國立交通大學

電子工程學系　電子研究所碩士班

碩 士 論 文

應用於全球互通微波存取通訊協定的面積優化雙位元迴旋渦輪解碼器

An Area-Efficient Double-Binary CTC Decoder for WiMAX Applications

學生：胡茗智

指導教授：李鎮宜 教授

中華民國九十七年七月

應用於全球互通微波存取通訊協定的面積優化雙位元迴旋渦輪解碼器

An Area-Efficient Double-Binary CTC Decoder for WiMAX Applications

研 究 生：胡茗智　　Student：Ming-Chih Hu

指導教授：李鎮宜教授　　Advisor：Chen-Yi Lee

國 立 交 通 大 學

電子工程學系 電子研究所 碩士班

碩 士 論 文

A Thesis

Submitted to Department of Electronics Engineering & Institute Electronics

College of Electrical and Computer Engineering

National Chiao Tung University

In Partial Fulfillment of the Requirements

for the Degree of

Master of Science

in

Electronics Engineering

July 2008

Hsinchu, Taiwan, Republic of China

中華民國九十七年七月

應用於全球互通微波存取通訊協定的面積優化雙位元迴旋渦輪解碼器

學生：胡茗智　　指導教授：李鎮宜 教授

國立交通大學

電子工程學系 電子研究所碩士班

摘　要

本論文介紹一個雙位元迴旋渦輪解碼理論，同時提出了一個應用於全球互通微波存取通訊協定符合所有種類的面積優化解碼器。我們提出的解碼器可以支援所有定義在 IEEE 802.16e 規格裡的編碼長度。藉由等比例縮減外在資訊，MAX-Log MAP 演算法可以在極小的效能降低下減低硬體複雜度。另外提出了假雙埠暫存檔案可以大量的省下記憶體的使用量以及解碼時所產生的延誤並且允許同時讀取及寫入資料在同一個解碼週期內。除此之外，我們所提出的簡化過後的交錯器只用到了簡單的加法及減法器可以大量的減少硬體的使用量。根據實驗結果，此解碼器在 90nm 製程下最高能達到 30Mb/s 的傳輸速度，晶片的面積是 1.12mm^2。此外，在 0.9V 的供應電壓、166MHz 操作頻率以及編碼長度 2400 下，功率的消耗經量測過後為 32.87mW。

本論文另外提供了一個應用隨機更新規則的柵狀解碼理論。藉由使用了隨機

運算方式，ACS 單位的硬體複雜度可以大大的被簡化。所提出的狀態記憶體增加了隨機切換活動力可避免鎖定在同一個固定的狀態，以及所引用的等比例降低雜訊相依因子可以消去地板錯誤現象。這兩種技術階可讓解碼性能大突的提高。相較於 (2, 1, 3) 的渦輪碼，實驗結果顯示隨機解碼器可以是一個做為降低硬體複雜度的解碼選項。

An Area-Efficient Double-Binary CTC Decoder for WiMAX Applications

Student：Ming-Chih Hu Advisor：Dr. Chen-Yi Lee

Department of Electronics Engineering
Institute of Electronics
National Chiao Tung University

ABSTRACT

Double-binary convolutional turbo code (CTC) decoding algorithm is introduced in this thesis, and a fully compliant and area-efficient CTC decoder for WiMAX 802.16e is proposed. The proposed decoder can support all code lengths specified in IEEE 802.16e system. By scaling the extrinsic information, the Max-Log MAP algorithm is used such that hardware complexity can be reduced with the minimized performance loss. For saving memory requirement and reducing decoding latency, the pseudo two-port register file is also demonstrated to allow read and write operation within one decoding cycle. Moreover, a simplified interleaver architecture which uses simple addition and subtraction instead of division is proposed to reduce the hardware area and decrease the critical path. Implemented in the 90-nm process, the proposed decoder chip occupied in 1.12mm^2 core area can achieve 30Mb/s decoding throughput. The power consumption according to post-layout simulation is 32.87mW operated at supply voltage 0.9V and clock rate 166MHz with block length of 2400.

Another trellis-based decoding algorithm using stochastic update rule is also presented in this thesis. By using the stochastic computation, the hardware complexity

can be reduced by simplifying ACS-unit operation. The proposed state memory can increase the random switching activity to avoid the state locked into a fixed state, and noise dependent scaling factor can further eliminate the error floor effect. Both techniques can greatly improve the performance compared to the Viterbi decoding algorithm. Through the simulation analysis and parameter decision for (2, 1, 3) convolutional code, the performance comparison shows that the stochastic decoding algorithm can be one of the candidates for low complexity iterative decoding.

誌　　謝

兩年的碩士班生涯轉眼間就過去了，感謝我的指導教授李鎮宜老師建立了 Si2Lab 良好的研究氣氛與實驗室環境並且總是能夠耐心的、慈祥的指引我走向正確方向，即便老師是如此的忙碌，還是不忘親切的關心每個人的研究進度讓我們能毫無壓力的做研究。感謝張錫嘉老師在我研究遇到挫折時給予我正面的思考方向，很高興遇上了這麼好的老師，給予了我很大的自由度讓我能做自己喜歡的研究；另外在求職的路上也提供了很多的寶貴意見，讓我不致於徬徨無助。

特別感謝陳志龍學長，從大一進交大到現在碩二要離開交大了都受到學長的照顧，記得錫嘉老師曾經說過「除了畢業的那張文憑，還有更多的習慣以及態度是需要在這幾年建立的」，感謝學長總是能不厭其煩的教導我正確的研究態度，很多的習慣及態度都是在碩士班這兩年所建立的。感謝林建青學長以及廖彥欽學姊所領導的 OCEAN 研究團隊，總是能給我許多的幫助，讓我能有信心的解決接踵而來的問題。

最後感謝 Si2Lab 以及 OCEAN group 的所有學長姊、學弟妹，以及我身邊的所有朋友，當然，還有最支持我的家人們，因為有你們的陪伴，讓我能不孤單的走完這兩年，謝謝你們。

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Research Motivation

The fundamental block diagram of a typical digital communication system is shown in Fig. 1.1. Signal transformation from the information source to the transmitter includes source encoding, channel encoding and modulation. The receiver will reverse the signal transformation by demodulation, channel decoding and source decoding. In order to eliminate the effects of noise disturbances, the channel encoder transforms the source codeword into the channel codeword by adding certain structural redundancy. These redundant bits can be used for detecting and correcting the errors. Theoretically, the encoding procedure provides the encoded signal with better distance properties than the un-coded one, and thus channel coding can improve the performance of the overall system.

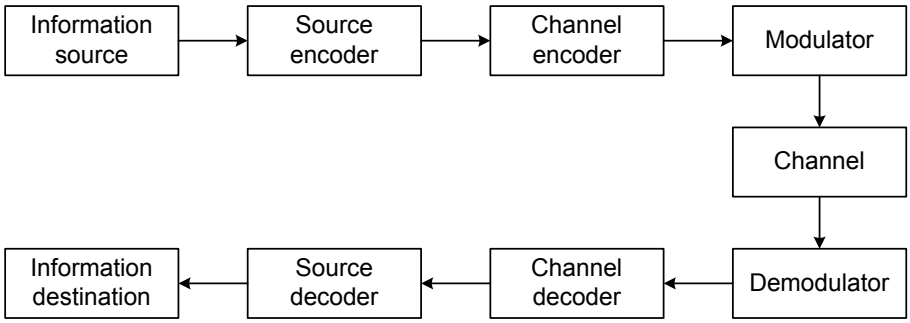


Figure 1.1: Block diagram of a typical digital communication system

In the last decade, trellis-based decoding algorithm applied to convolutional code or turbo code has been adopted in many standards because of its execellent error correction ability. Although the turbo code has outstanding error-correcting performance, the decoding efficiency and maximum throughput still cannot meet the standard requirement with higher throughput. Hence, double-binary convolutional turbo code is introduced in the recent years because of its high decoding efficiency and excellent error-correcting performance. The double-binary CTC decoder adopted in WiMAX 802.16e [1] standard which defined detail standard providing maximum throughput about 30Mb/s will be proposed in this thesis, and the hardware architecture and chip implementation result are also presented in the following chapter.

The main problem of double-binary convolutional turbo code is the higher hardware complexity on high-radix ACS-unit from single-binary to double-binary (even triple-binary). In order to design a low complexity trellis-based decoder, the stochastic computation will be applied to trellis-based decoding algorithm. Stochastic arithmetic introduced in 1960's can break speed bottleneck caused by recursive computation to increase the operating frequency. Besides, the error-correcting performance can be adjusted by decoding cycles. As a result, since stochastic decoding algorithm has been successfully applied to LDPC code, it might have potential to apply to convolutional code and would be introduced and applied on trellis diagram in this thesis.

1.2 Thesis Organization

This thesis consists of 5 chapters. In chapter 2, different kinds of trellis-based decoding algorithms are reviewed, such as single-binary turbo decdoing algorithm, double-binary turbo decoding algorithm, and stochasitc decoding algorithm using update rule. The stochastic decoder applied on trellis-based decoding algorithm is described in chapter 3. Further improvement and simulation analysis are also stated. Chapter 4 introduces the implementation of double-binary convolutional turbo code applied to WiMAX 802.16e system, including the performance comparison, the hardware architecture, and the chip implementation result. Finally, the conclusion is given in chapter 5. The parameters used in WiMAX 802.16e are also illustrated in appendix A.

Chapter 2

Trellis-based Decoding Algorithms

2.1 MAP Decoding Algorithm

2.1.1 The MAP Decoding Algorithm

The maximum *a posteriori* probability (MAP) decoding algorithm, also termed as $BCJR$ decoding algorithm, is developed by Bahl, Cocke, Jelinek, and Raviv in 1974 [2]. The MAP algorithm is optimal for estimating the states or the outputs of a Markov process observed under AWGN channel. It produces the sequence of *a posteriori* probabilities (APP) from the received sequence $\mathbf{r}$ over a discrete memoryless channel (DMC) and minimizes the symbol error probability. Assume for state transition from $S_{m'}^{t}$ at time t to $S_{m}^{(t+1)}$ at time $t+1$, we can estimate the joint probability

$$\begin{aligned}
\Pr\{S_{m'}^{(t)}, S_{m}^{(t+1)}, \mathbf{r}\} &= \Pr\{S_{m'}^{(t)}, S_{m}^{(t+1)}, \mathbf{r}_0^{t-1}, r_t, \mathbf{r}_{t+1}^{N-1}\} \\
&= \Pr\{\mathbf{r}_{t+1}^{N-1} | S_{m'}^{(t)}, S_{m}^{(t+1)}, \mathbf{r}_0^{t-1}, r_t\} \\
&\quad \times \Pr\{S_{m}^{(t+1)}, r_t | S_{m'}^{(t)}, \mathbf{r}_0^{t-1}\} \\
&\quad \times \Pr\{S_{m'}^{(t)}, \mathbf{r}_0^{t-1}\} \\
&= \Pr\{\mathbf{r}_{t+1}^{N-1} | S_{m}^{(t+1)}\} \Pr\{S_{m}^{(t+1)}, r_t | S_{m'}^{(t)}\} \Pr\{S_{m'}^{(t)}, \mathbf{r}_0^{t-1}\}
\end{aligned} \tag{2.1}$$

Notice that (m', m) means the state transition and $\mathbf{r}_0^{t-1}$ denotes the received sequence from time 0 to $t-1$, $\mathbf{r}_{t+1}^{N-1}$ denotes the received sequence from time $t+1$ to $N-1$, and

r_t denotes the codeword symbol at time t. We further redefine the equation in (2.1) :

$$\alpha(S_{m'}^{(t)}) = \Pr\{S_{m'}^{(t)}, \mathbf{r}_0^{t-1}\} \tag{2.2}$$

$$\gamma(S_{m'}^{(t)}, S_m^{(t+1)}) = \Pr\{S_m^{(t+1)}, r_t | S_{m'}^{(t)}\} \tag{2.3}$$

$$\beta(S_m^{(t+1)}) = \Pr\{\mathbf{r}_{t+1}^{N-1} | S_m^{(t+1)}\}, \tag{2.4}$$

and thus (2.1) can be rewritten as

$$\Pr\{S_{m'}^{(t)}, S_m^{(t+1)}, \mathbf{r}\} = \alpha(S_{m'}^{(t)})\gamma(S_{m'}^{(t)}, S_m^{(t+1)})\beta(S_m^{(t+1)}) \tag{2.5}$$

Now, we will derive the equations (2.2), (2.3), and (2.4) as follow:

$$\begin{aligned}
\alpha(S_m^{(t+1)}) &= \Pr\{S_m^{(t+1)}, \mathbf{r}_0^t\} \\
&= \sum_{S_{m'}^{(t)} \in \mathbf{S}} \Pr\{S_{m'}^{(t)}, S_m^{(t+1)}, \mathbf{r}_0^t\} \\
&= \sum_{S_{m'}^{(t)} \in \mathbf{S}} \Pr\{S_m^{(t+1)}, r_t, | S_{m'}^{(t)}, \mathbf{r}_0^{t-1}\} \Pr\{S_{m'}^{(t)}, \mathbf{r}_0^{t-1}\} \\
&= \sum_{S_{m'}^{(t)} \in \mathbf{S}} \Pr\{S_m^{(t+1)}, r_t, | S_{m'}^{(t)}\} \Pr\{S_{m'}^{(t)}, \mathbf{r}_0^{t-1}\} \\
&= \sum_{S_{m'}^{(t)} \in \mathbf{S}} \gamma(S_{m'}^{(t)}, S_m^{(t+1)})\alpha(S_{m'}^{(t)})
\end{aligned} \tag{2.6}$$

Similarly,

$$\begin{aligned}
\beta(S_{m'}^{(t)}) &= \sum_{S_m^{(t+1)} \in \mathbf{S}} \Pr\{S_m^{(t+1)}, \mathbf{r}_t^{N-1} | S_{m'}^{(t)}\} \\
&= \sum_{S_m^{(t+1)} \in \mathbf{S}} \Pr\{S_m^{(t+1)}, r_t, \mathbf{r}_{t+1}^{N-1}, S_{m'}^{(t)}\} / \Pr\{S_{m'}^{(t)}\} \\
&= \sum_{S_m^{(t+1)} \in \mathbf{S}} \Pr\{\mathbf{r}_{t+1}^{N-1} | S_m^{(t+1)}, r_t, S_{m'}^{(t)}\} \Pr\{S_m^{(t+1)}, r_t | S_{m'}^{(t)}\} \\
&= \sum_{S_m^{(t+1)} \in \mathbf{S}} \Pr\{\mathbf{r}_{t+1}^{N-1} | S_m^{(t+1)}\} \Pr\{S_m^{(t+1)}, r_t | S_{m'}^{(t)}\} \\
&= \sum_{S_m^{(t+1)} \in \mathbf{S}} \beta(S_m^{(t+1)})\gamma(S_{m'}^{(t)}, S_m^{(t+1)}),
\end{aligned} \tag{2.7}$$

where $\mathbf{S}$ is the set of all states. From the equations (2.6) and (2.7), we can find that the *forward metric* α and the *backward metric* β will be computed recursively in opposite

direction. Assume the trellis diagram diverges fom zero state at time 0 and converges to zero state at time N, the initial conditions are satisfied:

$$\begin{aligned} \alpha(S_0^{(0)}) = 1, \quad \alpha(S_x^{(0)}) = 0 \quad \text{for } S_x^{(0)} \in \mathbf{S} \backslash S_0 \\ \beta(S_0^{(N)}) = 1, \quad \beta(S_x^{(N)}) = 0 \quad \text{for } S_x^{(N)} \in \mathbf{S} \backslash S_0 \end{aligned} \tag{2.8}$$

Furthermore, the *branch metric* from state m' to m can be computed as

$$\begin{aligned} \gamma(S_{m'}^{(t)}, S_m^{(t+1)}) &= \frac{\Pr\{S_m^{(t+1)}, S_{m'}^{(t)}, r_t\}}{\Pr\{S_{m'}^{(t)}\}} \\ &= \frac{\Pr\{S_m^{(t+1)}, S_{m'}^{(t)}\}}{\Pr\{S_{m'}^{(t)}\}} \times \frac{\Pr\{S_m^{(t+1)}, S_{m'}^{(t)}, r_t\}}{\Pr\{S_m^{(t+1)}, S_{m'}^{(t)}\}} \\ &= \Pr\{S_m^{(t+1)} | S_{m'}^{(t)}\} \Pr\{r_t | S_m^{(t+1)}, S_{m'}^{(t)}\} \\ &= P(u_t) P(r_t | \hat{v}_t), \end{aligned} \tag{2.9}$$

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 for $0 \leq t \leq N$.

For the single-binary *Recursive Systematic Convolutional* (RSC) encoder input signal u_t after BPSK mapping, the log-likelihood ratio (LLR) can be defined as

$$L(u_t) \triangleq \ln \frac{\Pr\{u_t = +1 | \mathbf{r}\}}{\Pr\{u_t = -1 | \mathbf{r}\}} \tag{2.10}$$

Therefore, the equation can be further decomposed to

$$\begin{aligned} L(u_t) &= \ln \frac{\sum_{(m',m) \in \mathbf{B}_t^{+1}} \Pr\{S_{m'}^{(t)}, S_m^{(t+1)} | \mathbf{r}\}}{\sum_{(m',m) \in \mathbf{B}_t^{-1}} \Pr\{S_{m'}^{(t)}, S_m^{(t+1)} | \mathbf{r}\}} \\ &= \ln \frac{\sum_{(m',m) \in \mathbf{B}_t^{+1}} \Pr\{S_{m'}^{(t)}, S_m^{(t+1)}, \mathbf{r}\}}{\sum_{(m',m) \in \mathbf{B}_t^{-1}} \Pr\{S_{m'}^{(t)}, S_m^{(t+1)}, \mathbf{r}\}} \\ &= \ln \frac{\sum_{(m',m) \in \mathbf{B}_t^{+1}} \alpha(S_{m'}^{(t)}) \gamma(S_{m'}^{(t)}, S_m^{(t+1)}) \beta(S_m^{(t+1)})}{\sum_{(m',m) \in \mathbf{B}_t^{-1}} \alpha(S_{m'}^{(t)}) \gamma(S_{m'}^{(t)}, S_m^{(t+1)}) \beta(S_m^{(t+1)})}, \end{aligned} \tag{2.11}$$

where $\mathbf{B}_t^{+1}$ is the set of all (m', m) that indicate the state transitions are caused by $u_t = +1$, and $\mathbf{B}_t^{-1}$, the set of (m', m), denotes the state transitions are due to $u_t = -1$.

To decide the decoded output signal $\hat{u}_t$, make a hard decision to the value of LLR, then $\hat{u}_t$ can be estimated as

$$\hat{u}_t = \begin{cases} +1 & \text{if } L(u_t) \geq 0 \\ -1 & \text{if } L(u_t) < 0 \end{cases} \tag{2.12}$$

2.1.2 The Log-MAP Decoding Algorithm

From equations (2.6), (2.7), and (2.9), we can realize that the MAP algorithm requires complex hardware resource. In order to simplify hardware complexity, we can transform MAP decoding algorithm into logarithmic domain. At first, we need to transfer the *branch metric* γ in (2.9) to logarithmic domain; that is

$$\bar{\gamma}(S_{m'}^{(t)}, S_m^{(t+1)}) = \ln \gamma(S_{m'}^{(t)}, S_m^{(t+1)}) \tag{2.13}$$

Then, the *forward metric* α in (2.6) and the *backward metric* β in (2.7) can be further expressed as

$$\bar{\alpha}(S_{m'}^{(t+1)}) = \ln \alpha(S_{m'}^{(t+1)}) = \ln \sum_{S_{m'}^{(t)} \in \mathbf{S}} e^{\bar{\gamma}(S_{m'}^{(t)}, S_m^{(t+1)}) + \bar{\alpha}(S_{m'}^{(t)})}, \tag{2.14}$$

and

$$\bar{\beta}(S_m^{(t)}) = \ln \beta(S_m^{(t)}) = \ln \sum_{S_m^{(t+1)} \in \mathbf{S}} e^{\bar{\beta}(S_m^{(t+1)}) + \bar{\gamma}(S_{m'}^{(t)}, S_m^{(t+1)})} \tag{2.15}$$

As the path metrics have been changed, the initial conditions of metrics become

$$\begin{aligned} \bar{\alpha}(S_0^{(0)}) = 0, \quad \bar{\alpha}(S_x^{(0)}) = -\infty \quad \text{for } S_x^{(0)} \in \mathbf{S} \backslash S_0 \\ \bar{\beta}(S_0^{(N)}) = 0, \quad \bar{\beta}(S_x^{(N)}) = -\infty \quad \text{for } S_x^{(N)} \in \mathbf{S} \backslash S_0 \end{aligned} \tag{2.16}$$

Referring to (2.13), (2.14), and (2.15), the LLR in (2.11) can be rewritten as

$$\begin{aligned} L(u_t) = \ln \left[\sum_{(m',m) \in \mathbf{B}_t^{+1}} e^{\bar{\alpha}(S_{m'}^{(t)}) + \bar{\gamma}(S_{m'}^{(t)}, S_m^{(t+1)}) + \bar{\beta}(S_m^{(t+1)})} \right] \\ - \ln \left[\sum_{(m',m) \in \mathbf{B}_t^{-1}} e^{\bar{\alpha}(S_{m'}^{(t)}) + \bar{\gamma}(S_{m'}^{(t)}, S_m^{(t+1)}) + \bar{\beta}(S_m^{(t+1)})} \right] \end{aligned} \tag{2.17}$$

To simplify the logarithmic domain, we consider the Jacobian function [3]

$$\ln(e^{x_1} + e^{x_2}) \triangleq \max{}^*(e^{x_1}, e^{x_2}) = \max(x_1, x_2) + \ln(1 + e^{-|x_1 - x_2|}), \tag{2.18}$$

and the correction term $\ln(1+e^{-|x_1-x_2|})$ can be implemented by a lookup table to simplify hardware design. Apply the recursive procedure to (2.18), we can extend the Jacobian function to

$$\ln(e^{x_1} + e^{x_2} + \cdots + e^{x_b}) \triangleq \max{}^*(e^{x_1}, e^{x_2}, \ldots, e^{x_b}) \tag{2.19}$$

$$= \max{}^*(\cdots \max{}^*(\max{}^*(x_1, x_2), x_3) \cdots, x_b) \tag{2.20}$$

Apply (2.18) to (2.14) and (2.15)

$$\bar{\alpha}(S_{m'}^{(t+1)}) = \max{}^*{}_{S_{m'}^{(t)} \in \mathbf{S}}[\bar{\gamma}(S_{m'}^{(t)}, S_m^{(t+1)}) + \bar{\alpha}(S_{m'}^{(t)})] \tag{2.21}$$

$$\bar{\beta}(S_m^{(t)}) = \max{}^*{}_{S_m^{(t+1)} \in \mathbf{S}}[\bar{\beta}(S_m^{(t+1)}) + \bar{\gamma}(S_{m'}^{(t)}, S_m^{(t+1)})], \tag{2.22}$$

and therefore,

$$\begin{aligned} L(u_t) = &\max{}^*{}_{(m',m) \in \mathbf{B}_t^{+1}}[\bar{\alpha}(S_{m'}^{(t)}) + \bar{\gamma}(S_{m'}^{(t)}, S_m^{(t+1)}) + \bar{\beta}(S_m^{(t+1)})] \\ &- \max{}^*{}_{(m',m) \in \mathbf{B}_t^{-1}}[\bar{\alpha}(S_{m'}^{(t)}) + \bar{\gamma}(S_{m'}^{(t)}, S_m^{(t+1)}) + \bar{\beta}(S_m^{(t+1)})]. \end{aligned} \tag{2.23}$$

The MAP decoding algorithm based on (2.21), (2.22), and (2.23) is termed Log-MAP algorithm [4, 5].

2.1.3 The Max-Log-MAP Decoding Algorithm

The performance of the Log-MAP algorithm is equivalent to the MAP algorithm but the hardware complexity has been reduced considerably. However, the correction term $y = \ln(1 + e^{-|x_1 - x_2|})$ in (2.18) requires a lookup table to simplify the computation.

In Fig. 2.1, we can discover that y decreases rapidly as $x = |x_1 - x_2|$ increases. In order to further simplify the complexity of correction term, it is possible to discard y with some performance degradation because of the information loss.

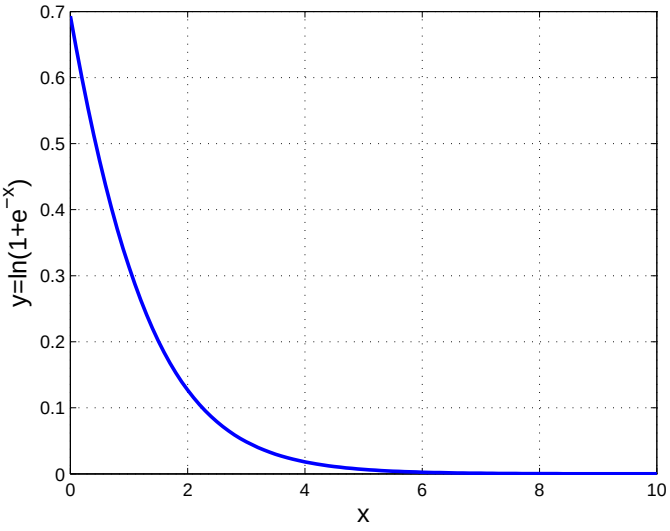


Figure 2.1: Correction Factor

Consequently, we have the following approximations:

$$\max{}^*(e^{x_1}, e^{x_2}) \approx \max(x_1, x_2) \tag{2.24}$$

$$\max{}^*(e^{x_1}, e^{x_2}, \ldots, e^{x_n}) \approx \max_{i=1\sim n}(x_i). \tag{2.25}$$

Applying (2.24) and (2.25), we can reduce the Log-MAP algorithm to the Max-Log-MAP algorithm that contains only the additions and the max functions. Therefore, we can rewrite (2.21), (2.22), and (2.23) as

$$\bar{\alpha}(S_{m'}^{(t+1)}) \approx \max_{S_{m'}^{(t)} \in \mathbf{S}} [\bar{\gamma}(S_{m'}^{(t)}, S_m^{(t+1)}) + \bar{\alpha}(S_{m'}^{(t)})] \tag{2.26}$$

$$\bar{\beta}(S_m^{(t)}) \approx \max_{S_m^{(t+1)} \in \mathbf{S}} [\bar{\beta}(S_m^{(t+1)}) + \bar{\gamma}(S_{m'}^{(t)}, S_m^{(t+1)})], \tag{2.27}$$

and

$$\begin{aligned} L(u_t) \approx & \max_{(m',m)\in \mathbf{B}_t^{+1}} [\bar{\alpha}(S_{m'}^{(t)}) + \bar{\gamma}(S_{m'}^{(t)}, S_m^{(t+1)}) + \bar{\beta}(S_m^{(t+1)})] \\ & - \max_{(m',m)\in \mathbf{B}_t^{-1}} [\bar{\alpha}(S_{m'}^{(t)}) + \bar{\gamma}(S_{m'}^{(t)}, S_m^{(t+1)}) + \bar{\beta}(S_m^{(t+1)})]. \end{aligned} \tag{2.28}$$

2.1.4 Sliding Window Approach

From the previous discussion, we can find that in the MAP-series decoding algorithm (including MAP algorithm, Log-MAP algorithm and MAX-Log-MAP algorithm), the calculation of LLR requires the forward metrics and backward metrics; all of the metrics should be kept to calculate all $L(u_t)$ with $t = 1 \sim N$. Since the backward recursive computation initials from the end of the decoding trellis, the LLR value cannot be calculated until the entire block metrics received. If the block length is large, it will lead to long output latency and require huge memory for hardware implementation.

To reduce the memory requirement, the sliding window algorithm [6–8] is applied to avoid storing the metrics corresponding to the entire codeword sequence. This algorithm utilizes the fact that the backward metrics can be highly reliable even without the initial condition if the length of backward recursive computation is long enough. In Fig. 2.2, the codeword sequence is divided into $\lceil N/T_w \rceil$ sub-blocks with sliding window length T_w which is also called the convergence length, and the dummy backward recursion β_d is employed to establish the initial conditions for the true backward recursion β. Although

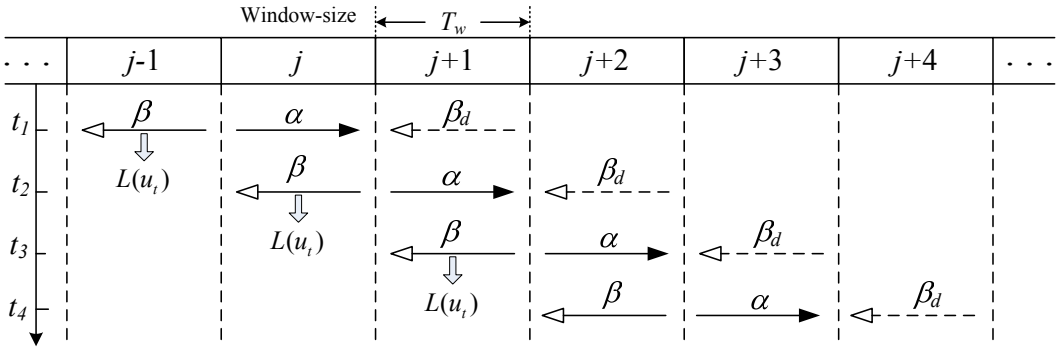


Figure 2.2: The process of sliding window MAP algorithm

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

$$\beta_d(S_m^{((j+1)\cdot T_w)}) = \frac{1}{M}, \quad \text{for all } S_m^{(j\omega)} \in \mathbf{S}, \tag{2.29}$$

where M is the state number in trellis diagram. After the backward recursive computation β_d process of T_w time instances, the initial metrics $\beta(S_m^{(j\cdot T_w)})$ 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 calculate 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-th sub-block. The sliding window length T_w which is set to be six times constraint length of component encoder in turbo code to ensure the reliable initialization for the β recursion [8].

2.2 Turbo Code

Turbo code, also named parallel concatenated convolutional code (PCCC), convolutional turbo code (CTC), or turbo convolutional code (TCC), was first proposed by C. Berrou, A. Glavieux and P. Thitimajshima in 1993 [9,10]. It has been proved that the performance of turbo code can be close to shannon limit with simple recursive systematic convolutional (RSC) codes concatenated by an interleaver whose length is N. The interleaver permutes the information sequence before the second encoding, introducing code diversity.

2.2.1 Turbo Encoder

The turbo encoder is composed of two RSC encoders and an interleaver to reorder the information sequence. Note that the RSC encoder must be recursive for better performance [11]. In Fig. 2.3, the information symbols are encoded to the systematic part $\mathbf{v}_0(D)$ and the parity part $\mathbf{v}_1(D)$; thus, $\mathbf{v}_0(D) = \mathbf{u}(D)$. And the second encoder encodes the interleaved information symbols $\tilde{\mathbf{u}}(D)$ to the parity part $\mathbf{v}_2(D)$.

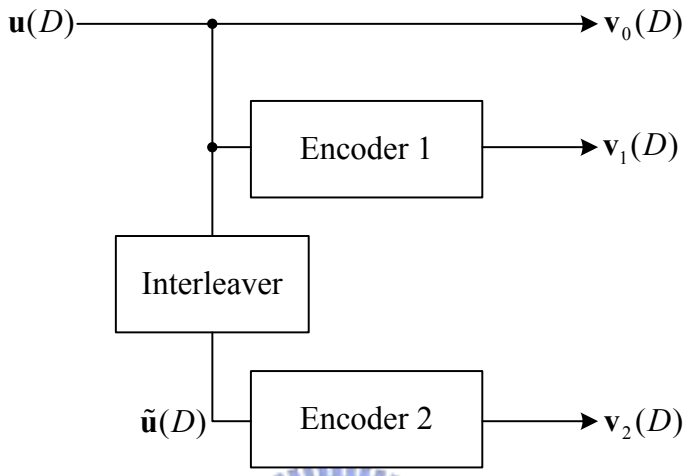


Figure 2.3: Turbo encoder

2.2.2 Turbo Interleaver

The main reason causing turbo code performance so close to shannon limit is the interleaver. As shown in Fig. 2.3, the interleaver permutes the information sequence $\mathbf{u}(D)$ to $\tilde{\mathbf{u}}(D)$. Therefore, the interleaver can spread out the burst errors and further eliminate the correlation of the input of two RSC encoders so that the iterative decoding algorithm based on exchanging un-correlated information between two decoders can be applied. Also, the interleaver can break low weight codewords to improve the coding gain.

The code distance spectrum dominates the error-correcting performance of the turbo code. Referring to [12], the process of the interleaver called spectral thinning can reduce the error probability of low weight codewords. If we assume the interleaver performs random permutation, the error probability can be reduced by a factor of $1/N$ [11, 13], where N is the interleaver size. And $1/N$ is also refered to the interleaver gain. 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. Consequently, the interelaver structure is desirable to break these input patterns. In such case, the input sequence to the second encoder, which is generated by the interleaver, will most likely produce a high weight parity check sequence and further increase the whole turbo codeword weight.

2.2.3 Turbo Decoder

The iterative turbo decoding process based on MAP algorithm is to exchange the soft information among soft-in/soft-out (SISO) decoders to calculate *a posteriori* probability of each information bit u_t [2]. For a code 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 $r_t^{(1)} \sim r_t^{(n-1)}$. If the *a priori* information is represented by

$$L_a(u_t) \triangleq \ln \frac{P(u_t = +1)}{P(u_t = -1)}, \tag{2.30}$$

additionally, the channel reliability value L_c is defined to be $\frac{4E_s}{N_0}$ for the AWGN channel [14], and the branch metric in logarithmic domain would be

$$\bar{\gamma}(S_{m'}^{(t)}, S_m^{(t+1)}) = \ln P(u_t)P(r_t|\hat{v}_t) = \frac{1}{2}(u_t L_a(u_t) + L_c u_t r_{0,t} + \sum_{i=1}^{n-1} L_c r_t^{(i)} \hat{v}_t^{(i)}), \tag{2.31}$$

which is from (2.9) and (2.13).

As a result, the APP information from the SISO decoder can be derived as follows:

$$\begin{aligned}
L(u_t) &= \ln \frac{\sum_{(m',m)\in \mathbf{B}_t^{+1}} \left[e^{\bar{\alpha}(S_{m'}^{(t)}) + \bar{\gamma}(S_{m'}^{(t)}, S_m^{(t+1)}) + \bar{\beta}(S_m^{(t+1)})} \right]}{\sum_{(m',m)\in \mathbf{B}_t^{-1}} \left[e^{\bar{\alpha}(S_{m'}^{(t)}) + \bar{\gamma}(S_{m'}^{(t)}, S_m^{(t+1)}) + \bar{\beta}(S_m^{(t+1)})} \right]} \\
&= \ln \frac{\sum_{(m',m)\in \mathbf{B}_t^{+1}} \left[e^{\frac{1}{2}((+1)L_a(u_t) + (+1)L_c r_{,0t})} \right] \left[e^{\bar{\alpha}(S_{m'}^{(t)}) + \frac{1}{2}\sum_{i=1}^{n-1} L_c r_t^{(i)} \hat{v}_t^{(i)} + \bar{\beta}(S_m^{(t+1)})} \right]}{\sum_{(m',m)\in \mathbf{B}_t^{-1}} \left[e^{\frac{1}{2}((-1)L_a(u_t) + (-1)L_c r_{0,t})} \right] \left[e^{\bar{\alpha}(S_{m'}^{(t)}) + \frac{1}{2}\sum_{i=1}^{n-1} L_c r_t^{(i)} \hat{v}_t^{(i)} + \bar{\beta}(S_m^{(t+1)})} \right]} \\
&= L_a(u_t) + L_c r_{0,t} + \ln \frac{\sum_{(m',m)\in \mathbf{B}_t^{+1}} \left[e^{\bar{\alpha}(S_{m'}^{(t)}) + \frac{1}{2}\sum_{i=1}^{n-1} L_c r_t^{(i)} \hat{v}_t^{(i)} + \bar{\beta}(S_m^{(t+1)})} \right]}{\sum_{(m',m)\in \mathbf{B}_t^{-1}} \left[e^{\bar{\alpha}(S_{m'}^{(t)}) + \frac{1}{2}\sum_{i=1}^{n-1} L_c r_t^{(i)} \hat{v}_t^{(i)} + \bar{\beta}(S_m^{(t+1)})} \right]} \\
&= L_a(u_t) + L_c r_{0,t} + L_e(u_t).
\end{aligned} \tag{2.32}$$

The term $L_e(u_t)$ is the *extrinsic* information corresponding to the information bit u_t [9,10].

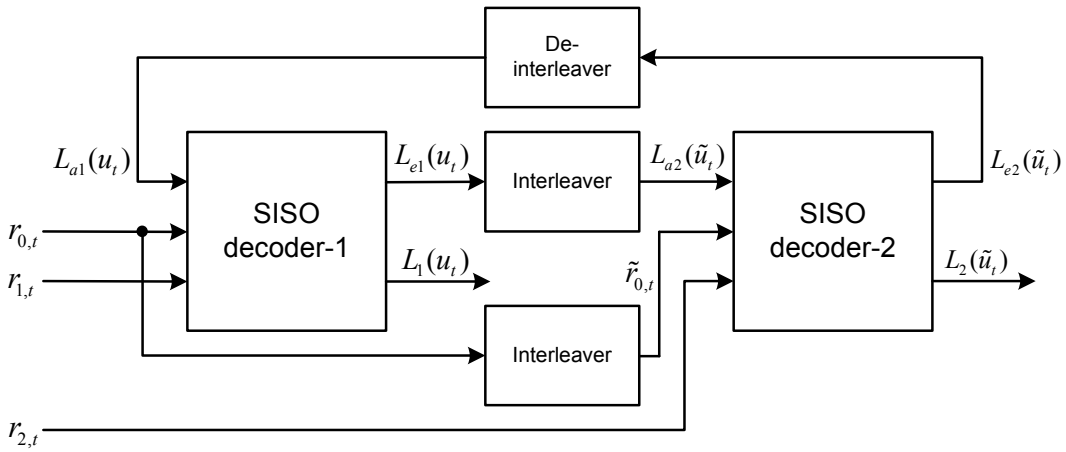


Figure 2.4: Turbo decoder

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. 2.4, there are two SISO decoders for the two constituent encoders in Fig. 2.3. Initially, we set the *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 (2.32), the extrinsic information $L_{e1}(u_t)$ can be obtained

$$L_{e1}(u_t) = L_1(u_t) - L_c r_{0,t} - L_{a1}(u_t), \tag{2.33}$$

where $L_{a1}(u_t) = 0$ 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

$$L_{e2}(\tilde{u}_t) = L_2(\tilde{u}_t) - L_c \tilde{r}_{0,t} - L_{a2}(\tilde{u}_t). \tag{2.34}$$

As shown in Fig. 2.4, 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 $L_2(\tilde{u}_t)$ through the de-interleaver is exported for hard decision. Notice that both SISO decoders in Fig. 2.4 will complete once within each decoding iteration.

The BER curve of turbo code can be divided into three regions [15], at very low SNRs, the signal is so greatly corrupted by channel noise that the decoder cannot improve the error rate and may even degrade it. The non-convergence region has an almost constant and high error probability. As the SNR increases, a waterfall region is encountered where the error rate drops sharply. As the SNR increases still further, a error floor region is encountered where the curve becomes less steep, limiting the performance gains. This error floor region is primarily a function of the distance properties of the code, which can be expressed by (2.35)

$$P_b \propto Q\left(\sqrt{2d_{free}R\frac{E_b}{N_0}}\right), \tag{2.35}$$

where d_{free} is the code minimum free distance, R is the code rate, and $\frac{E_b}{N_0}$ is the SNR.

2.3 Double-binary Convolutional Turbo Code Decoding Algorithm

Double-binary convolutional turco code (CTC) can provide better performance than single binary turbo code for equivalent compexity [16]. This section will introduce double-binary CTC with tail-biting technique which can avoid reducing the code rate and increasing the transmission bandwidth. Using double-binary CTC, the latency of the decoder is halved [17], and it could be easily adopted in many standards, such as DVB-RCS and WiMAX standards [1, 18].

2.3.1 Double-binary CTC Encoder

The bouble-binary CTC encoder is shown in Fig. 2.5. Compare to the conventional turbo code, there has two systematic bits, so the number of branches connected to each state in trellis diagram are increased from two to four.

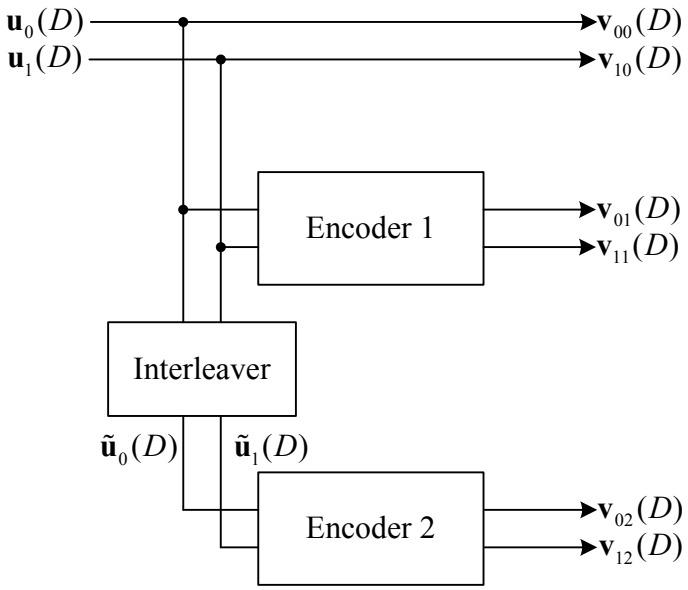


Figure 2.5: Double-binary Convolutional Turbo encoder

For conventional turbo encoder, we should add tail bits to force the trellis diagram to finish at zero state. The trellis termination makes sure that the initial state for the next block is the all-zero state, but the tail bits will decrease the code rate and degrade the transmission efficiency, and the degradation will be more for the shorter blocks. Using tail-biting application, also called circulation states, the state of the encoder at the beginning of the encoding process is not necessarily the all-zero state. The fundamental idea behind tail-biting is that the encoder is controlled in such a way that it starts and ends the encoding process in the same state [19].

The circular coding ensures that, at the end of the encoding operation, the encoder retrieves the initial state, so that data encoding may be represented by a circular trellis. Assume there exists such a circulation state S_c, if the encoder starts from state S_c, it comes back to the same state when the encoding process is finished. The derivation of circulation state S_c requires a pre-encoding operation. First, the encoder is initialized in the all zero state, and the data sequence of length N is encoded once, leading to a final state $S_m^{(N)}$. Second, we find S_c from the final state $S_m^{(N)}$ by the following equation [19]:

$$S_c = \left(I + G^N\right)^{-1} \times S_m^{(N)}, \tag{2.36}$$

where G is the generator matrix which comes from encoder, and I is the identity matrix. Finally, data are encoded starting from the state S_c calculated by (2.36).

2.3.2 Decoding Procedure for Double-binary CTC

According to the iterative decoding algorithm of turbo codes in section 2.2, we realize that the goal of the MAP decoding algorithm is to achieve the extrinsic and LLR values. Therefore, for the input signals $u_{0,t}$ and $u_{1,t}$, the LLR for $i = 1, 2, 3$ can be represented as

$$L_i\left(d_t\right) \triangleq \ln \frac{\Pr\left\{d_t = i|\mathbf{r}\right\}}{\Pr\left\{d_t = 0|\mathbf{r}\right\}}, \tag{2.37}$$

where d_t in $GF(2^2)$ is defined as the collection of input symbols $(u_{0,t}, u_{1,t})$ with elements $\{0, 1, 2, 3\}$ from time $(t-1)$ to time t (that is, $d_t = 00, 01, 10, 11$. We use decimal notation instead of binary for simplicity), and $\mathbf{r}$ is received symbol after QPSK mapping. The decomposition of the above equation will be

$$\begin{aligned} L_i(d_t) &= \ln \frac{\sum_{(m',m)\in\mathbf{B}_t^i} \Pr\{S_{m'}^{(t)}, S_m^{(t+1)}|\mathbf{r}\}}{\sum_{(m',m)\in\mathbf{B}_t^0} \Pr\{S_{m'}^{(t)}, S_m^{(t+1)}|\mathbf{r}\}} \\ &= \ln \frac{\sum_{(m',m)\in\mathbf{B}_t^i} \Pr\{S_{m'}^{(t)}, S_m^{(t+1)}, \mathbf{r}\}}{\sum_{(m',m)\in\mathbf{B}_t^0} \Pr\{S_{m'}^{(t)}, S_m^{(t+1)}, \mathbf{r}\}} \\ &= \ln \frac{\sum_{(m',m)\in\mathbf{B}_t^i} \alpha(S_{m'}^{(t)})\gamma(S_{m'}^{(t)}, S_m^{(t+1)})\beta(S_m^{(t+1)})}{\sum_{(m',m)\in\mathbf{B}_t^0} \alpha(S_{m'}^{(t)})\gamma(S_{m'}^{(t)}, S_m^{(t+1)})\beta(S_m^{(t+1)})}, \end{aligned} \tag{2.38}$$

where $\mathbf{B}_t^i$ is the set of all (m', m) that indicate the state transitions are caused by $d_t = i$, and $\mathbf{B}_t^0$, the set of (m', m), denotes the state transitions are due to $d_t = 0$.

Applying the Log-MAP algorithm to the (2.38), the LLR can be rewritten to

$$\begin{aligned} L_i(d_t) &= \ln \frac{\sum_{(m',m)\in\mathbf{B}_t^i} e^{\bar{\alpha}(S_{m'}^{(t)})+\bar{\gamma}(S_{m'}^{(t)}, S_m^{(t+1)})+\bar{\beta}(S_m^{(t+1)})}}{\sum_{(m',m)\in\mathbf{B}_t^0} e^{\bar{\alpha}(S_{m'}^{(t)})+\bar{\gamma}(S_{m'}^{(t)}, S_m^{(t+1)})+\bar{\beta}(S_m^{(t+1)})}} \\ &= \max{}^*{}_{(m',m)\in\mathbf{B}_t^i}[\bar{\alpha}(S_{m'}^{(t)}) + \bar{\gamma}(S_{m'}^{(t)}, S_m^{(t+1)}) + \bar{\beta}(S_m^{(t+1)})] \\ &- \max{}^*{}_{(m',m)\in\mathbf{B}_t^0}[\bar{\alpha}(S_{m'}^{(t)}) + \bar{\gamma}(S_{m'}^{(t)}, S_m^{(t+1)}) + \bar{\beta}(S_m^{(t+1)})], \end{aligned} \tag{2.39}$$

and the Max-Log-MAP approximation will become

$$\begin{aligned} L_i(d_t) \approx &\max_{(m',m)\in\mathbf{B}_t^i}[\bar{\alpha}(S_{m'}^{(t)}) + \bar{\gamma}(S_{m'}^{(t)}, S_m^{(t+1)}) + \bar{\beta}(S_m^{(t+1)})] \\ &- \max_{(m',m)\in\mathbf{B}_t^0}[\bar{\alpha}(S_{m'}^{(t)}) + \bar{\gamma}(S_{m'}^{(t)}, S_m^{(t+1)}) + \bar{\beta}(S_m^{(t+1)})]. \end{aligned} \tag{2.40}$$

Since the tail-biting is applied on circular trellis diagram, we have equally likely symbols. Thus, the initial condition of branch metrics become

$$\begin{aligned} \bar{\alpha}(S_x^{(0)}) &= 0 \quad \text{for } \forall\ S_x^{(0)} \in \mathbf{S} \\ \bar{\beta}(S_x^{(N)}) &= 0 \quad \text{for } \forall\ S_x^{(N)} \in \mathbf{S}. \end{aligned} \tag{2.41}$$

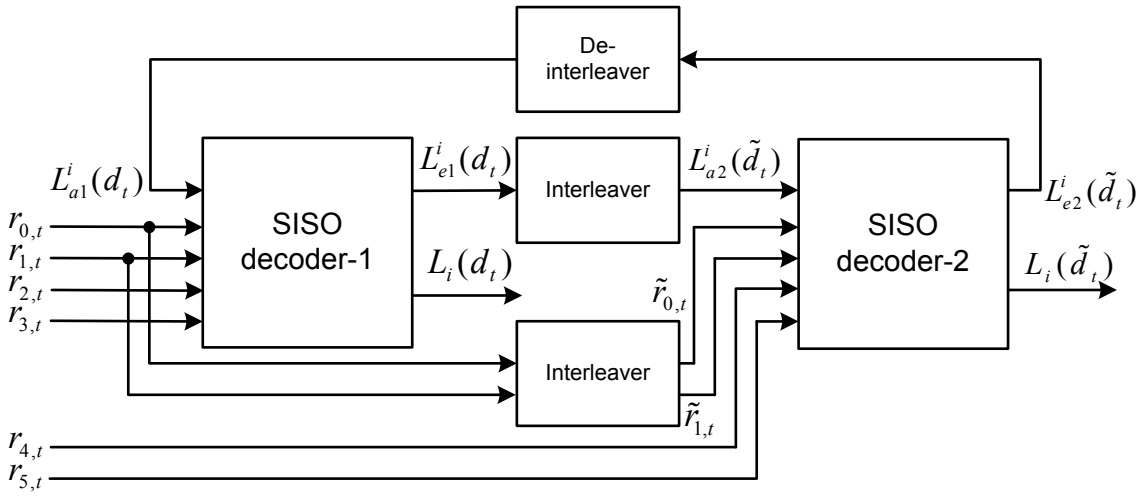


Figure 2.6: Double-binary CTC decoder

For a code rate $1/n$ double-binary RSC encoder, each codeword frame consists of two systematic bits and $2(n-1)$ parity bits. In the receiver, the received codeword has the systematic symbols $r_t^{(0)}$, $r_1^{(1)}$ and the parity symbols $r_t^{(2)} \sim r_t^{(2n-1)}$. Moreover, in order to reduce the computational complexity, to increase throughput, or to reduce the power consumption, we could further simplify the path metrics into

$$\bar{\gamma}(S_{m'}^{(t)}, S_m^{(t+1)}) = L_a^i(d_t) + \sum_{j=0}^{2n-1} b_j \cdot r_t^{(j)}, \tag{2.42}$$

where the value of $b_j \in \{+1, -1\}$ depends on the encoding polynomial after BPSK mapping and can be pre-calculated for all state transitions, respectively. The *a priori* information in (2.42) is represented by

$$L_a^i(d_t) \triangleq \ln \frac{P(d_t = i)}{P(d_t = 0)}. \tag{2.43}$$

From the decoding flow shown in Fig. 2.6, the extrinsic information for next stage can be calculated as

$$L_e^i(d_t) = L_i(d_t) - [(b_0 \cdot r_{0,t} + b_1 \cdot r_{1,t}) - (r_{0,t} + r_{1,t})] - L_a^i(d_t). \tag{2.44}$$

Compute symbol probabilities for the next decoder from previous decoder as

$$L_a^i(d_t) = L_e^i\left(\tilde{d}_t\right) = \ln \frac{P(d_t = i)}{P(d_t = 0)}, \tag{2.45}$$

to save the hardware resource, we can define $\ln P(d_t = 0)$ to 0. Hence, the *a priori* information can be rewritten as follows:

$$\begin{cases} \ln P(d_t = 1) = L_e^1\left(\tilde{d}_t\right) \\ \ln P(d_t = 2) = L_e^2\left(\tilde{d}_t\right) \\ \ln P(d_t = 3) = L_e^3\left(\tilde{d}_t\right) \end{cases} \tag{2.46}$$

Assume the information symbols are equal probability, so we initialize the *a priori* information for the first iteration:

$$\begin{cases} \ln P(d_t = 0) = 0 \\ \ln P(d_t = 1) = 0 \\ \ln P(d_t = 2) = 0 \\ \ln P(d_t = 3) = 0 \end{cases} \tag{2.47}$$

The double-binary 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 final decisions are made according to:

$$\tilde{d}_t = \begin{cases} 01, & \text{if } L\left(\tilde{d}_t\right) = L_1\left(\tilde{d}_t\right) > 0 \\ 10, & \text{if } L\left(\tilde{d}_t\right) = L_2\left(\tilde{d}_t\right) > 0 \\ 11, & \text{if } L\left(\tilde{d}_t\right) = L_3\left(\tilde{d}_t\right) > 0 \\ 00, & \text{else} \end{cases} \tag{2.48}$$

where

$$L\left(\tilde{d}_t\right) = \max\left(L_1\left(\tilde{d}_t\right), L_2\left(\tilde{d}_t\right), L_3\left(\tilde{d}_t\right)\right) \tag{2.49}$$

2.4 Stochastic Iterative Decoding Algorithm

Stochastic arithmetic was first introduced in 1960's as a method to design low-precision digital circuits [20]. Due to the hardware implementations of iterative decoding for error control code become more complex, much research effort has been invested to reduce hardware complexity. The major motivation for considering stochastic computation was the possibility of performing complex computations using only simple logic circuit. In stochastic computation, probabilities are represented as streams of random digital bits using Bernoulli sequences. With this representation, complex operations on probabilities

such as multiplication and division can be converted to operations on bits which can easily be implemented using simple stochastic gates, but to trade off between computation accuracy and computation time.

2.4.1 Stochastic Computation

In a stochastic computation, values are encoded as a Bernoulli sequence of bits. For an unsigned number N, the probability that any bit d_i in the Bernoulli sequence is a binary 1 is

$$P\left(d_i = 1\right) = \frac{N}{N_{\max}}, \tag{2.50}$$

for a probability value P_{in}, the probability of i-th bit d_i being a binary 1 is

$$P\left(d_i = 1\right) = P_{in}. \tag{2.51}$$

From the above equation, we can realize that the information is contained in the statistics of the bit stream, and there is no fixed mapping between probability value and encoded sequence. And the precision can be decided by the length of stochastic sequence, so we can increase the precision of stochastic streams by increasing the sequence length.

Consider the stochastic multiplier in Fig. 2.7. Let $P_a = \Pr\left(a_i = 1\right)$ and $P_b = \Pr\left(b_i = 1\right)$ be the input probability, and P_c is the output probability. The multiplication of two stochastic sequences can be performed with a single two-input AND gate [21].

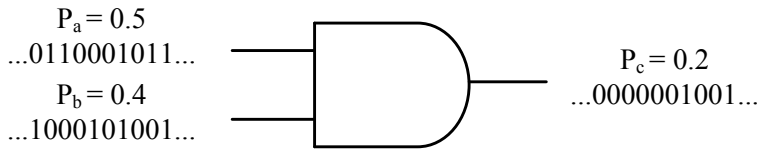


Figure 2.7: Multiplication of two stochastic sequences

The JK flip-flop shown in Fig. 2.8 can be used to perform stochastic division. The probability of random output transition from 0 to 1 and from 1 to 0 is $((1 - P_c)P_a)$ and (P_cP_b), respectively. Since the expected occurrence of random sequence in both direction must be equal, then we have

$$P_cP_b = \left(1 - P_c\right)P_a \rightarrow P_c = P_a/(P_a + P_b) \tag{2.52}$$

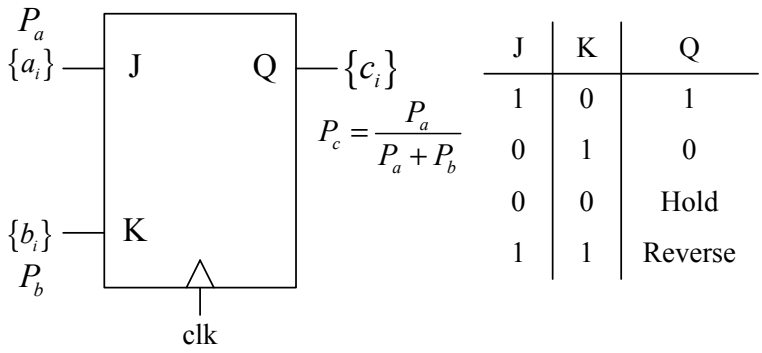


Figure 2.8: Division of two stochastic sequences

For the stochastic addition and subtraction, in order to ensure the operations to be closed on the probability interval of $[0, 1]$, therefore, these operations should be combined with a scaling operation for the outcome [21]. Addition with scaling is performed as

$$P_c = \sum_{i=1}^{N} S_i P_{Ai} \quad \text{where} \quad \sum_{i=1}^{N} S_i = 1, \tag{2.53}$$

The outcome is the scaled sum of the input probabilities. For $S_i = 1/N$, this operation can be implemented in hardware using a multiplexer as shown in Fig. 2.9, where RS refers to the random selection supplied by (pseudo) random number generators. Generating RS is straightforward when the N is a power of two.

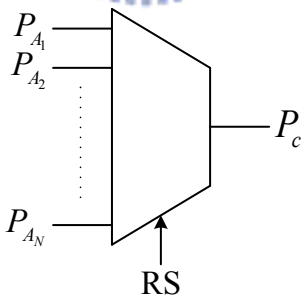


Figure 2.9: Stochastic (scaled) addition

2.4.2 Stochastic Stream Generation

For the implementation of the trellis-based stochastic decoder, the channel value y_i received from AWGN channel should be converted to LLR first

$$\begin{aligned} L_i &= \ln \frac{P(y_i = 1)}{P(y_i = 0)} \\ &= \ln(e^{-\frac{1}{2\sigma^2}((y_i+1)^2-(y_i-1)^2)}) \\ &= \frac{-1}{2\sigma^2}(4y_i) \\ &= \frac{-1}{N_0}(4y_i) \end{aligned} \tag{2.54}$$

The further conversion form LLR value in (2.54) to probability is shown below

$$\begin{aligned} P_i &= \frac{P(y_i = 1)}{P(y_i = 1) + P(y_i = 0)} \\ &= \frac{1}{1 + \frac{P(y_i=0)}{P(y_i=1)}} \\ &= \frac{1}{1 + e^{-L_i}} \end{aligned} \tag{2.55}$$

Assume we use N-bit representation for the received probabilities, these probabilities are converted to stochastic streams by using the structure shown in Fig. 2.10. This structure consists of a comparator which compares the channel probability, P, with a (pseudo) random number, R. The channel probability P is fixed during the decoding process, but R is a random number (with a uniform distribution) which is updated in every decoding cycle. The output bit of the comparator is equal to 1 if $P > R$, else it is equal to 0. Since R has a uniform distribution and can take a value from 0 to $2^N - 1$, each bit in the output stochastic stream is equal to 1 with a probability of $\frac{P}{2^N}$ [22].

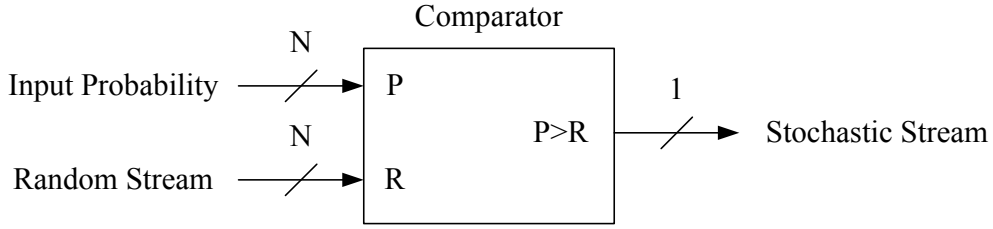


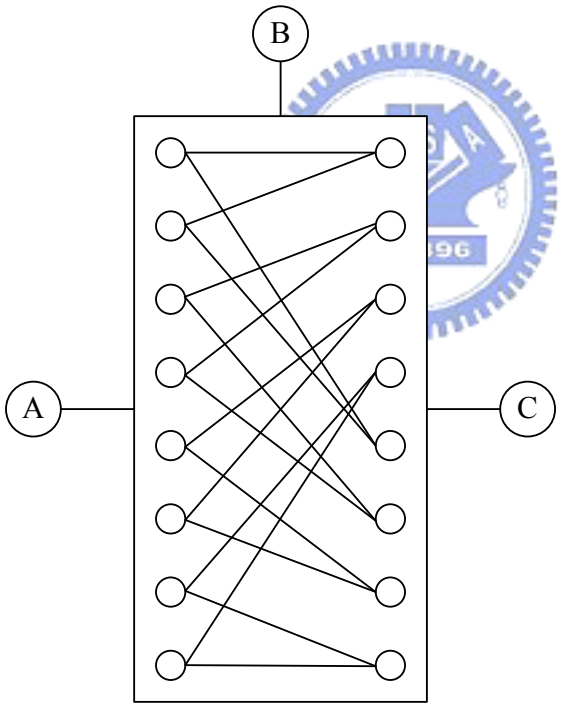
Figure 2.10: Converting channel probabilities to stochastic streams

2.4.3 Trellis-based Stochastic Decoding Algorithm

The stochastic decoding algorithm is a message-passing algorithm, which is based on the code constraint graph. To implement the stochastic message-passing algorithm, we use the following deterministic message update rule at each function node [23]. Consider the propagation of message from $(A_i(T), B_i(T))$ to $C_{i+1}(T+1)$ with $i = 1 \sim N$, where $A_i(T)$ and $B_i(T)$ are received messages, $C_{i+1}(T+1)$ is transmitted message, and N is the block length. Assume at time instance T, $A_i(T) = a$ and $B_i(T) = b$. Then

$$C_{i+1}(T+1) = \begin{cases} f_C(a,b) & \text{if } (a,b) \in \mathbf{S} \\ C_{i+1}(T) & \text{otherwise} \end{cases} \tag{2.56}$$

It is sometimes convenient to refer to the set **S** as the satisfaction of the constraint function f_C. For each row (a, b, c) in the satisfaction table in Fig. 2.11, there is a branch in the trellis which connects a with c, and which is labeled b. This relationship of a $(2, 1, 3)$ convolutional code is illustrated by Fig. 2.11.



(a)

a	b	c
0	0	0
0	3	4
1	3	0
1	0	4
2	2	1
2	1	5
3	1	1
3	2	5
4	3	2
4	0	6
5	0	2
5	3	6
6	1	3
6	2	7
7	2	3
7	1	7

(b)

Figure 2.11: An example of constrain node (a), showing a detailed trellis description of its constraint; and (b) the set S corresponding to this constraint

Apply the message update rule to stochastic decoding algorithm, we can define the

state transition parameter in (2.56) as follows:

- a : Source state
- b : Branch codeword
- $f_C(a, b)$: Destination state

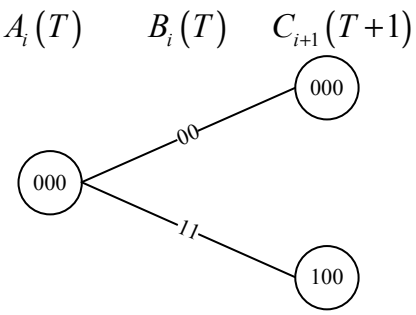


Figure 2.12: Update rule for stochastic decoding algorithm

A simple trellis diagram for a $(2, 1, 3)$ convolutional code is shown in Fig. 2.12, and the corresponding value sets of state transition are defined as follows:

$$\begin{aligned} \mathbf{A} &= \{000, 001, 010, \ldots, 111\} \\ \mathbf{B} &= \{(00), (01), (10), (11)\} \\ \mathbf{C} &= \{000, 001, 010, \ldots, 111\} \end{aligned} \tag{2.57}$$

Furthermore, assume at time instance T, the branch metrics become

$$B_i(T) = \{B_{i,0}(T), B_{i,1}(T), \ldots, B_{i,N-1}(T)\}, \tag{2.58}$$

where

$$\begin{aligned} i &= 0 \sim (\text{Block Length} - 1) \\ T &= 0 \sim (\text{Decoding Cycle} - 1) \end{aligned} \tag{2.59}$$

Referring to (2.56), there are three possible value for the destination state in Fig. 2.12

$$C_{i+1}(T+1) = \begin{cases} f_C(000, (00)) = 000 \\ f_C(000, (11)) = 100 \\ \quad C_{i+1}(T), \end{cases} \tag{2.60}$$

we can find that if the received codeword $(B_{i,0}, B_{i,1})$ (ignore the time instance T in notation for simplicity) is 00, the destination state C_{i+1} is 0; if the codeword is 11, the destination is 4, else the destination state would be remain the same state as last decoding cycle.

By using the update rule, the trellis-based stochastic decoder with matched codeword can be implemented in Fig. 2.13(a). We can find that branch metric $(B_{i,0}, B_{i,1})$ is stochastic stream which is generated by a comparator with input probability and random stream. With the matched codeword 11, the initial state is updated from initial state to state 3. At the same time, we increase the counter because of the transmitted bit of matched branch is "1", if the transmitted bit is "0", we decrease the counter. The main function of this counter is to make the final decision to convert stochastic stream to digital bit.

Furthermore, if there doesn't have any matched codeword as shown in Fig. 2.13(b), the destination state will remain the same state as the previous decoding cycle. Under this condition, the counter will remain the same value. When the maximum decoding cycle is reached, the counter is exported for the hard decision.

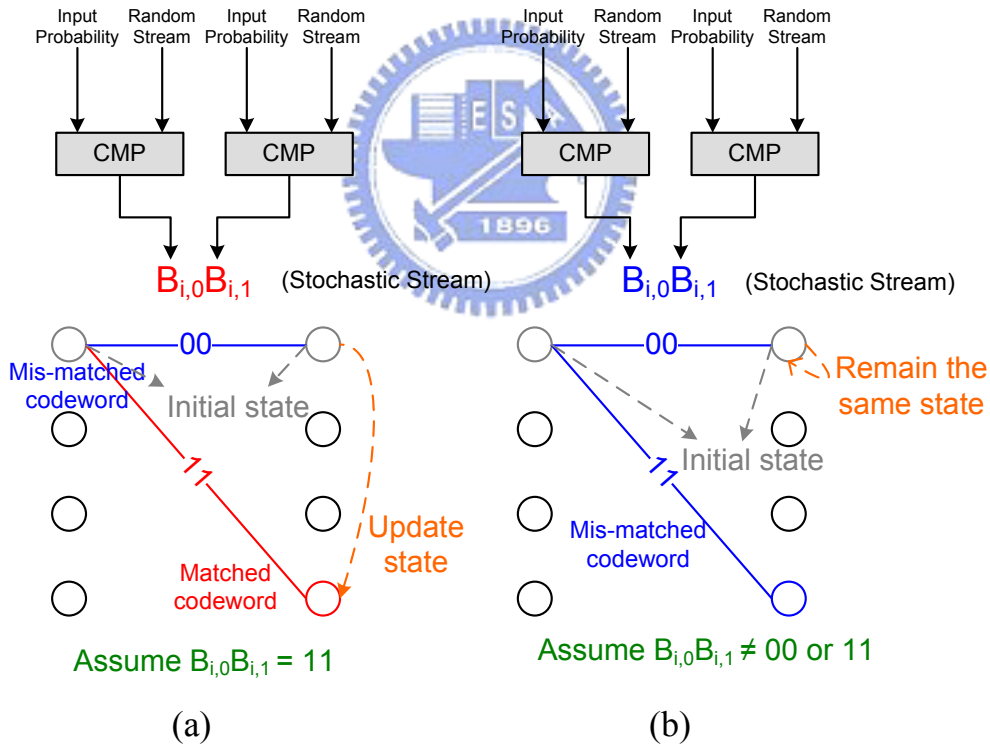


Figure 2.13: Trellis-based stochastic decoder. (a) With matched codeword. (b) With mis-matched codeword

Chapter 3

Trellis-based Stochastic Decoder

3.1 Analysis of Stochastic Update Rule

As we described in section 2.4, the error-correcting performance of trellis-based stochastic decoder can be adjusted by decoding cycles. Fig. 3.1 shows the performance comparison of the uncoded sequence, hard-decision Viterbi decoding algorithm, soft-decision Viterbi decoding algorithm and the trellis-based stochastic decoding algorithm with decoding cycle 2500. All of the simulation environment is under AWGN channel and BPSK mapping using $(2, 1, 3)$ convolutional code, and the random stream in Fig. 2.13 is generated by random number function in C++ programming. Besides the performance curve of stochastic decoder is fixed point simulation with quantization width 10, the other performance curves are floating point simulation.

From the simulation result, we can find that the performance of the stochastic decoder is even worse than the uncoded sequence. The reason might be the received Log-Likelihood Ratios (LLRs) become so large so that the corresponding probabilities approach "0" (or "1"). In this case, bits in stochastic sequences are mostly "0" (or "1"), hence random switching events become too rare for proper decoding [24]. In the following section, we will discuss some methods to improve the performance of trellis-based stochastic decoder.

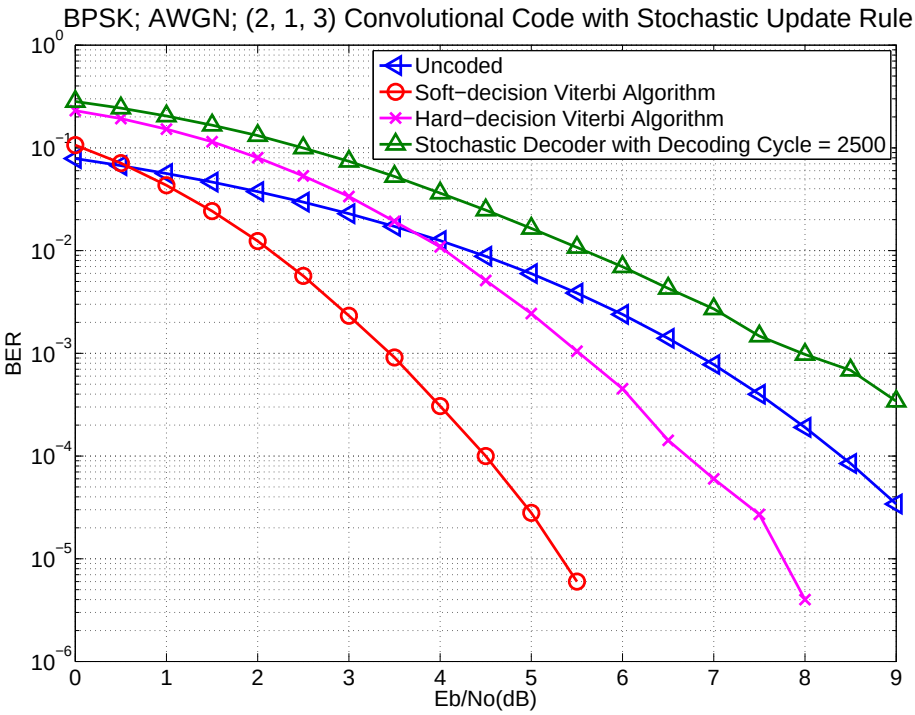


Figure 3.1: Performance of stochastic decoder

3.2 State Memory Method

One major difficulty observed in trellis-based stochastic decoding algorithm is the latching problem. The latching problem refers to the case where a cycle in the trellis diagram causes the state transition to lock into a fixed state. The mis-matched codeword caused the trellis diagram to remain the same state is the main reason of the latching problem.

To avoid the state transition to lock into a fixed state during decoding cycle and increase the random switching activity of stochastic sequences in the trellis diagram during latching, we propose the state memory to store the state with matched codeword transition. As shown in Fig. 3.2, the state memory store the unpdate state with matched codeword and increase the vlaid counter to record the memory index when the decoding cycle is bigger than T_{init} which is a parameter to reduce the chance of locking into a fixed state.

Fig. 3.3 shows the operation condition of state memory when the mis-matched codeword occurs in trellis diagram. To increase the sensitivity to the bit transition for proper decoding operation, in case of the updating rule is failed (codeword mis-matched), we

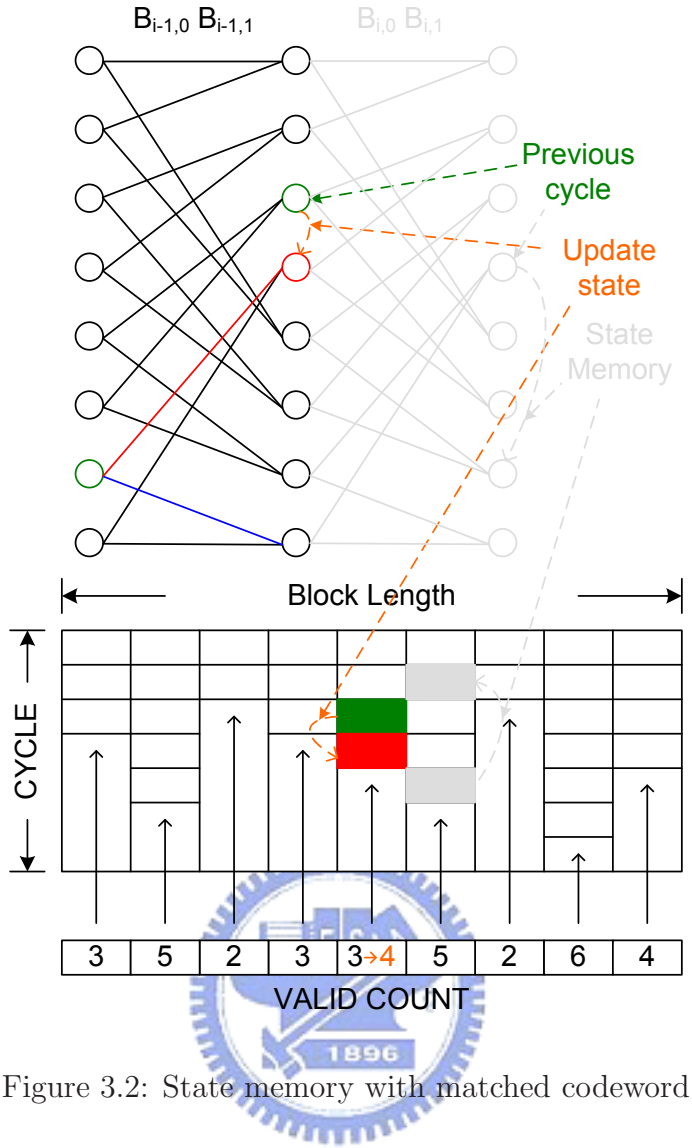


Figure 3.2: State memory with matched codeword

randomly select state from the state memory to update trellis diagram. This updating scheme reduces the chance of locking into a fixed state since every time the mis-matched codeword happens, the state is randomly chosen from those previous update sates which are not produced into latching problem.

To increase the state transition activity, the usage of state memory would be applied to both forward and backward transition in trellis diagram (just like the forward metric and backward metric in $BCJR$ decoding algorithm). Double-side state memory application is shown in Fig. 3.4, the forward transition and backward transition update state every decoding cycle simultaneously. The decisions of the forward and backward transition are also calculated in each decoding cycle. When the maximum decoding cycle is reached, the counter is exported for the hard decision.

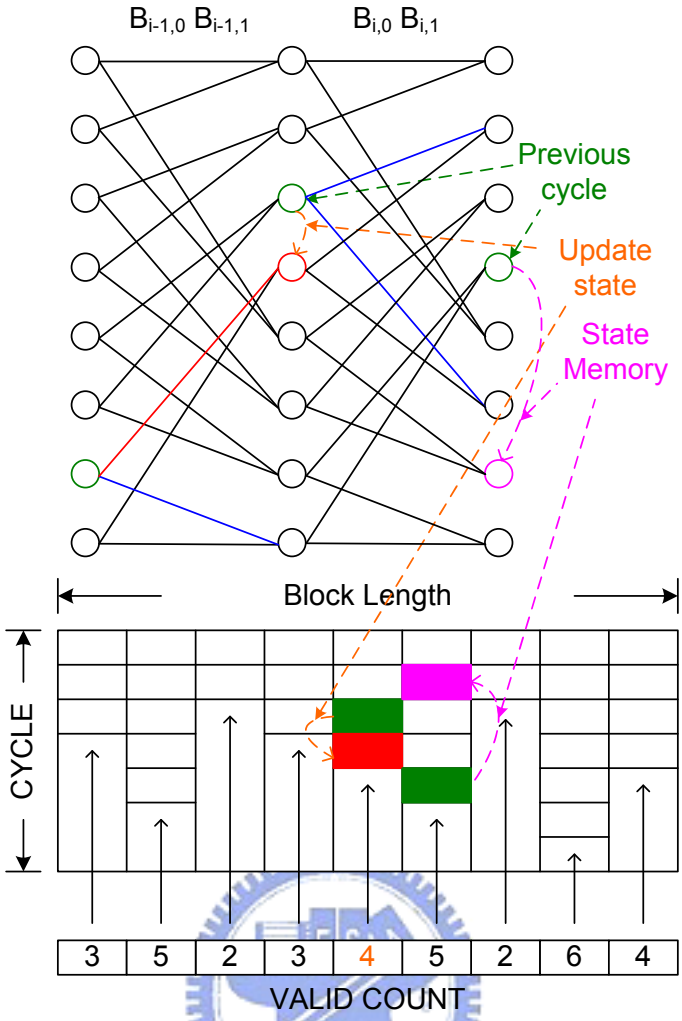


Figure 3.3: State memory with mis-matched codeword

The simulation result and performance comparison is show in Fig. 3.5, the simulation environment is the same as Fig. 3.1. As shown, the stochastic decoder with state memory improves the performance comparing to that without state meory and provides better performance at low SNRs with respect to hard-decision Viterbi decoding algorithm. But at high SNRs, the BER curve of stochastic decoder with state memory has error floor and the performance is worse than the hard-decision Viterbi decoding algorithm. In the next section, we will discuss this condition and introduce further improvement to solve this problem.

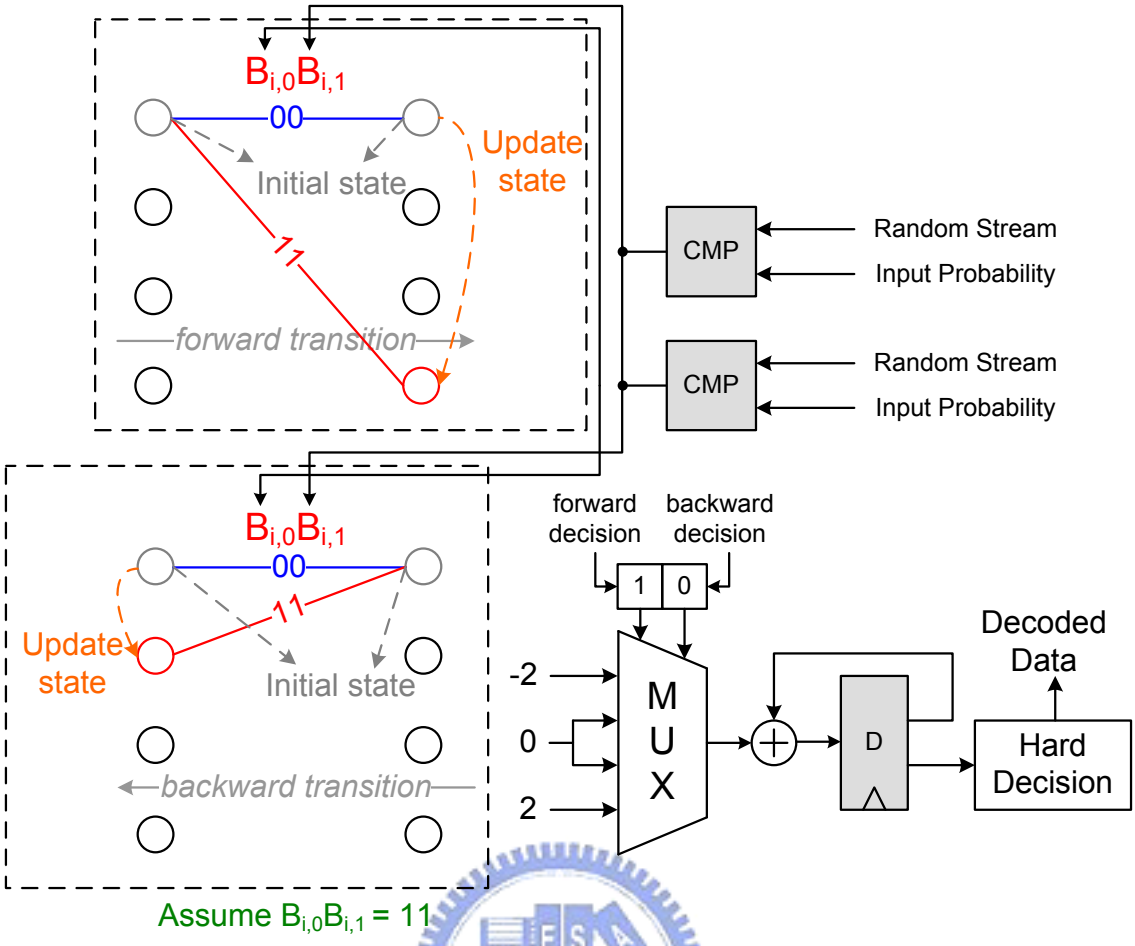


Figure 3.4: Double-side state memory application

3.3 Noise Dependent Scaling Factor

The Noise Dependant Scaling (NDS) factor proposed in [24] introduced that the received channel LLRs are down-scaled by a scaling factor which is proportional to the SNR. The down-scaled LLRs result in probabilities which introduce more switching activity in the stochastic decoder. Because the scaling factor is proportional to the noise level, it ensures a similar level of switching activity for different SNRs. Assuming a BPSK transmission over AWGN channel, the original LLR (L_i) is

$$L_i = \frac{-1}{N_0}(4y_i), \tag{3.1}$$

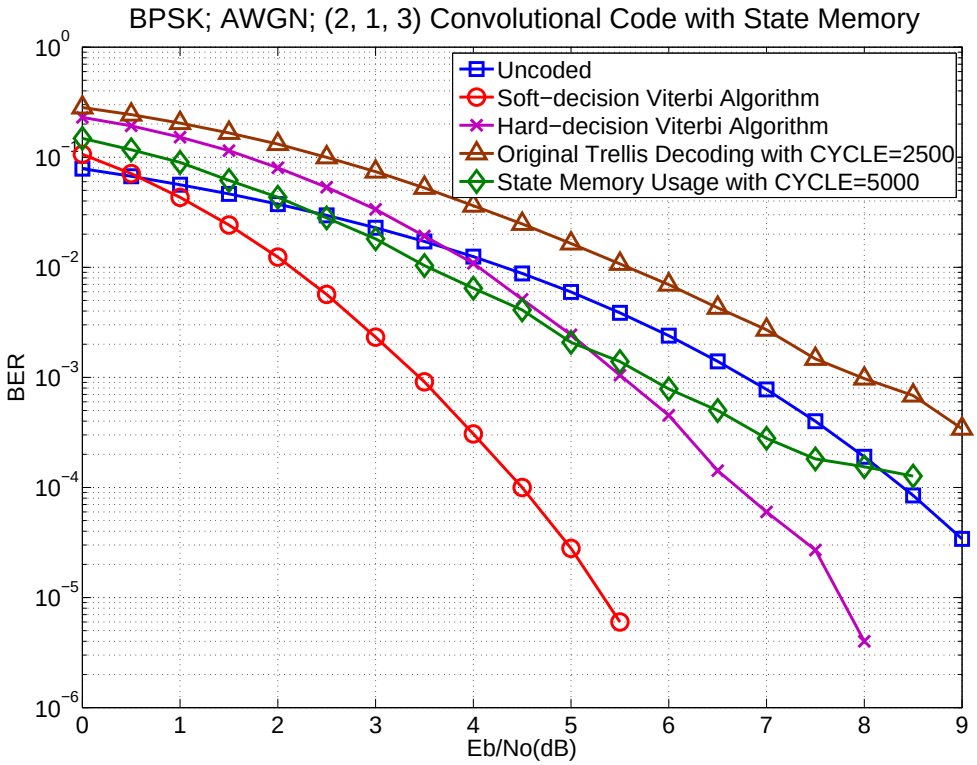


Figure 3.5: Stochastic decoder with state memory usage

where y_i is received symbol and N_0 is the single-sided noise power spectral density. The scaled LLR is calculated as

$$\begin{aligned} L_i' &= \left(\frac{\alpha N_0}{Y}\right) L_i \\ &= \left(\frac{\alpha N_0}{Y}\right) \frac{-1}{N_0}(4y_i) \\ &= \left(\frac{\alpha}{Y}\right)(-4y_i), \end{aligned} \tag{3.2}$$

Y is the fixed maximum value of the received symbols, and α is a constant factor. As a result, $\frac{\alpha}{Y}$ is the noise dependent scaling factor. Fig. 3.6 shows the peroformance of different NDS factors with state memory usage, the NDS factor 0.6 is the best case compared to others and without error floor effect when BER is 10^{-5}.

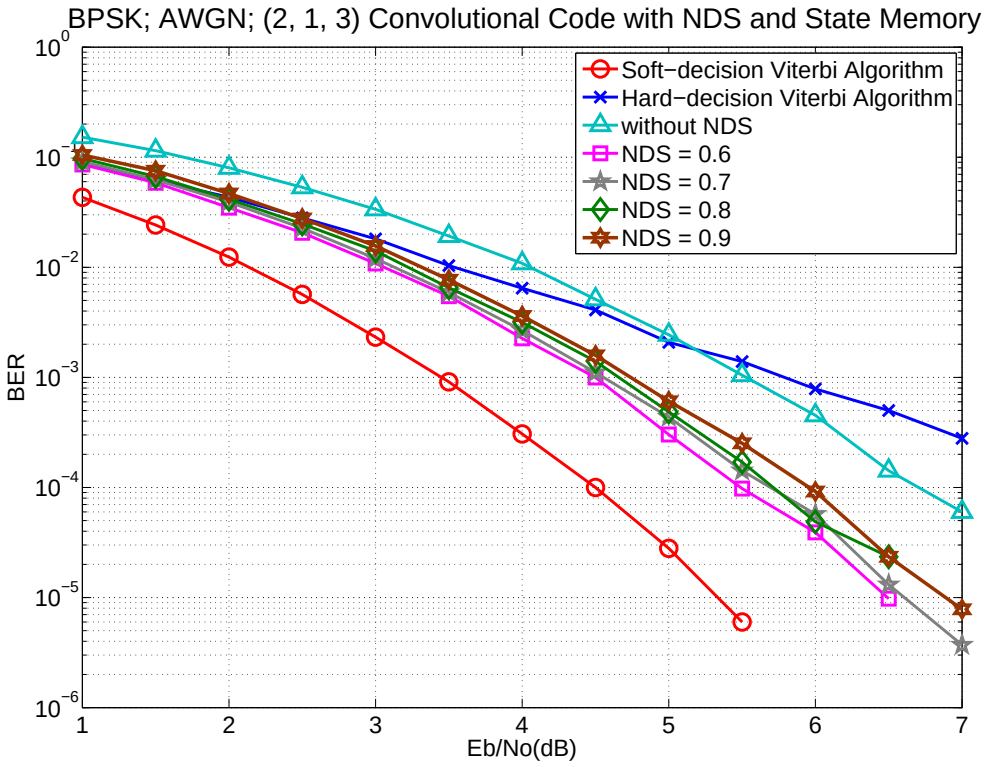


Figure 3.6: NDS factor comparison

3.4 Discussion

Further improvement with state memory and noise dependent scaling factor 0.6 applied to trellis-based stochastic decoder really enhance the possibilty to implement the stochastic decoding algorithm. Besides, the number of decoding cycles also affect the error-correcting performance and throughput of stochastic decoder, the analysis is shown in Fig. 3.7. Based on different SNRs (from 0 to 7), the decoding cycle 2500 seems to be enough to have outstanding error-correcting performance.

Fig. 3.8 shows the 1-stage stochastic decoder architecture, and the corresponding codeword and the destination state can be referred to LUT_B and LUT_C, respectively. Shorter critical path compared to conventional ACS-unit is also labeled in Fig. 3.8. The synthesis result shows that the 1-stage trellis-based stochastic decoder can be operated at 1.8GHz by using UMC 90nm CMOS process. Although the throughput can be enhanced by using stochastic decoding algorithm, there still has some dis-advantage for implementation. First, performance loss about 1dB at BER= 10^{-5} as compared with soft-decision Viterbi

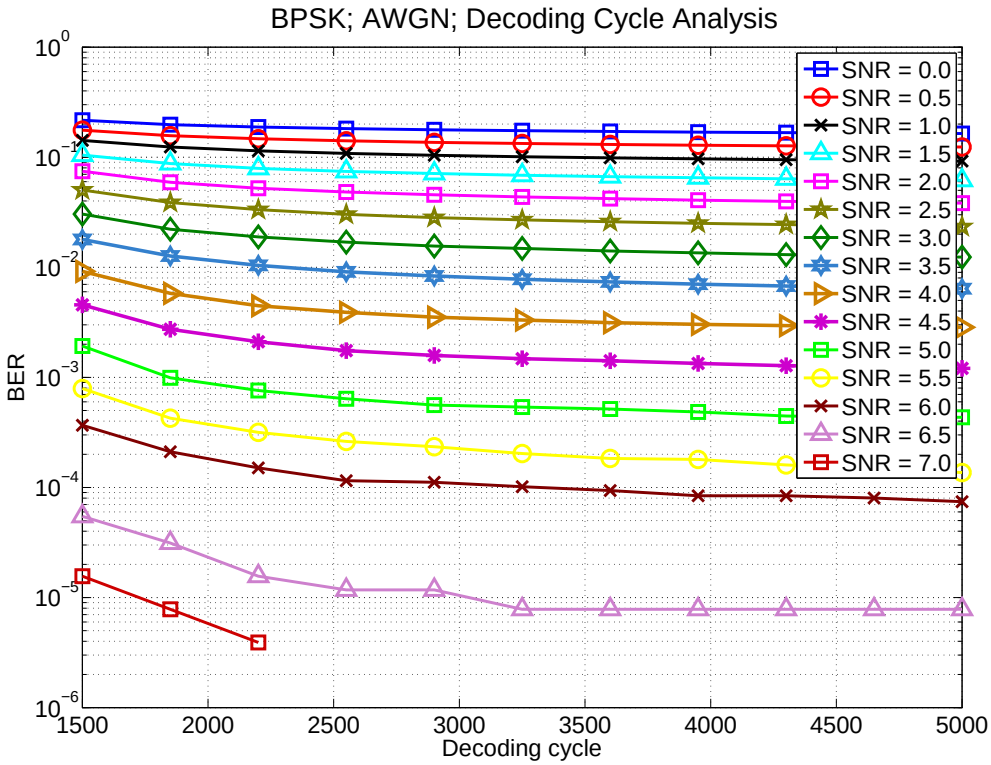


Figure 3.7: Decoding cycle analysis of stochastic decoder

algorithm. Second, the required functional units will be propotional to block length which may result in larger hardware cost about 187K gate counts. As a result of area-efficient design for WiMAX standard, the original decoding algorithm can achieve design requirement and would be adopted for hardware implementation. Furthermore, the stochastic decoding algorithm can be applied to the requirement of high throughput standard, such as IEEE 802.16m.

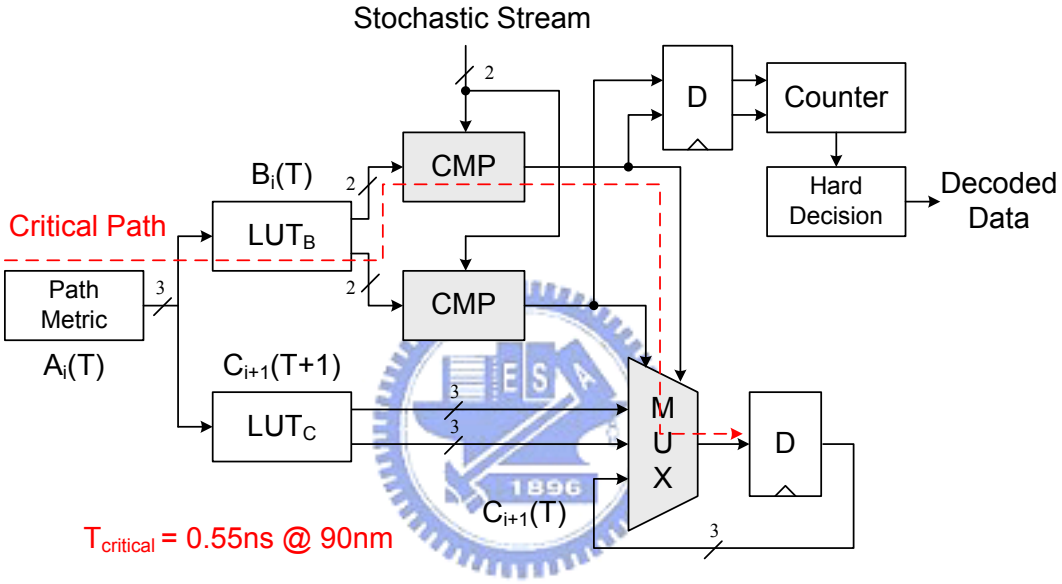


Figure 3.8: 1-stage trellis-based stochastic decoder architecture

Chapter 4

Double-binary CTC Decoder for WiMAX 802.16e Application

4.1 Introduction of WiMAX 802.16e Standard

In WiMAX 802.16e, channel coding considers convolutional turbo codes (CTC) as an optional code. It uses double-binary turbo codes to improve error correcting performance and decoding throughput. Besides double binary turbo code, 802.16e provides 17 modes to support different block sizes. In order to support various block sizes, the interleaver is designed as a function with five parameters.

Because the different parameters for different modes are challenges of hardware implementation, how to minimize a configurable parameter controller architecture is the main concern in this chapter. Moreover, since the block length ranges from 24 to 2400, the memory requirement is also a great issue.

Fig. 4.1 illustrates the turbo encoder block diagram [1]. It consists of two circular recursive encoders, and the code rate of CTC encoder is 1/3. Each encoder generates two additional parity bits using two information bits. The polynomials defining the connections and symbol notations are described as follows:

- For the feedback branch: $1 + D + D^3$
- For the Y parity bit: $1 + D^2 + D^3$
- For the W parity bit: $1 + D^3$

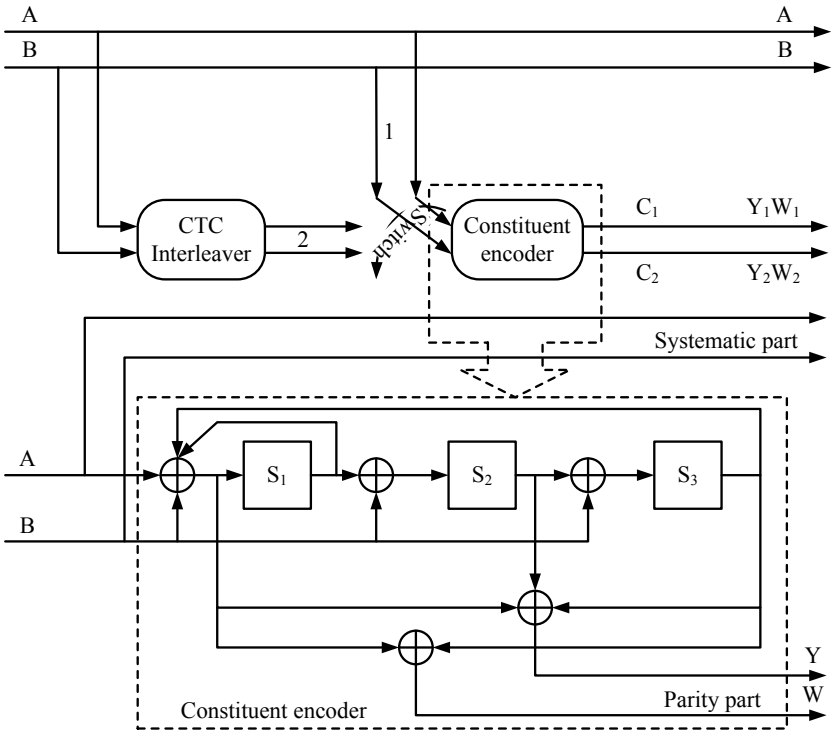


Figure 4.1: CTC encoder for WiMAX standard

The trellis diagram generated by the circular encoder is shown in Fig. 4.2. Each state receives four branch metrics (information symbol d_t= 00,01,10,11) and also sends four messages to other states. As a result, radix-4 ACS unit is required to decode the trellis diagram.

The state of the encoder is denoted S $(0 \leq S \leq 7)$ with S the value read binary (left to right) out of the constituent encoder memory (referring to Fig. 4.1). The circulation states S_c in (2.36) is determined by the following operations:

Step 1: Initialize the encoder with state 0. Encode the sequence in the natural order for the determination of S_c. Assume the final state of the encoder is $S_{0_{N-1}}$.

Step 2: According to the length N of the sequence, use Table 4.1 to find S_c.

In 802.16e, five parameters, including the block length N, P_0, P_1, P_2 and P_3 are specified in Table A.1 and Table A.2, and the parameters when supporting H-ARQ are specified in Table A.3. The interleaver address in CTC is shown as Table 4.2, where j is the index of memory address for MAP decoder 2 (for decoding interleaved data) and $P(j)$ is the index of memory address from MAP decoder 1 (for decoding deinterleaved data). The most important operation in this table is the *modulo* operation. It requires a

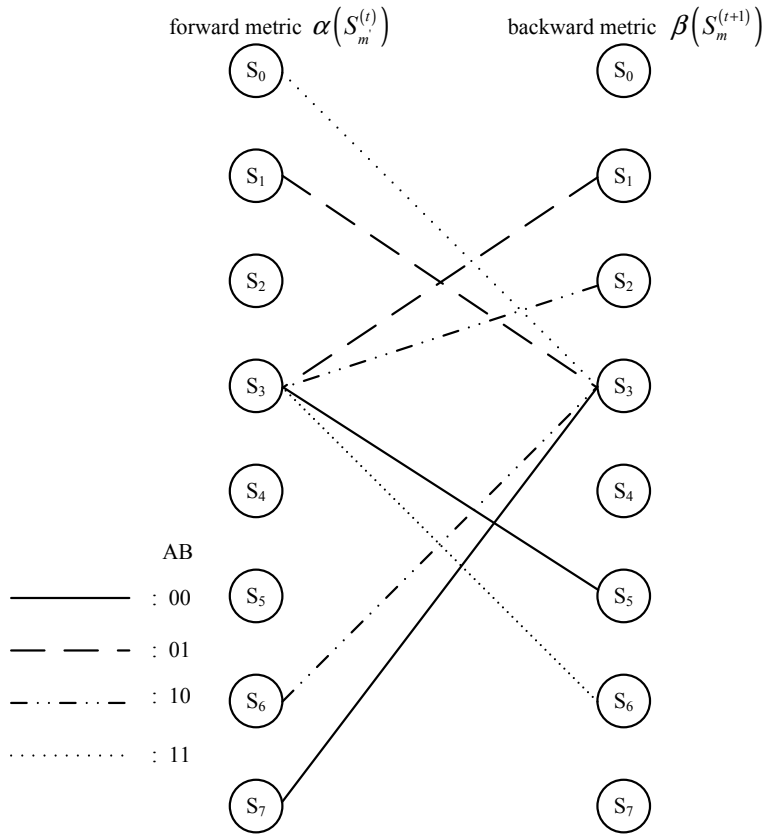


Figure 4.2: Trellis diagram of double-binary CTC

divider which occupies large area and increases the delay of critical path.

4.2 Simulation Analysis and Parameter Decision

Based on the double binary CTC decoding algorithm in section 2.3, the simulation result can be discussed in this section. In order to determine appropriate design parameters such as the bit widths of the path metric, branch metric, and the input symbol, the performance evaluation through simulations are necessary. In turbo decoding process, the iteration number and the sliding window size will directly influence not only the performance of turbo decoding but also the memory requirement of the design. The bit error rate (BER) curves of the floating point decoders under QPSK modulation and AWGN channel with block length of 2400 are presented in Fig. 4.3. In Fig. 4.3, we can realize that at the same iteration number 5, there is a 0.6dB loss between the sliding window size of 5 and 12 when the BER is 10^{-5}. However, the performance curves between the sliding window size 12

Table 4.1: Circulation state lookup table (S_c)

N_{mod_7}	$S_{0_{N-1}}$							
	0	1	2	3	4	5	6	7
1	0	6	4	2	7	1	3	5
2	0	3	7	4	5	6	2	1
3	0	5	3	6	2	7	1	4
4	0	4	1	5	6	2	7	3
5	0	2	5	7	1	3	4	6
6	0	7	6	1	3	4	5	2

Table 4.2: Interleaver Function

$for\ j = 1\ to\ N-1$

$Case(j\ mod\ 4)$

$Case0 : P(j) = (P_0 \times j + 1)\ mod\ N$

$Case1 : P(j) = (P_0 \times j + 1 + N/2 + P_1)\ mod\ N$

$Case2 : P(j) = (P_0 \times j + 1 + P_2)\ mod\ N$

$Case3 : P(j) = (P_0 \times j + 1 + N/2 + P_3)\ mod\ N$

and 20 are almost the same. Also, we compare the different iteration number at the same sliding window size 12, there is a 0.6dB loss between the iteration number 5 and 3 when the BER is 10^{-5}.

Although Max-Log MAP decoding algorithm introduced in section 2.1.3 can reduce the decoding complexity, it invokes the performance loss due to the approximation of max function. The approximation usually overestimates the value of messages. In order to compensate the performance loss, we introduce a scaling factor to scale down the extrinsic message. Therefore, the intrinsic information $L_a^i(d_t)$ can be formulated as follow:

$$L_a^i(d_t) = \beta \times L_e^i\left(\tilde{d}_t\right), \tag{4.1}$$

where β is the scaling factor. From Fig. 4.4 we can figure out that if the normalization factor is 0.75 in Max-Log MAP algorithm with block length of 2400, the performance has only less than 0.1dB loss and has more than 0.3dB performance gain from Max-Log MAP algorithm which will be very close to Log-MAP algorithm and this step would not cost a

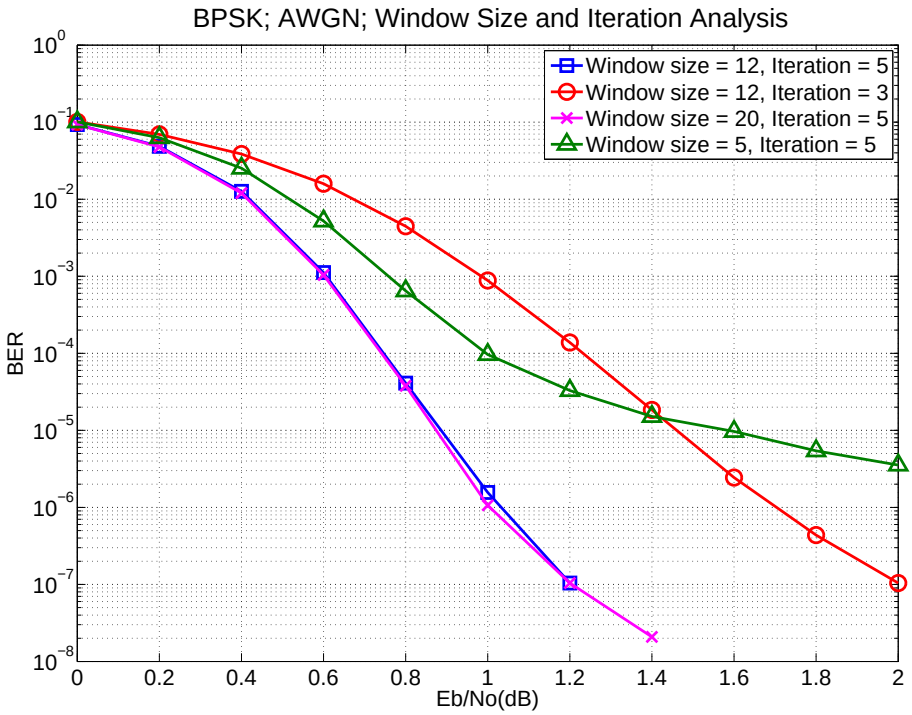


Figure 4.3: Comparison of iteration number and window size

lot of hardware area.

The fixed point representation of the internal variable in the MAP decoder is determined from the received symbol quantization. Fig. 4.5 shows the simulation result with the different input symbol quantization under QPSK modulation and AWGN channel, the block length of turbo decoder is 2400, the sliding window size is 12, and the iteration number is 5. Note that $(a.b)$ shown in the figure denotes the quantization scheme where a is the number of bits used in for the integer part a, and b is the number of bits used for the fractional part. Simulation result shows that performance of input symbol [4.2], intrinsic information [5.2], bit width of metrics 10 is the recommended for the double-binary CTC decoder which is close to the floating point Max-log MAP algorithm and we summarize the fixed representation in Table 4.3.

Table 4.3: Summary of fixed representation in turbo decoding

Quantities	Input symbols	Intrinsic information	Branch metrics	Path metrics
Width	6	7	10	10

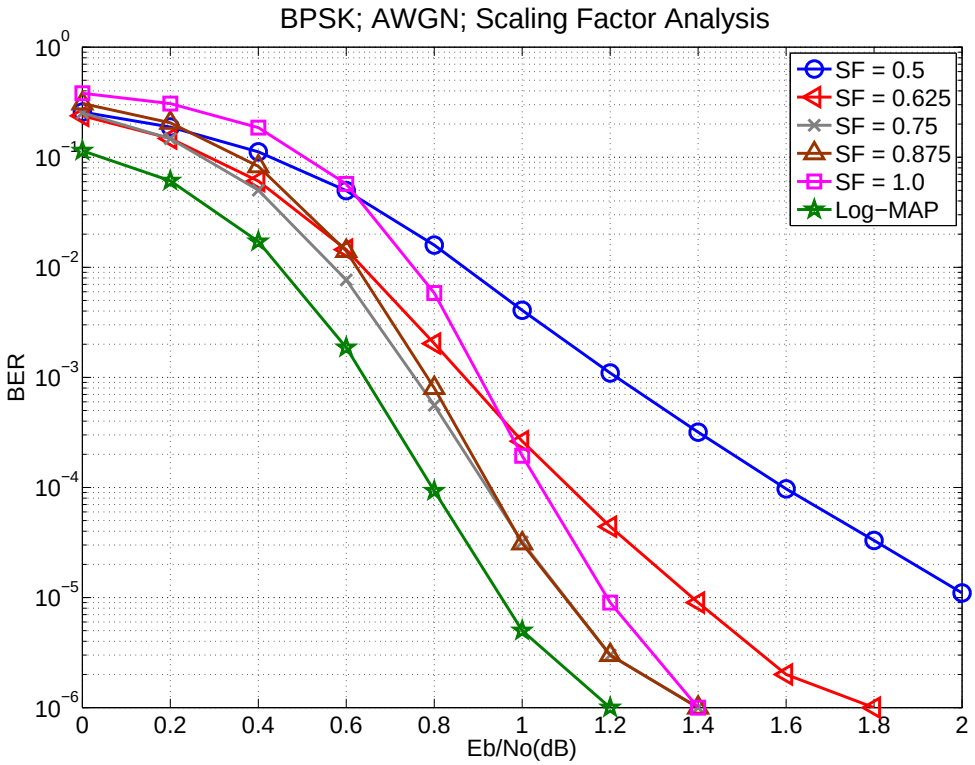


Figure 4.4: Scaling factor comparison

4.3 Proposed Architecture of WiMAX CTC Decoder

The block diagram of proposed architecture is illustrated in Fig. 4.6. There are four memory blocks for message storage, where store input information and extrinsic information generated by the SISO MAP decoder. The Finite-State-Machine (FSM) controls the iterative decoding procedure and decides which state is proceeding. Furthermore, two interleaver units are used to generate the read address (from MEM_EXT to MAP decoder) and the write address (from MAP decoder to MEM_EXT). By means of MEM ADDR control unit, the memory addresses are generated to store or access data.

4.3.1 MAP Decoder

Fig. 4.7 shows the architecture of MAP decoder, which consists of branch metric unit (BMU), add-compare-select (ACS) unit, log-likelihood-ratio (LLR) unit, and buffers. The BMUs compute branch metrics for ACS-α, ACS-β, and ACS-β_d and each ACS unit per-

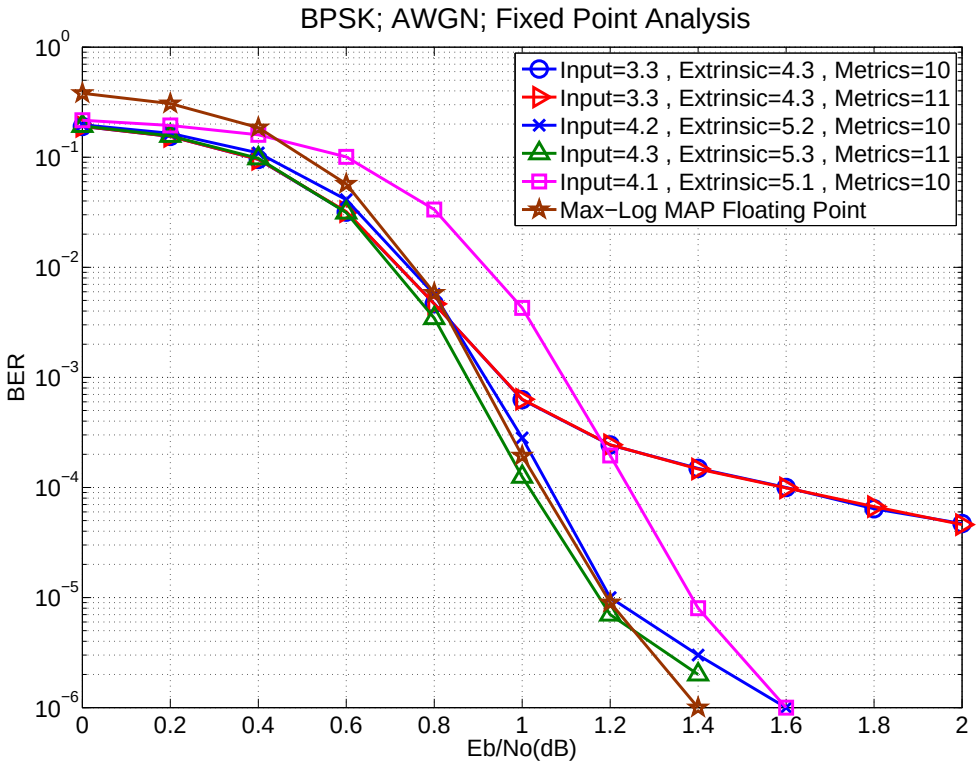


Figure 4.5: Fixed point comparison

forms Add-Compare-Select operation. ACS-α carries out the forward recursion and saves the results in the α-memory. ACS-β starts backward recursion from the initial conditions determined by the ACS-β_d previously. At the same time, the LLR calculator determines $L_i(d_t)$ and $L_e^i(d_t)$. Buffer units reorder the input sequence within one sliding window size, and the α buffer is a Last-In/First-Out (LIFO) buffer used to reorder each state of α value for LLR caculation.

To consider the sliding window approach in Fig. 2.2, the backward metrics β evaluation can be started when the required window of data have been stored. However, if we reverse the order of input sequence within a window size, the input buffer of the β_d calculation can be saved [25]. Fig. 4.8 is the timing flow of MAP decoder. In order to eliminate the IBUF for β_d-ACS, the input order of MAP decoding is from the end of the sliding window to the beginning of the window. After getting the branch metric, α and β, the LLR calculation can be finished without write-after-read (WAR) data hazard. As a result, the latency of MAP decoder is three times sliding window size.

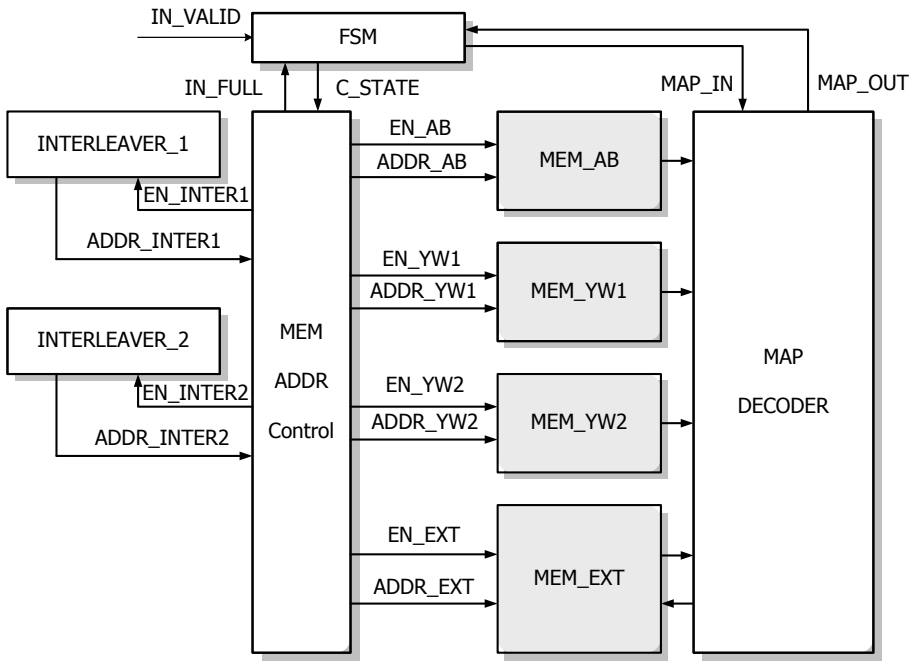


Figure 4.6: CTC decoder block diagram

4.3.2 Pseudo Two-port Register File

In order to increase the decoding speed and reduce the size of memory, pseudo two-port register file is used to read and write memory in one cycle. In Fig. 4.9(a), extrinsic memory operates at double clock rate. Write-address (from interleaver 2) and read-address (from interleaver 1) are generated at the original clock rate. A multiplexer selects correct address according to wheather the operation is read or write. Fig. 4.9(b) illustrates the timing diagram of read & write operation. As a result, we can eliminate one $2400 \times 21 = 50400$ bits (2400 is block length and 21 is total bits of extrinsic data) extrinsic memory such that 26.9% memory usage (original: $50400 + 50400 + 86400 = 187200$ bits) is saved. In the same way, the buffers in MAP decoder in Fig. 4.7 are also pseudo two-port register files to read & write in one cycle as shown in Fig. 4.9(c). The different between Fig. 4.9(b) and Fig. 4.9(c) is that Fig. 4.9(c) read & write at the same address in one cycle. By using this method, the buffers in MAP decoder can be replaced from two-port register file to single-port register file to reduce the hardware area and power consumption.

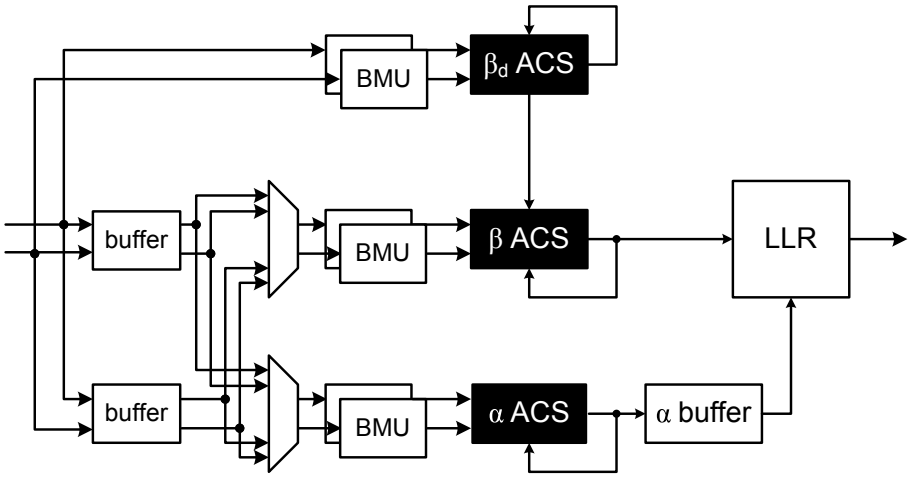


Figure 4.7: MAP Decoder Block Diagram

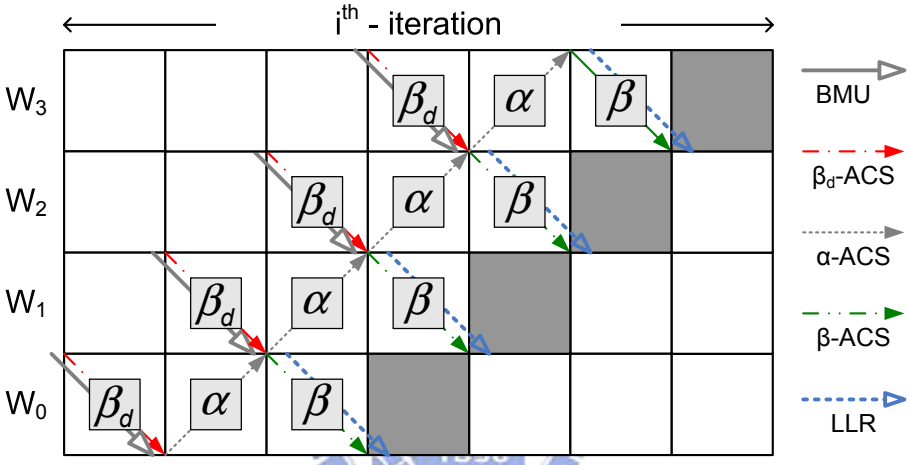
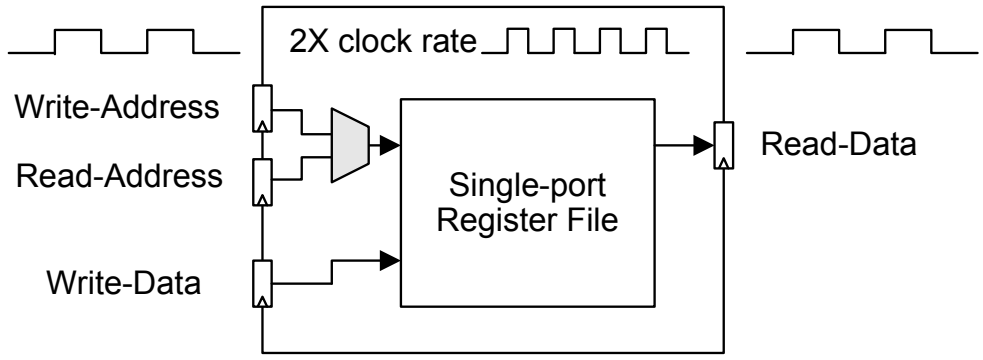


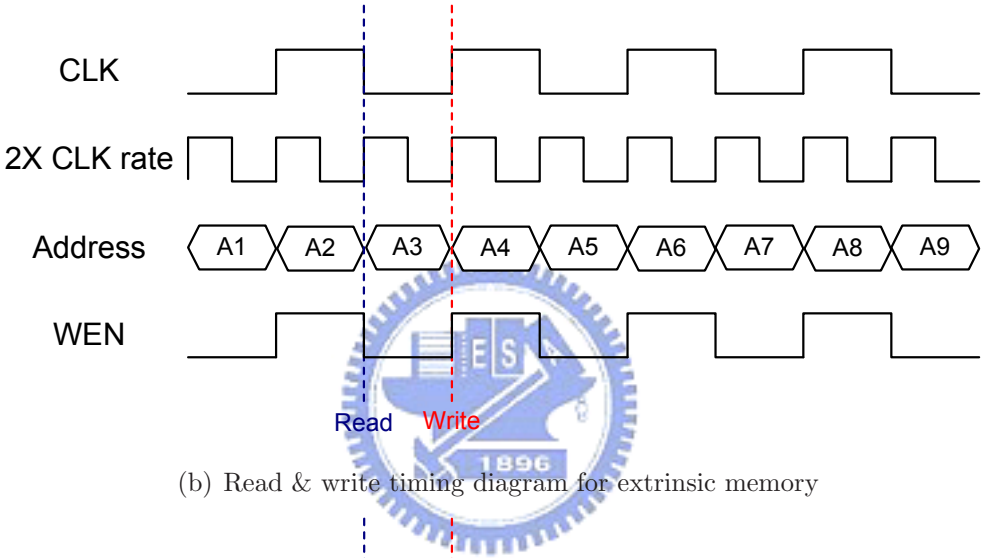
Figure 4.8: MAP Decoding Timing Flow

4.3.3 Interleaver Architecture

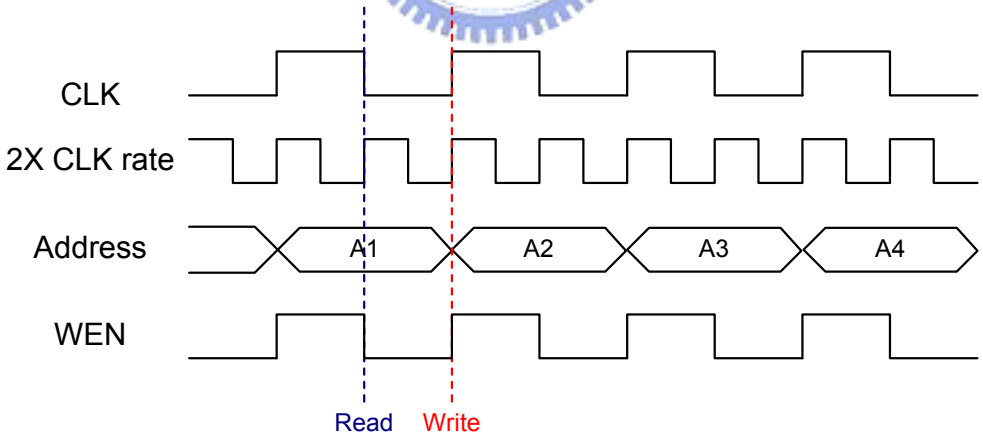
For the WiMAX interleaver in Table 4.2, the *modulo* operation is a critical problem for clock speed. Since we know all parameters used in Table 4.2 before decoding, some additions and divisions (simplified to shifter because the divisor is 2) can be derived as constant value before decoding. To minimize the critical path of interleaver operation, two adders and two subtractors are used instead of the divider. One adder is used to accumulate P_0, which adds one P_0 each cycle because in our design the interleavers only need to generate one read (write) address every cycle. Because the value of the accumulator ranges from 0 to $2N-1$, the *modulo* operation can be simplified to one subtraction and one multiplexer.



(a) Memory read & write architecture



(b) Read & write timing diagram for extrinsic memory



(c) Read & write timing diagram for MAP decoder

Figure 4.9: Pseudo Two-port Register File

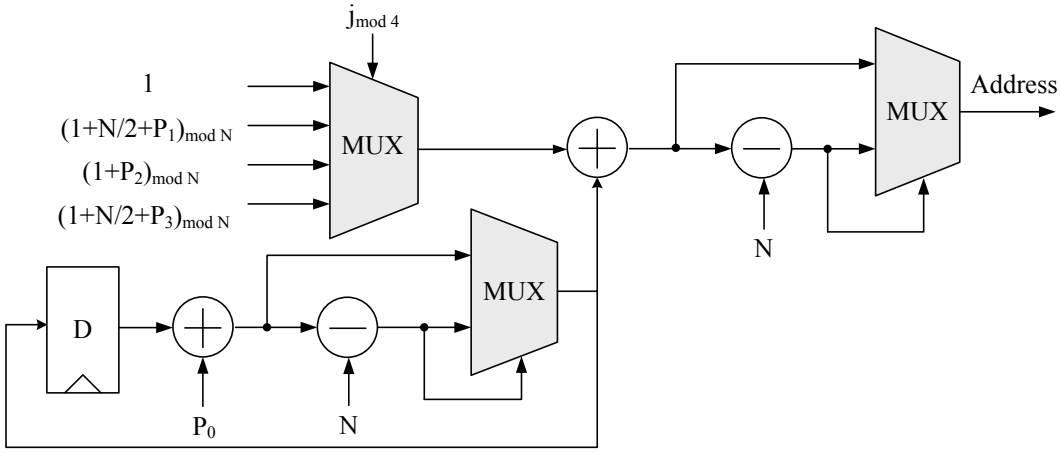


Figure 4.10: Interleaver Architecture

4.4 Chip Implementation Result

4.4.1 Chip Specification

Based on the architectures described above, we proposed an area-efficient double-binary turbo decoder with almost regular permutation (ARP) applied on WiMAX 802.16e. The proposed CTC decoder supports 17 different block lengths (24 to 2400) including hybrid automatic repeat-request (HARQ) modes. By means of scaling factor 0.75, we use smaller sliding window size (reducing the storage size of buffer and α buffer and the number of ACS unit) and smaller iterations (increasing the decoding throughput).

The primary chip specification of the double-binary turbo decoder is given in Table 4.4, which is implemented by the cell-based design flow, and fabricated in 90-nm 1P9M standard CMOS process. In CTC decoding process, two clock domains are used in memory and datapath respectively as we described in section 4.3.2. The higher clock rate is generated by a delay lock loop (DLL) circuit, which is applied to generate internal clock whose clock frequency is four times the external frequency. The other is generated clock which is the division of the higher clock rate. The total gate count is 303K (including the additional chip input buffer for testing) and the combinational logic part is only 30K while the memory occpies more than 80% area in our design. Fig. 4.11 is the chip layout of CTC decoder. By the proposed Max-Log MAP decoder and the simplified interleaver, the core size is 1.12mm^2 (1.4mm by 0.8mm) in 90-nm process. The operating frequency can achieve up to 166MHz and the maximum throughput is 30Mb/s which meets the

requirement of WiMAX standard. Furthermore, the average chip power consumption operated at 166MHz is 32.87mW with 0.9V supply voltage, which is estimated by post-layout simulation with the block length of 2400.

Table 4.4: WiMAX CTC decoder chip summary

Technology	UMC 90-nm 1P9M CMOS
Block length	24 to 2400
Support HARQ	Yes
Iteration	4.5
Sliding window	12
Input bit width	6bits
MAP	radix-4 MAP
Core area	1.12 mm^2
Gate count	303K
Max. throughput	30Mb/s
Supply voltage	1.0V
Clock rate	166MHz (@ 0.9V)
Power consumption	32.87mW (@ 0.9V and 166MHz) [1]
Utilization	65%

[1] Post-layout simulation with block length of 2400

4.4.2 Comparison with Other Relative Work

The comparison of the WiMAX 802.16e CTC decoders is shown in Table 4.5, where we list both synthesis and APR simulation result to compare the area with [26] and [27]. The synthesis result does not include the testing buffer unit in Fig. 4.11, so the core area is 0.619mm^2 operating at clock rate 166MHz with the maximum throughput 30Mb/s. Compared with other approaches, the proposed CTC decoder supports full-mode (include HARQ) block length and interleaver size for WiMAX 802.16e standard and is an area-efficient design. Besides, the power cnosumption estimated by post-layout simulation is 32.87mW and can be converted to energy efficiency. The energy efficiency is defined as the average energy consumed per bit within each decoding iteration (pJ/b/iter). For this

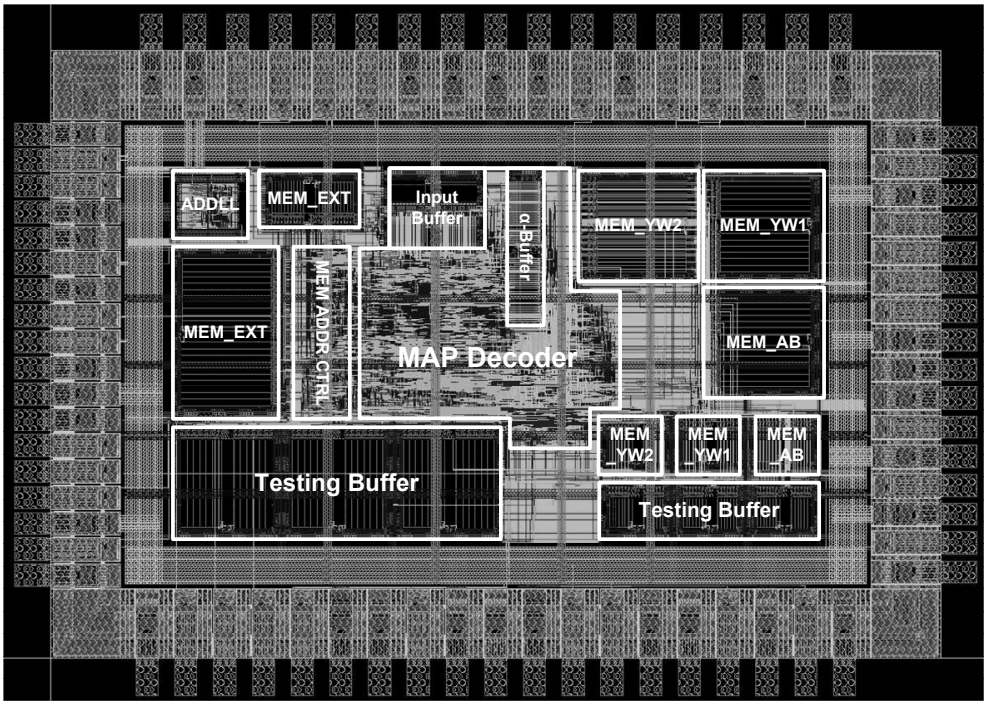


Figure 4.11: Chip layout photo of CTC decoder

decoder with 4.5 iterations, the energy efficiency will be calculated as

$$\frac{32.87\text{mW}}{4.5 \times 30\text{Mb/s}} = 243.4\text{pJ/b/iter}$$

Consequently, due to the area-efficient design with pseudo two-port register file usage and suitable parameter decision, the power consumption and the energy efficiency become smallest. The proposed design provides a fully compliant and area-efficient CTC decoder for WiMAX 802.16e application.

Table 4.5: Comparison among WiMAX CTC decoders

	Proposed Design	VLSI-DAT'08 [26]	ASPDAC'08 [27]
Technology	90nm	0.13μm	0.13μm
Block length	24 to 2400	24 to 240	2400
Support HARQ	Yes	No	N/A
Iteration	4.5	4	8
Sliding window	12	24	32
Input bit width	6bits	7bits	6bits
MAP	radix-4 MAP	radix-4 MAP	radix-4 MAP
Operating frequency	166MHz	100MHz	200MHz
Core area (mm^2)	0.619 [1] / 1.12	5.12	2.24
Gate count	219K [1] / 303K	N/A	N/A
(Combinational part) [2]	(39.7K) / (30K)	N/A	(65.7K)
Power consumption (mW)	32.87	183.7 [3]	274.138
Energy efficiency (pJ/bit/iter)	243.4 @ 0.9V	397.9	1412.5 [4]
Max. throughput	30Mb/s	115.4Mb/s	24.26Mb/s
Status	APR	Synthesis	APR

[1] Synthesis result without testing buffer

[2] Exclude memory unit

[3] Estimated by Prime Power with block length of 240

[4] Estimated by Power Compiler with block length of 2400

Chapter 5

Conclusion

This thesis proposed a fully compliant and area-efficient CTC decoder for the WiMAX 802.16e system. The modified Max-Log MAP algorithm is used to reduce hardware complexity and keeps almost the same performance as Log-MAP algorithm. In the proposed architecture, pseudo two-port register file is adopted to reduce storage memory and decoding latency. In MAP decoder, data reordering (reversing the input data order within a sliding window) is also used to decrease the latency and eliminate the β_d buffer requirement. The modified interleaver uses simplified addition and subtraction instead of divider to reduce the critical path and the hardware complexity. Implemented in 90-nm technology, the CTC decoder chip with 303K gate counts can achieve throughput 30Mb/s to meet the requirement of WiMAX standard. Besides, the power consumption is 32.87mW (@ 0.9V and 166MHz) estimated by post-layout simulation and block length of 2400.

An innovative trellis-based decoding algorithm using iterative stochastic computation is also proposed in this thesis. The state memory and noise dependent scaling factor are also applied on stochastic update rule. By using the further improvement, the performance of stochastic decoder with fixed point bit-width 10 can be better than hard-decision Viterbi algorithm and approach soft-decision Viterbi algorithm with floating point. Although the proposed improvement validates the stochastic decoding algorithm can be applied to trellis-based decoder, there still has some questions to be resolved. For the hardware issue, many possible simplifications and update rules can be applied to the algorithm. Besides, the number of decoding cycles and the proformance of the stochastic decoder can still be improved by increasing state number in trellis diagram and applying to high-radix trellis

decoding algorithm. These issues raise interesting research areas, with significant research potential to improve the hardware complexity of trellis-based stochastic decoders.

Appendix A

WiMAX 802.16e Parameter

Table A.1: CTC channel coding per modulation

Modulation	Block size [1]	Encoded block size [1]	Code rate	N	P_0	P_1	P_2	P_3
QPSK	6	12	1/2	24	5	0	0	0
QPSK	12	24	1/2	48	13	24	0	24
QPSK	18	36	1/2	72	11	6	0	6
QPSK	24	48	1/2	96	7	48	24	72
QPSK	30	60	1/2	120	13	60	0	60
QPSK	36	72	1/2	144	17	74	72	2
QPSK	48	96	1/2	192	11	96	48	144
QPSK	54	108	1/2	216	13	108	0	108
QPSK	60	120	1/2	240	13	120	60	180
QPSK	9	12	3/4	36	11	18	0	18
QPSK	18	24	3/4	72	11	6	0	6
QPSK	27	36	3/4	108	11	54	56	2
QPSK	36	48	3/4	144	17	74	72	2
QPSK	45	60	3/4	180	11	90	0	90
QPSK	54	72	3/4	216	13	108	0	108

[1] bytes

Table A.2: CTC channel coding per modulation (cont.)

Modulation	Block size [1]	Encoded block size [1]	Code rate	N	P_0	P_1	P_2	P_3
16-QAM	12	24	1/2	48	13	24	0	24
16-QAM	24	48	1/2	96	7	48	24	72
16-QAM	36	72	1/2	144	17	74	72	2
16-QAM	48	96	1/2	192	11	96	48	144
16-QAM	60	120	1/2	240	13	120	60	180
16-QAM	18	24	3/4	72	11	6	0	6
16-QAM	36	48	3/4	144	17	74	72	2
16-QAM	54	72	3/4	216	13	108	0	108
64-QAM	18	36	1/2	72	11	6	0	6
64-QAM	36	72	1/2	144	17	74	72	2
64-QAM	54	108	1/2	216	13	108	0	108
64-QAM	24	36	2/3	96	7	48	24	72
64-QAM	48	72	2/3	192	11	96	48	144
64-QAM	27	36	3/4	108	11	54	56	2
64-QAM	54	72	3/4	216	13	108	0	108
64-QAM	30	36	5/6	120	13	60	0	60
64-QAM	60	72	5/6	240	13	108	60	180

[1] bytes

Table A.3: CTC channel coding per modulation when supporting H-ARQ

Block size [1]	N	P_0	P_1	P_2	P_3
6	24	5	0	0	0
12	48	13	24	0	24
18	72	11	6	0	6
24	96	7	48	24	72
36	144	17	74	72	2
48	192	11	96	48	144
60	240	13	120	60	180
120	480	53	62	12	2
240	960	43	64	300	824
360	1440	43	720	360	540
480	1920	31	8	24	16
600	2400	53	66	24	2

[1] bytes

Bibliography

[1] *IEEE Standard for Local and Metropolitan Area Networks - Part 16: Air Interface for Fixed Broadband Wireless Access Systems*, Std. 802.16e, 2005.

[2] 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.

[3] 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, pp. 1261–1271, Feb./Mar./Apr. 1994.

[4] 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.

[5] B. Vucetic and J. Yuan, *Turbo codes, principles and applications.* Boston: Kluwer Academic, 2000.

[6] S. A. Barbulescu, "Sliding window and interleaver design," *Electron. Lett.*, vol. 37, no. 21, pp. 1299–1300, Oct. 2001.

[7] ——, "Iterative decoding of turbo codes and other concatenated codes," Ph.D. dissertation, Univ. South Australia, 1996.

[8] 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.

[9] 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.

[10] 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.

[11] S. Benedetto and G. Montorsi, "Design of paralle concatenated convolutional codes," *IEEE Trans. Commun.*, vol. 44, no. 5, pp. 591–600, May 1996.

[12] 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.

[13] 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.

[14] 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.

[15] S. Benedetto, G. Montorsi, and D. Divsalar, "Concatenated convloutional codes with interleavers," *IEEE Commun. Mag.*, vol. 41, no. 8, pp. 102–109, Aug. 2003.

[16] C. Berrou and M. Jezequel, "Non binary convolutional codes for turbo coding," *IEEE Electronic Letters*, vol. 35, no. 1, pp. 39–40, Jan. 1999.

[17] M. R. Soleymani, Y. Gao, and U. Vilaipornsawai, *Turbo coding for satellite and wireless communications*. Boston: Kluwer Academic, 2002.

[18] *Digital Video Broadcasting (DVB); Interaction channel for satellite distribution systems*, ETSI Std. EN 301 790, Rev. 1.3.1, 2003.

[19] C. Weib, C. Bettstetter, S. Riedel, and D. Costello, "Turbo decoding with tail-biting trellises," *in Proc. URSI Int. Symposium on Signals, Systems, and Electronics*, pp. 343–348, Oct. 1998.

[20] B. Gaines, *Advances in Information Systems Science*. New York: Plenum, 1969.

[21] S. S. Tehrani, S. Mannor, and W. J. Gross, "Survey of stochastic computation on factor graphs," *Proceedings of the 37th IEEE International Symposium on Multiple-Valued Logic (ISMVL 2007)*, p. 54, May. 2007.

[22] ——, "An area-efficient fpga-based architecture for fully-parallel stochastic ldpc decoding," *IEEE Workshop on Signal Processing Systems (SiPS 2007)*, pp. 255–260, Oct. 2007.

[23] C. Winstead, V. Gaudet, A. Rapley, and C. Schlegel, "Stochastic iterative decoders," *IEEE Trans. Inform. Theory*, pp. 1116–1120, Sept. 2005.

[24] S. S. Tehrani, S. Mannor, and W. J. Gross, "Stochastic decoding of ldpc dodes," *IEEE Electronic Letters*, vol. 10, no. 10, pp. 716–718.

[25] 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.

[26] C. H. Lin, C. Y. Chen, and A. Y. Wu, "High-throughput 12-mode ctc decoder for wimax standard," Apr. 2008, pp. 216–219.

[27] J. Kim and I. Park, "Duo-binary circular turbo decoder based on border metric encoding for wimax," *in Proc. of Asia and South Pacific Design Automation Conf. (ASPDAC'08)*, pp. 109–110, 2008.