C_1, C_2,$\cdots$, C_d with symbol variables x_1, x_2, $\cdots$, x_d. Assume the messages $\mu_{C_j \to C_0}(x_j)$ with $j = 1 \sim d$ have been obtained from $C_1 \sim C_d$, we can calculate $\mu_{C_0 \to C_i}$ by

$$\mu_{C_0 \to C_i}(x_i) = \sum_{\substack{\mathbf{x} \backslash x_i \\ \mathbf{x} \in \mathbf{S}_{C_0}}} \prod_{\substack{j=1, \\ j \neq i}}^{d} \mu_{C_j \to C_0}(x_j) \tag{4.19}$$

where $\mathbf{S}_{C_0}$ is the constraint set for C_0, and $\mathbf{x} = (x_1, x_2, \ldots, x_d)$. For $i = 1 \sim d$, the messages $\mu_{C_0 \to C_i}(x_i)$ can be found and become the instrinsic probability inputs for the vertices $C_1 \sim C_d$.

The above discussion of the message passing algorithm is based on the graph without cycles. In the graph with cycles [150], the message passing algorithm will be somewhat different from that in the graph without cycles. In Fig. 4.4, the graph has four vertices that

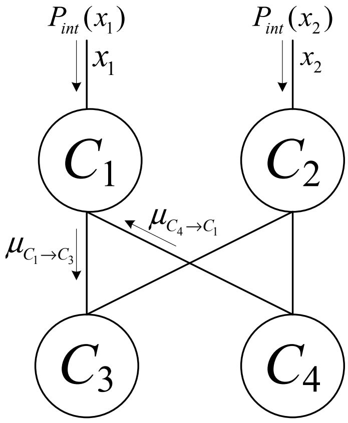


Figure 4.4: Graph with cycles

form a cycle and and two left edges having probability inputs. The message calculation form C_1 to C_3 will need $\mu_{C_4 \to C_1}$, the message form C_4. Nevertheless, $\mu_{C_4 \to C_1}$ also requires the messages from C_2 as well as C_3 that depends on $\mu_{C_1 \to C_3}$. It is impossible to derive $\mu_{C_4 \to C_1}$ with only the probabilities on the left edges. Hence we have to directly initialize $\mu_{C_4 \to C_1}$ to a constant probability, and the message passing algorithm can proceed as in the graph without cycles. If the variables $x_i \in \mathbf{A}$ for all i, the initial value of $\mu_{C_4 \to C_1}$ can be set to be $\frac{1}{|\mathbf{A}|}$, where $|\mathbf{A}|$ denotes the order of $\mathbf{A}$. Furthermore, the independency assumption among all edge variables would no longer hold, and the extrinsic probability based on the sum-product computation becomes an approximation.

The message passing schedule determines the computation order for a graph and also effects the computational complexity as well as the final results. The detailed discussion can be found in [147], [149], and [28]. Generally, some stopping criteria should be defined to terminate the message passing algorithm. Each block will be detailed in the following.

4.2 Turbo code

The parallel concatenated convolutional codes (PCCC), named turbo code [17,18], has been widely adopted in wireless communication systems [59,63,151,152]. Turbo code can achieve an excellent coding performance with simple constituent codes concatenated by a interleaver whose length is N.

4.2.1 Turbo encoder

Fig. 4.5 is a turbo encoder with two recursive systematic convolutional (RSC) encoders and an interleaver. It is shown in [153] that the constituent encoder must be recursive for better performance. In the first encoder, the information symbols are encoded to the systematic part $\mathbf{v}_0(D)$ and the parity $\mathbf{v}_1(D)$; thus, $\mathbf{v}_0(D) = \mathbf{u}(D)$. The second encoder encodes $\tilde{\mathbf{u}}(D)$, the information sequence $\mathbf{u}(D)$ after interleaving. However, the systematic part which is also $\tilde{\mathbf{u}}(D)$ will be discarded during transmission because $\mathbf{v}_0$ has carried the information sequence. The two constituent encoders are typically identical; nevertheless, they can be different and have been shown to achieve better performance [154,155]. If the code rates of encoder 1 and encoder 2 are R_1 and R_2 respectively, the overall code rate R in Fig. 4.5 will satisfy

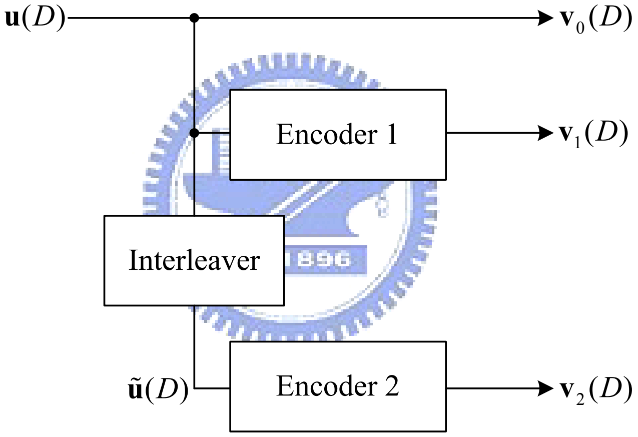


Figure 4.5: Turbo encoder

$$\frac{1}{R} = \frac{1}{R_1} + \frac{1}{R_2} - 1. \tag{4.20}$$

The interleaver permutes the information sequence $\mathbf{u}(D)$ to a new one $\tilde{\mathbf{u}}(D)$. The size and the permutation will considerably affect the turbo code performance. At low SNRs, the interleaver size has the most important effect, whereas the permutation would dominate the error performance at high SNRs. Conceptually, the interleaver is used to generate the code with a long block length from smaller constituent codes, and it decorrelates the two smaller encoders, or $\mathbf{u}(D)$ and $\tilde{\mathbf{u}}(D)$, to improve the decoding quality based on the iterative message passing algorithm. If we assume the interleaver performs random permutation, the error probability can be reduced by a factor of $1/N$ [103, 153], which is also referred to the interleaver gain. The graph model for the turbo encoder in Fig. 4.5 is also illustrated in Fig. 4.6 [27,28,156]. The graph is separated by the interleaver. The upper part comprise the vertices for the information bits $\mathbf{u}(D)$, the state sequence of the first encoder, and the parity symbols $\mathbf{v}_1(D)$. In the lower part, the permuted information $\tilde{\mathbf{u}}(D)$ is used as the inputs to produce the state sequence in the second encoder as well as the parity symbols $\mathbf{v}_2(D)$.

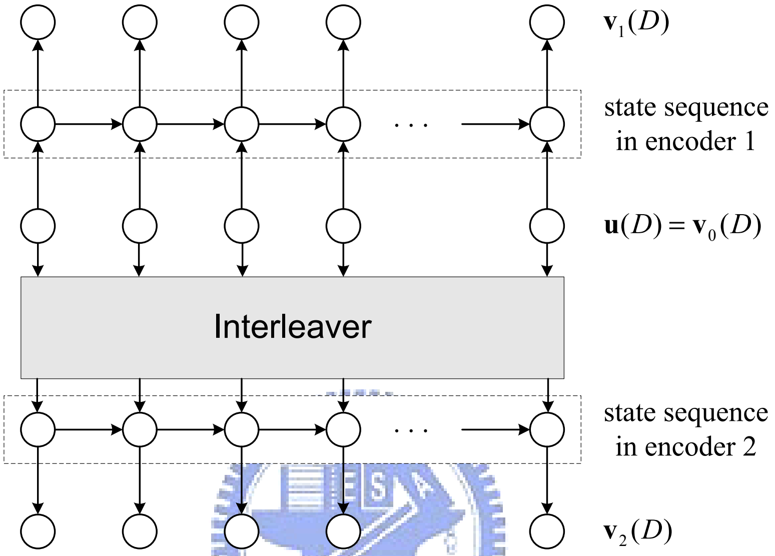

$\mathbf{v}_1(D)$
state sequence in encoder 1
$\mathbf{u}(D) = \mathbf{v}_0(D)$
Interleaver
state sequence in encoder 2
$\mathbf{v}_2(D)$

Figure 4.6: Graph representation of the turbo encoder

Due to the iterative decoding algorithm using soft-in/soft-out (SISO) decoders [144], an excellent coding performance can be achieved with simple constituent decoders concatenated by interleavers. The coding gain of turbo codes is better than that of convolutional codes on the basis of comparable complexity. Therefore, turbo codes are used in many mobile communication devices due to their outstanding performance and moderate complexity. However, the iterative decoding in turbo decoders limits the decoding speed and increases the decoding latency.

4.2.2 Iterative decoding of turbo codes

The decoder iteratively decodes the PCCC with the MAP algorithm that calculates *a posteriori* probability of each information bit u_t [16]. For a rate $1/n$ RSC encoder, each codeword frame consists of one systematic bit and $(n-1)$ parity bits. In the receiver, the received codeword has the systematic symbol $r_{0,t}$ and the parity symbols $r_t^{(1)} \sim r_t^{(n-1)}$, ans then the branch metric in logarithmic domain should be

$$\bar{\gamma}(S_{m'}^{(t)}, S_m^{(t+1)}) = \ln P(u_t)P(r_t|\hat{v}_t) = \frac{1}{2}(u_t L_a(u_t) + L_c u_t r_{0,t} + \sum_{i=1}^{n-1} L_c r_t^{(i)} \hat{v}_t^{(i)}) \tag{4.21}$$

which is from (3.179). As a result, the APP information from the SISO decoder can be derived as follows:

$$\begin{aligned} L(u_t) &= \ln \frac{\sum_{(m',m)\in \mathbf{B}_t^{+1}} \left[e^{\bar{\alpha}(S_{m'}^{(t)}) + \bar{\gamma}(S_{m'}^{(t)}, S_m^{(t+1)}) + \bar{\beta}(S_m^{(t+1)})} \right]}{\sum_{(m',m)\in \mathbf{B}_t^{-1}} \left[e^{\bar{\alpha}(S_{m'}^{(t)}) + \bar{\gamma}(S_{m'}^{(t)}, S_m^{(t+1)}) + \bar{\beta}(S_m^{(t+1)})} \right]} \\ &= \ln \frac{\sum_{(m',m)\in \mathbf{B}_t^{+1}} \left[e^{\frac{1}{2}((+1)L_a(u_t) + (+1)L_c r_{,0t})} \right] \left[e^{\bar{\alpha}(S_{m'}^{(t)}) + \frac{1}{2}\sum_{i=1}^{n-1} L_c r_t^{(i)} \hat{v}_t^{(i)} + \bar{\beta}(S_m^{(t+1)})} \right]}{\sum_{(m',m)\in \mathbf{B}_t^{-1}} \left[e^{\frac{1}{2}((-1)L_a(u_t) + (-1)L_c r_{0,t})} \right] \left[e^{\bar{\alpha}(S_{m'}^{(t)}) + \frac{1}{2}\sum_{i=1}^{n-1} L_c r_t^{(i)} \hat{v}_t^{(i)} + \bar{\beta}(S_m^{(t+1)})} \right]} \\ &= L_a(u_t) + L_c r_{0,t} + \ln \frac{\sum_{(m',m)\in \mathbf{B}_t^{+1}} \left[e^{\bar{\alpha}(S_{m'}^{(t)}) + \frac{1}{2}\sum_{i=1}^{n-1} L_c r_t^{(i)} \hat{v}_t^{(i)} + \bar{\beta}(S_m^{(t+1)})} \right]}{\sum_{(m',m)\in \mathbf{B}_t^{-1}} \left[e^{\bar{\alpha}(S_{m'}^{(t)}) + \frac{1}{2}\sum_{i=1}^{n-1} L_c r_t^{(i)} \hat{v}_t^{(i)} + \bar{\beta}(S_m^{(t+1)})} \right]} \\ &= L_a(u_t) + L_c r_{0,t} + L_e(u_t). \end{aligned} \tag{4.22}$$

The term $L_e(u_t)$ is the *extrinsic* information corresponding to the information bit u_t [17, 18].

In the decoder, we receive the systematic sequence $\mathbf{r}_0(D)$ as well as the parity sequences $\mathbf{r}_1(D)$ and $\mathbf{r}_2(D)$ from encoder 1 and encoder 2. In the decoding flow shown in Fig. 4.7, there are two SISO decoders for the two constituent encoders in Fig. 4.5. Initially, we set the

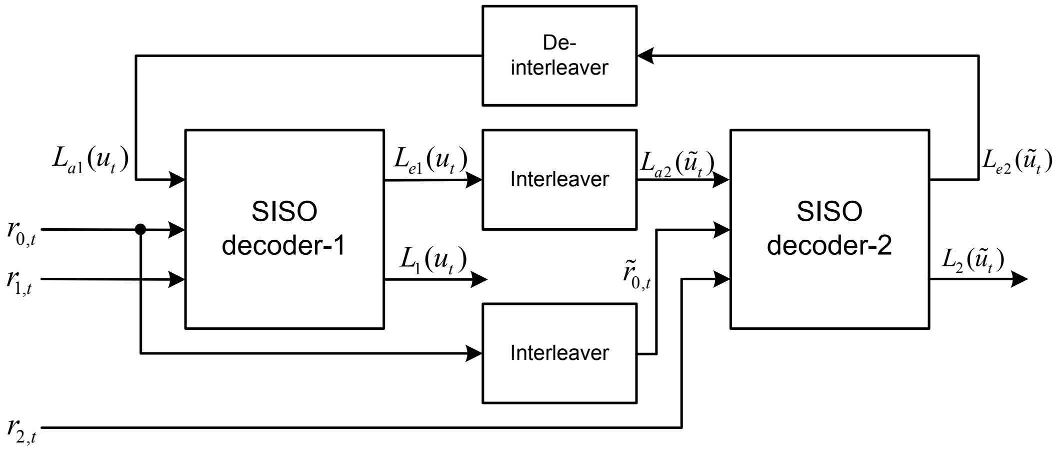


Figure 4.7: The turbo decoding flow

a priori information $L_{a1}(u_t)$ for the first decoder to zero and apply the BCJR algorithm to calculate the *a posteriori* information $L_1(u_t)$. From (4.22), the extrinsic information $L_{e1}(u_t)$ can be obtained

$$L_{e1}(u_t) = L_1(u_t) - L_c r_{0,t} - L_{a1}(u_t), \tag{4.23}$$

where $L_{a1}(u_t)i = 0$ initially. In the SISO decoder-2, the inputs are $\tilde{\mathbf{r}}_0(D)$ permuted from the systematic part $\mathbf{r}_0(D)$ and the parity sequence $\mathbf{r}_2(D)$, while the *a priori* information $L_{a2}(\tilde{u}_t)$ is the extrinsic output $L_{e1}(u_t)$ from decoder-1 after permutation. Consequently, we can evaluate the *a posteriori* output $L_2(\tilde{u}_t)$ and the extrinsic information $L_{e2}(\tilde{u}_t)$ corresponding to the second constituent code by

$$L_{e2}(\tilde{u}_t) = L_2(\tilde{u}_t) - L_c \tilde{r}_{0,t} - L_{a2}(\tilde{u}_t). \tag{4.24}$$

As shown in Fig. 4.7, the information $L_{e2}(\tilde{u}_t)$ can be regarded as the the *a priori* information $L_{a1}(u_t)$ for SISO decoder-1 after being reordered by the de-interleaver. The BCJR algorithm proceeds again for the first constituent code based on the information $L_{a1}(u_t)$ from SISO decoder-2. The turbo decoding proceeds iteratively with the extrinsic information passing between the two SISO decoders. When the stopping criteria are reached, which may be the maximum iteration number or a correctly decoded codeword, the APP information $L_2(\tilde{u}_t)$ is

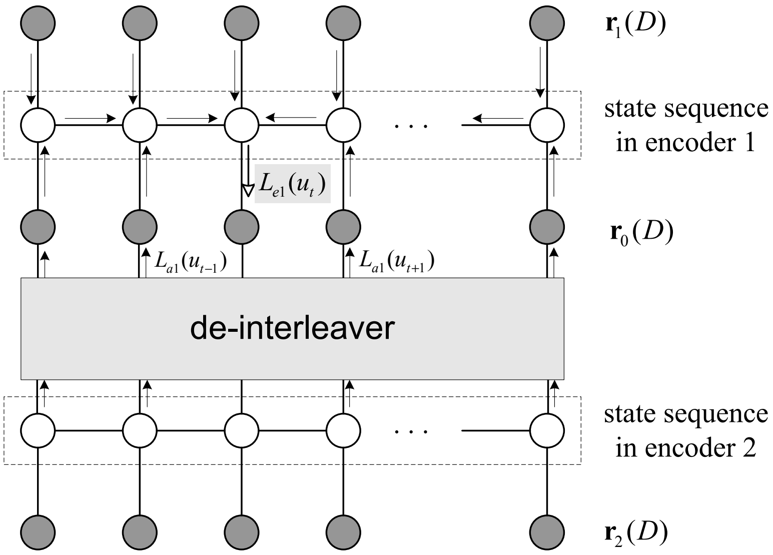


Figure 4.8: Graph representation of the turbo decoding

exported for hard decision. Fig. 4.8 also presents the message passing on the graph when the SISO decoder-1 computes the extrinsic information $L_{e1}(u_t)$ with the *a priori* information $L_{a1}(u_t)$ from the SISO decoder-2.

The turbo encoder specified in the 3GPP2 system [63] is demonstrated in Fig. 4.9. The constituent RSC encoder is

$$G(D) = \begin{bmatrix} 1 & \frac{1+D+D^3}{1+D^2+D^3} & \frac{1+D+D^2+D^3}{1+D^2+D^3} \end{bmatrix} \tag{4.25}$$

whose rate is 1/3. Since the systematic symbol $\tilde{v}_{0,t}$ in the second encoder is dropped during the codeword transmission, the overall code rate of the turbo encoder is $R = 1/5$. The maximum interleaver size in $N = 20730$, and thus each information sequence

$$\mathbf{u}(D) = \sum_{t=1}^{N} u_t D^t \tag{4.26}$$

has 20730 bits. The codeword is composed of the systematic sequence

$$\mathbf{v}_0(D) = \sum_{t=1}^{N} v_{0,t} D^t = \mathbf{u}(D), \tag{4.27}$$

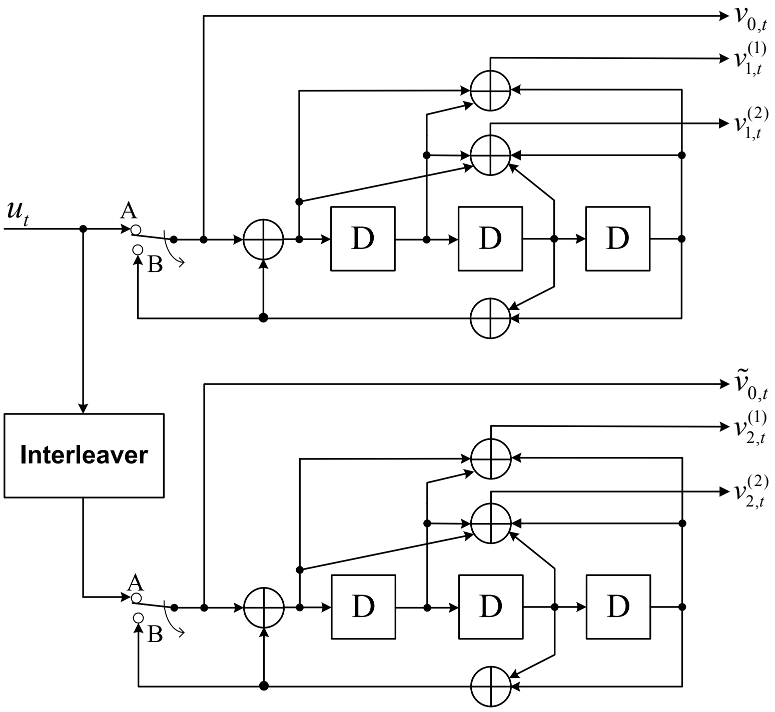


Figure 4.9: The turbo encoder specified in the 3GPP2 system

the first parity sequence

$$\mathbf{v}_1(D) = \sum_{t=1}^{N} (v_{1,t}^{(1)}, v_{1,t}^{(2)}) D^t, \tag{4.28}$$

and the second parity sequence

$$\mathbf{v}_2(D) = \sum_{t=1}^{N} (v_{2,t}^{(1)}, v_{2,t}^{(2)}) D^t. \tag{4.29}$$

After the encoding process, we charge the switch positions to B so that the states of both constituent codes returns to zero states. Hence additional 18 tail bits should be transmitted [63]; the first nine tail bits are generated from the first constituent encoder outputs $(v_{0,t}, v_{1,t}^{(1)}, v_{1,t}^{(2)})$, while the last nine bits are from $(\tilde{v}_{0,t}, v_{2,t}^{(1)}, v_{2,t}^{(2)})$ in the second constituent encoder. In the turbo decoder, we apply the iterative decoding based on the BCJR algorithm as shown in Fig. 4.7. Assume the 16-QAM modulation and the AWGN channel, the BER performance is presented in Fig. 4.10, including different iteration numbers. Notice that both SISO decoders in Fig. 4.10 will complete once within each decoding iteration. We can

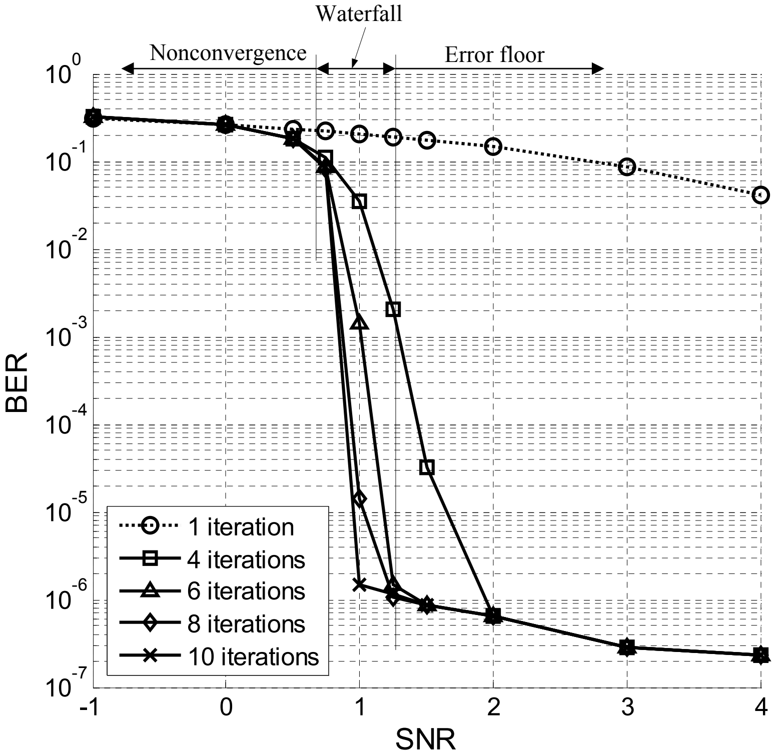


Figure 4.10: The BER performance based on the iterative decoding

find that the error performance will improve as the iteration number increases. The BER curve can be divided into three regions [157], which are pointed out for the 8 iterations curve in Fig. 4.10. The non-convergence region has an almost constant and high error probability. In the waterfall region, the error probability sharply drops to a lower value. Moreover, the error floor region reveals a slowly decreasing error probability. The waterfall region is determined by the interleaver gain, and the error floor region is by the minimum distance of the code [158], which is also related to the interleaver. Therefore, there are two design criteria for the interleaver [159]; the first one is the optimization of the weight distribution for the code, and the second is the decorrelation between the constituent encoders. The former tends to improve the minimum distance for lower error floor, while the latter increases the convergence rate that dominates the curve slope in the waterfall region. Generally, the convergence rate can be enhanced by increasing the interleaver size N. The interleaver designs for the turbo code can be found in [143, 159–164].

4.2.3 Turbo decoder

In the Log-MAP algorithm, which is equivalent to the MAP algorithm, the channel reliability value $L_c = \frac{4E_s}{N_0}$ requires the SNR estimation for the AWGN channel [165]. Nevertheless, the real channel condition is hard to be accurately judged. Furthermore, in the max* function (3.169), we need to implement a table for the nonlinear term $\ln(1 + e^{-|x_1 - x_2|})$ that also depends on L_c. Multiple tables are necessary for various SNRs, leading to a much complex circuit design. Alternatively, we can apply the Max-Log-MAP algorithm where max* is reduced to the maximum (max) function, revealed in (3.180). The value L_c becomes ineffective in the Max-Log-MAP algorithm because of the maximum function [166]; however, the performance loss will be encountered. In [167] and [168], the approach that scales the extrinsic information is introduced to improve the turbo decoder based on the Max-Log-MAP algorithm.

The number of bits to represent the quantities in decoding algorithms affects the performance, memory size, and chip area. In [169], the upper bounds of quantities in SISO decoding is derived theoretically. The fixed-point performance of turbo decoding in 3GPP2 is also presented in [170] by using Log-MAP algorithm. We analyze the Max-Log-MAP algorithm to achieve the optimal performance after quantizing the quantities.

The internal precision of both SISO decoders in Fig. 4.7 are the same if we assume only one SISO decoder is implemented. The width of symbols in SISO decoder is strongly dependent on the range of input symbols as well as the fixed-point representation. The notation $n_i.n_f$ indicates the symbol is quantized to $(n_i + n_f)$ bits where n_i bits are integer part, and n_f bits are fractional part. After quantization, we first define the maximum absolute value of the input symbols $r_{0,t}$ and $r_t^{(i)}$ as $A_i n$ and that of the *a priori* input $L_a(u_t)$ as A_a. For example, $A_{in} = 2^{n_i - 1}$ if $r_{0,t}$ and $r_t^{(i)}$ are represented by $(n_i + n_f)$. The branch metric is obtained by (4.21), and the maximum difference between any two branch metrics at time t

$$\Delta\gamma^{(t)} = \max_{(m_1',m_1),(m_2',m2)\in\mathbf{B}_t} |\bar{\gamma}(S^{(t)}_{m_1'}, S^{(t+1)}_{m_1}) - \bar{\gamma}(S^{(t)}_{m_2'}, S^{(t+1)}_{m_2})| \tag{4.30}$$

is upper bounded by

$$\Delta\gamma^{(t)} \leq n \cdot A_{in} + A_a. \tag{4.31}$$

Note that $\mathbf{B}_t = \{\mathbf{B}_t^{+1}, \mathbf{B}_t^{-1}\}$ is the state index set of all branches between time t and $t+1$, and we also assume all the received symbols are equally quantized.

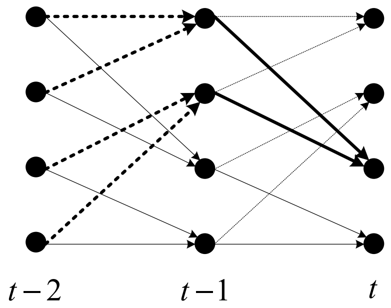


Figure 4.11: A trellis diagram example

For a convolutional code with memory order m, the paths entering each state at time t stem from all states at time $t-m$. Fig. 4.11 shows an example with $m=2$. Therefore, we can bound the difference of forward metrics α as follows:

$$\Delta\alpha^{(t)} \leq m \cdot A_a + d_m \cdot A_{in}, \tag{4.32}$$

and

$$\Delta\alpha^{(t)} = \max_{S^{(t)}_{m_1}, S^{(t)}_{m_2}\in\mathbf{S}} |\alpha(S^{(t)}_{m_1}) - \alpha(S^{(t)}_{m_2})|. \tag{4.33}$$

The value $d_m \leq nm$ indicates the maximum Hamming distance between any two paths from time $t-m$ to t. Similarly, for the backward metric, we have

$$\Delta\beta_t \leq m \cdot A_a + d_m \cdot_{in} . \tag{4.34}$$

Form the calculation of $L(u_t)$ in (3.168), we can derive the following upper bound:

$$\begin{aligned} L(u_t) &\leq \ln\left[\sum_{(m',m)\in\mathbf{B}_t^{+1}} e^{\alpha_{max}^{(t)}+\gamma_{max}^{(t)}+\beta(S_m^{(t+1)})}\right] - \ln\left[\sum_{(m',m)\in\mathbf{B}_t^{-1}} e^{\alpha_{min}^{(t)}+\gamma_{min}^{(t)}+\beta(S_m^{(t+1)})}\right] \\ &= (\alpha_{max}^{(t)}+\gamma_{max}^{(t)}) - (\alpha_{min}^{(t)}+\gamma_{min}^{(t)}) \qquad (4.35) \\ &\quad + \left(\ln \sum_{(m',m)\in\mathbf{B}_t^{+1}} e^{\beta(S_m^{(t+1)})}\right) - \left(\ln \sum_{(m',m)\in\mathbf{B}_t^{-1}} e^{\beta(S_m^{(t+1)})}\right) \end{aligned}$$

We can also find that the two branches coming into a state at time $t+1$ will correspond to the information bits $u_t = +1$ and $u_t = -1$. As a result,

$$\ln \sum_{(m',m)\in\mathbf{B}_t^{+1}} e^{\beta(S_m^{(t+1)})} = \ln \sum_{(m',m)\in\mathbf{B}_t^{-1}} e^{\beta(S_m^{(t+1)})}, \qquad (4.36)$$

and the bound in (4.35) is reduced to

$$L(u_t) \leq \Delta\alpha^{(t)} + \Delta\gamma^{(t)} \qquad (4.37)$$

Accordingly, the lower can also be obtained as follows:

$$L(u_t) \geq -(\Delta\alpha^{(t)} + \Delta\gamma^{(t)}). \qquad (4.38)$$

Combining the upper bound, we summarize the bound of $L(u_t)$ as

$$|L(u_t)| \leq \Delta\alpha^{(t)} + \Delta\gamma^{(t)} \leq (m+1)(n \cdot A_{in} + A_a), \qquad (4.39)$$

in which we use $d_m \leq nm$. Since the extrinsic value is obtained by,

$$L_e(u_t) = L(u_t) - L_c r_{0,t} - L_a(u_t), \qquad (4.40)$$

the bound for $L_e(u_t)$ can be written as

$$|L_e(u_t)| \leq |L(u_t)| \tag{4.41}$$

if $L(u_t)$, $r_{0,t}$, and $L_a(u_t)$ are assumed to have the same sign.

From the above discussions, we have derive the differences for α and β, and the modulo normalization scheme [130, 131] can be applied to avoid metric overflow. Furthermore, once the received symbols have been quantized to fixed point values, we can acquire the data width requirements for α, γ, and β based on the modulo normalization; on the other hand, $L(u_t)$ and $L_e(u_t)$ can also be determined by the data widths of α and γ. Note that the upper bound in (4.41) shows that $|L_e(u_t)|$ is smaller than $|L(u_t)$, and we can use the representation of $L(u_t)$ for $L_e(u_t)$. In turbo decoding, the *a priori* information $L_a(u_t)$ comes from the extrinsic information $L_e(u_t)$ of the other decoder. Generally, $|L_e(u_t)|$ is smaller than $|L(u_t)$ in real applications [170, 171], and thus we may use smaller widths for $L_e(u_t)$ and $L_a(u_t)$.

The turbo decoder architecture is presented in Fig. 4.12, consisting of the single SISO decoder, the interleaver/de-interleaver for the extrinsic data $L_e(u_t)$, the interleaver for the systematic symbols $r_{0,t}$, and the cache buffer for the BMUs. In the SISO decoder, there are three ACS groups for α, β, and β_d recursions in the sliding windowed MAP algorithm (see Fig. 3.53). The SISO decoder processes three consecutive sub-blocks concurrently for different strategies in the BCJR algorithm. ACSU-α carries out the forward recursion and saves the results in the α-memory . ACSU-β starts backward recursion from the initial conditions determined by the ACSU-β_d previously. At the same time, the LLR calculator determines $L(u_t)$ and $L_e(u_t)$, which is formulated in (3.174) and (4.40). Because of the Max-Log-MAP algorithm, the ACSU is identical to that in the Viterbi decoder. The BMUs compute branch metrics for ACSU-α, ACSU-β, and ACSU-β_d according to (4.21).

Since the modulo normalization is applied, the evaluation of $L(u_t)$ should be modified in

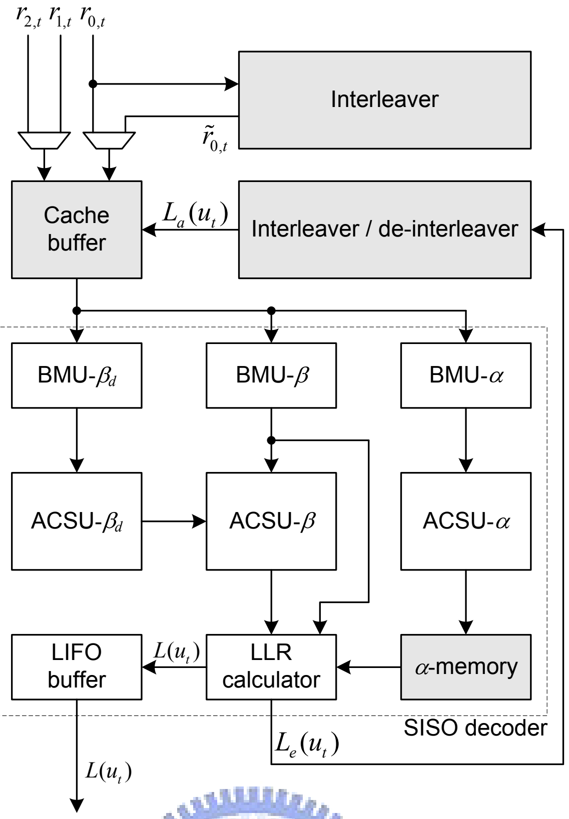


Figure 4.12: The turbo decoder architecture with a single SISO decoder

the LLR calculator. Considering the following operation

$$S = \max(\alpha_1 + \gamma_1 + \beta_1, \alpha_2 + \gamma_2 + \beta_2), \tag{4.42}$$

we have to determine the sign of

$$(\alpha_1 + \gamma_1 + \beta_1) - (\alpha_2 + \gamma_2 + \beta_2) = (\alpha_1 - \alpha_2) + (\beta_1 - \beta_2) + (\gamma_1 - \gamma_2). \tag{4.43}$$

If α_1, α_2, β_1 and β_2 are modulo normalized, only the difference of them are useful during calculations. As shown in (4.43), we should first find the difference in the left-hand side instead of the summation first in the right-hand side. The operation is also illustrated in Fig. 4.13.

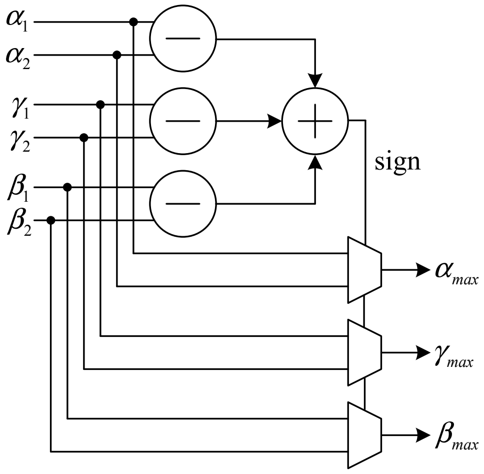


Figure 4.13: The comparator cell in the LLR calculator

In the $L(u_t)$ computation based on the Max-Log-MAP algorithm, we can write

$$\begin{aligned} L(u_t) &= \max_{(m',m)\in \mathbf{B}_t^{+1}} [\alpha(S_{m'}^{(t)}) + \gamma(S_{m'}^{(t)}, S_m^{(t+1)}) + \beta(S_m^{(t+1)})] \\ &\quad - \max_{(m',m)\in \mathbf{B}_t^{-1}} [\alpha(S_{m'}^{(t)}) + \gamma(S_{m'}^{(t)}, S_m^{(t+1)}) + \beta(S_m^{(t+1)})] \\ &= (\alpha_{max}^{(+1)} + \gamma_{max}^{(+1)} + \beta_{max}^{(+1)}) - (\alpha_{max}^{(-1)} + \gamma_{max}^{(-1)} + \beta_{max}^{(-1)}) \\ &= (\alpha_{max}^{(+1)} - \alpha_{max}^{(-1)}) + (\gamma_{max}^{(+1)} - \gamma_{max}^{(-1)}) + (\beta_{max}^{(+1)} - \beta_{max}^{(-1)}). \end{aligned} \tag{4.44}$$

For $u = +1$ or $u = -1$, $\alpha_{max}^{(u)}$, $\gamma_{max}^{(u)}$, and $\beta_{max}^{(u)}$ are selected from

$$\{\alpha(S_{m'}^{(t)}), \gamma(S_{m'}^{(t)}, S_m^{(t+1)}), \beta(S_m^{(t+1)}) | (m', m) \in \mathbf{B}_t^u\} \tag{4.45}$$

according to the maximum function

$$\max_{(m',m)\in \mathbf{B}_t^u} [\alpha(S_{m'}^{(t)}) + \gamma(S_{m'}^{(t)}, S_m^{(t+1)}) + \beta(S_m^{(t+1)})]. \tag{4.46}$$

Consequently, we can derive $L(u_t)$ with sum of the difference between metrics. The extrinsic information $L_e(u_t)$ is also computed by (4.40). Fig. 4.14 is the architecture of the LLR calculator where the comparator (CMP) cell is similar to the comparator in Fig. 4.13, but

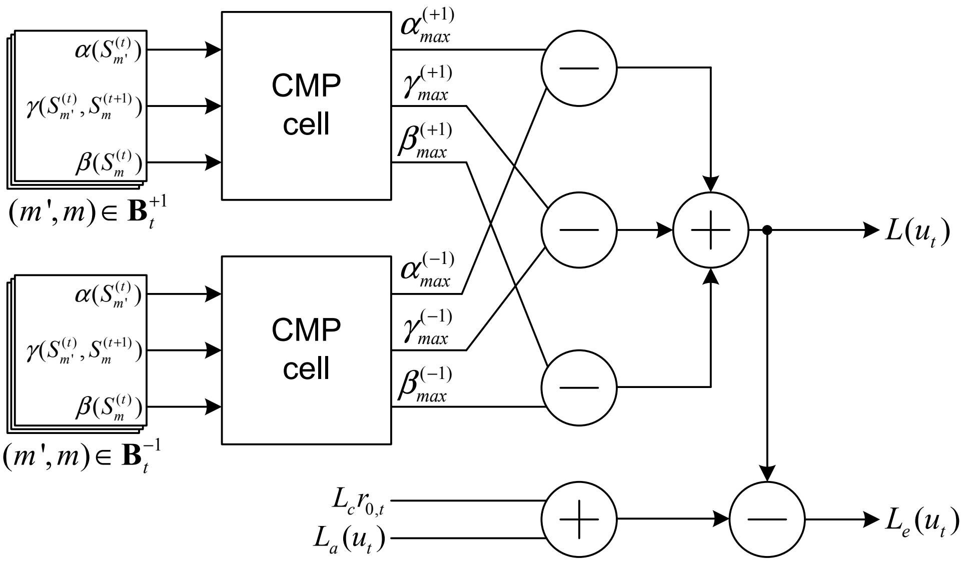
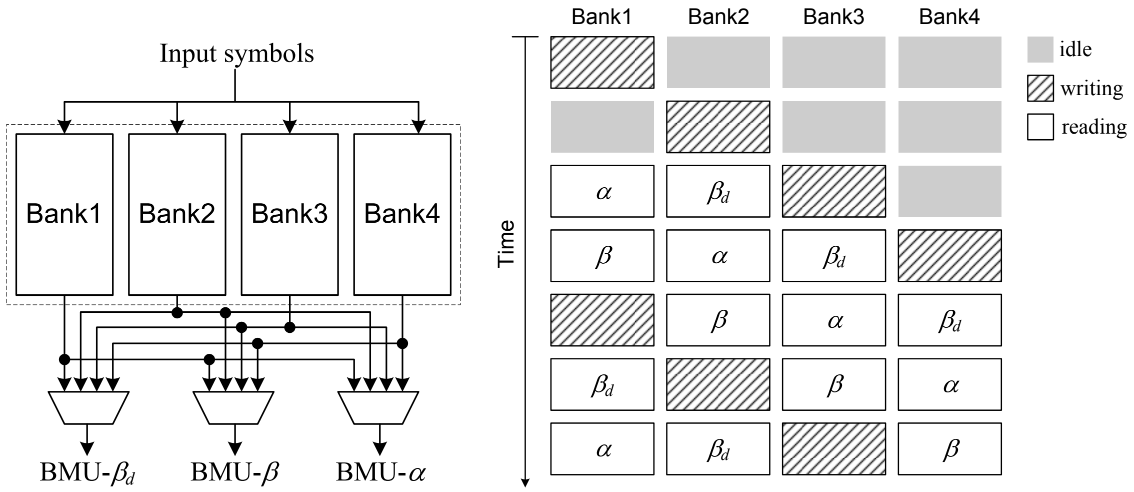


Figure 4.14: The LLR calculator architecture

has more inputs depending on $\mathbf{B}_t^{+1}$ and $\mathbf{B}_t^{-1}$. The CMP cell will perform the function (4.46) and generate the selection signals to identify $\alpha_{max}^{(u)}$, $\gamma_{max}^{(u)}$, and $\beta_{max}^{(u)}$.

In Fig. 3.53, each sub-block needs to be read by ACSU-β_d, ACSU-α, and ACSU-β separately. At the same time slot, from t_i to t_{i+1}, three consecutive sub-blocks are read by the BMUs. The minimum data bandwidth to the external codeword memory should be $3f_c$M symbols per second (MS/s) , assuming a f_cMHz working frequency in the ACSUs. Therefore, an input cache is implemented to reduce the repeated access of the external memory [171]. With the four banks memory model, the behavior of each bank can be expressed by Fig. 4.15 where each bank has T_{sb} words and should be connected to the three TMUs with multiplexers. Codewords are written to the memory and read by the BMU-α, the BMU-β_d, and the BMU-β for branch metric calculations. The data bandwidth of the cache is f_cMS/s for inputs and $3f_c$MS/s for outputs. Accordingly, a multi-port memory or a higher working frequency can be applied to reduce the interconnection between the cache and the BMUs. However, both methods may lead to larger area or more power consumption. The reading by BMU-β and the codeword writing is further combined by avoiding the write-after-read (WAR) data hazard; as a result, the memory size can be reduced from $4T_{sb}$to $3T_{sb}$ words.



Bank1	Bank2	Bank3	Bank4
▨	░	░	░
░	▨	░	░
α	β_d	▨	░
β	α	β_d	▨
▨	β	α	β_d
β_d	▨	β	α
α	β_d	▨	β

(a) Physical model

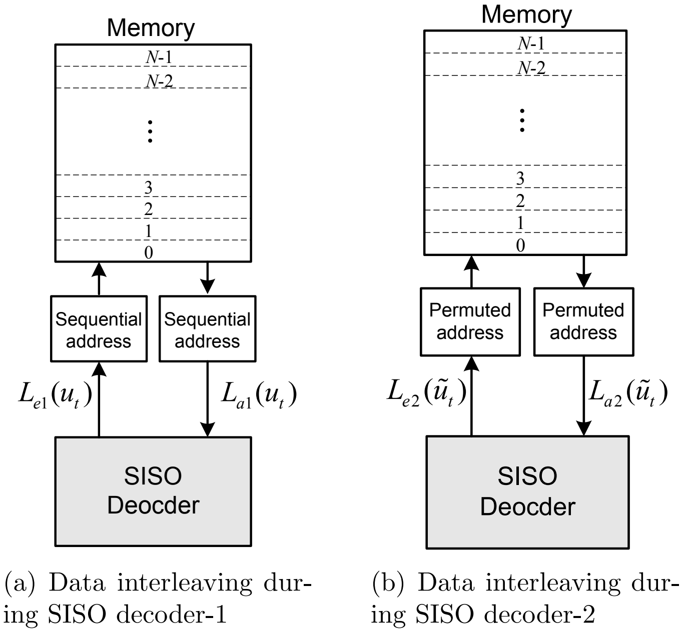
(b) Behavior

Figure 4.15: The multi-bank cache model

The last-in/first-out (LIFO) buffer is included in Fig. 4.12 to reorder the output sequence that is the inverse of the original data sequence because of the computational schedule in the sliding windowed MAP algorithm.

The embedded interleaver/de-interleaver is designed to reduce the amount of time required to permute symbols. The interleaver size N can achieve lower bit error rate (BER), but requires larger memory size. We use a single memory block for both interleaving and de-interleaving functions, which is explained in Fig. 4.16. In SISO decoder-1, the extrinsic information is read and written in a sequential order, while the extrinsic information is accessed in a permuted order in SISO decoder-2. Therefore, the data in the memory are always in sequence regardless of the permutation. Note that the SISO decoder in Fig. 4.12 performs both SISO decoder-1 and SISO decoder-2 functions in different time slots, leading to no data hazard. The memory require one reading port and one writing port in this configuration, and can be either a dual-port SRAM (DP-SRAM) or a single-port SRAM (SP-SRAM) working at higher clock rates.

The permutation realized by address management operates on-the-fly with the SISO decoder and induces no additional delay within each iteration. However, in some cases [63], the address generator (AG) may produce invalid addresses and stall the SISO decoder. This



(a) Data interleaving during SISO decoder-1

(b) Data interleaving during SISO decoder-2

Figure 4.16: The operation of interleaver during SISO decoding

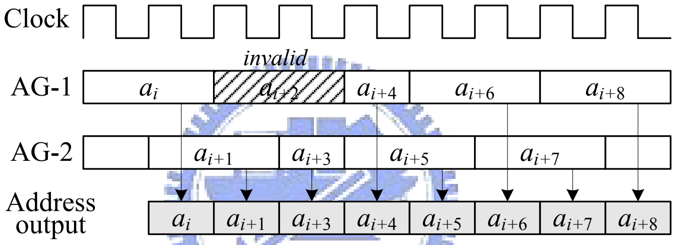


Figure 4.17: The removal of invalid addresses with two AGs

can be solved by using two AGs as illustrated in Fig. 4.17. While an invalid address is observed, the address from the other generator is adopted.

4.3 Low-density parity check code

Low-density parity-check (LDPC) code, a linear block code defined by a very sparse parity-check matrix, was first introduced by Gallager [20,21] and rediscovered by MacKay [23,24]. It has engaged much research interest recently because the sparse property of parity check matrix H makes the decoding algorithm simple and practical at good communication rates [24]. Similar to turbo code [17], LDPC code can achieve a capacity approaching performance as the block length becomes large [25, 172]; however, LDPC decoders, which are highly parallelizable, have a much higher decoding speed [65] than turbo decoders [60]. LDPC decoders based on sum-product algorithm (SPA) are capable of parallel implementation, leading to a much higher decoding speed than turbo decoders. Such distinct dominance of LDPC code can be employed to enhance system performances for high speed wireless communications. Consequently, more high speed communication systems have considered employing LDPC code to enhance performance.

An (N,K) LDPC code over $GF(2)$ is often represented by the bipartite graph [22, 150] where N bit nodes and M $(\geq N-K)$ check nodes are connected by edges according to a $M \times N$ matrix H. In the following parity check matrix of a (10,5) LDPC code, for example,

$$H = \begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \end{bmatrix}, \tag{4.47}$$

and

$$[\ x_0 \quad x_1 \quad x_2 \quad x_3 \quad x_4 \quad x_5 \quad x_6 \quad x_7 \quad x_8 \quad x_9\ ] \times H^T = [\ c_0 \quad c_1 \quad c_2 \quad c_3 \quad c_4\ ], \tag{4.48}$$

Fig. 4.18 represents the five parity check equations

$$\begin{aligned} c_0 :& \quad x_3 + x_4 + x_7 + x_9 = 0 \\ c_1 :& \quad x_0 + x_1 + x_5 + x_9 = 0 \\ c_2 :& \quad x_1 + x_3 + x_6 + x_8 = 0 \\ c_4 :& \quad x_2 + x_4 + x_5 + x_6 = 0 \\ c_5 :& \quad x_0 + x_2 + x_7 + x_8 = 0 \end{aligned} \tag{4.49}$$

in the bipartite graph with 10 bit nodes and 5 check nodes. The column weight of H determines the number of edges (or degree) for each bit node connected to check nodes, while the row weight of H determines the connections for each check node. A LDPC code with equal weights for columns and rows is a regular code, as shown in (4.47), and otherwise is termed irregular. It has been shown that irregular codes can outperform those based on regular graphs [173].

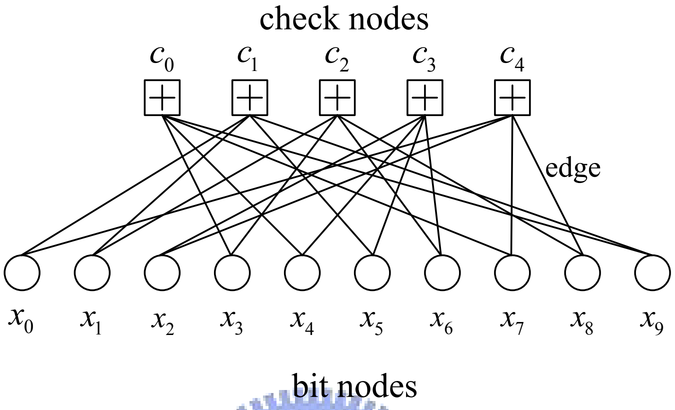


Figure 4.18: The bipartite graph for the (10,5) LDPC code

4.3.1 LDPC decoding algorithm

The decoding of LDPC codes is based on sum-product algorithm (SPA) [24, 26], or message passing (MP) algorithm, which iteratively updates the *a posteriori* probabilities of the bit nodes. We first consider the message passing for the check nodes. Fig. 4.19 is a check node

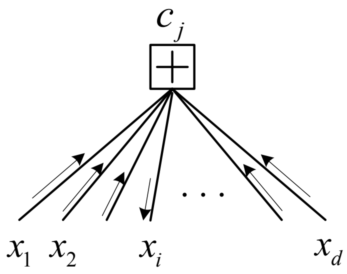


Figure 4.19: The check node with degree d

with d edges, and each edge corresponds to a variable in $GF(2)$. The constraint set for the node is

$$\mathbf{S}_{c_j} = \{(x_1, x_2, \ldots, x_d) | x_1 + x_2 + \cdots + x_d = 0\}; \tag{4.50}$$

therefore, the output message on the edge x_i should be

$$\mu_{c_j \to x_i}(x_i) = P_{ext}(x_i) = P(x_1 + \cdots + x_{i-1} + x_{i+1} + \cdots + x_d = x_i) \tag{4.51}$$

Before deriving (4.51), we begin with the two variables condition:

$$\begin{aligned} P(x_1 + x_2 = 0) &= P_{int}(x_1 = 0) P_{int}(x_2 = 0) + P_{int}(x_1 = 1) P_{int}(x_2 = 1) \\ &= (1 - p_1)(1 - p_2) + p_1 p_2, \end{aligned} \tag{4.52}$$

where $p_i = P_{int}(x_i = 1)$. Moreover, the above equation can be expressed as

$$2P(x_1 + x_2 = 0) - 1 = (1 - 2p_1)(1 - 2p_2) \tag{4.53}$$

If we assume

$$\begin{aligned} 2P(x_1 + x_2 + \cdots + x_j = 0) - 1 = 2\Pi_j - 1 \\ &= (1 - 2p_1)(1 - 2p_2) \cdots (1 - 2p_j) \\ &= \prod_{l=1}^{j} (1 - 2p_l), \end{aligned} \tag{4.54}$$

the following probability will be

$$\begin{aligned} \Pi_{j+1} &= P(x_1 + x_2 + \cdots + x_{j+1} = 0) \\ &= P(x_1 + x_2 + \cdots + x_j = 0)(1 - p_{j+1}) + P(x_1 + x_2 + \cdots + x_j = 1) p_{j+1} \\ &= \Pi_j (1 - p_{j+1}) + (1 - \Pi_j) p_{j+1} \end{aligned} \tag{4.55}$$

As a result, we can obtain

$$2\Pi_{j+1} - 1 = (2\Pi_j - 1)(1 - 2p_{j+1}) = \prod_{l=1}^{j+1}(1 - 2p_l) \tag{4.56}$$

from (4.54). By induction, we conclude that

$$\Pi_k = P(x_1 + x_2 + \cdots + x_d = 0) = \frac{1}{2}[1 + \prod_{j=1}^{d}(1 - 2p_i)] \tag{4.57}$$

for any $d \geq 1$. The probability in (4.51) can then be written as

$$\mu_{c_j \to x_i}(x_i = 0) = \frac{1}{2}\left[1 + \prod_{l=1, l \neq i}^{d} (1 - 2\mu_{x_l \to c_j}(x_l = 1))\right] \tag{4.58}$$

$$\mu_{c_j \to x_i}(x_i = 1) = \frac{1}{2}\left[1 - \prod_{l=1, l \neq i}^{d} (1 - 2\mu_{x_l \to c_j}(x_l = 1))\right], \tag{4.59}$$

where we let $p_l = \mu_{x_l \to c_j}(x_i = 1)$, the message from x_l.

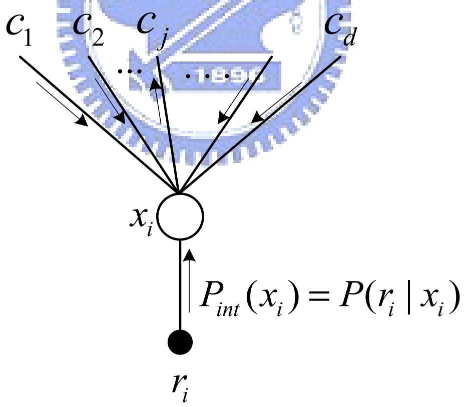


Figure 4.20: The bit node with degree d

For the message passing at the bit node as shown in Fig. 4.20, the node x_i will receive messages from the check nodes connecting to itself. Since the constraint set for x_i is

$$\mathbf{S}_{x_i} = \{x_i = a | a \in GF(2)\}, \tag{4.60}$$

the output message from x_i to c_j will be

$$\mu_{x_i \to c_j}(x_i = 0) = \rho_b \cdot P_{int}(x_i = 0) \prod_{l=1, l \neq j}^{d} \mu_{c_l \to x_i}(x_i = 0) \tag{4.61}$$

$$\mu_{x_i \to c_j}(x_i = 1) = \rho_b \cdot P_{int}(x_i = 1) \prod_{l=1, l \neq j}^{d} \mu_{c_l \to x_i}(x_i = 1), \tag{4.62}$$

and

$$\rho_b = \sum_{x_i} P_{int}(x_i) \prod_{l=1, l \neq j}^{d} \mu_{c_l \to x_i}(x_i). \tag{4.63}$$

The instrinsic probability $P_{int}(x_i)$ comes from the received symbol r_i, and $P_{int}(x_i) = P(r_i|x_i)$.

In the logarithmic domain, we can use the log-likelihood ratio to represent the messages. However, the ratio is redefined to be

$$L(x) = \ln \frac{P(x = 0)}{P(x = 1)}, \tag{4.64}$$

and

$$P(x = 1) = \frac{1}{e^{L(x)} + 1}. \tag{4.65}$$

Alternatively, we can write

$$1 - 2P(x = 1) = \frac{e^{L(x)} - 1}{e^{L(x)} + 1} = \tanh(\frac{L(x)}{2}), \tag{4.66}$$

in which the hyperbolic tangent is

$$\tanh(\frac{x}{2}) = \frac{e^x - 1}{e^x + 1}. \tag{4.67}$$

According to the definition of (4.64), the messages from check node c_j to bit node x_i can be

$$\begin{aligned} L_{c_j \to x_i}(x_i) &= \ln \frac{1 + \prod_{l=1, l\neq i}^{d} (1 - 2\mu_{x_l \to c_j}(x_l = 1))}{1 - \prod_{l=1, l\neq i}^{d} (1 - 2\mu_{x_l \to c_j}(x_l = 1))} \\ &= \ln \frac{1 + \prod_{l=1, l\neq i}^{d} \tanh(\frac{L_{x_l \to c_j}(x_l)}{2})}{1 - \prod_{l=1, l\neq i}^{d} \tanh(\frac{L_{x_l \to c_j}(x_l)}{2})} \\ &= 2 \tanh^{-1} \Bigg(\prod_{l=1, l\neq i}^{d} \tanh(\frac{L_{x_l \to c_j}(x_l)}{2}) \Bigg). \end{aligned} \tag{4.68}$$

Note that the inverse hyperbolic tangent

$$\tanh^{-1}(y) = \frac{1}{2} \ln \frac{1+y}{1-y} \tag{4.69}$$

is applied to the above equation. We further define another function for $x > 0$:

$$\Psi(x) = \Psi^{-1}(x) = \ln \frac{1 + e^{-x}}{1 - e^{-x}} = -\ln(\tanh(\frac{x}{2})), \tag{4.70}$$

and decompose the following function

$$\begin{aligned} \prod_{l=1, l\neq i}^{d} \tanh(\frac{L_{x_l \to c_j}(x_l)}{2}) &= \prod_{l=1, l\neq i}^{d} A_l \\ &= \Bigg(\prod_{l=1, l\neq i}^{d} \mathrm{sgn}(A_l) \Bigg) \exp \Bigg(\sum_{l=1, l\neq i}^{d} \ln |A_l| \Bigg) \\ &= \Bigg(\prod_{l=1, l\neq i}^{d} \mathrm{sgn}(L_{x_l \to c_j}(x_i)) \Bigg) \exp \Bigg(\sum_{l=1, l\neq i}^{d} \ln[\tanh(\frac{|L_{x_l \to c_j}(x_l)|}{2})] \Bigg) \end{aligned} \tag{4.71}$$

because the sign of $A_l = \tanh(\frac{L_{x_l \to c_j}(x_l)}{2})$ is consistent with $L_{x_l \to c_j}(x_l)$. Moreover, we also note that for any integer s

$$(-1)^s \Psi^{-1}(x) = \ln \frac{1 + (-1)^s e^{-x}}{1 - (-1)^s e^{-x}}. \tag{4.72}$$

If we let

$$x = -\sum_{l=1, l \neq i}^{d} \ln[\tanh(\frac{|L_{x_l \to c_j}(x_l)|}{2})] \tag{4.73}$$

in (4.72), (4.68) can be then rewritten as

$$\begin{aligned} L_{c_j \to x_i}(x_i) &= \Big(\prod_{l=1, l \neq i}^{d} \operatorname{sgn}(L_{x_l \to c_j}(x_l))\Big)\Psi^{-1}\Big(-\sum_{l=1, l \neq i}^{d} \ln[\tanh(\frac{|L_{x_l \to c_j}(x_l)|}{2})]\Big) \\ &= \Big(\prod_{l=1, l \neq i}^{d} \operatorname{sgn}(L_{x_l \to c_j}(x_i))\Big)\Psi^{-1}\Big(\sum_{l=1, l \neq i}^{d} \Psi(|L_{x_l \to c_j}(x_l)|)\Big), \end{aligned} \tag{4.74}$$

where we use the function $\Psi(x) = -\ln(\tanh(\frac{x}{2}))$ again. As compared with (4.68), the multiplications has been converted to the additions in (4.74). The messages from bit node x_i to check node c_j can also be expressed as

$$\begin{aligned} L_{x_i \to c_j}(x_i) &= \ln \frac{\rho_b \cdot P_{int}(x_i = 0) \prod_{l=1, l \neq j}^{d} \mu_{c_l \to x_i}(x_i = 0)}{\rho_b \cdot P_{int}(x_i = 1) \prod_{l=1, l \neq j}^{d} \mu_{c_l \to x_i}(x_i = 1)} \\ &= L_{int}(x_i) + \sum_{l=1, l \neq j}^{d} L_{c_l \to x_i}(x_i). \end{aligned} \tag{4.75}$$

In the AWGN channel with variance $2\sigma^2 = N_0/E_s$, the value $L_{int}(x_i)$, also termed channel value, can be obtained by

$$L_{int}(x_i) = \ln \frac{P(r_i | x_i = -1)}{P(r_i | x_i = +1)} = \frac{2}{\sigma^2} r_i = L_c r_i, \tag{4.76}$$

assuming 1 is mapped to -1, and 0 to $+1$.

Fig. 4.21 shows a bipartite graph for decoding an (N,K) LDPC code,and $M = N - K$. The messages on the ordinary edges between bit nodes and check nodes are initialized to

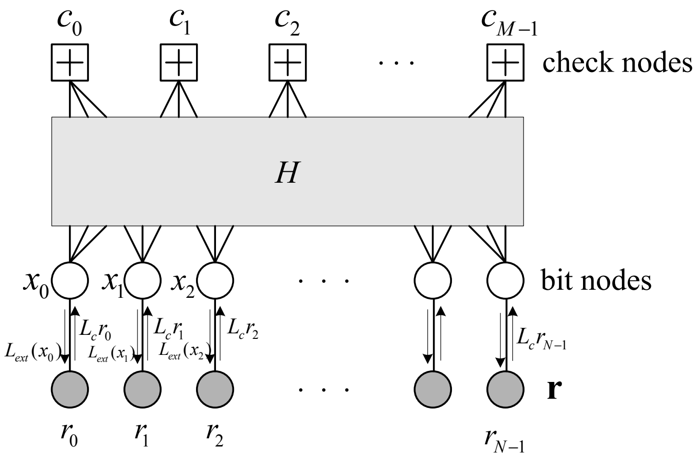


Figure 4.21: Graph representation of LDPC decoding

zero, $L_{c_l \to x_i}(x_l) = 0$ for all l as well as $i = 0 \sim (N-1)$. Moreover, the received sequence $\mathbf{r} = (r_0, r_1, \ldots, r_{N-1})$ provides the channel values $L_c r_i$ for the bit nodes. Beginning with iteration number $J = 1$, we can summarize the iterative LDPC decoding as follows:

1. The messages from check nodes to bit nodes is updated by

$$L_{x_i \to c_j}(x_i) = L_{int}(x_i) + \sum_{l \in \Phi(i) \setminus \{j\}} L_{c_l \to x_i}(x_i), \tag{4.77}$$

   where the set $\Phi(i)$ denotes the indexes of all the check nodes connecting to x_i.

2. For each check node, we calculate the messages conveyed to the bit notes with

$$L_{c_j \to x_i}(x_i) = \left(\prod_{l \in \mathbf{B}(j) \setminus \{i\}} \operatorname{sgn}(L_{x_l \to c_j}(x_l)) \right) \Psi^{-1} \left(\sum_{l \in \mathbf{B}(j) \setminus \{i\}} \Psi(|L_{x_l \to c_j}(x_l)|) \right). \tag{4.78}$$

   The set $\mathbf{B}(j)$ comprises all the indexes of the bit nodes that involve the check node c_j.

3. The *a posteriori* information for the codeword symbol x_i is obtained by

$$L_{post}(x_i) = L_{int}(x_i) + L_{ext}(x_i) = L_{int}(x_i) + \sum_{l \in \Phi(i)} L_{c_l \to x_i}(x_i). \tag{4.79}$$

Notice that

$$L_{ext}(x_i) = \sum_{l \in \Phi(i)} L_{c_l \to x_i}(x_i) \tag{4.80}$$

is the extrinsic information from the LDPC decoder. Finally, we use $L_{post}(x_i)$ to decide the codeword symbols; that is,

$$\hat{x}_i = \begin{cases} 1 & \text{if } L_{post}(x_i) < 0 \\ 0 & \text{if } L_{post}(x_i) \geq 0 \end{cases} \tag{4.81}$$

4. If the parity check is satisfied

$$\mathbf{x} \cdot H^T = \mathbf{0}, \tag{4.82}$$

or the iteration number J reaches the predefined value J_m, the LDPC decoding is completed, and the estimated codeword $\hat{\mathbf{x}} = (\hat{x}_0, \hat{x}_1, \ldots, \hat{x}_{N-1})$ is outputted. Otherwise, the decoder repeats the steps $1 \sim 3$ for the next decoding iteration, and J is increased by one.

The message passing algorithm on the bipartite graph Fig. 4.21 with cycles becomes sub-optimal due to the loss of the independence criteria. Nevertheless, the message passing algorithm in most practical designs can provide precisely decoding. Generally, the length of the shortest cycle in a graph is referred to the *girth* [150] of the graph. The error performance of the graph based iterative decoding is significantly affected by the girth, and a large girth will improve the decoding quality [174–178]. The design of parity check matrices should maximize the girth for better decoding performance.

The implementation of (4.78) is the most complicated part of the LDPC decoder and often accomplished by the table look-up approach [65]. An alternative is the sub-optimal expression for (4.74) [144]:

$$L_{c_j \to x_i}(x_i) \approx \left(\prod_{l=1, l \neq i}^{d} \operatorname{sgn}(L_{x_l \to c_j}(x_i)) \right) \min_{l \in \mathbf{B}(j) \backslash \{i\}} (|L_{x_l \to c_j}(x_i)|). \tag{4.83}$$

The approximation in (4.83) is based on the property of $\Psi(x)$ shown in Fig. 4.22. The smaller $|L_{x_l \to c_j}(x_i)|$ will dominate the summation in (4.78). If

$$|L_{min}(x_i)| = \min_{l \in \mathbf{B}(j) \backslash \{i\}} (|L_{x_l \to c_j}(x_i)|), \tag{4.84}$$

we can approximate the summation

$$\begin{aligned} \sum_{l \in \mathbf{B}(j) \backslash \{i\}} \Psi(|L_{x_l \to c_j}(x_i)|) &= \Psi(|L_{min}(x_i)|) + \delta \\ &\approx \Psi(|L_{min}(x_i)|), \end{aligned} \tag{4.85}$$

and the residual $\delta > 0$.

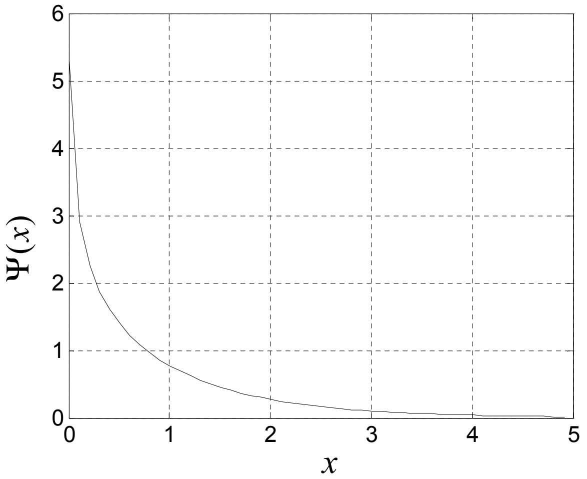


Figure 4.22: Plot of the $\Psi(x)$ function

The decoding procedure based on (4.77) and (4.83), referred to min-sum algorithm [156], is more practical for implementation because of its simplicity, although there is a performance degradation at low SNR conditions. The more accurate approximations, including table look-up [179] and normalization [180–182] approaches, have been proposed to enhance performance with an additional correction process. A further improvement using dynamic normalization technique is reported in [183].

4.3.2 LDPC decoder

An LDPC decoder mainly consists of the message memory keeping messages on the edges, the bit node unit (BNU) that computes (4.77), and the check node unit (CNU) that performs (4.78). The management of the message memory depends on the structure of the parity check matrix H, the maximum decoding throughput, and the parallelism in the computational units. The implementation ranges from the partially parallel architectures [66, 184] to the fully parallel ones [65]. Among the partially parallel designs, the serial implementation using a single processing element has a simple memory architecture, but the decoding throughput is limited. More processing elements are necessary for higher decoding speed; however, more memory bandwidth or access ports are required, leading to much critical memory designs in terms of area, timing, and power consumption. Furthermore, the random structure of parity check matrices also complicates the message processing schedule. Th fully parallel design assigns N BNUs and $M = N - K$ CNUs for parallel computation. However, the randomness of bipartite graph causes complicated signal connections during circuit implementation and requires more area for signal wires. The fully parallel implemen-

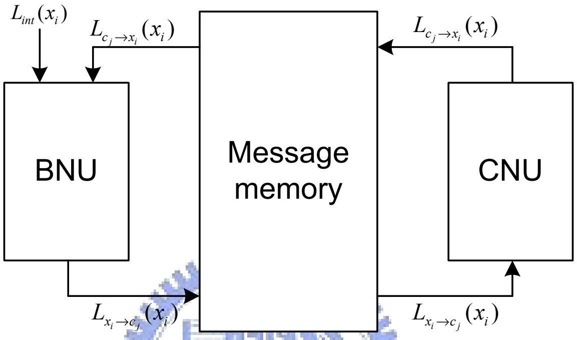


Figure 4.23: The LDPC decoder architecture

tation in [65] demands large area to accommodate interconnections, leading to only 50% chip density. The partially parallel architecture with the high level parallelism also suffers from the same complication. Moreover, the critical path delay induced by global routing decreases the maximum achievable throughput.

Each component will be detailed in the follow-up descriptions. We first consider the

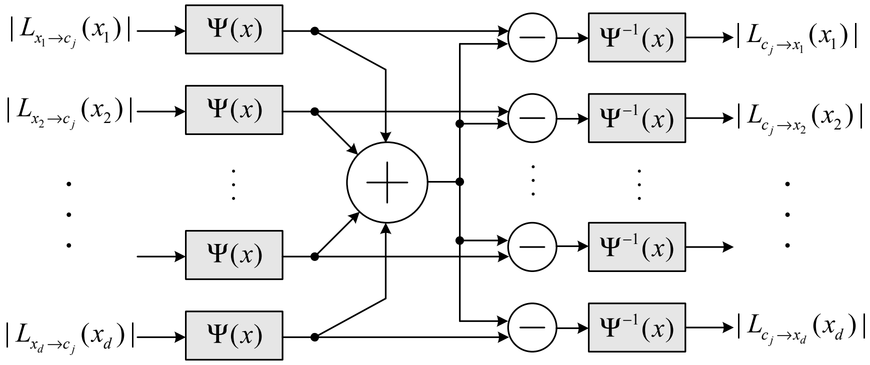


Figure 4.24: The CNU architecture based on the table look-up approach

CNU that implements the function (4.78). Fig. 4.24 shows the architecture that realizes the message magnitude calculations for the bit nodes $x_1, x_2, \ldots, x_d$ connecting to the check node c_j with the degree d. The $\Psi(x)$ and $\Psi^{-1}(x)$ blocks are look-up tables (LUT) that maps their inputs to the Ψ function and its inverse Ψ^{-1}. The resolution of LUTs determines the decoding performance, but also the table sizes. Generally, more resolution can result in more accurate messages, however, the table size increases exponentially with the resolution. The sign operation in (4.78) can be implemented with exclusive-or function, which is illustrated in Fig. 4.25.

Alternatively, the CNU can be designed by a sorter that search the minimum magnitude according to the sub-optimal function (4.83). As shown in Fig. 4.26, the sorter searches for the minimum (min) and the second minimum ($min2$) values among the magnitudes

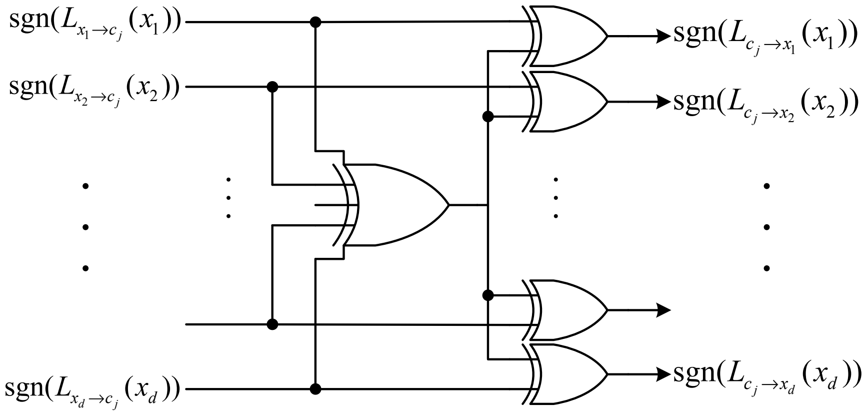

$\mathrm{sgn}(L_{x_1\to c_j}(x_1))$
$\mathrm{sgn}(L_{x_2\to c_j}(x_2))$
$\mathrm{sgn}(L_{x_d\to c_j}(x_d))$
$\mathrm{sgn}(L_{c_j\to x_1}(x_1))$
$\mathrm{sgn}(L_{c_j\to x_2}(x_2))$
$\mathrm{sgn}(L_{c_j\to x_d}(x_d))$

Figure 4.25: The sign operation in CNU

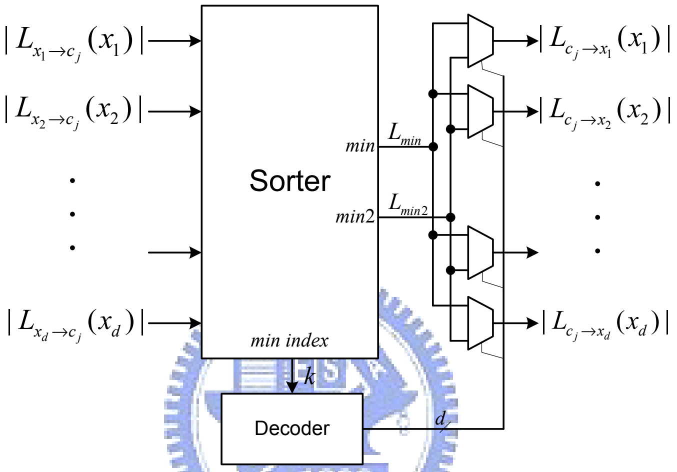


Figure 4.26: The CNU architecture based on the sorter

$|L_{x_i\to c_j}(x_i)|$ for $i = 1 \sim d$ [66]. Assume $1 \le k \le d$, we can also write

$$L_{min} = |L_{x_k\to c_j}(x_k)| = \min_{i=1\sim d}(|L_{x_i\to c_j}(x_i)|) \tag{4.86}$$

$$L_{min2} = \min_{i=1\sim d, i\neq k}(|L_{x_i\to c_j}(x_i)|), \tag{4.87}$$

and the magnitude outputs for $i = 1 \sim d$ will be

$$|L_{c_j\to x_i}(x_i)| = \begin{cases} L_{min} & \text{if } i \neq k \\ L_{min2} & \text{if } i = k \end{cases} \tag{4.88}$$

It is considerably difficult to achieve the sorting function for many parallel inputs. Therefore, we divide the inputs into groups of four, and search the minimum and the second minimum among the four values. Fig. 4.27 is an example with 16 inputs. The CMP-4 cell will identify the two smaller values from its four inputs. After the first compare operations, we have four minimum values from which the CMP-4 can be applied to find the final minimum and the candidates for the final second minimum. The output second minimum is finally picked out with a two-input comparator (CMP-2).

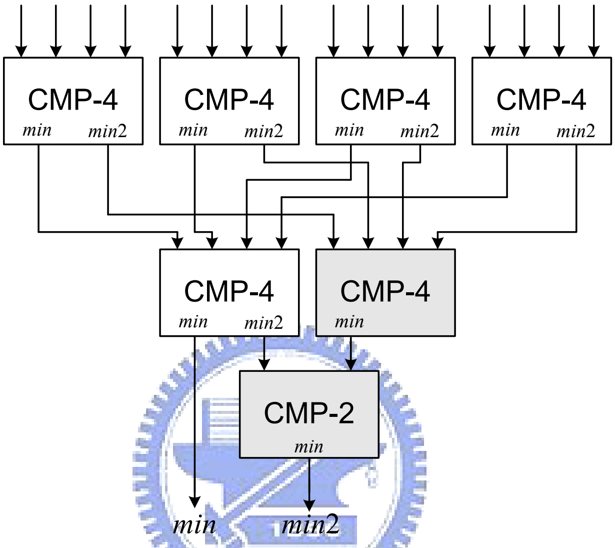


Figure 4.27: The sorter architecture with 16 inputs

The message memory unit (MMU) structure depends the parallelism of processing elements as well as the decoding throughput. Fig. 4.28 is the parallel decoder architecture with N BNUs and M CNUs that perform parallel processing for all the bit nodes and the check nodes. Note that the number of data read or written from the memory equals to the edge number of the bipartite graph or to the number of 1s in the parity check matrix H. For better performance, the code is often designed to have large block length N; if H has a fixed column degree d, the edge number during decoding will be $N \times d$. The huge interconnections between the memory and the computational units cause the difficulty in signal routing.

Fig. 4.29 shows a partially parallel decoder where much less BNUs and CNUs are al-

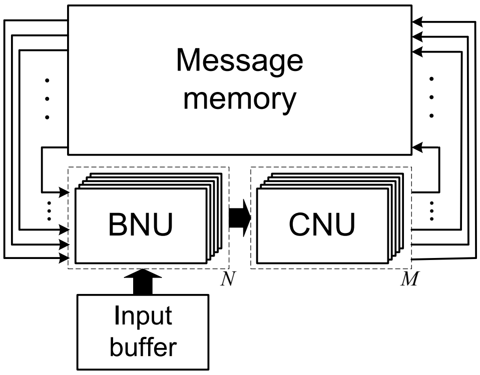


Figure 4.28: The fully parallel LDPC decoder

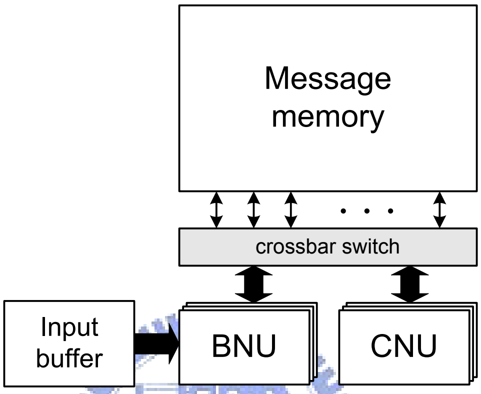


Figure 4.29: The partially parallel LDPC decoder

located for computations, leading to less area and much less signal routing. Nevertheless, the operations of the BNU and the CNU have message dependency and therefore should work separately. In each decoding iteration, most of the messages from the BNUs should be written into the memory before the CNUs can proceed, leading to some throughput degradation.

For the partially parallel design with high level parallelism, we may face the large signal connections that enlarge the chip area and decrease the decoding throughput. We provide the memory management approach as well as the architecture for the partially parallel decoder with high decoding speed. The $M \times N$ parity check matrix H is divided into four $\frac{M}{2} \times \frac{N}{2}$ sub-matrices: h_{00}, h_{01}, h_{10}, and h_{11}, which is shown in Fig. 4.30. The sequence of data processed by CNUs are $\{h_{00}, h_{01}\}$ and $\{h_{10}, h_{11}\}$, whereas the sequence of data in BNUs

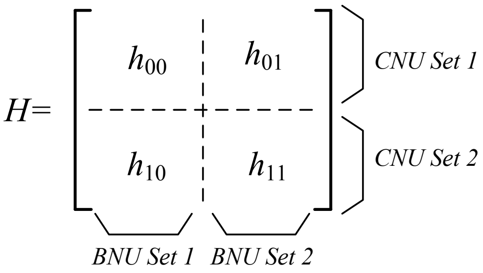


Figure 4.30: The sub-matrices of H

should be $\{h_{00}, h_{10}\}$ and $\{h_{01}, h_{11}\}$. Fig. 4.31 illustrates the main structure of the LDPC decoder, containing $M/2$ CNUs, $N/2$ BNUs, an input buffer connected to BNUs, and two dedicated message memory units (MMU).

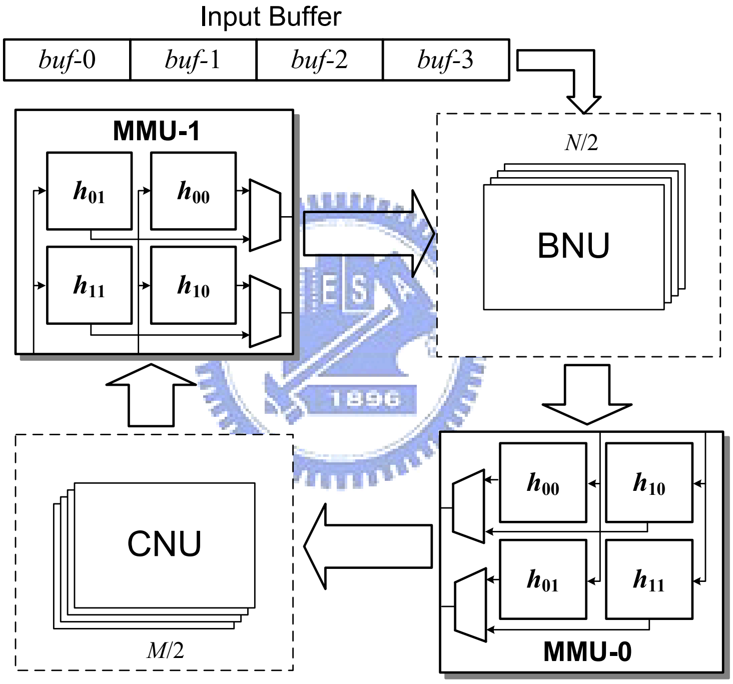
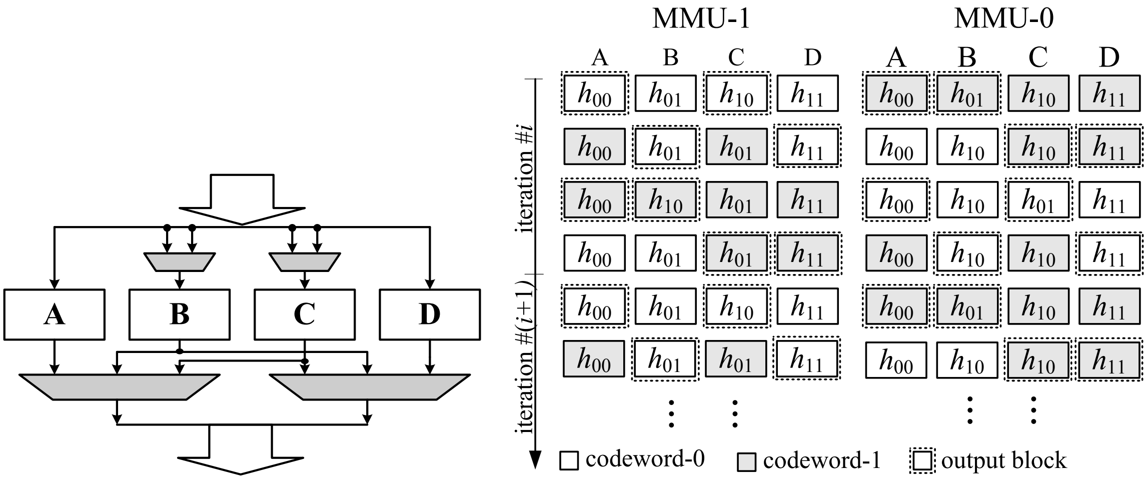


Figure 4.31: The partially parallel LDPC decoder with $N/2$ BNUs and $M/2$ CNUs

MMU is a storage unit that keeps message values in the bipartite graph. With two MMUs, the decoder in Fig. 4.31 can achieve higher decoding speed due to less critical path delay and parallel decoding of two distinct codewords. Each MMU that is divided into four sub-blocks according to Fig. 4.30 receives data from one computational unit and delivers them to another after reordering. As described below, the memory management strategies switching data sequence between BNU and CNU include multiplexers and register exchange (RE) schemes, which have different level of routing complexity.

1. Multiplexer (MUX):



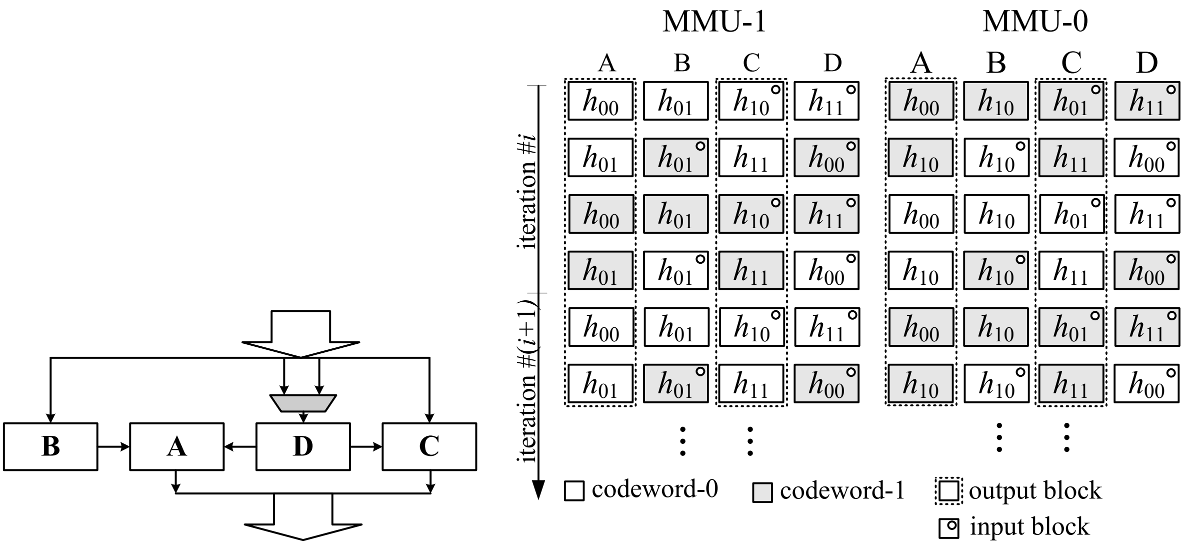
(a) MMU-0 and MMU-1 (b) Timing diagram

Figure 4.32: The architecture and timing diagram of MUX-based MMU

MMUs capture data from datapath and reorder them by multiplexers as shown in Fig. 4.32(a). The detail operation is also illustrated in Fig. 4.32(b), where MMU-0 and MMU-1 interchange two codewords iteratively without stalls. Hence the decoder can achieve the maximum throughput based on the allocated computing resources. What should be noted in Fig. 4.32(a) is the coherent interconnection and complicated data bus between all four sub-blocks and datapaths. The data inputted B and C should be switched since either h_{01} or h_{10} will get into these two sub-blocks. Furthermore, the output of MMUs can be {A,B}, {C,D}, {A,C}, or {B,D} at different time instance. As a result, the data switching causes a large number of signal connections as well as high routing complexity.

2. Four blocks register exchange (RE4):

Fig. 4.33(a) shows the organization of sub-blocks based on the proposed RE approach. Since data movement is accomplished through register exchange, the multiplexers can


MMU-1
MMU-0
A B C D A B C D
h_{00} h_{01} h_{10} h_{11} h_{00} h_{10} h_{01} h_{11}
h_{01} h_{01} h_{11} h_{00} h_{10} h_{10} h_{11} h_{00}
h_{00} h_{01} h_{10} h_{11} h_{00} h_{10} h_{01} h_{11}
h_{01} h_{01} h_{11} h_{00} h_{10} h_{10} h_{11} h_{00}
h_{00} h_{01} h_{10} h_{11} h_{00} h_{10} h_{01} h_{11}
h_{01} h_{01} h_{11} h_{00} h_{10} h_{10} h_{11} h_{00}
iteration #i
iteration #(i+1)
□ codeword-0
□ codeword-1
output block
input block
B
A
D
C

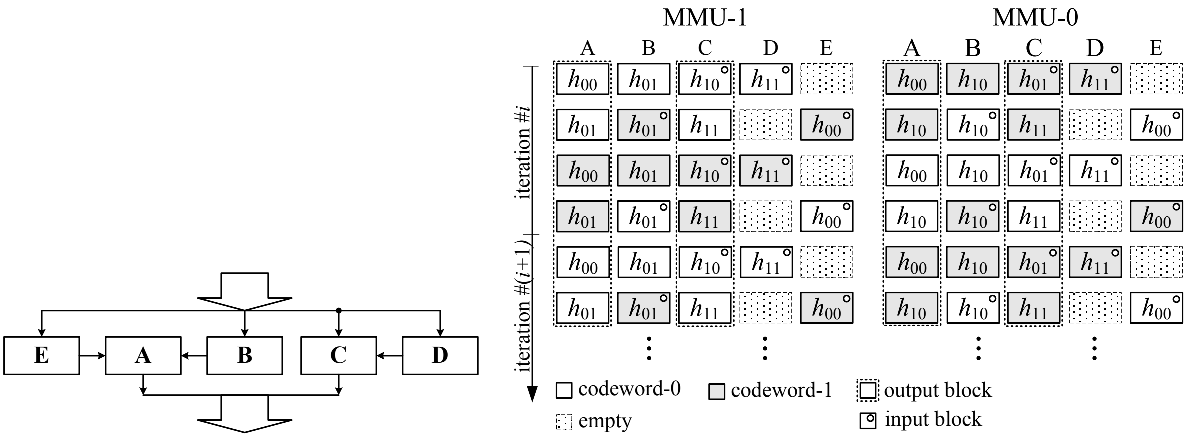
(a) MMU-0 and MMU-1 (b) Timing diagram

Figure 4.33: The architecture and timing diagram of RE4-based MMU

be reduced by exchanging data among four sub-blocks. Fig. 4.33(b) also shows a detail timing diagram about the data flow in MMUs. The outputs of BNUs or CNUs will be fed into sub-blocks B, C and D, while sub-blocks A and C export data after reordering. It can be noticed that sub-block A receives data from either B or D instead of MMU input. The output of MMU is {A,C} without any data switching. Consequently, many multiplexers are eliminated to achieve a much simpler signal wiring.

3. Five blocks register exchange (RE5):

The register exchange scheme based on five sub-blocks is proposed in Fig. 4.34(a) to achieve a multiplexer free data bus between MMUs and datapaths. The multiplexer in Fig. 4.33(a) can be further reduced by inserting sub-block E to MMU. Note that h_{00} and h_{11} in D of the RE4-based MMU (Fig. 4.33) have been dispatched to D and E of the RE5-based MMU to avoid data switching. Therefore, the message values associated with the four sub-matrices of H can be individually captured by different sub-blocks, B, C, D, and E. Similarly only A and C serves as outputs of MMU. Fig. 4.34(b) also illustrates the detail operation of RE5-based MMU. The connection of MMUs and


E
A
B
C
D
MMU-1
MMU-0
A B C D E
iteration #i
iteration #$(i+1)$
h_{00} h_{01} h_{10} h_{11} | h_{00} h_{10} h_{01} h_{11}
h_{01} h_{01} h_{11} h_{00} | h_{10} h_{10} h_{11} h_{00}
h_{00} h_{01} h_{10} h_{11} | h_{00} h_{10} h_{01} h_{11}
h_{01} h_{01} h_{11} h_{00} | h_{10} h_{10} h_{11} h_{00}
h_{00} h_{01} h_{10} h_{11} | h_{00} h_{10} h_{01} h_{11}
h_{01} h_{01} h_{11} h_{00} | h_{10} h_{10} h_{11} h_{00}
codeword-0
codeword-1
output block
empty
input block

(a) MMU-0 and MMU-1 (b) Timing diagram

Figure 4.34: The architecture and timing diagram of RE5-based MMU

datapaths becomes immediate and simple as a result of the removal of multiplexers on data bus.

In summary, with MUX or RE approach, the proposed MMU architectures not only store message values but reorder the data sequence as well. Moreover, to lower routing complexity, the RE4 or RE5 based architecture is applied.

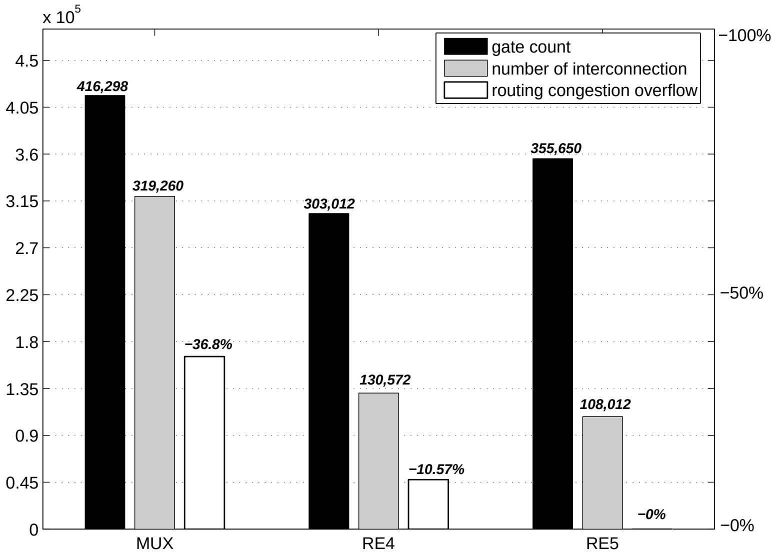


Figure 4.35: The comparison of different MMU architectures

Fig. 4.35 shows a comparison among the three MMU architectures, assuming the message value is 6 bits. The gate count and interconnection are measured only from MMU-0 and MMU-1, whereas the routing congestion overflow is investigated through implementing the decoder within a 25mm^2 1P6M 0.18-μm chip. The negative value in routing congestion overflow indicates more chip area ($>$ 25mm^2) is demanded to accommodate more signal wires. As compared with the MUX-based approach, the RE4 and RE5 based architectures have a 15% $\sim$ 23% decrease in gate count due to the elimination of multiplexers. Furthermore, there is a significant drop of interconnections in RE-based approaches; as a result, the routing congestion can be dramatically improved. This enhancement will facilitate the circuit implementation for the RE-based MMU. The solution to switch data sequence also enables the decoder to process two codewords concurrently without stalls.

Input buffer is also a storage component that receives and keeps channel values for iterative decoding. According to (4.77), the BNUs require channel values $L_{int}(x_i)$ in every iteration, and the input buffer in Fig. 4.31 connected to BNUs, will provide $L_{int}(x_i)$.

Based on the matrix partition of H, four $\frac{M}{2} \times \frac{N}{2}$ sub-matrices, the codeword should be separated into two parts of $N/2$ symbols; consequently, codeword-0 is divided into C00 and C01, while codeword-1 is divided into C10 and C11. Moreover, according to the codeword segmentation, this buffer keeping two codewords should be partitioned into four segments, buf-0, buf-1, buf-2, and buf-3. In the proposed decoder architecture, C00, C01, C10, and C11 should be delivered to BNUs at different time instance. The buffer management strategies will be presented as follows.

1. Multiplexer:

   Fig. 4.36 shows the MUX-based input buffer where each buffer segment contains 600 channel values. The channel values should be hold in the buffer during iterative decoding, and different segment is delivered to BNUs through four-to-one multiplexers. Under this organization, there is a close link between all four buffer segments and BUNs.

2. Register exchange:

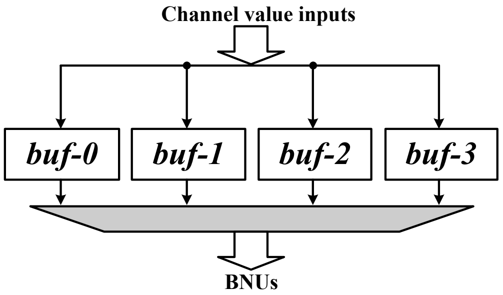
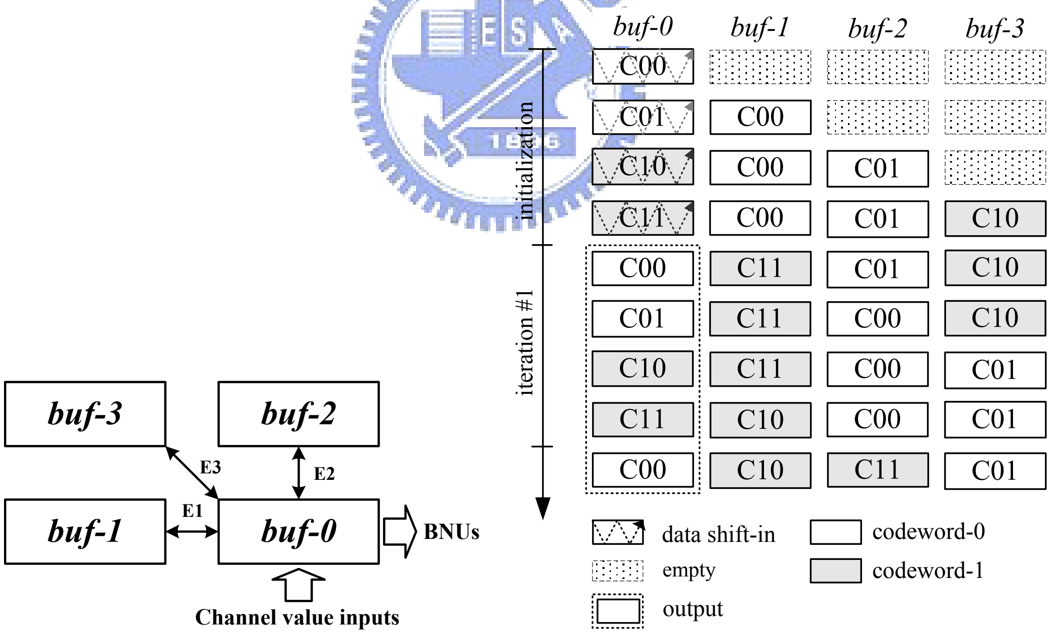


Figure 4.36: The architecture of MUX-based input buffer

Fig. 4.37 shows the buffer structure of RE-based approach and the timing diagram of data exchange. During the initialization, the codewords are serially shifted into buf-0 and conveyed to the other buffer segments while buf-0 is full. Each buffer segment will exchange its contents with buf-0 when the decoding proceeds. The exchange sequence is always {E1,E2,E3,E1,E2,E3,...} where Ei is the exchange operation between buf-i and buf-0 . In this scheme, since only buf-0 is connected to BNUs, the multiplexers can be removed to simplify the signal connection.



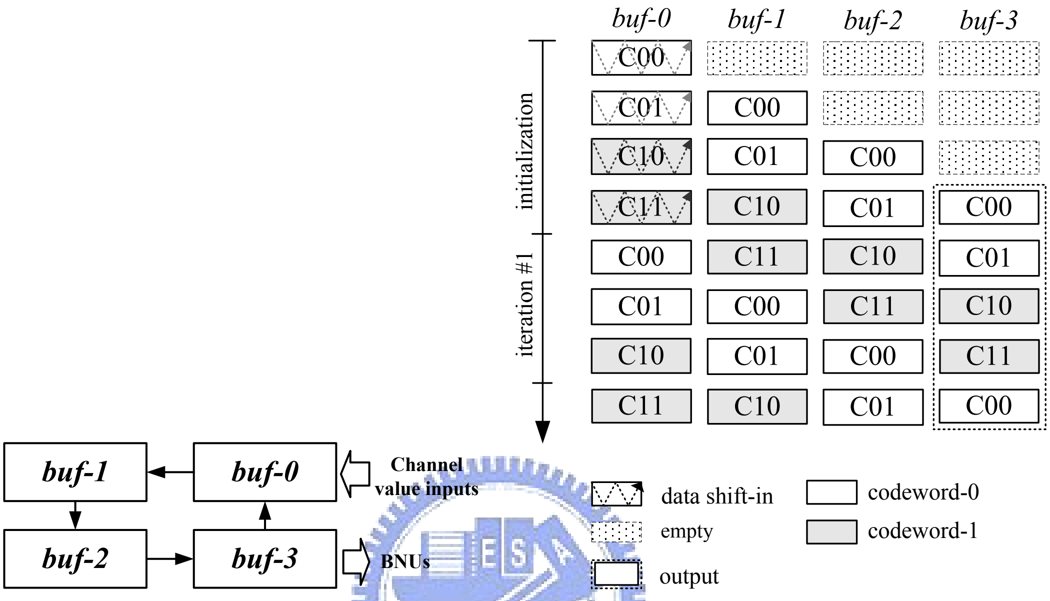
buf-0	buf-1	buf-2	buf-3
C00			
C01	C00		
C10	C00	C01	
C11	C00	C01	C10
C00	C11	C01	C10
C01	C11	C00	C10
C10	C11	C00	C01
C11	C10	C00	C01
C00	C10	C11	C01

(a) Input buffer (b) Timing diagram

Figure 4.37: The architecture and timing diagram of RE based input buffer

3. Register shifting:

In order to reduce the interconnection of *buf*-0 in Fig. 4.37(a), a register shifting (RS) based architecture is proposed in Fig. 4.38. *buf*-0 is a shift register that serially receives the channel values and *buf*-3 transports the associated channel values to BNUs. Channel values of two different codewords are shifted within the four buffer segments as shown in Fig. 4.38(b). Therefore, a much simpler signal connection is achieved.



buf-0	buf-1	buf-2	buf-3
C00			
C01	C00		
C10	C01	C00	
C11	C10	C01	C00
C00	C11	C10	C01
C01	C00	C11	C10
C10	C01	C00	C11
C11	C10	C01	C00

(a) Input buffer

(b) Timing diagram

Figure 4.38: The architecture and timing diagram of RS based input buffer

With the 0.18-μm standard cell library [135], Fig. 4.39 compares the input buffer structures with $N = 1200$ and 5 bits channel values. The RS-based architecture can achieve about 20% gate count and 30% interconnection reduction as compared to the MUX-based buffer.

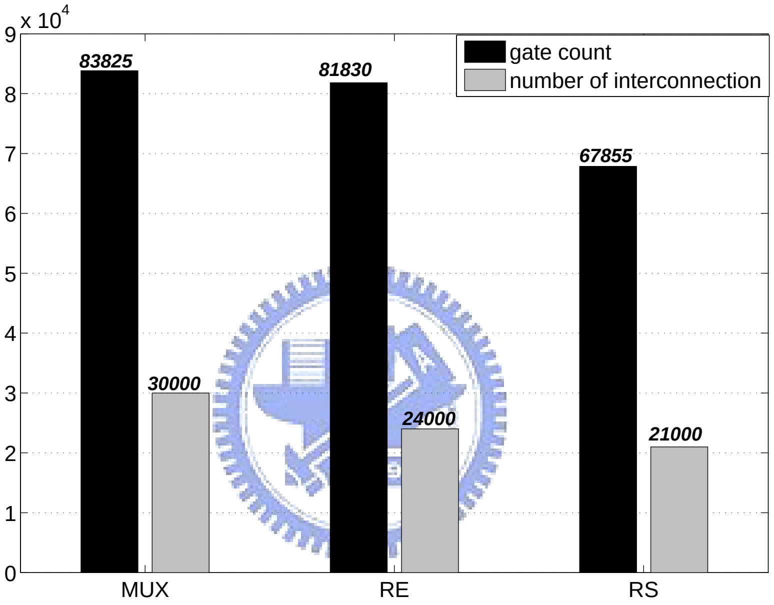


Figure 4.39: The comparison of different input buffer architectures

$$\boldsymbol{H} = \left[\begin{array}{c:c:c:c} \boldsymbol{h}_{00} & \boldsymbol{h}_{01} & \cdots & \boldsymbol{h}_{0x} \\ \hdashline \boldsymbol{h}_{10} & \boldsymbol{h}_{11} & \cdots & \boldsymbol{h}_{1x} \\ \hdashline \vdots & \vdots & \ddots & \vdots \\ \hdashline \boldsymbol{h}_{x0} & \boldsymbol{h}_{x1} & \cdots & \boldsymbol{h}_{xx} \end{array}\right]$$

Figure 4.40: The H with $(x+1)^2$ sub-matrices

The register exchange (RE) approach for the H with four sub-matrices has been presented. When less parallelism is required, the RE methods can also be generalized for the H with $(x+1)^2$ sub-matrices as shown in Fig. 4.40. The corresponding operation of the MMU capturing data from the BNUs and delivering data to the CNUs is illustrated in Fig. 4.41. The MMU is also partitioned into $(x+1)^2$ sub-blocks labeled by M_{ij} where i and j are integers, and $0 \leq i, j \leq x$. In Fig. 4.41, the MMU keeps the messages of the codeword-0, $\mathrm{M}_{ij} = h_{ij}$ for all i and j, and serially shifts out the messages in the vertical direction. At the same time, the outputs of the BNUs are also shifted into the MMU in the diagonal direction. In this scheme, only $\mathrm{M}_{00} \sim \mathrm{M}_{0x}$ are connected to the CNUs while all M_{ix} and M_{xj} should be linked with the BNUs. Moreover, there are many multiplexers exist between the MMU and the BNUs. For example, M_{x1} needs a 2-to-1 multiplexer since two inputs, $h_{(x-1)0}$ and h_{x1} are possible. In general, for $1 \leq i, j \leq x$, the sub-blocks M_{ix} and M_{xj} require $(i+1)$-to-1 multiplexers and $(j+1)$-to-1 multiplexers respectively.

Similar to the RE5 based approach, the multiplexers between the MMU and the BNUs can be further removed by replicating the sub-blocks connecting to the BNUs. According to the input number of multiplexers, each M_{ix} should replicate i copies, and each M_{xj} needs to have j replicas. Therefore, the extra number of sub-blocks to eliminate multiplexers is

$$\begin{aligned} \Delta_s &= (\sum_{i=1}^{x-1} i) + x + (\sum_{j=1}^{x-1} j) && (4.89) \\ &= x^2 && (4.90) \end{aligned}$$

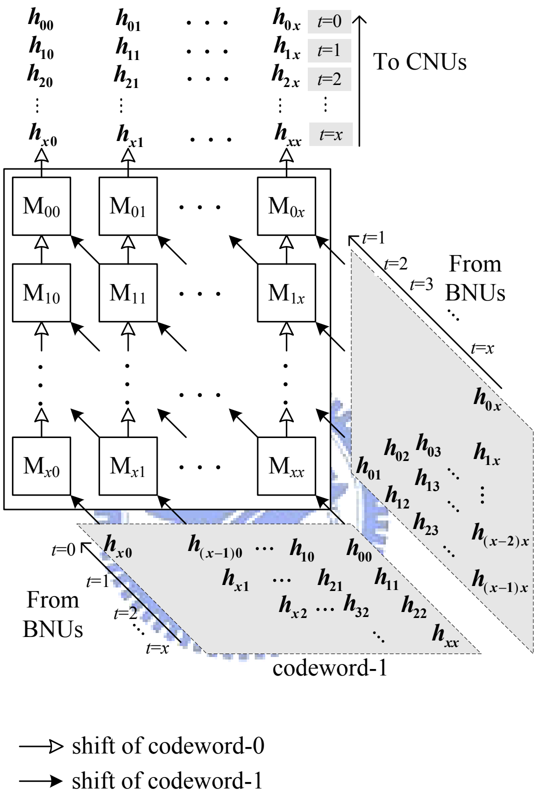
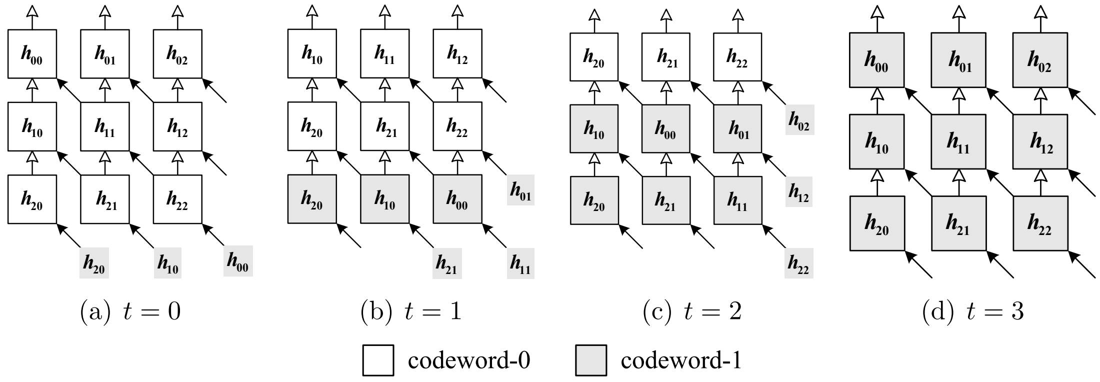

h_{00} h_{01} . . . h_{0x} t=0
h_{10} h_{11} . . . h_{1x} t=1
h_{20} h_{21} . . . h_{2x} t=2
h_{x0} h_{x1} . . . h_{xx} t=x
To CNUs
M$_{00}$ M$_{01}$ M$_{0x}$
M$_{10}$ M$_{11}$ M$_{1x}$
M$_{x0}$ M$_{x1}$ M$_{xx}$
t=1 t=2 t=3 t=x
From BNUs
h_{0x} h_{02} h_{03} h_{1x} h_{01} h_{13} h_{12} h_{23} $h_{(x-2)x}$ $h_{(x-1)x}$
t=0 t=1 t=2 t=x
From BNUs
h_{x0} $h_{(x-1)0}$ ⋯ h_{10} h_{00}
h_{x1} ⋯ h_{21} h_{11}
h_{x2} ⋯ h_{32} h_{22}
h_{xx}
codeword-1
shift of codeword-0
shift of codeword-1

Figure 4.41: The RE operation for the MMU with $(x+1)^2$ sub-blocks


h_{00} h_{01} h_{02}
h_{10} h_{11} h_{12}
h_{20} h_{21} h_{22}
h_{20} h_{10} h_{00}
h_{10} h_{11} h_{12}
h_{20} h_{21} h_{22}
h_{01}
h_{20} h_{10} h_{00}
h_{21} h_{11}
h_{20} h_{21} h_{22}
h_{02}
h_{10} h_{00} h_{01}
h_{12}
h_{20} h_{21} h_{11}
h_{22}
h_{00} h_{01} h_{02}
h_{10} h_{11} h_{12}
h_{20} h_{21} h_{22}
□ codeword-0
▨ codeword-1

(a) $t = 0$ (b) $t = 1$ (c) $t = 2$ (d) $t = 3$

Figure 4.42: The MMU operation for $x = 2$

Note that in (4.89) the first summation comes from the sub-blocks $\mathrm{M}_{x1} \sim \mathrm{M}_{x(x-1)}$, the x from M_{xx}, and the second summation from $\mathrm{M}_{(x-1)x} \sim \mathrm{M}_{1x}$. Hence the overhead increases with the growing x. In addition, we provide an example for $x = 2$ in Fig. 4.42 at different time instances, $t = 0 \sim 3$. The MMU initially stores the information of codeword-0, and the operation proceeds according to Fig. 4.41. At $t = 3$, the messages from the BNUs are all shifted into the MMU.

Chapter 5

Applications of Channel Decoders

5.1 Universal Reed-Solomon decoder

The Reed-Solomon code is well acceptable in many storage applications and digital communication systems for its excellent burst error correction capability. Those systems have different code specifications depending on the performance requirements. Table 5.1 lists some applications for RS codes. We can find that the differences are not only in the code rates, but in the finite field definitions as well. Therefore, based on the universal finite field arithmetic, we investigate the cost efficient RS decoder that can meets various system specifications.

Table 5.1: RS code specifications in various applications

Application		Code specification
DVB-T [185], DVB-S [186]		(204,188) RS code over $GF(2^8)$
ITU-T J.83 [187]	Annex A, C	(204,188) RS code over $GF(2^8)$
	Annex B	(128,122) extended RS code over $GF(2^7)$
	Annex D	(207,187) RS code over $GF(2^8)$
G.975 FEC [188]		(255,239) RS code over $GF(2^8)$
Flash memory		(526,518) RS code over $GF(2^10)$
DVD	row	(182,172) RS code over $GF(2^8)$
	column	(208,192) RS code over $GF(2^8)$
Blu-ray Disc	LDC	(248,216) RS code over $GF(2^8)$
	BIS	(62,30) RS code over $GF(2^8)$

In Table 5.1, since most RS codes are over $GF(2^8)$, we define the universal (n, k) RS decoder that supports $n \leq 255$, $t \leq 16$ with or without erasures, and $GF(2^m)$ with $m \leq 8$ as well as any irreducible polynomials.

Fig. 5.1 is the universal RS decoder based on the Montgomery multiplication algorithm and look-up table based divider [38]. The dual-bank static RAM (SRAM) of size 1k bytes is embedded to buffer four received codewords waiting for being corrected. In the syndrome calculator, there are 16 syndrome cells that concurrently compute syndrome values $\tilde{S}_1 \sim \tilde{S}_{16}$, which is shown in Fig. 2.5. In the case of $t = 16$ with erasures, the 16 syndrome cells are sufficient, however, they can simultaneously supports the $t = 8$ error only decoding. Since the key equation solver is designed to accommodate the error location polynomial with degree 16, we can modified the syndrome calculator for decoding at most 16 errors without erasures ($t = 16$). According to (2.71), only the FG_1 and the FG_2 need to be configured for calculating $\tilde{S}_{17} \sim \tilde{S}_{32}$. Consequently, $\tilde{S}_1 \sim \tilde{S}_{16}$ are first calculated form the received codeword that is also written into the FIFO memory, and $\tilde{S}_{17} \sim \tilde{S}_{32}$ are subsequently obtained from the same codeword read form the FIFO memory. The erasure generator produces the erasure information $\alpha^{l_1+8} \sim \alpha^{l_u+8}$ according to the erasure location in the codeword. Generally, we can use the cell in Fig. 2.6 to generate these erasure information delivered to the key equation solver. Based on the Berlekamp-Massey algorithm, we implement the key equation solver to determine the Forney syndrome polynomial $\tilde{T}(x)$, errata location polynomial $\tilde{\Lambda}(x)$, and the error value polynomial $\tilde{\omega}(x)$. As shown in Fig. 2.7, the design is the inversionless decomposed

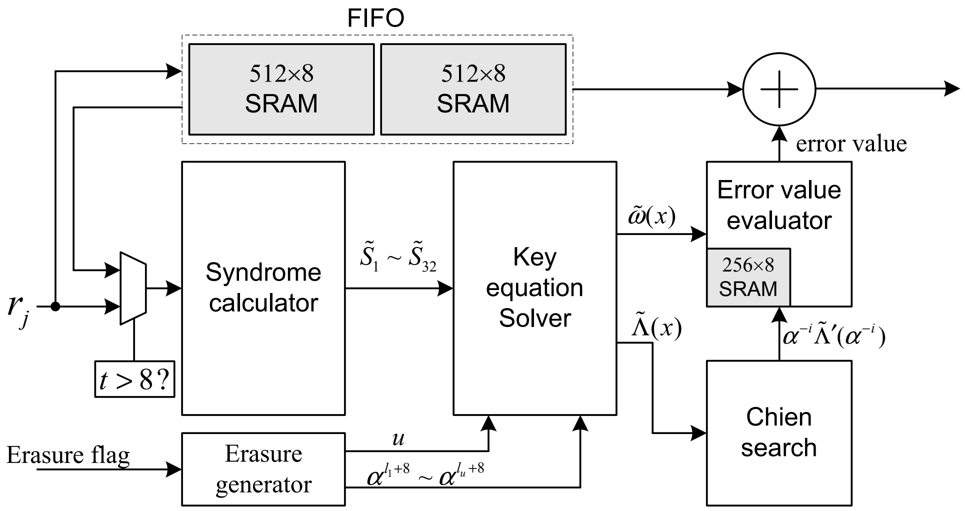


Figure 5.1: The universal RS decoder architecture

architecture with only three universal finite field multipliers [33]. In the Chien search module, we use the architecture in Fig. 2.8 that not only checks the roots of $\tilde{\Lambda}(x) = 0$, but also generates $\alpha^{-i}\tilde{\Lambda}'(\alpha^{-i})$ for error value evaluation. Finally, according to (2.91) or (2.92), the error value evaluator will estimate the error values e_κ and the erasure values e_ρ with the architecture in Fig. 2.9. In the present RS decoder, the divider in Fig. 2.9 is based on the table look-up approach; therefore, we allocate a 256×8 SRAM for storing the inversion table that is created by the α^{-i+8} and the α^i generators (see Fig. 2.10).

The universal RS decoder is implemented with the standard 0.18-μm 1P6M CMOS process and measured to achieve the maximum 160MHz clock rate at the supply voltage 1.62V$\sim$ 1.98V. The chip summary is also listed in Table 5.2. If the chip works in the $GF(2^8)$ mode,

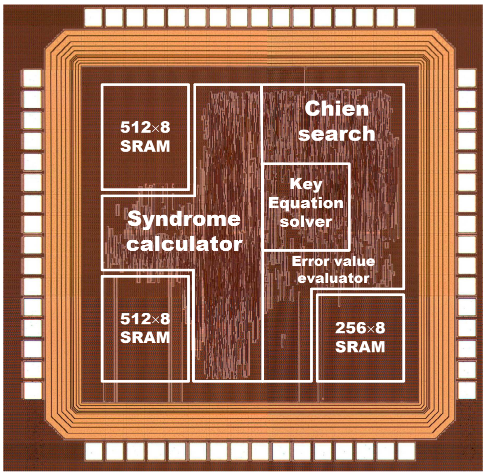


Figure 5.2: The 0.18-μm universal RS decoder microphoto

Table 5.2: The universal RS decoder chip summary

Technology	0.18-μm 1P6M CMOS
Chip size	2.25 mm^2
Core size	1.21 mm^2
Gate count	46.4k
Embedded SRAM	8k bits (FIFO memory) 2k bits (Inversion table)
Supply voltage	1.62V$\sim$ 1.98V
Clock rate	160MHz
Power consumption	68.1mW (1.8V and 160MHz)

the maximum decoding throughput is 8bits × 160MHz = 1.28Gb/s.

We also conduct an experiment based on the 0.13-μm 1P8M CMOS process. After static timing analysis (STA), the universal RS decoder in this experiment can achieve the maximum 300MHz clock rate while considering the worst speed corner and the coupling noise among signal wires at the 1.02V supply. The core area is about 0.36mm^2, and the average power consumption is 20.2mW, assuming 300MHz clock rate and the 1.2V supply voltage. We also compare different RS decoder implementations in Table 5.3. Compared with other approaches, the proposed design has more flexibility while still has high decoding speed. The decoder in [36] applies the serial architecture to achieve the universality; hence, the throughput is limited. Notice that the gate count of the present decoder is also comparable with other single mode or multi-mode decoders.

Table 5.3: Comparison among RS decoders

Design	[189]	[190]	[36]	Proposed	Proposed
Technology	0.25-μm	0.35-μm	0.25-μm	0.13-μm	0.18-μm
(n, k)	single	variable	variable	variable	variable
t	8	$1 \sim 8$	$1 \sim 8$	$1 \sim 16$	$1 \sim 16$
Erasure	No	No	No	$1 \sim 16$	$1 \sim 16$
$GF(2^m)$	$m = 8$	$m = 8$	$m = 1 \sim 8$	$m = 1 \sim 8$	$m = 1 \sim 8$
$p(x)$	single	single	variable	variable	variable
Max. throughput	1.6Gb/s	800Mb/s	48Mb/s	2.4Gb/s	1.28Gb/s
Gate count	21k	34k	44k	54k	46k

We present a universal architecture for the error and erasure decoder. The proposed architecture can accommodate variable codeword length, correctable errors , different finite field degrees, and different primitive polynomials. Furthermore, the proposed decoder can support erasure correction without increasing any finite field multipliers. In summary, the decoder is not only flexible but cost efficient as well.

5.2 Low power Viterbi decoder for the IEEE 802.11a WLAN

The present Viterbi decoder targets the WLAN system specified in IEEE 802.11a [129]. Based on the orthogonal frequency division multiplexing (OFDM) and the forward error correction (FEC) coding, the system is able to transmit data with data rates up to 54Mb/s. Both phase-shift keying (PSK) modulation and quadrature amplitude modulation (QAM) are included to provide various data rates listed in Table 5.4. The FEC coding employs rate 1/2 convolutional encoder

$$G(D) = [\ 1 + D^2 + D^3 + D^5 + D^6 \quad 1 + D + D^2 + D^3 + D^6\ ] \tag{5.1}$$

and derives higher rates from it by puncturing. The design signal-to-noise ratios (SNRs) for FEC are the targets to achieve a packet error rate (PER) of 10% in additive white Gaussian noise (AWGN) channel. As the requirement of data rates increase in wireless applications, the power consumption becomes an obvious design issue in system-level integrated circuit. Therefore, we explore the system level behavior to remove redundant operations and achieve a better system architecture in terms of power dissipation and complexity.

Table 5.4: Transmission modes of the IEEE 802.11a WLAN

Data rate (Mb/s)	Modulation	Code rate (R)	Design SNR for FEC (dB)
6	BPSK	1/2	$1 \sim 2$
9	BPSK	3/4	$3 \sim 4$
12	QPSK	1/2	$4 \sim 5$
18	QPSK	3/4	$6 \sim 7$
24	16-QAM	1/2	$9 \sim 10$
36	16-QAM	3/4	$13 \sim 14$
48	64-QAM	2/3	$17 \sim 18$
54	64-QAM	3/4	$18 \sim 19$

Fig. 5.3 shows the architecture of the proposed design. The de-puncture unit can sup-

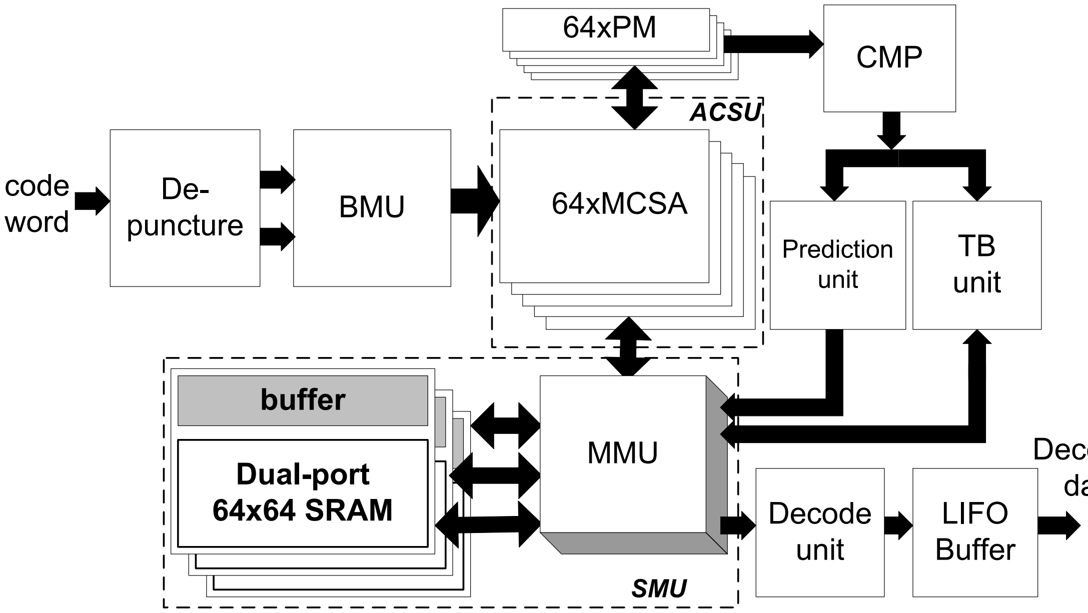


Figure 5.3: Block diagram of the proposed Viterbi decoder

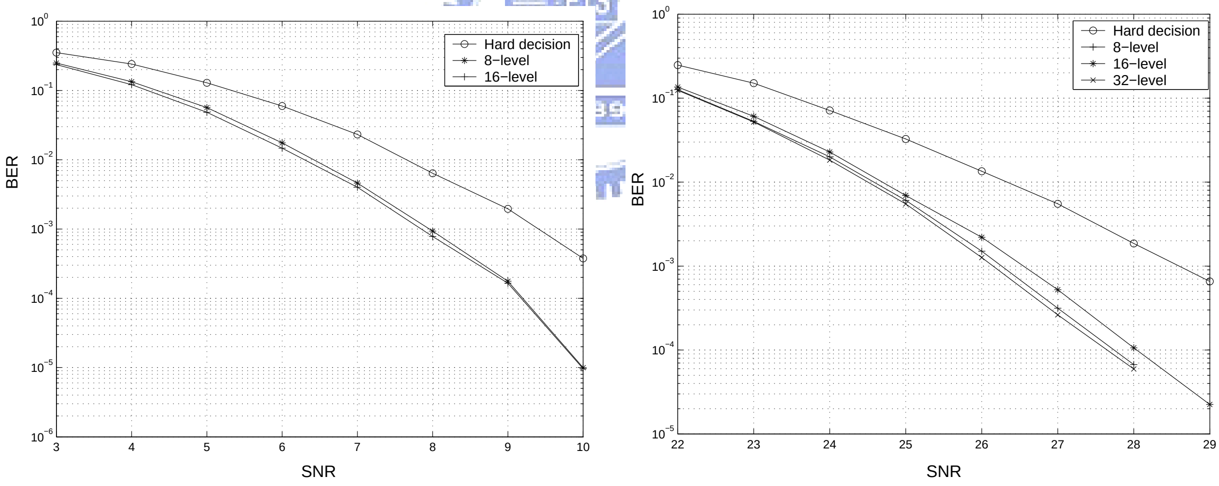
port various coding rates defined in [129]. The BMU calculates the distance between the received symbols and codewords on the branches. As illustrated in Fig. 3.23 and Fig. 3.24 the 64 parallel simplified MCSAs perform comparison among candidate paths to determine survivors and compute the corresponding path metrics. The SMU based on the k-pointer even algorithm with k=3 is constructed by the 6-bank memory architecture [126]. Each memory bank equips a state buffer which retains the state sequence that would probably be reused. The memory management unit (MMU) governs the operations of SMU, including the path merging and path prediction operations. The comparator (CMP) computes the minimum path metric and outputs the corresponding state to the prediction unit as well as the TB unit. The prediction unit will generate a possible state sequence to increase buffer reuse efficiency.

In the BMU, we have to minimize the quantization loss in terms of hardware cost, increasing linearly with the quantization bit number. The quantization level and stepsize vary with modulation types and channel conditions. For different quantization levels and modulation types, Table 5.5 summarizes the performance improvement in terms of SNR over the hard decision decoding. All of the quantization schemes are set to be uniform

Table 5.5: Improvement of soft-decision Viterbi decoder compared to the hard decision decoding

	8-level	16-level	32-level
BPSK	1.7dB	1.9 dB	2dB
QPSK	1.7dB	1.9 dB	2dB
16-QAM	2.3dB	2.87 dB	2.96dB
64-QAM	2.2dB	2.6 dB	2.75dB

quantization and optimal stepsize [128]. While considering the indoor multipath channel, we use the Rayleigh fading with a root mean square (rms) delay spread of 50ns. Fig. 5.4 shows the simulation results of two extreme cases. Note that there are slight improvements from 8-level to 16-level in BPSK case and from 16-level to 32-level in 64-QAM case. In order to achieve a good compromise between performance and complexity, 16-level soft decision will be our choice.



(a) BPSK and $R = 1/2$

(b) 64-QAM and $R = 3/4$

Figure 5.4: Simulation results of the soft decision Viterbi decoder in multipath channel

The overhead that implements the buffer and additional control circuit in the SMU is less than 6%. Since the maximum truncation length is set to $T = 64$, each bank sized to 32×64 bits contains a 38 bits buffer that reduces a lot of memory read operations. The reduction of memory access in the 54Mb/s mode and the AWGN channel is presented in Fig. 5.5. In the conventional SMU, the memory access number is constant. On the other hand, in the proposed SMU the number of memory read operations degrades as the channel condition becomes better. If the SNR is grater than 17dB, the average memory access will be dominated by writing operations.

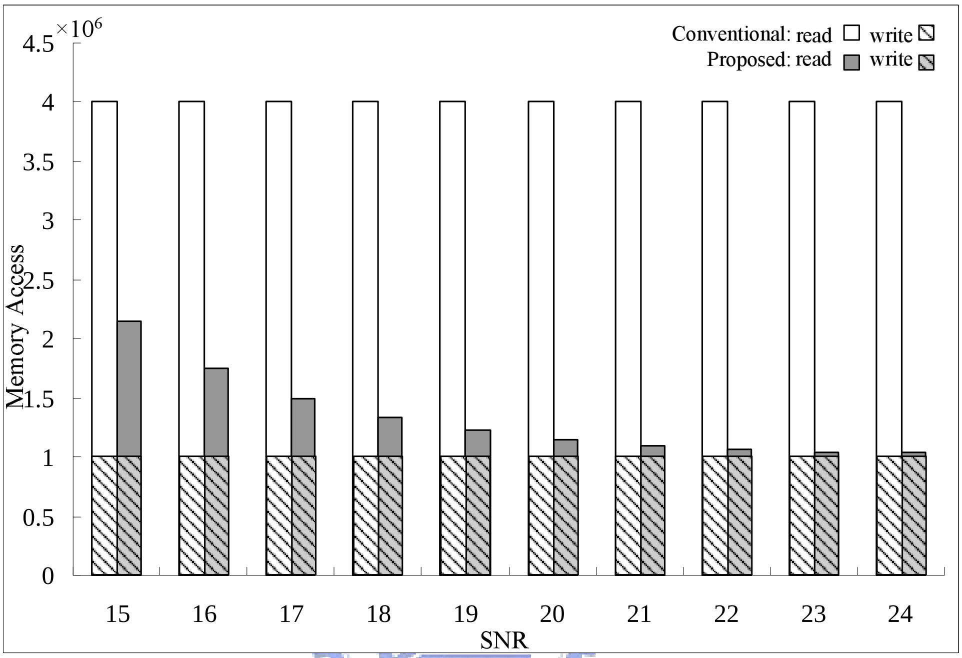


Figure 5.5: Comparison of the memory access operations

The decoder is implemented and fabricated in the 0.18-μm 1P6M CMOS process. The chip shown in Fig. 5.6 has been measured and summarized in Table 5.6. The core size is 3.06mm^2 containing 49k gates and 12k bits embedded SRAM. With the 1.64V $\sim$ 1.98V supply, the chip is measured to work at the 100MHz clock rate, which is equivalent to the 75Mb/s decoding data rate when $R = 3/4$. Fig. 5.7 shows the measured power consumption in terms of different data rates listed in Table 5.4. The conventional data in Fig. 5.7 is obtained from this chip with the path merging and path prediction functions being turned off. The more detailed information is illustrated in Fig. 5.8 where 54Mb/s data rate is measured at SNR=19dB. The increased power dissipation in clock tree and CMP unit is due to the additional buffer and the computation of the minimum path metric. The power

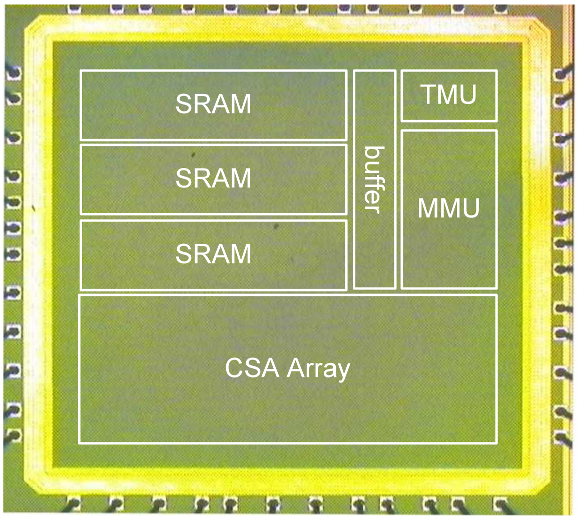


Figure 5.6: Microphoto of the Viterbi decoder test chip

consumption which varies with the channel condition has a 30%∼40% reduction as compared to conventional designs.

Table 5.6: The Viterbi decoder chip summary

Technology	0.18-μm 1P6M CMOS
Chip size	5.62mm^2
Core size	3.06mm^2
Gate count	49k
Embedded SRAM	12k bits
Supply voltage	1.64V $\sim$ 1.98V
Clock rate	100MHz
Power consumption	68mW at 1.8V (54Mb/s and SNR=19dB)

A high performance and power reduction Viterbi decoder is presented for the WLAN applications. The modified CSA architecture reduces the hardware complexity as well as the power dissipation. Furthermore, with the path merging and the path prediction features, the memory access drops more than 70% on the average. As a result, the power consumption decreases due to the reduced memory access operations. The proposed design not only considers the error correction capacity, but also provides a high speed and power efficient solution.

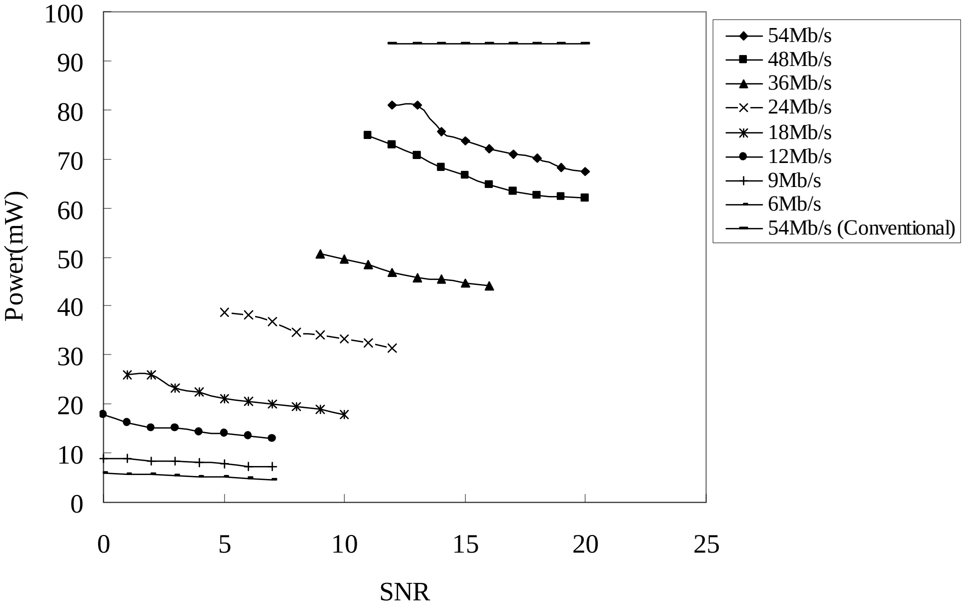


Figure 5.7: Measured power consumption at different data rates defined in the IEEE 802.11a

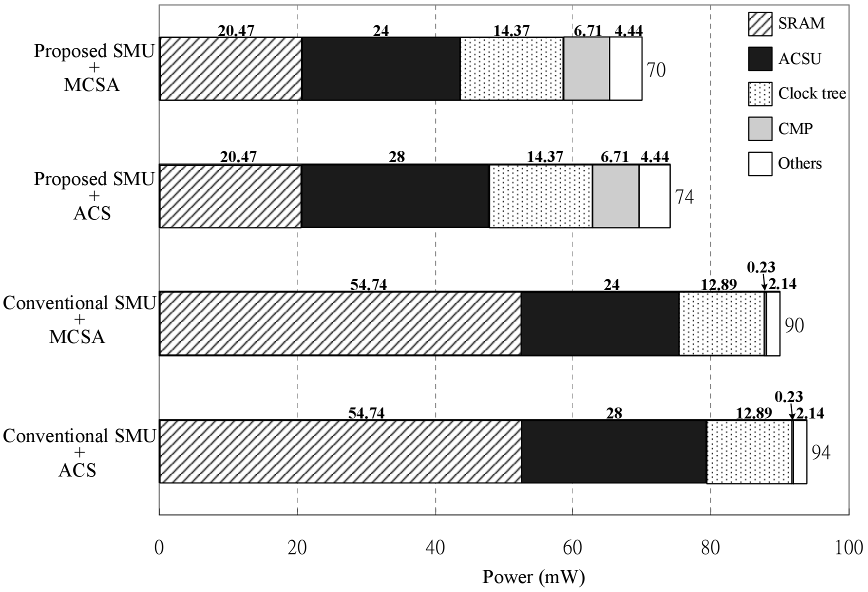


Figure 5.8: Difference in power consumption with and without the proposed algorithm

5.3 A Turbo/Viterbi decoder for the 3GPP2 mobile communication

In the third generation (3G) mobile wireless communication [63], both turbo and convolutional codes are specified for high speed data and speech transmission. Higher data rates and larger block lengths in the turbo code indicate more design challenges due to large memory size and bandwidth. On the other hand, the higher data rate requires much more memory bandwidth, resulting in more design complexity and higher power dissipation.

The major concerns in building mobile wearable devices are the size and weight where the battery is a large portion [191]. Thus, the key to reduce the battery size is the lower power constraint. Low power designs of turbo decoders in [192–195] incorporate the early termination of the iterative decoding for better channel conditions. In [196,197], the memory blocks are optimized to achieve a significant power reduction. The sub-optimal approaches that reduce the number of states or paths in trellis are also presented as power saving techniques, but the performance becomes degraded. The turbo decoders with the block length 5,114 are also reported in [58] and [60]. For 3G application, the integration of turbo and Viterbi decoders is also reported in [58]. However, there is little research available on the implementation of the large turbo code in 3GPP2 system [63].

We present a channel decoder that integrates both the turbo and the Viterbi decodings with the optimized memory organization as well as the low power dissipation. The decoder is designed with a single SISO decoder architecture based on the Max-Log-MAP algorithm, and the embedded interleaver is implemented with the modest memory size. It also features a cache buffer to increase the bandwidth efficiency for the SISO decoder and reduce the external memory access.

The trellis decoding structure of both decoder enables the resource sharing of the ACS units and the memory units, leading to a area efficient architecture. In the Max-Log-MAP algorithm, the forward metric and the backward metric computations in Fig. 3.52 should be

$$\bar{\alpha}(S_0^{(t)}) = \max[\bar{\alpha}(S_0^{(t-1)}) + \bar{\gamma}(S_0^{(t-1)}, S_0^{(t)}), \bar{\alpha}(S_1^{(t-1)}) + \bar{\gamma}(S_1^{(t-1)}, S_0^{(t)})] \tag{5.2}$$

$$\bar{\beta}(S_0^{(t)}) = \max[\bar{\beta}(S_0^{(t+1)}) + \bar{\gamma}(S_0^{(t+1)}, S_0^{(t)}), \bar{\beta}(S_2^{(t+1)}) + \bar{\gamma}(S_2^{(t+1)}, S_0^{(t)})], \tag{5.3}$$

which are ACS operations. Consequently, the Viterbi decoding algorithm that applies ACS computaitons to update path metrics can also utilize (5.2) or (5.3).

Fig. 5.9 shows the decoder architecture in the turbo decoding (TD) mode where the active components are highlighted. The TD consists of three ACS groups for α, β, and β_d recursions in Fig. 3.53, and each ACS group contains eight ACS units according to the turbo encoder in (4.25). The SISO decoder processes three consecutive sub-blocks concurrently for different strategies in the windowed MAP algorithm.

As shown in Fig. 4.15, the cache buffer require four data access ports for three read operations and one write operation. From the previous discussion, either the multi-port memory or the high working frequency can provide sufficient memory bandwidth for the cache buffer. However, the former has large area overhead while the latter would complicate circuit implementation. We use a hybrid cache solution where a dual-port memory works at the double clock frequency to provide the quadruple-port function. Fig. 5.10 represents the cache architecture where Since the data ports and BMUs are directly connected, the multiplexers are eliminated, leading to less signal routing complexity as well as power dissipation.

In the Viterbi decoding (VD) mode, 256 states trellis decoding is implemented with 1/2, 1/3, 1/4, and 1/6 coding rates. As shown in Fig. 5.11, The ACS-α and ACS-β_d that contains 16 ACS units perform these 256 ACS operations in 16 cycles. The memory for the interleaver of TD is treated as the survivor memory. The traceback (TB) read operation is performed separately from ACS operations due to the limited memory bandwidth and takes additional two cycles based on the 3-point even algorithm [47, 126]. The decode read follows after the second traceback read and outputs a decoded bit. The decoding flow is illustrated in Fig. 5.12. Averagely, to decode one data bit, it takes 19 cycles where the ACS units take 16 cycles to write new decisions, the TB read operation spends two cycles, and the decode read operation needs one cycle. In a 100MHz clock rate, the Viterbi decoder can achieve the maximum throughput of 5.26Mb/s.

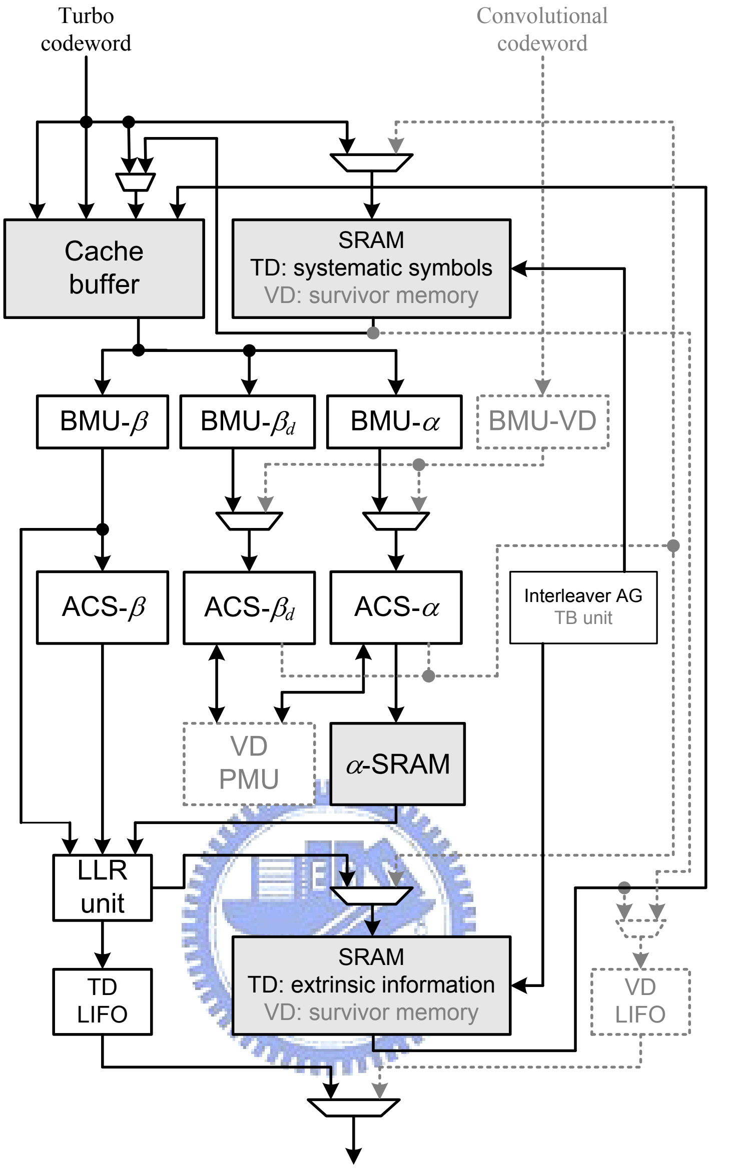


Figure 5.9: The decoder in the turbo mode

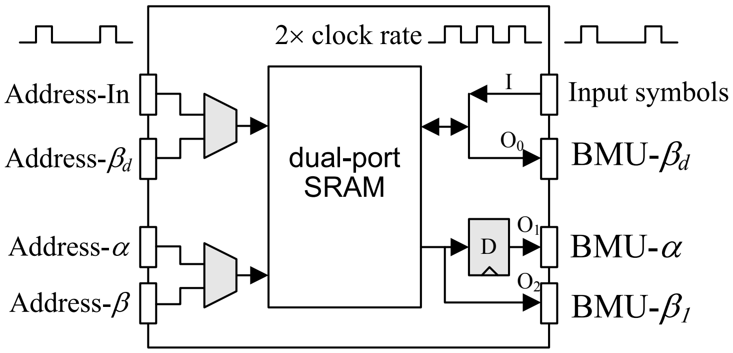


Figure 5.10: The quadruple-port cache buffer architecture

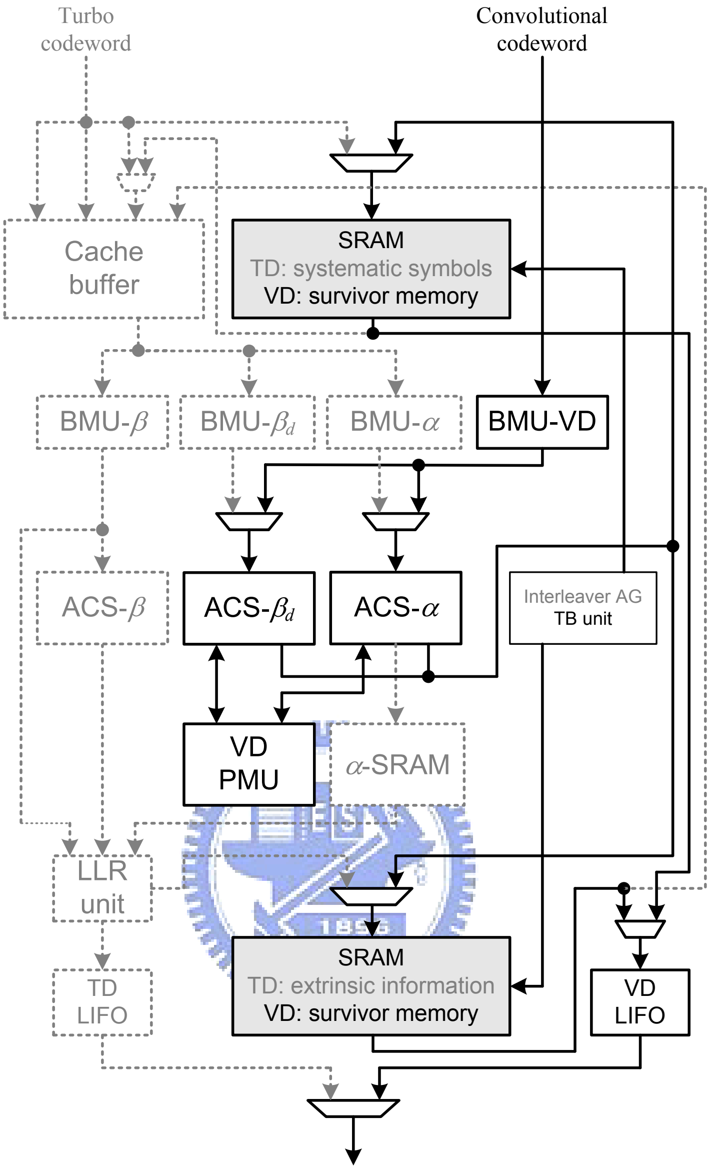


Figure 5.11: The decoder in the Viterbi mode

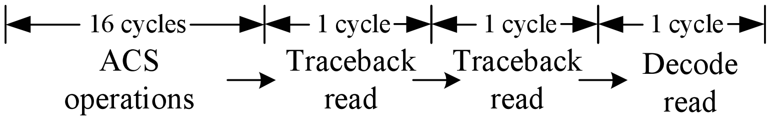


Figure 5.12: The timing diagram of the Viterbi decoder

Table 5.7: Performance loss of fixed representations in turbo decoding

Format	Input symbols		$L_e(u_t)$	α
	$R = 1/2$	$R = 1/5$	$R = 1/5$	$R = 1/5$
7.2	< 0.01dB	< 0.01dB	< 0.01dB	< 0.01dB
6.2	< 0.01dB	< 0.01dB	< 0.01dB	< 0.01dB
5.2	< 0.01dB	< 0.01dB	< 0.01dB	> 10dB
4.2	0.2dB	0.017dB	0.05dB	> 10dB
3.3	< 0.01dB	0.015dB	2dB	> 10dB
3.2	0.2dB	0.02dB	4.3dB	> 10dB

The fixed point representation of the quantities in the turbo decoder is determined from the received symbols. We also use the notation $n_i.n_f$ to denote the $(n_i + n_f)$ bits quantization. Many different formats of input symbols were simulated in additive white Gaussian noise (AWGN) channel and summarized in Table 5.7. The performance loss due to the quantization is measured in terms of the signal to noise ratio (SNR) at BER=2×10^{-6}, and the coding rate $R = 1/2$ is derived from $R = 1/5$ by puncturing [63]. In Table 5.7, we first noticed that the 5.2 format is optimal for both the $R = 1/2$ and the $R = 1/5$ cases. Nevertheless, the 3.3 format is still suitable for the $R = 1/2$ code, but induces additional 0.005dB loss in the $R = 1/5$ code. In order to reduce n_i for $L_e(u_t)$, α, and β, we select the 3.3 scheme and deduced the other quantities form this format.

According to the upper bounds in (4.31) and (4.32), we know that both $\Delta\alpha_t$ and $\Delta\gamma_t$ are determined by $L_a(u_t)$ which is the $L_e(u_t)$ output from the other SISO decoder. The extrinsic information $L_e(u_t)$ also decides the memory size of interleaver and de-interleaver that occupy a large chip area. Consequently, we first bound $L_e(u_t)$ and optimize its range to reduce the memory size. Table 5.7 shows the simulation results of different fixed representations for $L_e(u_t)$. Although the 5.2 format has the least loss, the 4.2 format will be chosen to save the memory size due to the large block length (N=20,730). As a result, $L_e(u_t)$ is bounded to ±8 and the input symbols are bounded to ±4; in addition, the range of $\Delta\alpha_t$ and $\Delta\gamma_t$ can be determined as 96 and 40, and $|L(u_t)| \leq 136$ based on (4.35), indicating that α and γ require

Table 5.8: Summary of fixed representation in turbo decoding

quantities	Input symbols	$L_a(u_t)$	α	β	γ
width	6(3.3)	6(4.2)	8(6.2)	8(6.2)	8(6.2)

$n_i = 7$ and $n_i = 6$ respectively. For α, as shown in Table 5.7, the 6.2 format is sufficient due to the looser bound in the noisy channel [170]. The backward metric β can also be analyzed and will obtain the same results as α. Table 5.8 summarizes the fixed representations of quantities in the turbo decoder. As shown in Fig. 5.13, the performance loss is 0.06dB in the waterfall region and 0.5dB when BER=2×10^{-6}.

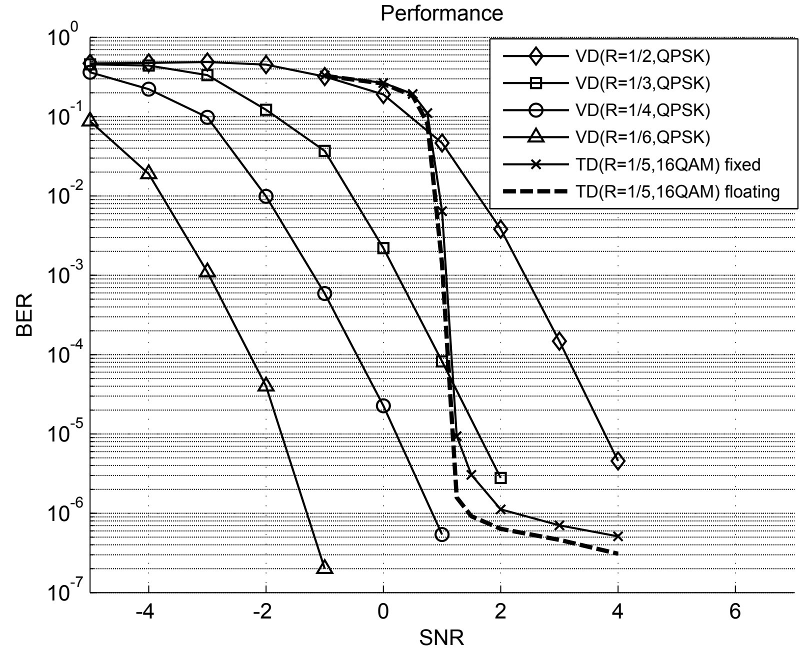


Figure 5.13: The BER performance of the channel decoder

The performance of the Viterbi decoder is determined by the precision of input symbols and path metrics. For the quadrature amplitude modulation (QAM), four bits or more are required to minimize the performance loss [47]. The word length of path metrics depends on the code rate R, the precision of input symbols, and the trackback length T. For the code with constraint length $K = 9$, the value of T should be $32 \sim 64$, depending on the channel condition [100]. Hence, with the modulo normalization scheme [131], the four bits soft input,

Table 5.9: Performance loss of different path metric precisions

Number of bits	13	11	10	9
Loss(BER=2×10^{-6})	$\sim$ 0dB	$\sim$ 0 dB	$\sim$ 0dB	0.4dB

$R = 1/6$, and $T = 64$, the word length path metrics should be 13 bits. Nevertheless, the simulation results in Table 5.9 show that 10 bits are sufficient in the AWGN channel. We also provide the performance the fixed point Viterbi decoder in Fig. 5.13.

The decoder is implemented with the 0.18-μm standard CMOS process. In the TD mode, the sub-block length T_{sb} is set to 20, and two clock domains are used in the memory and the datapath respectively. Since the double clock rate provides the memory with higher bandwidth, the single-port memory is sufficient in the proposed design except the cache memory.

The specification report shows the dual-port memory in Fig. 5.10 is 0.103mm^2, leading to 30% area reduction from Fig. 4.15 whose area is 0.146mm^2. Two SP-SRAMs of 20,730 words are included in the decoder for the systematic symbols and the extrinsic information. The input and the output ports are implemented by the time division multiplexing approach that avoids the use of multi-port memories. As compared with DP-SRAM design, the proposed SP-SRAM approach has only 1/3 area with the double clock rate.

In Fig. 5.14, the chip size is 11.56mm^2, and the core size is 7.29mm^2. The total gate count is about 115k including the path metric memory for the Viterbi decoder. Three SP-SRAMs and one DP-SRAM are embedded in the chip with the total size of 251.64k bits. Table 5.10 summaries the chip features where the maximum data rate is obtained from the post-layout simulation and verified with the chip measurement. The chip has been tested at 100MHz (50MHz in datapath) under the 1.60$\sim$1.98V supply and can provide the 4.52Mb/s turbo decoding with six iterations and the 5.26Mb/s Viterbi decoding. In the 0.18-μm technology, a 10% drop on the power grid voltage (IR-drop) will lead to 5% $\sim$ 6% changes in timing [198]. For wearable device, the supply voltage will vary with the capacity of batteries. Therefore,

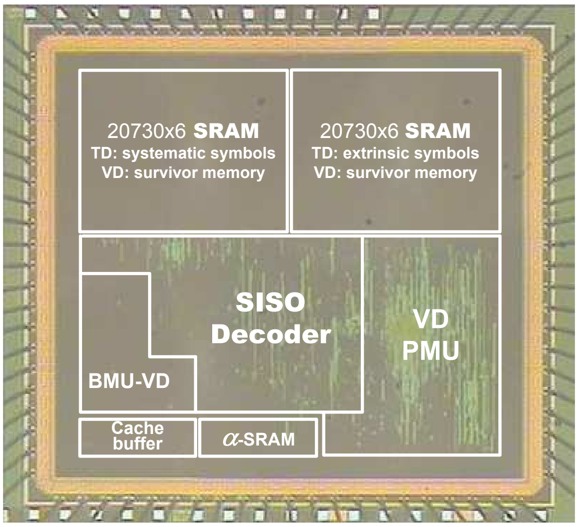


Figure 5.14: The microphoto of the 0.18-μm decoder chip

Table 5.10: Summary of the decoder chip

Technology	0.18-μm 1P6M CMOS
Supply voltage	1.60V$\sim$ 1.98V
Chip size	11.56mm^2
Core size	7.29mm^2
Embedded SRAM	251.64k bits
Supported coding rate	1/5 for the turbo decoding 1/2,1/3,1/4,1/6 for Viterbi decoding
Maximum data rate	4.52Mb/s in the turbo decoder [1] 5.26Mb/s in the Viterbi decoder

[1] 6 iterations

the power integrity is also analyzed to insure functionality and timing with the non-ideal battery property and IR-drop. Table 5.11 shows the analysis results, including IR drop and electron-migration (EM) risk. The IR drop analysis make sure the transistors in this chip will have proper supply voltage during operation. The worst operating voltage due to IR drop is 1.603V while the supply is 1.62V. The EM risk is a reliability measure and defined as the probability that chip will fail within ten years due to EM. The worst value is 1.35×10^{-5} while the supply is 1.98V. The power distribution of the major blocks is also illustrated in Fig. 5.15 where the TD is simulated with the $N = 20,730$, six decoding iterations, the 16-

Table 5.11: Power Integrity Analysis

Function	R	Type	Supply Voltage		
			1.62V	1.8V	1.98V
Turbo decoding [1]	$\frac{1}{5}$	IR drop	17mV	18.78mV	22.89mV
		EM risk	2.23×10^{-6}	4.46×10^{-6}	1.35×10^{-5}
Viterbi decoding[2]	$\frac{1}{6}$	IR drop	5.39mV	5.97mV	7.59mV
		EM risk	1.99×10^{-12}	5.43×10^{-12}	5.67×10^{-11}
	$\frac{1}{4}$	IR drop	5.03mV	5.57mV	7.15mV
		EM risk	1.12×10^{-12}	3.08×10^{-12}	4.38×10^{-11}
	$\frac{1}{3}$	IR drop	4.58mV	5.06mV	6.55mV
		EM risk	3.10×10^{-13}	9.08×10^{-13}	3.03×10^{-11}
	$\frac{1}{2}$	IR drop	4.44mV	4.91mV	6.35mV
		EM risk	1.97×10^{-13}	5.89×10^{-13}	2.46×10^{-11}

[1] 4.52Mb/s with 6 iterations
[2] 1Mb/s

QAM, and the input SNR of 1dB, while VD is simulated with the $R = 1/6$, the quadrature phase-shift keying (QPSK) modulation, and the SNR=-2dB.

With 1.8V supply, Table 5.12 shows the power consumption while decoding turbo and convolutional codes, and table 5.13 summarizes the differences between the proposed design and other turbo decoder chips. The energy efficiency is defined as the average energy consumed per bit within each decoding iteration (nJ/b/iter.). For this decoder with six iterations, the energy efficiency will be

$$\frac{\text{83mW}}{6 \times \text{3.1Mb/s}} = \text{4.46nJ/b/iter.}\ .$$

In this section, we present a unified turbo and Viterbi decoder chip with less memory usage and low power consumption. The memory size is reduced by data scheduling for the interleaver and the single SISO decoder. Furthermore, the power consumption is improved by the efficient memory design and the less data bandwidth for the codeword input. At the

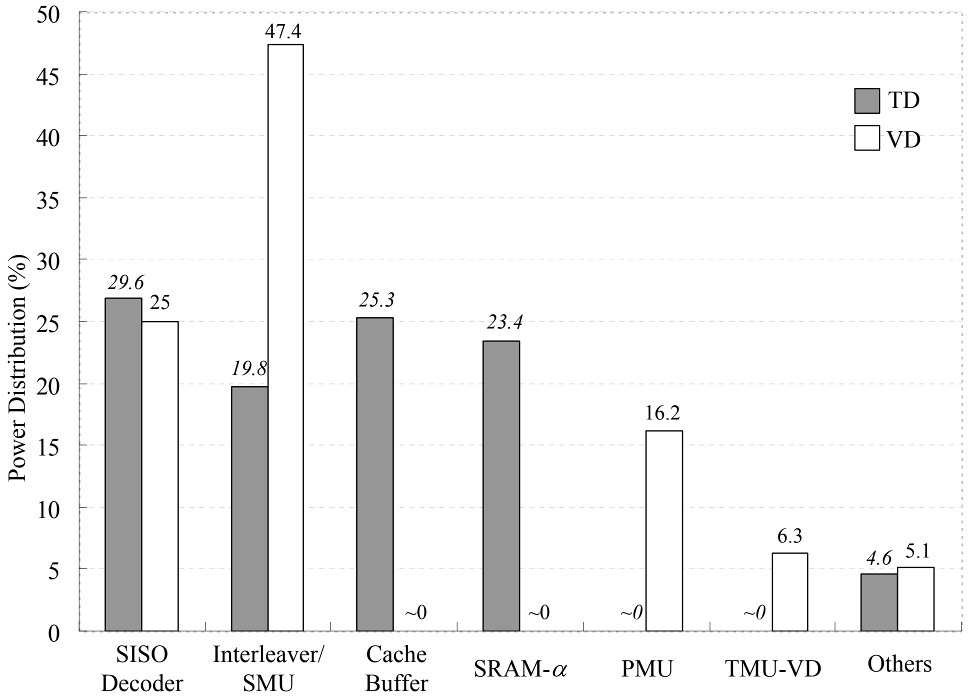


Figure 5.15: The power distribution of the major blocks

3.1Mb/s data rate, the power consumption is about 83mW in decoding a turbo code with the block length of 20,730. The chip is also designed to work reliably with the wider supply voltage range.

Table 5.12: Power consumption of the decoder chip

Mode	Data rate	Power	SNR
Turbo[1] mode	4.52Mb/s	121 mW	1dB
	3.1Mb/s	83 mW	1dB
	1Mb/s	29.5 mW	1dB
Viterbi[2] mode	5.26Mb/s	116.46 mW	3dB
	1Mb/s	25.1 mW	3dB

[1] R=1/5 and 16-QAM

[2] R=1/2 and QPSK

Table 5.13: Comparison of different turbo decoder chip

	Proposed design	[58]	[60]
Coding rate (R)	1/5	1/3	1/3
Block length	20,730	5,114	5,114
Data rate (Mb/s)	4.52 6 iterations	2.048 10 iterations	24 6 iterations
Technology	0.18-μm	0.18-μm	0.18-μm
Chip core size (mm^2)	7.29	9	14.5[1]
Energy efficiency (nJ/b/iter.)	4.46	14.25	10

[1] Without Viterbi decoder

5.4 High speed channel decoders for UWB applications

Ultra-wideband (UWB) is an emerging wireless physical(PHY)-layer technology that uses a very large bandwidth [199, 200]. By its rule-making proposal in 2002, the Federal Communications Commission (FCC) unleashed 3.1GHz to 10.6GHz RF band for increasing high-speed data transmission. The multi-band orthogonal frequency-division multiplexing (MB-OFDM) PHY-layer proposal indicates the coded OFDM based baseband solution can provide up to 480Mb/s within 2m desired range for 528MHz UWB system [136]. To enhance overall system performance, the convolutional codes and interleaving techniques are applied in the forward error correction (FEC) mechanism, whose block diagram is shown in Fig. 5.16.

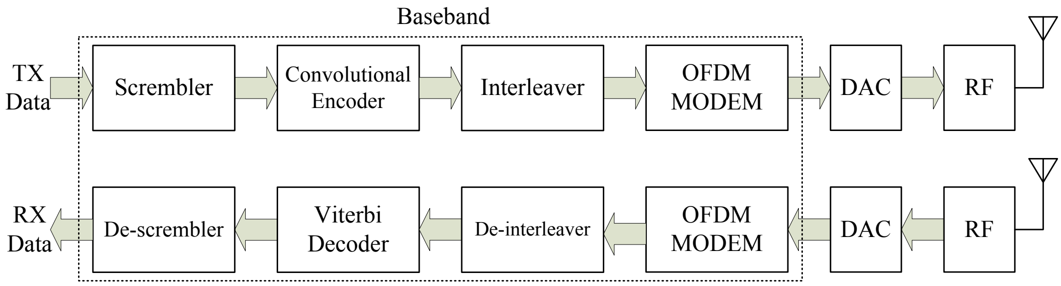


Figure 5.16: Block diagram of MB-OFDM UWB systems.

The convolutional encoder is based on the rate 1/3 mother encoder

$$G(D) = [\ 1 + D^2 + D^3 + D^5 + D^6 \quad 1 + D^2 + D^3 + D^4 + D^6 \quad 1 + D + D^2 + D^3 + D^6\ ], \tag{5.4}$$

and also punctured to $R = 1/2, 5/8, 3/4$.

Since the QPSK modulation is applied [136], we choose the 3 bits quantization for the input symbols, referring to the results in Table 5.5. If the truncation length $T = 48 \approx 6K$, the bit number of path metrics should be $10 \sim 11$ bits. Form the simulation results, we find nine bits are sufficient to achieve 0.1dB∼0.2dB performance loss in SNR. The parameters of the Viterbi decoder are listed in Table 5.14, and the performance figure in the AWGN channel is also in Fig. 5.17.

Table 5.14: Parameters of the Viterbi decoder

State number	64
Coding rate (R)	1/3, 1/2, 5/8, 3/4
Path Metric	9 bits
Branch Metric	5 bits
Input Symbol	3 bits
Truncation length	48 (R=1/3)
	64 (R=1/2)
	80 (R=5/8)
	96 (R=3/4)

The circuit implementation of Viterbi decoders are completed based on the proposed high radix and 2-D ACS structures where $M = 4$ and $p = q = 2$. Considering the high throughput requirement, the register-exchange approach is applied to the SMU with quite different structures in radix-16 and radix-4×4 designs. Within the $t \sim t+4$ trellis, there are 64 16-to-1 multiplexers in the radix-16 SMU whereas only 4-to-1 multiplexers are necessary in the radix-4×4 SMU, and their number is 128. If the 16-to-1 multiplexer is realized with five 4-to-1 multiplexers, the multiplexer number in the radix-16 SMU is $\frac{5 \times 64}{128} = 2.5$ times as many as that in the radix-4×4 SMU.

The Viterbi decoders have been implemented by using the 1.8V 0.18-μm 1P6M CMOS

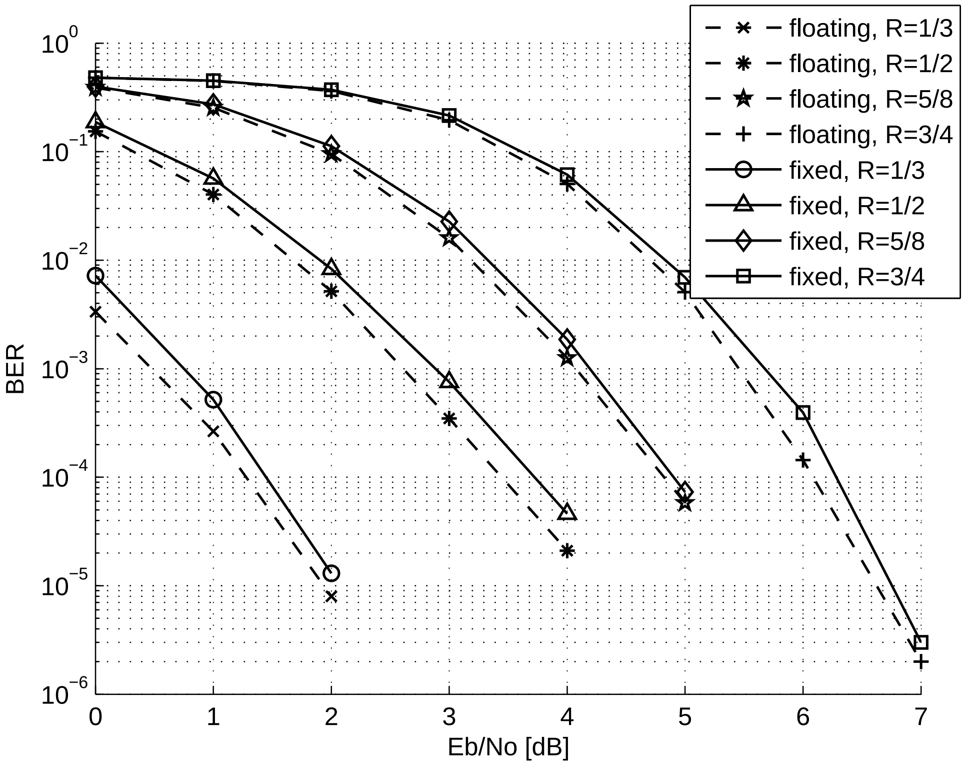


Figure 5.17: The Viterbi decoder performance based on Table 5.14

technology and the 1.2V 0.13-μm 1P8M CMOS technology. Data throughput is estimated form the static timing analysis (STA) while considering the 1.62V supply for 0.18-μm, 1.08V supply for 0.13-μm, the worst speed corner, and the coupling noise due to the crosstalk effect on signal wires. Table 5.15 and Table 5.16 summarize the results with tight timing constraints. The gate count(G_N) is a measure of the total standard cell area after the layout implementation, and ΔG_N indicates the gate count increase before and after the physical

Table 5.15: Implementation results with timing critical constraints

0.18-μm 1P6M	Data rate (Mb/s)	Area (mm^2)	N_G	ΔN_G	Density
ACS-16	513	9.30	740.0k	151.1k	0.79
ACS-16(RT-1)	623	15.21	1129.9k	190.4k	0.74
ACS-4 × 4[1]	427	4.41	310.4k	76.4k	0.77
ACS-4 × 4(RT-2)	553	5.29	398.6k	36.6k	0.75
ACS-4 × 4(RT-3)	731	6.76	533.1k	114.6k	0.79

[1] This chip was fabricated, and the results showed the 500Mb/s data rate is achieved under 1.8V supply.

Table 5.16: Implementation results of timing critical constraints

0.13-μm 1P8M	Data rate (Mb/s)	Area (mm^2)	G_N	ΔG_N	Density
ACS-16	933	3.61	647.3k	84.7k	0.92
ACS-16(RT-1)	1,038	5.36	945.7k	212.3k	0.90
ACS-4 × 4	923	1.28	239.2k	29.1k	0.96
ACS-4 × 4(RT-2)	986	1.85	349.9k	45.6k	0.97
ACS-4 × 4(RT-3)	1,105	1.96	358.0k	43.9k	0.94

Table 5.17: Implementation results of 500Mb/s data rate

	Area (mm^2)	N_G	ΔN_G	Density	Power(mW)[a]
ACS-16	2.66	491.7k	66.1k	0.94	344
ACS-16(RT-1)	3.84	685.5k	82.0k	0.92	533
ACS-4 × 4	0.90	165.5k	5.2k	0.94	119
ACS-4 × 4(RT-2)	1.38	247.7k	8.4k	0.94	169
ACS-4 × 4(RT-3)	1.44	263.2k	9.9k	0.94	195

[a] 1.2V supply and 500Mb/s data rate

design. The density is measure from the chip implementation, and

$$\text{Density} = \frac{\text{total standard cell area}}{\text{Core area}} \times 100\% \tag{5.5}$$

Both ACS-16 with RT-1 and ACS-4 × 4 with RT-3 are shown to achieve the higher data rates, over 1Gb/s in the 0.13-μm technology. Notice that the ACS-4 × 4 Viterbi decoder is much smaller than the ACS-16 based one. Additionally, ACS-4 × 4 with RT-2 has the throughput similar to ACS-16, but requires only half area. The ACS-4 × 4 based decoders are more area efficient than the ACS-16 based ones because the less computational units and the simple signal routing result in not only the smaller G_N and ΔG_N, but also the higher chip density. Table 5.17 also shows the results when the Viterbi decoders target 500Mb/s data throughput. We can find that the ACS-4 × 4 based decoders have much smaller area and ΔG_N than the ACS-16 based ones. All these results confirm the analysis in Table 3.2, and the ACS-$2^p \times 2^q$ structure is shown to be more cost efficient for the high radix Viterbi decoders.

In Table 5.15, the ACS-4 × 4 Viterbi decoder chip fabricated with the 0.18-μm process

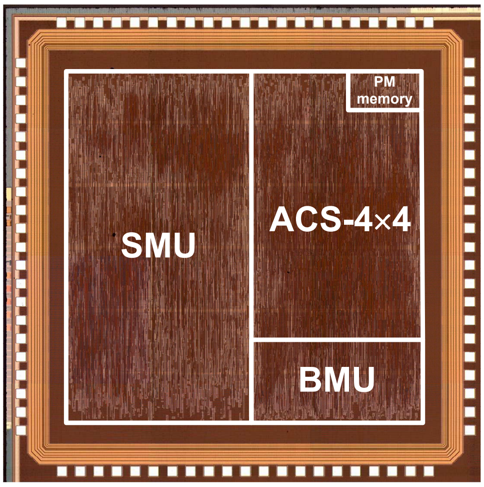


Figure 5.18: Microphoto of the 0.18-μm ACS-4 × 4 Viterbi decoder

is shown in Fig. 5.18. The chip summary is also listed in Table 5.18.

Table 5.18: The Viterbi decoder chip summary

Technology	0.18-μm 1P6M CMOS
Chip size	7.34mm^2
Core size	4.41mm^2
Gate count	310.4k
Supply voltage	1.62V $\sim$ 1.98V
Clock rate	100MHz
Power consumption	300mW at 1.8V (400Mb/s)

The ACS-$2^p \times 2^q$ structure and the retiming mechanism facilitate the Viterbi decoder implementation based on the high-radix trellis decoding. The chip area is reduced for less branch number, and the retiming techniques reduce the critical path delay of the ACS units. The results shows a significant area reduction for the designs with 2-D ACS unit and a considerable improvement in throughput with the retiming technique. The 0.18-μm chip design shows RT-3 can improve the throughput of ACS-4 × 4 by about 71%. In the 0.13-μm technology, both the ACS-16 and the ACS-4 × 4 decoders with retiming can accomplish the 1Gb/s data rate; however, the later results in only 37% area as compared to the former.

Among the well-know error-correcting codes, the LDPC code, which can reach a capacity

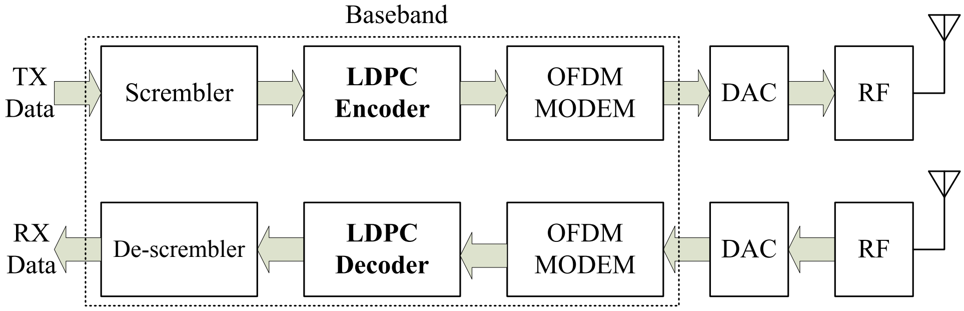
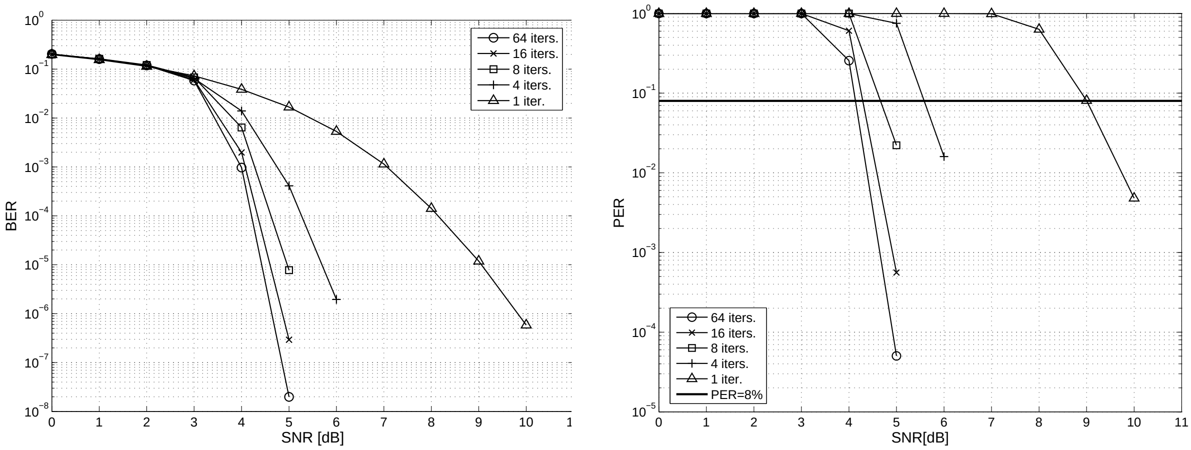


Figure 5.19: Block diagram of the proposed LDPC-COFDM UWB systems.

approaching performance by the iterative decoding algorithm [20, 25], undoubtedly engage the most research interest recently. For improving PHY-layer capacity, LDPC codes can increase the throughput to over 500Mb/s in future WLAN applications [201]. Therefore, we apply the (1200,720) LDPC code to the MB-OFDM UWB system [136] illustrated in Fig. 5.19, where the original convolutional codes and the bit interleaver have been replaced. For the relatively small block length, the irregular code is constructed by the progressive edge-growth (PEG) algorithm [174] to deliver better performance.

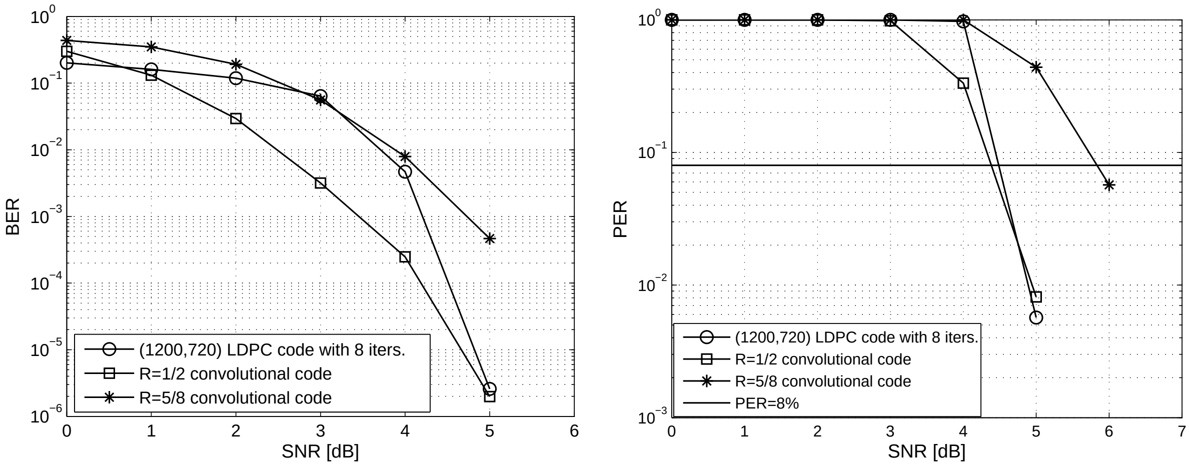


(a) BER performance

(b) PER performance

Figure 5.20: Performance of the (1200,720) LDPC code

Based on the referenced MB-OFDM system, the performance including the bit error rate (BER) and the packet error rate (PER) is shown in Fig. 5.20 with the AWGN channel model and 1024 bytes data in each packet. Assume the min-sum decoding algorithm, the system is required to achieve 8% PER specified in [136]. The required signal to noise ratio (SNR) is reduced as the iteration (iter.) number increases, but the improvement tends to be insignificant after 8 iterations. In Fig. 5.21, the performance is also compared to the convolutional code (5.4) [136] where two different rates $R = 1/2$ and $R = 5/8$ after puncturing the $R = 1/3$ mother code are selected as the references. It shows that the (1200,720) LDPC code can achieve the comparable performance to the punctured convolutional code with only 8 decoding iterations. The short block length and small decoding iterations will facilitate the implementation of high speed LDPC decoders.



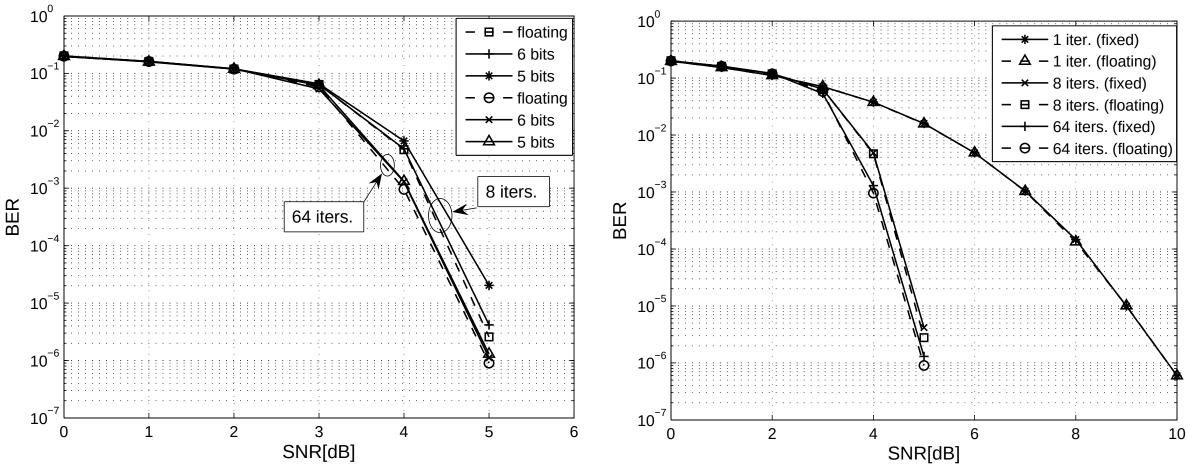
(a) BER

(b) PER

Figure 5.21: Comparison of different codes

Fig. 5.22 presents the bit error rate (BER) performance in AWGN channel. The comparison in Fig. 5.22(a) reveals the required quantization level of message in the decoder. As compared with theoretical curve, both 5 bits and 6 bits quantization schemes have less than 0.07dB loss of signal-to-noise ratios (SNR) after 64 decoding iterations (iters.). However, there is a 0.3dB loss with 8 decoding iterations while quantizing message into 5 bits. Therefore, six bits scheme is considerably the better choice for circuit implementation. Fig. 5.22(b) also shows the performance of the proposed LDPC decoder, applying 6 bits quantization in different decoding iterations.

The decoder architecture has been shown in Fig. 4.31. In order to reduce signal routing



(a) Different quantization level

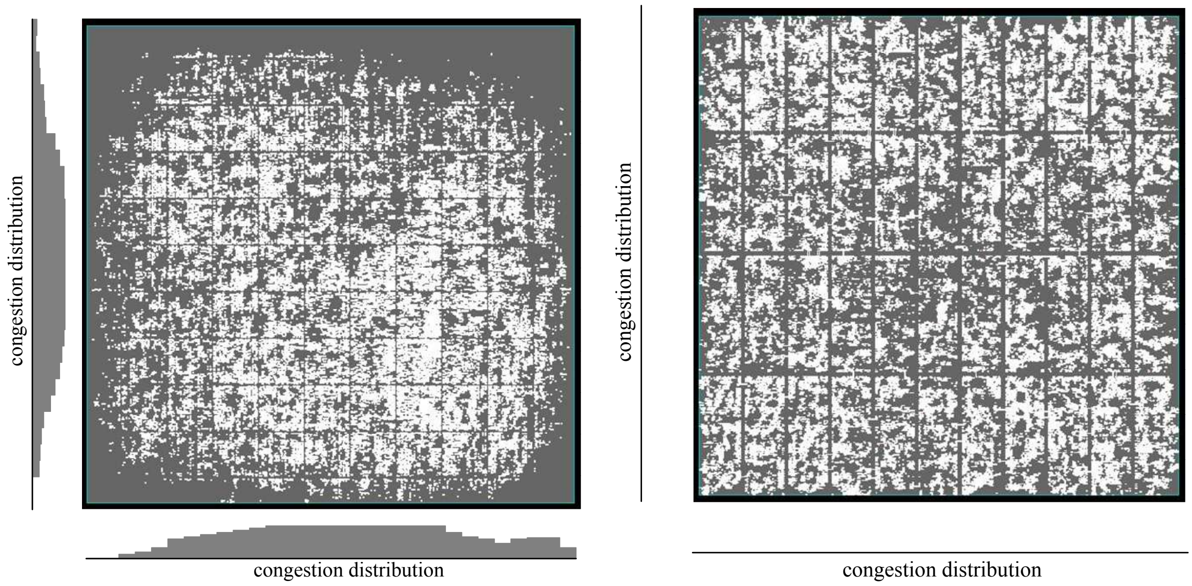
(b) Proposed decoder

Figure 5.22: The BER performance in the MB-OFDM UWB system

congestion, the RE5-based MMU and RS-based input buffer are adopted for circuit implementation. With the 0.18-μm cell library [135], the distributions of MMUs based on MUX and RE5 approaches are demonstrated in Fig. 5.23 where the chip size constrained to 25mm^2. In addition, the routing congestion profile is measured in both horizontal and vertical axes. The MUX-based MMU in Fig. 5.23(a) has serious signal routing difficulty almost within the whole chip, whereas the RE5-based one in Fig. 5.23(b) has not only uniform MMU distribution but also zero congestion overflow in both axes. In short, the 25mm^2 chip area is sufficient for the RE5-based approach, but is still inadequate for the MUX-based approach due to a large amount of signal wires.

A test chip has been fabricated in the 1.8V, 0.18-μm 1P6M CMOS technology. The chip size is 25mm^2 while the core occupies 21.23mm^2. The total gate count is 1.15M including two MMUs while the chip core density is about 71.2%. After static timing analysis (STA) and post-layout simulation, the decoder achieves 3.33Gb/s throughput with 8 decoding iterations under 1.62V supply and worst speed corner. The estimation also includes crosstalk analysis for signal wires that cause coupling noise.

A second test chip is implemented in a 1.2V, 0.13μm 1P8M CMOS technology. The maximum decoding speed has been improved to 5.92Gb/s with 8 decoding iterations. Moreover,



(a) MUX

(b) RE-5B

Figure 5.23: The distribution of MMUs and routing congestion distribution

the chip size becomes $13.5mm^2$ where the core area is $10.24mm^2$. The chip density grows to about 75.4% because of two more metal layers. Table 5.19 summarizes the LDPC decoder chips in this paper and makes a comparison with the fully parallel design in [65].

Table 5.19: Summary of the LDPC decoder chips

		Chip 1	Chip 2	[65]
Block length		1200	1200	1024
Technology		0.13-μm	0.18-μm	0.16-μm
Data rate	8 iters.	5.92Gb/s [1]	3.33Gb/s [2]	N.A.
	64 iters.	820Mb/s [1]	461Mb/s [2]	512Mb/s
Power consumption		268mW @1.2V	644mW @1.8V	690mW @1.5V
Gate count		1.15M	1.15M	1.75M
Chip size		13.5mm^2	25mm^2	52.5mm^2
Core size		10.4mm^2	21.16mm^2	N.A.
Chip density		75.4%	71.2%	50%

[1] 1.02V supply
[2] 1.62V supply

We present the high speed and area efficient LDPC decoder architecture. The message memories architecture permits parallel decoding of two codewords and diminishes the routing congestion issues. Additionally, the data rescheduling minimizes the signal routing between datapaths and memory units. Consequently, the chip becomes smaller due to the the increased chip density. After implementation in the 0.18-μm technology, the chip can achieve the 3.33Gb/s data rate with 8 decoding iterations. Furthermore, the 0.13-μm chip reaches the maximum 5.92Gb/s data rate with 13.5mm^2 area and 268mW power consumption.

Chapter 6

Conclusion

The research on the channel decoder design and implementation is reported in this dissertation. We investigate the Reed-Solomon code with algebraic decoding, the convolutional code with probabilistic decoding, and the iterative decoding based turbo code and LDPC code.

6.1 Summary

The universal Reed-Solomon decoder architecture is proposed with the Montgomery algorithm by which the multiplications over any $GF(2^m)$ are feasible for a predefined constant d and $1 \leq m \leq d$. Furthermore, the low complexity constant multiplier based on Montgomery algorithm is also introduced for the syndrome calculator and the Chien search for less circuit area. The key equation solver is a decomposed architecture using only three universal multipliers, and the finite field inversion in the error value evaluator can be either the Fermat's identity or the memory based look-up table, depending on the table size. An (n, k) error and erasure decoder for $n \leq 255$ and $t \leq 16$ is constructed and implemented with the 0.18-μm and the 0.13-μm processes. As compared with the convectional designs, the gate count increase due to the universal features is less than 50%. Additionally, this decoder can fully support most communication and storage applications.

We provide two design strategies for the Viterbi decoding algorithm. The low power Viterbi decoder with the path merge and the path prediction methods is optimized to achieve the dynamic truncation length according to the channel condition. As a result, most redundant memory access is avoided to reduce the power consumption in the survivor memory unit. The computational unit, ACS unit, is also transformed to the compare-select-add structure for less gate count as well as less signal switching power. The Viterbi decoder for theChien Ching Lin IEEE 802.11a system is implemented and measured to achieve 30%∼40% power reduction. The 2-D ACS structure with retiming techniques is also introduced to accommodate the requirement for high decoding throughput. The ACS-4 × 4 is designed to the parallelism equivalent to the ACS-16 architecture. The circuit implementation of ACS-4 × 4 based decoder is shown to accomplish the 500Mb/s data rate within the 4.41mm^2 0.18-μm silicon area. Applying the retiming approaches, the ACS-4 × 4 decoder can be improved to 731Mb/s and 1105Mb/s data rates in the 0.18-μm and the 0.13-μm CMOS technology. Note that the area of 1.96mm^2 is required to reach the over 1000Mb/s data throughput. Furthermore, the major feature of the ACS-4 × 4 decoder is the much smaller area, or gate count, as compared to the ACS-16 based one.

The soft iterative decoding for the turbo code and the LDPC code can deliver the performance approaching the Shannon limit. However, they have quite different decoding schemes; the turbo decoding is based on the BCJR algorithm over trellis diagram, and the LDPC decoding is based on the sum-product algorithm over the bipartite graph. We consider the 3GPP2 mobile communication for the turbo code that has the considerably large interleaver size. The cost and the power consumption will be the design constraints, especially the embedded memory for data interleaving. In the turbo decoder design, we prefer the simple SISO decoder architecture, the efficient memory hierarchy, and the embedded interleaver with the modest memory size. Furthermore, the Viterbi decoding is integrated without additional datapaths because of the trellis based decoding algorithm. The decoder after the 0.18-μm chip implementation shows the core size 7.29mm^2 and the power consumption 83mW while decoding a turbo coded data stream at the 3.1Mb/s data rate.

On the other hand, the LDPC code is applied to the high speed UWB data communication. The LDPC decoding on the bipartite graph facilitates the parallel decoder implementation due to the simple computations at the bit nodes and the check nodes after approximation. Hence the (1200,720) LDPC code with the non-structured parity check matrix is selected for implementation. Because of the random and the large number of connections in the bipartite graph, the signal routing becomes very difficult, and the decoding speed is also hard to be enhanced. More chip area is necessary for signal routing and leads to the low chip density, especially for those based on the architecture. We exploit the register exchange scheme for the message memory to eliminate multiplexers between datapaths and memory units. The interconnection becomes simple and direct in the partially-parallel decoder. Consequently, the chip implementations using the 0.18-μm with six metals and the 0.13-μm with eight metals respectively accomplish the 71.2% and the 75.4% chip density. The 0.18-μm chip has the core size 21.16mm^2 and can reach the 3.33Gb/s data rate. Furthermore, the 0.13-μm decoder can provide the 5.92Gb/s decoding speed within the 10.2mm^2 silicon area.

6.2 Future work

The decoding algorithm for Reed-Solomon codes in Chapter 2 is based on the hard decision symbols $GF(q)$ converted from the channel output signals, assuming the encoder outputs are also over $GF(q)$. Generally, the maximum likelihood (ML) decoding for Reed-Solomon is NP-hard [202]. The trellis construction [203,204] provides the ML decoding for the Reed-Solomon code [205], however, the complexity limits the code length to be ≤ 15. The non-algebraic soft decoding algorithm in [206–208] has the computational complexity exponentially growing with the code length. Koetter and Vardy introduced the algebraic soft-decision decoding algorithm [209] based on the list decoding algorithm [210] that can correct an error number larger than half the minimum distance for a code. Alternatively, the belief propagation algorithm can also be applied to the iterative Reed-Solomon decoding [211]. The combination of the algebraic soft-decision decoding and the belief propagation algorithms is also reported in [212]. The other soft decision decoding algorithms can also be found in [213–218]. As compared to the hard decision decoding, the soft decoding algorithms can provide up to 3dB coding gain with much complex decoder implementations [219–221]. Many research efforts are still necessary for the algorithms as well as the decoder designs in terms of complexity, performance, and power consumption.

The low power Viterbi decoder is still motivated by the future wireless applications. The scarce state transition (SST) scheme [41] can be applied to reduce the dynamic power of ACS units. The path merge approach is also applicable to the register exchange based SMU; moreover, the circuit techniques, power gating and voltage scaling [222], will lower the power dissipation due to data movement in the SMU.

The iterative decoding and the interleaver design dominate the decoding speed of turbo decoders. Therefore, multiple SISO decoders are deployed for parallel processing [61, 223], and the interleaver should be modified for multiple data access [224,225] while the error performance meets the system requirement. Furthermore, the SISO decoder can also have internal parallelism based on the high radix trellis or the two-dimensional trellis structures [57].

In some applications, the LDPC codes, based on the structured parity check matrices, has have large code length [226, 227]. The large parity check matrix constructed form many smaller sub-matrices [228–233] will facilitate the implementation of both encoders and decoders. Therefore, the matrix structure not only effect the decoder error performance, but also determines the decoder architecture. The co-deign of the code structure and the decoder would facilitate the implementation and improve the circuit performance. Moreover, for large LDPC codes, the embedded memory saving messages would occupy a large circuit area; therefore, a scheduling may be necessary to reduce the memory size.

In the probabilistic based decoding algorithm, the probabilities or messages should be quantized for digital signal processing and digital circuit implementation. However, the quantization introduces some error during decoding, and each value should be represented with multiple bits that leads to large chip area. Hence many researchers prefer the analog decoding techniques [234–241]. The floating-point quantities are represented with the analog signals without quantization; therefore, the decoding algorithm is more optimal, and the decoder requires less area as well as power dissipation. Nevertheless, the codes processed by analog decoders are relative small as compared with those in current applications. The analog decoders are still motivated by the increasing transmission speed and low power requirement.

References

[1] T. M. Cover and J. A. Thomas, *Elements of Information Theory.* New York: Wiley, 1991.

[2] R. W. Yeung, *A First Course in Information Theory.* New York: Kluwer Academic, 2006.

[3] C. E. Shannon, "A mathematical theory of communication," *Bell Syst. Tech. J.*, vol. 27, pp. 379–428(Part I), July 1948.

[4] ——, "A mathematical theory of communication," *Bell Syst. Tech. J.*, vol. 27, pp. 623–656(Part II), July 1948.

[5] R. E. Blahut, *Theory and practice of error control codes.* Reading: Addison-Wesley, 1983.

[6] R. W. Hamming, "Error detecting and error correcting codes," *Bell Syst. Tech. J.*, vol. 29, pp. 147–160, Apr. 1950.

[7] D. E. Muller, "Applications of boolean algrbra to switching circuits design and to error detection," *IRE Trans.*, vol. EC-3, pp. 6–12, Sept. 1954.

[8] I. S. Reed, "A class of multile-error-correcting codes and the decoding scheme," *IRE Trans.*, vol. IT-4, pp. 38–49, Sept. 1954.

[9] R. C. Bose and D. K. Ray-Chaudhuri, "On a class of error correcting binary group codes," *Inform. Control*, vol. 2, pp. 68–69, Mar. 1960.

[10] A. Hocquenghem, "Codes corecteurs d'erreurs," *Chiffres*, vol. 2, pp. 147–156, 1959.

[11] I. S. Reed and G. Solomon, "Polynomial codes over certain finite fields," *J. Soc. Ind. Appl. Math.*, vol. 8, pp. 300–304, June 1960.

[12] D. C. Gorenstein and N. Zierler, "A class of cyclic linear error-correcting codes in p^m symbols," *J. Soc. Indust. Appl. Math.*, vol. 9, pp. 207–214, June 1961.

[13] P. Elias, "Coding for noisy channels," *IRE Conv. Rec.*, vol. pt.4, pp. 37–47, 1955.

[14] J. M. Wozencraft and B. Reiffen, *Sequential decoding.* Cambridge: MIT Press and John Wiley, 1961.

[15] A. J. Viterbi, "Error bounds for convolutional codes and asymptotically optimum decoding algorithm," *IEEE Trans. Inform. Theory*, vol. IT-13, no. 2, pp. 260–269, Apr. 1967.

[16] L. R. Bahl, J. Cocke, F. Jelinek, and J. Raviv, "Optimal decoding of linear codes for minimizing symbol," *IEEE Trans. Inform. Theory*, no. IT-20, pp. 284–287, Mar. 1974.

[17] C. Berrou and A. Glavieux, "Near optimum error correcting coding and decoding: turbo-codes," *IEEE Trans. Commun.*, vol. 44, no. 10, pp. 1261–1271, Oct. 1996.

[18] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near Shannon limit error-correcting coding and decoding: turbo-codes," in *IEEE Int. Conf. Communications (ICC)*, May 1993, pp. 1064–1070.

[19] J. Hagenauer and P. Hoeher, "A Viterbi algorithm with soft-decision outputs and its applications," in *IEEE GLOBECOM*, Dallas, TX, Nov. 1989, pp. 47.1.1–47.1.7.

[20] R. G. Gallager, *Low-Density Parity-Check Codes.* MA: MIT Press, 1963.

[21] ——, "Low-density parity-check codes," *IEEE Trans. Inform. Theory*, vol. 8, pp. 21–28, Jan. 1962.

[22] R. M. Tanner, "A recursive approach to low complexity codes," *IEEE Trans. Inform. Theory*, vol. IT-27, no. 5, pp. 399–431, Sept. 1981.

[23] D. J. C. MacKay and R. M. Neal, "Near Shannon limit performance of low density parity check codes," *Electron. Lett.*, vol. 33, no. 6, pp. 457–458, Mar. 1997.

[24] D. J. C. MacKay, "Good error-correcting codes based on very sparse matrices," *IEEE Trans. Inform. Theory*, vol. 45, no. 2, pp. 399–431, Mar. 1999.

[25] S. Y. Chung, G. D. Forney, Jr., T. J. Richardson, and R. Urbanke, "On the design of low-density parity-check codes within 0.0045 db of the Shannon limit," *IEEE Commun. Lett.*, vol. 5, no. 2, pp. 58–60, Feb. 2001.

[26] J. Pearl, *Probabilistic Reasoning in intelligent systems: networks of plausible inference.* San Mateo: Morgan Kaufmann, 1988.

[27] R. J. McEliece, D. J. C. MacKay, and J. F. Cheng, "Turbo decoding as a instanec of Pearl's blief propagation algorithm," *IEEE J. Select. Areas Commun.*, vol. 16, no. 2, pp. 140–152, Feb. 1998.

[28] F. R. Kschischang and B. J. Frey, "Iterative decoding of compound codes by probability propagation in graphical models," *IEEE J. Select. Areas Commun.*, vol. 16, no. 2, pp. 219–230, Feb. 1998.

[29] L. Song, M. L. Yu, and M. S. Shaffer, "A 10 Gb/s and 40 Gb/s forward-error-correction device for optical communications," *IEEE J. Solid-State Circuits*, vol. 37, pp. 1565–1573, Nov. 2002.

[30] K. Seki *et al.*, “Single-chip 10.7 gb/s FEC CODEC LSI using timemultiplexed RS decoder,” in *Proc. IEEE Custom Integrated Circuits Conf. (CICC)*, 2001, pp. 289–292.

[31] H. C. Chang, C. C. Chung, C. C. Lin, and C. Y. Lee, “A high speed Reed-Solomon decoder chip using inversionless decomposed architecture for Euclidean algorithm,” in *Proc. 28th Eur. Solid State Circuits Conf. (ESSCIRC)*, Sept. 2002, pp. 24–26.

[32] I. Reed, M. Shih, and T. K. Truong, “VLSI design of inverse-free Berlekamp-Massey algorithm,” *Proc. Inst. Elect. Eng. pt. E*, vol. 138, pp. 295–298, Sept. 1991.

[33] H. C. Chang, C. B. Shung, and C. Y. Lee, “A Reed-Solomon product-code (RS-PC) decoder chip for DVD applications,” *IEEE J. Solid-State Circuits*, vol. 36, pp. 229–237, Feb. 2001.

[34] F. K. Chang, W. C. Hsu, C. C. Lin, and H. C. Chang, “Design and implementation of a reconfigurable architecture for (528, 518) Reed-Solomon codec IP,” in *IEEE NEWCAS Conf.*, June 2005, pp. 87–90.

[35] H. C. Chang, C. C. Lin, and C. Y. Lee, “A low-power Reed-Solomon decoder for STM-16 optical communications,” in *IEEE Asia-Pacific Conf. ASIC (APASIC)*, 2002.

[36] J. C. Huang, C. M. Wu, M. D. Shieh, and C. H. Wu, “An area-efficient versatile Reed-Solomon decoder for ADSL,” in *IEEE Int. Symop. Circuits and Systems (ISCAS)*, June 1999, pp. 517–520.

[37] L. .Song, K. K. Parhi, I. Kuroda, and T. Nishitani, “Hardware/software codesign of finite field datapath for low energy Reed-Solomon codecs,” *IEEE Trans. VLSi Syst.*, vol. 8, pp. 160–172, Apr. 2000.

[38] F. K. Chang, C. C. Lin, H. C. Chang, and C. Y. Lee, “Universal architectures for Reed-Solomon error-and-erasure decoder,” in *Proc. IEEE Asia Solid State Circuits Conf. (ASSCC)*, Nov. 2005.

[39] M. Kawokgy and C. A. T. Salama, “Low-power asynchronous Viterbi decoder for wireless applications,” in *Int. Symop. Low Power Electronics and Design (ISLPED)*, 2004, pp. 286–289.

[40] P. A. Riocreux, L. E. M. Brackenbury, M. Cumpstey, and S. B. Furber, “A low-power self-timed Viterbi decoder,” in *Int. Symop. Asynchronous Circuits and Systems*, 2001, pp. 15–24.

[41] K. Seki, S. Kubota, M. Mizoguchi, and S. Kato, “Very low power consumption Viterbi decoder LSIC empolying the SST (scarce state transition) scheme for multimedia mobile communications,” *Electron. Lett.*, vol. 30, no. 8, pp. 637–639, Apr. 1994.

[42] 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 Eur. Solid State Circuits Conf. (ESSCIRC)*, Sept. 2003, pp. 723–726.

[43] C. Tsui, R. S.-K. Cheng, and C. Ling, “Low power ACS unit design for the Viterbi decoder,” in *IEEE Int. Sympo. Circuits and Systems (ISCAS)*, vol. 1, May 1999, pp. 137–140.

[44] D. A. El-Dib and M. I. Elmasry, “Modified register-exchange Viterbi decoder for low-power wireless communications,” *IEEE Trans. Circuit and Syst. I*, vol. 51, no. 2, pp. 371–378, Feb. 2004.

[45] R. Henning and C. Chakrabarti, “An approach for adaptively approximating the Viterbi algorithm to reduce power consumption while decoding convolutional codes,” *IEEE Trans. Signal Processing*, vol. 52, no. 5, pp. 1443–1451, May 2005.

[46] S. J. Simmons, “Breadth-first trellis decoding with adaptive effort,” *IEEE Trans. Commun.*, vol. 38, pp. 3–12, Jan. 1990.

[47] C. C. Lin, Y. H. Shih, H. C. Chang, and C. Y. Lee, “Design of a power-reduction Viterbi decoder for WLAN applications,” *IEEE Trans. Circuit and Syst. I*, vol. 52, no. 6, pp. 1148–1156, June 2005.

[48] J. Tang and K. K. Parhi, “Viterbi decoder for high-speed ultra-wideband communication systems,” in *IEEE Int. Conf. Acoustics, Speech, and Signal Processing*, 2005, pp. 37–40.

[49] 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.

[50] S. W. Choi and S. S. Choi, “200Mbps Viterbi decoder for UWB,” in *Int. Conf. Advanced Commun. Tech.*, vol. 2, 2005, pp. 904–907.

[51] A. K. Yeung and J. M. Rabaey, “A 210Mb/s radix-4 bit-level pipelined Viterbi decoder,” in *IEEE Int. Solid-State Circuit Conf. (ISSCC) Dig. Tech. Papers*, Feb. 1995, pp. 88–89.

[52] V. S. Gierenz, O. Weiss, T. G. Noll, I. Carew, J. Ashley, and R. Karabed, “A 550 Mb/s radix-4 bit-level pipelined 16-state 0.25-μm CMOS Viterbi decoder,” in *Int. Conf. Application-Specific Syst., Architectures, and Processors*, 2000, pp. 195–201.

[53] N. Bruels, E. Sicheneder, M. Loew, J. Gliese, and C. Sauer, “A 2.8 Gb/s, 32-state, radix-4 Viterbi decoder add-compare-select unit,” in *Symop. VLSI Circuits Dig. Tech. Papers*, 2004, pp. 170–173.

[54] P. J. Black and T. H. Meng, “A 140-Mb/s, 32-state, radix-4, Viterbi decoder,” *IEEE J. Solid-State Circuits*, vol. 27, no. 12, pp. 1877–1885, Dec. 1992.

[55] 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.

[56] P. J. Black and T. H. Meng, "A 1 Gb/s, four-state, sliding block Viterbi decoder," *IEEE J. Solid-State Circuits*, vol. 32, no. 6, pp. 797–805, June 1997.

[57] C. L. Chen, C. C. Lin, H. C. Chang, and C. Y. Lee, "High-speed Viterbi decoder based on the two-dimensional ACS structure," submitted to IEEE Trans. Circuit and Syst. I.

[58] M. A. Bickerstaff, D. Garrett, T. Prokop, C. Thomas, B. Widdup, G. Zhou, L. M. Davis, G. Woodward, C. Nicol, and R. H. Yan, "A unified turbo/Viterbi channel decoder for 3GPP mobile wireless in 0.18um CMOS," *IEEE J. Solid-State Circuits*, vol. 37, no. 11, pp. 1555–1564, Nov. 2002.

[59] *Technical Specification Group Radio Access Network; Multiplexing and channel coding (FDD)*, 3GPP Std. TS 25.212, Rev. 6.4.0, 2005.

[60] 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. (ISSCC) Dig. Tech. Papers*, 2003, pp. 151–484.

[61] P. Urard *et al.*, "A generic 350Mb/s turbo-codec based on a 16-states SISO decoder," in *IEEE Int. Solid-State Circuit Conf. (ISSCC) Dig. Tech. Papers*, 2004, pp. 424–536.

[62] Z. Wang, Z. Chi, and K. K. Parhi, "Area-efficient high speed decoding schemes for turbo decoders," *IEEE Trans. VLSI Syst.*, vol. 10, no. 6, pp. 902–912, Dec. 2002.

[63] *Physical Layer Standard for cdma2000 Spread Spectrum Systems*, 3GPP2 Std. C.S0002-C, Rev. 1.0, 2002.

[64] C. C. Lin, Y. H. Shih, H. C. Chang, and C. Y. Lee, "A dual mode channel decoder for 3GPP2 mobile wireless communications," in *Proc. 30th Eur. Solid State Circuits Conf. (ESSCIRC)*, Sept. 2004, pp. 483–486.

[65] A. Blanksby and C. Howland, "A 690-mw 1-Gb/s 1024-b, rate-1/2 low-density parity-check code decoder," *IEEE J. Solid-State Circuits*, vol. 37, no. 3, pp. 404–412, Mar. 2002.

[66] C. C. Lin, K. L. Lin, H. C. Chang, and C. Y. Lee, "A 3.33Gb/s (1200,720) low-density parity check code decoder," in *Proc. 31st Eur. Solid State Circuits Conf. (ESSCIRC)*, Sept. 2005, pp. 211–214.

[67] R. J. McEliece, *Finite field for computer scientists and engineers.* Boston: Kluwer Academic, 1987.

[68] F. J. MacWilliams and N. J. A. Sloane, *The theory of error-correcting codes.* Amsterdam: North-Holland, 1977.

[69] R. J. McEliece, *The theory of information and coding.* Cambridge, UK: Cambridge University Press, 2004.

[70] W. W. Peterson, “Encoding and error-correction procedures for the Bose-Chaudhuri codes,” *IRE Trans. Inform. Theory*, vol. IT-6, pp. 459–470, Sept. 1960.

[71] R. Chien, “Cyclic decoding procedure for the Bose-Chaudhuri-Hocquenghem codes,” *IEEE Trans. Inform. Theory*, vol. IT-10, pp. 357–363, Oct. 1964.

[72] E. R. Berlekamp, *Algrbraic coding theory.* New York: McGraw-Hill, 1968.

[73] G. D. Forney, Jr., “On decoding BCH codes,” *IEEE Trans. Inform. Theory*, vol. IT-11, pp. 549–557, Oct. 1965.

[74] E. Berlekamp, “On decoding binary Bose-Chaudhuri-Hocquenghem codes,” *IEEE Trans. Inform. Theory*, vol. IT-11, pp. 577–579, Oct. 1965.

[75] J. Massey, “Step-by-step decoding of the Bose-Chaudhuri-Hocquenghem codes,” *IEEE Trans. Inform. Theory*, vol. IT-11, pp. 580–585, Oct. 1965.

[76] ——, “Shift-register synthesis and bch decoding,” *IEEE Trans. Inform. Theory*, vol. IT-15, pp. 122–127, Jan. 1969.

[77] H. C. Chang, “Research on Reed-Solomon decoder-design and implementation,” Ph.D. dissertation, National Chiao Tung Univ., Hsinchu, Taiwan, 2002.

[78] Y. Sugiyama, M. Kasahara, S. Hirasawa, and T. Namekawa, “A method for solving key equation for decoding Goppa codes,” *Inform. Contr.*, vol. 27, pp. 87–99, Jan. 1975.

[79] L. Welch and R. Scholtz, “Continued fractions and Berlekamp’s algorithm,” *IEEE Trans. Inform. Theory*, vol. IT-25, pp. 19–27, Jan. 1979.

[80] W. C. Gore, “Transmitting binary symbols with Reed-Solomon codes,” in *Proc. Conf. Inform. Sci. and Syst.*, Princeton, NJ, 1973, pp. 495–497.

[81] S. Choomchuay and B. Arambepola, “Time domain algorithms and architectures for Reed-Solomon decoding,” *IEE Proc.-I in Commun., Speech, and Vision*, vol. 140, pp. 189–196, June 1993.

[82] J. M. Hsu and C. L. Wang, “An area-efficient pipelined VLSI architecture for decoding of Reed-Solomon codes based on time-domain algorithm,” *IEEE Trans. Circuits Syst. Video Technol.*, vol. 7, pp. 864–871, Dec. 1997.

[83] C. Seki *et al.*, “Single-chip FEC codec using a concatenated BCH code for 10 Gb/s long-haul optical transmission systems,” in *Proc. IEEE Custom Integrated Circuits Conf. (CICC)*, 2003, pp. 21–24.

[84] H. Burton, “Inversionless decoding of binary BCH codes,,” *IEEE Trans. Inform. Theory*, vol. IT-17, pp. 464–466, July 1971.

[85] T. K. Truong, J. H. Jeng, and K. C. Hung, “Inversionless decoding of both errors and erasures of Reed-Solomon code,” *IEEE Trans. Commun.*, vol. 46, pp. 973–976, Aug. 1998.

[86] C. C. Lin, F. K. Chang, H. C. Chang, and C. Y. Lee, "A universal VLSI architecture for bit-parallel computation in $GF(2^m)$," in *Proc. IEEE Asia-Pacific Conf. Circuits and Syst.*, Dec. 2004, pp. 6–9.

[87] P. L. Montgomery, "Modular multiplication without trial division," *Mathematics of Computation*, vol. 44, no. 170, pp. 519–521, Apr. 1985.

[88] C. Wang, T. K. Truong, H. Shao, L. J. Deutsch, J. K. Omura, and I. Reed, "VLSI architectures for computing multiplications and inverses in $GF(2^m)$," *IEEE Trans. Comput.*, vol. C-34, pp. 709–717, Aug. 1985.

[89] J. H. Jeng and T. K. Truong, "On decoding of both errors and erasures of a Reed-Solomon code using an incersion-free berlekamp-massey algorithm," *IEEE Trans. Commun.*, vol. 47, no. 10, pp. 1488–1494, Oct. 1999.

[90] J. L. Massey, *Threshold decoding.* MA: MIT Press, 1963.

[91] R. M. Fano, "A heruistic discussion of probabilistic decoding," *IEEE Trans. Inform. Theory*, vol. 9, pp. 64–74, Apr. 1963.

[92] F. Jelinek, "Fast sequential decoding algorithm using a stake," *IBM J. Res. and Dev.*, vol. 13, pp. 675–685, Nov. 1969.

[93] K. S. Zigangirov, "Some sequential decoding procedures," *Probl. Peredach. Inform.*, vol. 2, pp. 13–25, 1966.

[94] A. Dholakia, *Introduction to convolutional codes with applications.* Boston: Kluwer Academic, 1994.

[95] P. Piret, *Convolutional codes: an algebraic approach.* MA: MIT Press, 1988.

[96] R. Johannesson and Z. X. Wan, "A linear algebra approach to minimum convolutional encoders," *IEEE Trans. Inform. Theory*, vol. 39, no. 4, pp. 37–47, July 1993.

[97] C. B. Schlegel and L. C. Pérez, *Trellis and turbo coding.* NJ: IEEE Press, 2004.

[98] G. D. Forney, Jr., "Convolutional codes I: algebraic structure," *IEEE Trans. Inform. Theory*, vol. IT-16, pp. 720–738, Nov. 1970.

[99] ——, "Structural analyses of convolutional codes via dual codes," *IEEE Trans. Inform. Theory*, vol. IT-19, pp. 512–518, Jul. 1973.

[100] A. J. Viterbi and J. K. Omura, *Principles of Digital Communication and Coding.* New York: McGraw-Hill, 1979.

[101] D. J. Costello, Jr., "Construction of convolutional codes for sequential decoding," Ph.D. dissertation, Univ. Notre Dame, Norte Dame, IN, 1969.

[102] J. L. Massey and M. K. Sain, "Inverses of linear sequential circuits," *IEEE Trans. Computers*, vol. C-17, pp. 330–337, Apr. 1968.

[103] S. Benedetto and G. Montorsi, "Unveiling turbo-codes: some results on parallel concatenated coding schemes," *IEEE Trans. Inform. Theory*, vol. 42, no. 2, pp. 409–428, Mar. 1996.

[104] G. D. Forney, Jr., "The Viterbi algorithm," *Proc. IEEE*, vol. 61, no. 3, pp. 268–278, Mar. 1973.

[105] S. Lin and D. J. Costello, Jr., *Error control coding: fundamentals and applications*, 2nd ed. NJ: Pearson Education, 2004.

[106] A. J. Viterbi, "Convolutional codes and their performance in commumication systems," *IEEE Trans. Commun. Tech.*, vol. COM-19, no. 5, pp. 751–772, Oct. 1971.

[107] I. S. Gradshteyn and I. M. Ryzhik, *Table of integrals, series, and products*, 5th ed. San Diego, CA: Academic Press, 1994.

[108] J. M. Wozencraft and I. M. Jacobs, *Principles of Communication Engineering.* New York: John Wiley, 1965.

[109] M. Chiani, D. Dardari, and M. K. Simon, "New exponential bounds and approximations for the computation of error probability in fading channels," *IEEE Trans. Wireless Commun.*, vol. 2, pp. 840–845, Jul. 2003.

[110] T. K. Blankenship and B. Classon, "Improved exponential bounds and approximation for the q-function with application to average error probability computation," in *IEEE GLOBECOM*, Taipei, Taiwan, Nov. 2002, pp. 1399–1402.

[111] N. Ermolova and S. G. Haggman, "Simplified bounds for the complementary error function; application to the performance evaluation of signal processing systems," in *Eur. Signal Processing Conf.(EUSIPCO)*, Vienna, Austria, Sept.. 2004, pp. 1087–1090.

[112] R. G. Gallager, "A simple derivation of the coding theorem and some applications," *IEEE Trans. Inform. Theory*, vol. 11, pp. 3–18, Jan. 1965.

[113] ——, *Information Theory and Reliable Communication.* New York: Wiley, 1968.

[114] J. G. D. Forney, "Convolutional codes II: Maximum-likelihood decoding," *Information and Control*, vol. 25, pp. 222–266, July 1974.

[115] F. Hemmati and D. J. Costello, Jr, "Truncation error probability in Viterbi decoding," *IEEE Trans. Commun.*, vol. 25, pp. 530–532, May 1977.

[116] R. J. McEliece and I. M. Onyszchuk, "Truncation effects in Viterbi decoding," in *Military Commun. Conf.*, vol. 2, Boston, MA, 1989, pp. 541–545.

[117] J. B. Cain, G. C. Clark, and J. M. Geist, "Punctured convolutional codes of rate $(n-1)/n$ and simplified maximum likelihood decoding," *IEEE Trans. Inform. Theory*, vol. IT-25, no. 1, pp. 97–100, Jan. 1979.

[118] Y. Yasuda, K. Kashiki, and Y. Hirata, “High-rate punctured convolutional codes for soft decision Viterbi decoding,” *IEEE Trans. Commun.*, vol. COM-32, no. 3, pp. 315–319, Mar. 1984.

[119] J. K. Hole, “Punctured convolutional codes for the 1-D partial-response channel,” *IEEE Trans. Inform. Theory*, vol. 37, no. 3, pp. 808–817, May 1991.

[120] ——, “Rate $k/(k+1)$ punctured convolutional encoders,” *IEEE Trans. Inform. Theory*, vol. 37, no. 3, pp. 653–655, May 1991.

[121] ——, “An algorithm for determining if a rate $(n-1)/n$ punctured convolutional encoder is catastrophic,” *IEEE Trans. Commun.*, vol. 39, no. 3, pp. 386–389, Mar. 1991.

[122] G. Fettweis and H. Meyr, “A 100MBit/s Viterbi decoder chip: Novel architecture and its relization,” in *IEEE Int. Conf. Communications (ICC)*, vol. 2, Aug. 1990, pp. 463–467.

[123] C. M. Rader, “Memory management in a Viterbi decoder,” *IEEE Trans. Commun.*, vol. 29, pp. 1399–1401, Sept. 1981.

[124] R. Cypher and C. B. Shung, “Generalized trace back techniques for survivor memory management in the Viterbi algorithm,” in *IEEE GLOBECOM*, vol. 2, San Diego, CA, Dec. 1990, pp. 1318–1322.

[125] G. Feygin and P. G. Gulak, “Survivor sequence memory management in Viterbi decoders,” in *IEEE Int. Symop. Circuits and Systems (ISCAS)*, vol. 5, June 1991, pp. 2967–2970.

[126] G. Feygin and P. Gulak, “Architectural tradeoffs for survivor sequence memory management in Viterbi decoders,” *IEEE Trans. Commun.*, vol. 41, no. 3, pp. 425–429, Mar. 1993.

[127] J. A. Heller and I. M. Jacobs, “Viterbi decoding for satellite and space communication,” *IEEE Trans. Commun. Tech.*, vol. COM-19, no. 5, pp. 835–848, Oct. 1971.

[128] I. M. Onyszchuk, K. M. Cheung, and O. Collins, “Quantization loss in convolutional decoding,” *IEEE Trans. Commun.*, vol. 41, no. 2, pp. 261–265, Feb. 1993.

[129] *Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specification*, IEEE Std. 802.11a, 1999.

[130] A. P. Hekstra, “An alternative to metric rescaling in viterbi decoders,” *IEEE Trans. Commun.*, vol. 37, no. 11, pp. 1220–1222, Nov. 1989.

[131] C. Shung, P. Siegel, G. Ungerboeck, and H. Thapar, “VLSI architertures for metric normalization in the Viterbi algorithm,” in *Int. Conf. Communications*, vol. 4, Atlanta, CA, Apr. 1990, pp. 1723–1728.

[132] S. Sridharan and L. R. Carley, “A 110 MHz 350 mW 0.6 μm cmos 16-state generalized-target viterbi detector for disk drive read channels,” *IEEE J. Solid-State Circuits*, vol. 35, no. 3, pp. 362–370, Mar. 2000.

[133] G. Fettweis, R. Karabed, P. H. Siegel, and H. K. Thapar, “Reduced-complexity viterbi detector architectures for partial response signalling,” in *IEEE GLOBECOM*, vol. 1, Singapore, Nov. 1995, pp. 559–563.

[134] K. Page and P. M. Chau, “Improved architectures for the add-compare-select operation in long constraint length viterbi decoding,” *IEEE J. Solid-State Circuits*, vol. 33, no. 1, pp. 151–155, Jan. 1998.

[135] *UMC 0.18μm Process 1.8-Volt SAGE-X^{TM} Standard Cell Library Databook*, Artisan Components, Inc., Aug. 2000.

[136] A. Batra *et al.*, “Multi-band OFDM physical layer proposal for IEEE 802.15 task group 3a,” submitted to IEEE P802.15 working group for WPANs, Sept. 2004.

[137] R. Fisher *et al.*, “DS-UWB physical layer submission to 802.15 task group 3a,” submitted to IEEE P802.15 working group for WPANs, Mar. 2004.

[138] 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.

[139] ——, “A new architecture for the fast Viterbi algorithm,” in *IEEE GLOBECOM*, vol. 3, San Francisco, CA, 2000, pp. 1664–1668.

[140] T. Conway, “Implementation of high speed Viterbi detectors,” *Electron. Lett.*, vol. 35, no. 24, pp. 2089–2090, Nov. 1999.

[141] 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.

[142] P. Robertson, E. Villebrun, and P. Honher, “A comparison of optimal and suboptimal map decoding algorithms operating in the log domain,” in *IEEE Int. Conf. Communications*, June 1995, pp. 1009–1013.

[143] B. Vucetic and J. Yuan, *Turbo codes, principles and applications.* Boston: Kluwer Academic, 2000.

[144] J. Hagenauer, E. Offer, and L. Papke, “Iterative decoding of binary block and convolutional codes,” *IEEE Trans. Inform. Theory*, vol. 42, no. 2, pp. 429–445, Mar. 1996.

[145] S. A. Barbulescu, “Iterative decoding of turbo codes and other concatenated codes,” Ph.D. dissertation, Univ. South Australia, 1996.

[146] A. J. Viterbi, “A intuitive justification and a simplified implementation of the map decoder for convolutional codes,” *IEEE J. Select. Areas Commun.*, vol. 16, no. 2, pp. 260–264, Feb. 1998.

[147] J. L. Fan, *Constrained coding and soft iterative decoding.* Netherlands: Kluwer Academic, 2001.

[148] G. D. Forney, Jr., “Codes on graphs: Normal realizations,” *IEEE Trans. Inform. Theory*, vol. 47, no. 2, pp. 520–548, Feb. 2001.

[149] F. R. Kschischang, B. J. Frey, and H. A. Loeliger, “Factor graphs and the sum-product algorithm,” *IEEE Trans. Inform. Theory*, vol. 47, no. 2, pp. 498–519, Feb. 2001.

[150] D. B. West, *Introduction to graph theory*, 2nd ed. NJ: Prentice-Hall, 2001.

[151] *Digital Video Broadcasting (DVB); Interaction channel for satellite distribution systems*, ETSI Std. EN 301 790, Rev. 1.3.1, 2003.

[152] *CCSDS recommandation for telemetry channel coding*, CCSDS Std. 101.0-B-6, Oct. 2002.

[153] S. Benedetto and G. Montorsi, “Design of paralle concatenated convolutional coddes,” *IEEE Trans. Commun.*, vol. 44, no. 5, pp. 591–600, May 1996.

[154] O. Y. Takeshita, O. M. Collins, P. C. Massey, and D. J. Costello, Jr., “A note on asymmetric turbo-codes,” *IEEE Commun. Lett.*, vol. 3, no. 3, pp. 69–71, Mar. 1999.

[155] ——, “Asymmetric turbo-codes,” in *Proc. IEEE Int. Sympo. Inform. Theory*, Aug. 1998, p. 179.

[156] N. Wiberg, “Codes and decoding on general graphs,” Ph.D. dissertation, Univ. Linkoping, Sweden, 1996.

[157] S. Benedetto, G. Montorsi, and D. Divsalar, “Concatenated convloutional codes with interleavers,” *IEEE Commun. Mag.*, vol. 41, no. 8, pp. 102–109, Aug. 2003.

[158] L. C. Perez, J. Seghers, and D. J. Costello, “A distance spectrum interpretation of turbo codes,” *IEEE Trans. Inform. Theory*, vol. 42, no. 6, pp. 1698–1709, Nov. 1996.

[159] H. R. Sadjadpour, N. J. A. Sloane, M. Salehi, and G. Nebe, “Interleaver design for turbo codes,” *IEEE J. Select. Areas Commun.*, vol. 19, no. 5, pp. 831–837, May 2001.

[160] D. Divsalar and F. Pollara, “Turbo codes for pcs applications,” in *IEEE Int. Conf. Communications (ICC)*, June 1995, pp. 54–59.

[161] S. Dolinar and D. Divsalar, “Weight distributions for turbo codes using random and nonrandom permutations,” TDA Progress Report, Jet propulsion Lab., pp. 42–122, Aug. 1995.

[162] J. Yuan, B. Vucetic, and W. Feng, “Combined turbo codes and interleaver design,” *IEEE Trans. Commun.*, vol. 47, no. 4, pp. 484–487, Apr. 1999.

[163] J. Hokfelt, O. Edfors, and T. Maseng, “A turbo code interleaver design criterion based on the performance of iterative decoding,” *IEEE Commun. Lett.*, vol. 5, no. 2, pp. 484–487, Feb. 2001.

[164] J. Yu, M. L.Boucheret, R. Vallet, A. Duverdier, and G. Mesnager, “Interleaver design for turbo codes from convergence analysis,” *IEEE Trans. Commun.*, vol. 54, no. 4, pp. 619–624, Apr. 2006.

[165] 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.

[166] A. Worm, P. Hoeher, and N. Wehn, “Turbo-decoding without snr estimation,” *IEEE Commun. Lett.*, vol. 4, no. 6, pp. 193–195, June 2000.

[167] J. Vogt and A. Finger, “Improving the max-log-MAP turbo decoder,” *Electron. Lett.*, vol. 36, no. 23, pp. 1937–1939, Nov. 2000.

[168] P. H. Y. Wu and S. M. Pisuk, “Implementation of a low complexity, low power, integer-based turbo decoder,” in *IEEE GLOBECOM*, vol. 2, San Antonio, TX, Nov. 2001, pp. 946–951.

[169] 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.

[170] T. K. Blankenship and B. Classon, “Fixed-point performance of low-complexity turbo decoding algorithms,” in *IEEE Vehic. Tech. Conf.*, vol. 2, May 2001, pp. 1483–1487.

[171] C. C. Lin, Y. H. Shih, H. C. Chang, and C. Y. Lee, “A low power turbo/Viterbi decoder for 3GPP2 applications,” *IEEE Trans. VLSI Syst.*, vol. 14, no. 4, pp. 426–430, Apr. 2006.

[172] T. J. Richardson, M. A. Shokrollahi, and R. L. Urbank, “Design of capacity-approaching irregular low-density parity-check codes,” *IEEE Trans. Inform. Theory*, vol. 47, no. 2, pp. 619–637, Feb. 2001.

[173] M. G. Luby, M. Mitzenmacher, M. A. Shokollahi, and D. A. Spielman, “Improved low-density parity-check codes using irregular graphs,” *IEEE Trans. Inform. Theory*, vol. 47, no. 2, pp. 585–598, Feb. 2001.

[174] X. Y. Hu, E. Eleftheriou, and D. M. Arnoldx, “Regular and irregular progressive edge-growth tanner graphs,” *IEEE Trans. Inform. Theory*, vol. 51, no. 1, pp. 386–398, Jan. 2005.

[175] X. Y. Hu, E. Eleftheriou, and D. M. Arnold, “Irregular progressive edge-growth (PEG) Tanner graphs,” in *IEEE Int. Sympo. Inform. Theory*, 2002, p. 480.

[176] ——, “Progressive edge-growth Tanner graphs,” in *IEEE GLOBECOM*, vol. 2, 2001, pp. 995–1001.

[177] H. Xiao and A. H. Banihashemi, “Improved progressive-edge-growth (PEG) construction of irregular LDPC codes,” *IEEE Commun. Lett.*, vol. 8, no. 12, pp. 715–717, Dec. 2004.

[178] ——, “Improved progressive-edge-growth (PEG) construction of irregular LDPC codes,” in *IEEE GLOBECOM*, vol. 1, 2004, pp. 489–492.

[179] X. Y. Hu, E. Eleftheriou, D. M. Arnold, and A. Dholakia, “Efficient implementations of the sum-product algorithm for decoding ldpc codes,” in *IEEE GLOBECOM*, vol. 2, San Antonio, TX, 2001, pp. 1036–1036E.

[180] J. Chen and M. P. C. Fossorier, “Near optimum universal belief propagation based decoding of low-density parity check codes,” *IEEE Trans. Commun.*, vol. 50, no. 3, pp. 406–414, Mar. 2002.

[181] J. Heo and K. M. Chugg, “Optimization of scaling soft information in iterative decoding via density evolution methods,” *IEEE Trans. Commun.*, vol. 53, no. 6, pp. 957–961, June 2005.

[182] J. Heo, “Analysis of scaling soft information on low density parity check code,” *Electron. Lett.*, vol. 39, no. 2, pp. 219–221, Jan. 2002.

[183] Y. C. Liao, C. C. Lin, C. W. Liu, and H. C. Chang, “A dynamic normalization technique for decoding LDPC codes,” in *IEEE Workshop Signal Processing Syst.*, Nov. 2005, pp. 768–772.

[184] E. Yeo, P. Pakzad, B. Nikolic, and V. Anantharam, “VLSI architectures for iterative decoders in magnetic recording channels,” *IEEE Trans. Magnetics*, vol. 37, no. 2, pp. 748–755, Mar. 2001.

[185] *Digital Video Broadcasting (DVB); Framing structure, channel coding and modulation for digital terrestrial television*, ETSI Std. EN 300 744, Rev. 1.1.2, 1998.

[186] *Digital Video Broadcasting (DVB); Framing structure, channel coding and modulation for 11/12 GHz satellite services*, ETSI Std. EN 300 421, Rev. 1.1.2, 1997.

[187] *Digital multiprogramme systems for television sound and data services for cable distribution*, ITU-T Std. J.83, 1997.

[188] *Forward error correction for submarine systems*, ITU-T Std. G.975, 2000.

[189] A. G. M. Strollo, N. Petra, D. D. Caro, and E. Napoli, “An area-efficient high-speed Reed-Solomon decoder in 0.25μm CMOS,” in *Proc. 30th Eur. Solid State Circuits Conf. (ESSCIRC)*, Sept. 2004, pp. 479–482.

[190] H. Y. Hsu and A. Y. Wu, “VLSI design of a reconfigurable multi-mode Reed-Solomon codec for high-speed communication systems,” in *Proc. IEEE Asia-Pacific Conf. Circuits and Syst.*, 2002, pp. 359–362.

[191] T. L. Martin and D. P. Siewiorek, “Nonideal battery properties and their impact on software design for wearable computers,” *IEEE Trans. Comput.*, vol. 52, no. 8, pp. 979–984, Aug. 2003.

[192] S. Hong, J. Yi, and W. E. Stark, “VLSI design and implementation of low-complexity adaptive turbo-code encoder and decoder for wireless mobile communication applications,” in *IEEE Workshop Signal Processing Syst.*, Oct. 1998, pp. 233–242.

[193] Z. Wang, H. Suzuki, and K. K. Parhi, “VLSI implementation issues of turbo decoder design for wireless applications,” in *IEEE Workshop Signal Processing Syst.*, Oct. 1999, pp. 503–512.

[194] O. Y. H. Leung, C. Y. Tsui, and R. S. K. Cheng, “Reducing power consumption of turbo-code decoder using adaptive iteration with variable supply voltage,” *IEEE Trans. VLSI Syst.*, vol. 9, no. 1, pp. 34–41, Feb. 2001.

[195] J. Kaza and C. Chakrabarit, “Design and implementation of low-energy turbo decoders,” *IEEE Trans. VLSI Syst.*, vol. 12, no. 9, pp. 968–977, Sept. 2004.

[196] 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, June 2002.

[197] C. Schurgers, F. Catthoor, and M. Engels, “Memory optimization of map turbo decoder algorithms,” *IEEE Trans. VLSI Syst.*, vol. 9, no. 2, pp. 305–312, Apr. 2001.

[198] R. Saleh, S. Z. Hussian, S. Rochel, and D. Overhauser, “Clock skew verification in the presence of IR-drop in the power distribution network,” *IEEE Trans. Computer-Aided Design.*, vol. 19, no. 6, pp. 635–644, June 2000.

[199] A. Batra, J. Balakrishnan, G. R. A. J. R. Foerster, and A. Dakbak, “Design of a multiband OFDM system for realistic UWB channel environments,” *IEEE Trans. Microwave Theory Tech.*, vol. 52, no. 9, pp. 2123–2138, Sept. 2004.

[200] L. Yang and G. Giannakis, “Ultra-wideband communications,” *IEEE Signal Processing Mag.*, pp. 26–54, Nov. 2004.

[201] D. Krishnaswamy and J. Vicente, “Scalable adaptive wireless networks for multimedia in the proactive enterprise,” *Intel Technology Journal*, vol. 8, pp. 291–302, 2004.

[202] V. Guruswami and A. Vardy, “Maximum-likelihood decoding of Reed-Solomon codes is NP-hard,” *IEEE Trans. Inform. Theory*, vol. 51, no. 7, pp. 1757–1767, July 2005.

[203] J. K. Wolf, “Efficient maximum likelihood decoding of linear block codes using a trellis,” *IEEE Trans. Inform. Theory*, vol. 24, no. 1, pp. 76–80, Jan. 1978.

[204] D. J. Muder, “Minimal trellises for block codes,” *IEEE Trans. Inform. Theory*, vol. 34, no. 5, pp. 1049–1053, Sept. 1988.

[205] S. K. Shin and P. Sweeney, “Soft decision decoding of Reed-Solomon codes using trellis methods,” *IEE Proc. Commun.*, vol. 14, no. 5, pp. 303–308, Oct. 1994.

[206] A. Vardy and Y. Be’ery, “Bit-level soft-decision decoding of Reed-Solomon codes,” *IEEE Trans. Commun.*, vol. 39, no. 3, pp. 440–444, Mar. 2002.

[207] V. Ponnampalam and B. Vucetic, “Soft decision decoding of Reed-Solomon codes,” *IEEE Trans. Commun.*, vol. 50, no. 11, pp. 1758–1768, Nov. 2002.

[208] T. R. Halford, V. Ponnampalam, A. J. Grant, and K. M. Chugg, “Soft-in soft-out decoding of Reed-Solomon codes based on Vardy and Be’ery’s decomposition,” *IEEE Trans. Inform. Theory*, vol. 51, no. 12, pp. 4363–4368, Dec. 2005.

[209] R. Koetter and A. Vardy, “Algebraic soft-decision decoding of reed-solomon codes,” *IEEE Trans. Inform. Theory*, vol. 49, no. 11, pp. 2809–2825, Nov. 2003.

[210] V. Guruswami and M. Sudan, “Improved decoding of Reed-Solomon and algebraic-geometry codes,” *IEEE Trans. Inform. Theory*, vol. 45, no. 6, pp. 1757–1767, Sept. 1999.

[211] J. Jiang and K. R. Narayanan, “Iterative soft decoding of Reed-Solomon codes,” *IEEE Commun. Lett.*, vol. 8, no. 4, pp. 244–246, Apr. 2004.

[212] M. El-Khamy and R. J. McEliece, “Iterative algebraic soft-decision list decoding of Reed-Solomon codes,” *J. Select. Area Commun.*, vol. 24, no. 3, pp. 481–490, Mar. 2006.

[213] G. D. Forney, Jr., “Generalized minimum distance decoding,” *IEEE Trans. Inform. Theory*, vol. 11, no. 2, pp. 125–131, Apr. 1966.

[214] D. Chase, “Class of algorithms for decoding block codes with channel measurement information,” *IEEE Trans. Inform. Theory*, vol. 18, no. 1, pp. 170–182, Jan. 1972.

[215] M. P. C. Fossorier and S. Lin, “Soft-decision decoding of linear block codes based on ordered statistics,” *IEEE Trans. Inform. Theory*, vol. 41, no. 5, pp. 1379–1396, Sept. 1995.

[216] Y. Liu, “MAP decoding of linear block codes, iterative decoding of Reed-Solomon codes and interactive concatenated turbo coding systems,” Ph.D. dissertation, Univ. Hawaii, 1999.

[217] N. Kamiya, “On algebraic soft-decision decoding algorithms for BCH codes,” *IEEE Trans. Inform. Theory*, vol. 41, no. 1, pp. 45–58, Jan. 2001.

[218] H. Tang, Y. Liu, M. P. C. Fossorier, and S. Lin, “On combining Chase-2 and GMD decoding algorithms for nonbinary block codes,” *IEEE Commun. Lett.*, vol. 5, no. 5, pp. 209–211, May 2001.

[219] W. J. Gross, F. R. Kschischang, R. Koetter, and R. G. Gulak, “A VLSI architecture for interpolation in soft-decision list decoding of Reed-Solomon codes,” in *IEEE Workshop Signal Processing Syst.*, 2002, pp. 39–44.

[220] A. Ahmed, R. Koetter, and N. R. Shanbhag, “VLSI architectures for soft-decision decoding of Reed-Solomon codes,” in *IEEE Int. Conf. Commun. (ICC)*, vol. 5, 2004, pp. 2584–2590.

[221] L. Boulianne and W. J. Gross, “SIMD implementation of interpolation in algebraic soft-decision Reed-Solomon decoding,” in *IEEE Workshop Signal Processing Syst.*, 2005, pp. 750–755.

[222] C. Yen and Y. S. Kang, “Cell-based layout techniques supporting gate-level voltage scaling for low power,” *IEEE Trans. VLSI Syst.*, vol. 9, no. 6, pp. 983–986, Dec. 2001.

[223] 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, July 2002.

[224] A. Giulietti, L. van der Perre, and A. Strum, “Parallel turbo coding interleavers: avoiding collisions in accesses to storage elements,” *Electron. Lett.*, vol. 38, no. 5, pp. 232–234, Feb. 2002.

[225] 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.

[226] *Digital Video Broadcasting (DVB); Second generation framing structure, channel coding and modulation systems for Broadcasting, Interactive Services, News Gathering and other broadband satellite applications*, ETSI Std. EN 302 307, Rev. 1.1.1, 2005.

[227] P. Urard *et al.*, “A 135Mb/s DVB-S2 compliant codec based on 64800b LDPC and BCH codes,” in *IEEE Int. Solid-State Circuit Conf. (ISSCC) Dig. Tech. Papers*, 2005, pp. 446–447,609.

[228] M. P. C. Fossorier, “Quasicyclic low-density parity-check codes from circulant permutation matrices,” *IEEE Trans. Inform. Theory*, vol. 50, no. 8, pp. 1788–1793, Aug. 2004.

[229] S. Myung, K. Yang, and J. Kim, “Quasi-cyclic LDPC codes for fast encoding,” *IEEE Trans. Inform. Theory*, vol. 51, no. 8, pp. 2894–2901, Aug. 2005.

[230] R. M. Tanner, D. Sridhara, A. Sridharan, and J. D. J. Fuja, T. E. amd Costello, “LDPC block and convolutional codes based on circulant matrices,” *IEEE Trans. Inform. Theory*, vol. 50, no. 12, pp. 2966–2984, Dec. 2004.

[231] Z. Li, L. Chen, L. Zeng, S. Lin, and W. H. Fong, “Efficient encoding of quasi-cyclic low-density parity-check codes,” *IEEE Trans. Commun.*, vol. 54, no. 1, pp. 71–81, Jan. 2005.

[232] Y. Kou, S. Lin, and M. P. C. Fossorier, “Low-density parity-check codes based on finite geometries: A rediscovery and new results,” *IEEE Trans. Inform. Theory*, vol. 47, no. 7, pp. 2711–2736, Nov. 2001.

[233] S. Lin, L. Chen, J. Xu, and I. Djurdjevic, “Near shannon limit quasi-cyclic low-density parity-check codes,” in *IEEE GLOBECOM*, vol. 4, 2003, pp. 2030–2035.

[234] A. S. Acampora and R. P. Gilmore, “Analog Viterbi decoding for high speed digital satellite channels,” *IEEE Trans. Commun.*, vol. 26, no. 10, pp. 1463–1470, Oct. 1978.

[235] A. Demosthenous and J. Taylor, “Low-power CMOS and BiCMOS circuits for analog convolutional decoders,” *IEEE Trans. Circuits and Syst. II: Analog and Digital Signal Processing*, vol. 46, no. 8, pp. 1077–1080, Aug. 1999.

[236] ——, “A 100-Mb/s 2.8-V CMOS current-mode analog Viterbi decoder,” *IEEE J. Solid-State Circuits*, vol. 37, no. 7, pp. 904–910, July 2002.

[237] M. H. Shakiba, D. A. Johns, and K. W. Martin, “General approach to implementing analogue Viterbi decoders,” *Electron. Lett.*, vol. 30, no. 22, pp. 1823 – 1824, Oct. 1995.

[238] D. Vogrig, A. Gerosa, A. Neviani, A. G. i Amat, G. Montorsi, and S. Benedetto, “A 0.35-μm CMOS analog turbo decoder for the 40-bit rate 1/3 UMTS channel code,” *IEEE J. Solid-State Circuits*, vol. 40, no. 3, pp. 753–762, Mar. 2005.

[239] S. Hemati and A. H. Banihashem, “Dynamics and performance analysis of analog iterative decoding for low-density parity-check (LDPC) codes,” *IEEE Trans. Commun.*, vol. 54, no. 1, pp. 61–70, Jan. 2006.

[240] F. Lustenberger, “On the design of analog VLSI iterative decoders,” Ph.D. dissertation, Swiss Federal Institute of Technology, Zurich, 2000.

[241] C. Winstead, “Analog iterative error control decoders,” Ph.D. dissertation, Univ. of Alberta, 2000.