

國立交通大學

電機與控制工程學系

碩士論文

適用於 3GPP 之 Radix-4 渦輪碼解碼器

A Radix-4 Turbo Decoder for 3GPP

研究生：廖盈超

指導教授：蔡尚澤 教授

中華民國九十七年十二月

適用於 3GPP 之 Radix-4 渦輪碼解碼器  
A Radix-4 Turbo Decoder for 3GPP

研究生：廖盈超

Student : Ying-Chao Liao

指導教授：蔡尚濶

Advisor : Shang-Ho Tsai

國立交通大學  
電機與控制工程學系  
碩士論文

A Thesis

Submitted to Department of Electrical and Control Engineering

College of Electrical Engineering and Computer Science

National Chiao Tung University

in Partial Fulfillment of the Requirements

for the Degree of

Master

In Electrical and Control Engineering

December 2008

Hsinchu, Taiwan, Republic of China

中華民國九十七年十二月

# 適用於 3GPP 之 Radix-4 渦輪碼解碼器

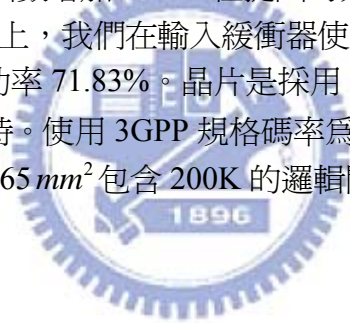
學生：廖盈超

指導教授：蔡尚澤

國立交通大學電機與控制工程學系（研究所）碩士班

## 摘 要

渦輪碼已經廣範使用在通訊系統，因為它有極佳的錯誤修正能力。為了增加扇出數和減少所需的記憶，開始研究渦輪碼 radix-4 架構。可是在 radix-4 的渦輪解碼器所需的計算路徑較長，使得 radix-4 渦輪解碼器的扇出數無法高於 radix-2 渦輪解碼器 2 倍。在這篇論文中我們在遞迴架構中提出一個查表方針，使得扇出數增加 62%，在提出的方法下效能僅比 Log-MAP 差 0.025dB。應用在超大型積體電路上，我們在輸入緩衝器使用 dual-RAM 取代成 single-RAM，這樣可以減少面積 57.8%及減少功率 71.83%。晶片是採用 TSMC 0.18  $\mu\text{m}$  CMOS 製程，操作頻率在 167MHz，電壓為 1.62 伏特。使用 3GPP 規格碼率為 1/3，扇出數為 22Mb/s 下，消耗功率為 135mW，而晶片的面積為 2.65  $\text{mm}^2$  包含 200K 的邏輯閘數。



# A Radix-4 Turbo Decoder for 3GPP

Student : Ying-Chao Liao

Advisors : Dr. Shang-Ho Tsai

Department ( Institute ) of Electrical and Control Engineering  
National Chiao Tung University

## ABSTRACT

Turbo code has been widely used in communication systems, because of its outstanding error correction performance. To increase throughput and decrease the required memory. Radix-4 architecture for Turbo decoder was studied. However, the critical path of the recursive architecture in Radix-4 turbo decoder is long, As a result conventional Radix-4 architecture [15] cannot achieve twice throughput over the conventional Radix-2 architecture. In this thesis, we proposed a Look-Up Table scheme for the recursive architecture and the throughput increases up to 62%. The performance of the proposed scheme is worse than the Log-MAP (optimal) by only 0.025dB. In VLSI implementation, we propose a method for input buffer and it can reduce the dual-RAM by the single-RAM to save area and power. The proposed method can reduce the area by 57.8% and the power by 71.83%. The chip is fabricated in TSMC 0.18  $\mu\text{m}$  CMOS process, operating at 167MHz clock rate with voltage supply 1.62V. The power consumption is 135mW at decoding rate 22Mb/s, with code rate 1/3 for 3GPP standard. The core area is 2.65  $\text{mm}^2$ , contain 200K gate counts.

## 誌 謝

經過了兩年研究所的生涯終於告一段落了，此篇論文能夠順利的完成非常感謝的是我的指導教授蔡尚濶教授，在兩年的研究生活中，在研究上遭遇到很多困難，但老師很仔細的了解原因所在並一一地幫忙解決，使得我可以繼續研究下去。並感謝我的口試委員：林源倍教授、簡鳳村教授、董蘭榮教授提供我寶貴意見，以彌補論文不善之處。

另外，感謝陳宇文、葉柏賢同學，因為有你們的幫忙，讓我在研究之路上得到幫忙和勉勵，使我獲益良多。也感謝學弟妹們的加入，因為有你們的加入使我們研究室的氣氛更加和樂。

最後，我要感謝的是我的父母，因為有他們給我無後顧之憂才使得研究可以完成，也因為有你們的支持與鼓勵讓我遇到困難都能迎刃而解。



# A Radix-4 Turbo Decoder for 3GPP

Ying-Chao Liao

Advisor: Dr. Shang-Ho Tsai  
Department of Electrical and Control Engineering  
National Chiao Tung University

December 4, 2008

## Abstract

Turbo code has been widely used in communication systems, because of its outstanding error correction performance. To increase throughput and decrease the required memory. Radix-4 architecture for Turbo decoder was studied. However, the critical path of the recursive architecture in Radix-4 turbo decoder is long, As a result conventional Radix-4 architecture [15] cannot achieve twice throughput over the conventional Radix-2 architecture. In this thesis, we proposed a Look-Up Table scheme for the recursive architecture and the throughput increases up to 62%. The performance of the proposed scheme is worse than the Log-MAP (optimal) by only 0.025dB. In VLSI implementation, we propose a method for input buffer and it can reduce the dual-RAM by the single-RAM to save area and power. The proposed method can reduce the area by 57.8% and the power by 71.83%. The chip is fabricated in TSMC 0.18  $\mu\text{m}$  CMOS process, operating at 167MHz clock rate with voltage supply 1.62V. The power consumption is 135mW at decoding rate 22Mb/s, with code rate 1/3 for 3GPP standard. The core area is 2.65  $\text{mm}^2$ , contain 200K gate counts.

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Turbo Code</b>	<b>3</b>
2.1	Turbo Code Encoder . . . . .	3
2.1.1	Recursive Systematic Convolution Codes . . . . .	3
2.1.2	Termination of encoding process . . . . .	4
2.2	Decoding Criterion . . . . .	5
2.2.1	MAP criteria . . . . .	5
2.2.2	Log-MAP and Max-Log-MAP criteria . . . . .	7
2.3	Decoding Algorithm . . . . .	11
2.3.1	Radix-2 algorithm . . . . .	11
2.3.2	Radix-4 algorithm . . . . .	12
2.3.3	Deriving LLR for Radix-4 . . . . .	14
2.4	Decoding Architecture . . . . .	17
2.4.1	Sliding Window . . . . .	20
<b>3</b>	<b>Radix-4 Recursive Architecture</b>	<b>21</b>
3.1	Conventional Architecture . . . . .	21
3.2	Proposed Architecture . . . . .	21
3.2.1	Performance Comparison . . . . .	26
3.3	Fixed point Analysis . . . . .	28
<b>4</b>	<b>VLSI Implementation</b>	<b>30</b>
4.1	Hardware Design for 3GPP . . . . .	30
4.1.1	Input Buffer . . . . .	30
4.1.2	BMU(branch metric unit ) . . . . .	34
4.1.3	OACS(Offset-Add-Compare-Select) . . . . .	36
4.1.4	LLR (Log-Likelihood Ratio ) . . . . .	36
4.1.5	Extrinsic Information and a Priori Information . . . . .	37
4.1.6	Interleaver and De-interleaver . . . . .	39
4.1.7	Hard Decision . . . . .	39
4.1.8	Sliding Window . . . . .	40

4.2	Design flow . . . . .	41
4.2.1	System model . . . . .	41
4.2.2	RTL code . . . . .	44
4.2.3	BIST . . . . .	44
4.2.4	Synthesis . . . . .	44
4.2.5	Gate-level simulation . . . . .	44
4.2.6	DFT . . . . .	45
4.2.7	ATPG . . . . .	45
4.2.8	APR . . . . .	45
4.2.9	DRC and LVS . . . . .	45
4.2.10	Post-layout level . . . . .	46
4.3	Chip Layout and Comparison . . . . .	47
<b>5</b>	<b>Conclusion</b>	<b>49</b>





# List of Figures

2.1	The Turbo Code Encoder for 3GPP. . . . .	4
2.2	A general form of $max^*(\cdot)$ . . . . .	9
2.3	The trellis diagram of $\bar{\alpha}$ . . . . .	12
2.4	The trellis diagram of $\bar{\beta}$ . . . . .	13
2.5	The Radix-2 and Radix-4 trellis diagram. . . . .	15
2.6	The trellis diagram of LLR unit for stage $t$ : (a) $LLR_t^1$ and (b) $LLR_t^0$ . . . . .	18
2.7	The trellis diagram of LLR unit for stage $t + 1$ : (a) $LLR_{t+1}^1$ and (b) $LLR_{t+1}^0$ . . . . .	18
2.8	An architecture of the turbo decoder. . . . .	19
2.9	The sliding window diagram. . . . .	20
3.1	Conventional Radix-4 Architecture. . . . .	22
3.2	Radix-4 recursion architecture of [20]. . . . .	23
3.3	Architecture of the proposed LUT used in [20]. . . . .	24
3.4	Architecture of the proposed LUT. . . . .	25
3.5	Performance comparison among the Log-MAP and four approximated algorithms. . . . .	29
4.1	The turbo decoder architecture with a single SISO decoder. . . . .	31
4.2	The input data flow. . . . .	33
4.3	The proposed ROM and RAM scheme to achieve two-read and two-write in one clock cycle. . . . .	34
4.4	Timing diagram for the proposed RAM and ROM schemes. . . . .	35
4.5	The architecture of $\bar{\gamma}$ . . . . .	36
4.6	The normalization of OACS. . . . .	37
4.7	The Architecture of LLR. . . . .	38
4.8	The hardware architecture of $max^*(\cdot)$ . . . . .	38
4.9	Timing diagram of a priori information for two Dual-RAMs. . . . .	39
4.10	The architecture of hard decision. . . . .	40
4.11	Calculating BMU, OACS and LLR. . . . .	41
4.12	Timing diagram of Sliding Window. . . . .	42
4.13	IC design flow. . . . .	43
4.14	Chip layout of the proposed Radix-4 Turbo Decoder for 3GPP. . . . .	47

# List of Tables

3.1	Approximation of [20]. . . . .	24
3.2	The values of $g(x)$ , $u(v)$ , $d_1$ , $d_0$ and $p_1$ , $p_0$ . . . . .	27
3.3	Comparison of various recursive architectures. . . . .	28
3.4	Quantization format for the proposed Turbo Decoder. . . . .	28
4.1	Summary of interleaver process with four case. . . . .	32
4.2	Comparison of area and power for implementing the input buffer by dual-RAM and singl-RAM. . . . .	33
4.3	The expected turbo decoder chip summary. . . . .	48
4.4	Chip comparison. . . . .	48



# Chapter 1

## Introduction

The basic concept of channel coding is to add redundancy bits along with information bits before transmission. These redundancy bits can help the receiver to decode data correctly with higher probability. In 1984, Shannon proposed a limit on the maximum achievable data rate over a channel. Many researchers attempt various methods to close the Shannon limit. In 1993, Turbo Code was proposed by Berrou, Glavieux and Thimajashima [1], it is a powerful error correcting codes whose performance is close to Shannon limit. In many mobile communication systems, turbo code has been adopted to gain better performance, such as in WCDMA, CDMA2000 [2], WiMAX and 3G [3] standards.

The turbo encode consists of two Recursive Systematic Convolutional (RSC) encoders [18] and one interleaver. For the turbo decoder, it consists of two soft-input soft-output (SISO) decoders and one interleaver/deinterleaver between them. The SISO decoder is used in turbo decoder. In addition, it is also applied in some other algorithm such as SOVA (Soft Out Viterbi Algorithm) [4]-[6], Log-MAP, Max-Log-MAP [7] and improved Max-Log-MAP [8] (approximations to the MAP algorithm).

Interleaver design for 3GPP was proposed by [10]-[12], to support full block length. In 2003, Lucent Bell Labs [15] proposed the radix-4 algorithm for turbo decoder. The algorithm has two advantages. One is doubling the throughput for a given clock rate over the radix-2 architecture, and the other is reducing the memory. Hence, in recent years, radix-4 turbo code is studied, e.g. see [20], [21]

which proposed methods to improve the recursive architecture for radix-4 turbo code decoder. As for VLSI implementation for turbo decoders, the sliding window algorithm ([16], [17]) is proposed to avoid storing the metrics corresponding to the entire codeword sequence to reduce the memory requirement.

In this thesis, we use Radix-4 algorithm, approximated Log-MAP [20] and sliding window technique to implement turbo decoder for 3GPP standard. Moreover, we proposed a Look-Up Table scheme for the recursive architecture and the throughput increases up to 62% over the traditional Radix2 algorithm. In VLSI implementation, we propose a method for input buffer and it can reduce the dual-RAM by the single-RAM to save area and power. The proposed method can reduce the area by 57.8% and the power by 71.83%. The chip is fabricated in TSMC 0.18  $\mu\text{m}$  CMOS process. The expected clock rate is 167MHz, throughput is 22Mb/s, and the power consumption is 135mW with code rate 1/3, block length 512. The core area is 2.65  $\text{mm}^2$ , containing 200K gate counts.

The chapters are organized as follows. In Chapter 2, we describe the MAP criteria, Log-MAP criteria and Max-Log-MAP criteria. Also we compare decoding algorithm for radix-2 and radix-4. Operation of sliding window technique is also described here. In Chapter 3 we introduce the proposed scheme for radix-4 recursive architecture, and compare it to various recursive architectures. Chapter 4 introduces the VLSI implementation, we describe how to use one single RAM to achieve Dual-RAM operation for Radix-4 turbo decoder. Also we describe the decoding flow with hardware architecture and show the chip layout as well as the corresponding chip performance comparison.

# Chapter 2

## Turbo Code

In 1993, The turbo code was introduced by Berrou, Glavieux, and Thitimajshima [1], and achieved a bit-error probability of  $10^{-5}$  with a code rate of  $1/2$  over an AWGN (additive white Gaussian noise) channel and BPSK modulation at an  $E_b/N_0$  of 0.7dB. Turbo code has been adopted in many mobile communication systems, such as WCDMA, CDMA2000, WiMAX, 3G mobile. In 3GPP [3] systems, the turbo encoder consists of two Recursive Systematic Convolutional (RSC) [18] codes in parallel and an interleaver unit. For turbo decoder, it consists of two Maximum A Posteriori (MAP) decoders connected in series with a feedback loop from the second output to the first input. Let us introduce the codec more detailed in the following subsections.

### 2.1 Turbo Code Encoder

#### 2.1.1 Recursive Systematic Convolution Codes

Fig. 2.1 is the turbo code encoder structure in 3GPP systems and code rate is  $1/3$ . The encoder consists of two RSC codes and an interleaver. The generator matrix of the RSC encode is:

$$G(D) = \left[ 1, \frac{g_1(D)}{g_0(D)} \right], \quad (2.1)$$

where

$$g_0(D) = 1 + D^2 + D^3 \quad (2.2)$$

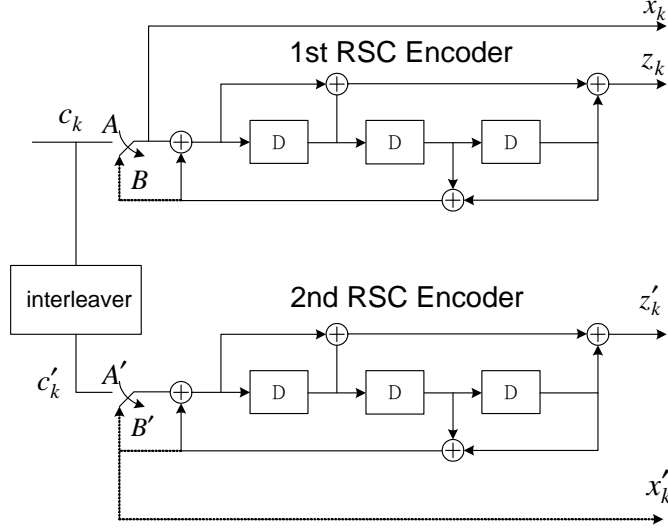


Figure 2.1: The Turbo Code Encoder for 3GPP.

is the feedback polynomial and

$$g_1(D) = 1 + D + D^3 \quad (2.3)$$

is the forward polynomial.

Initially, the registers from RSC must be zero, and upper switch and lower switch switch to A and A'. After  $K$  numbers are inputted, thus the order of the output from the turbo encoder is  $x_1, z_1, z'_1, x_2, z_2, z'_2, \dots, x_K, z_K, z'_K$  where  $x_1, x_2, \dots, x_K$  are input bits and  $K$  is the number of input bits.

### 2.1.2 Termination of encoding process

The termination scheme is to let the encoder come back to zero state and thus it can decrease the bit error rate. When the  $K$  bits complete encoding process, both the two RSCs need to generate 12 tail bits. First, the lower RSC in Fig. 2.1 is disabled and the switch in the upper RSC is changed from position A to position B, and then six tail bits are generated. Second, the last six tail bits are generated by turning off the upper RSC and the switch in the lower RSC is

changed from position  $A$  to position  $B$ . The 12 tail bits shall be:

$$x_{K+1}, z_{K+1}, x_{K+2}, z_{K+2}, x_{K+3}, z_{K+3}, x'_{K+1}, z'_{K+1}, x'_{K+2}, z'_{K+2}, x'_{K+3}, z'_{K+3}. \quad (2.4)$$

## 2.2 Decoding Criterion

### 2.2.1 MAP criteria

The MAP algorithm has been developed by Bahl, Cocke, Jelinek, and Raviv in 1974 [13] and is termed as BCJR algorithm. Consider a situation that we received a signal  $r$  over a discrete memoryless channel. The state transitions from state  $m$  to state  $m'$ , i.e.  $S_m(t)$  at time  $t$  to  $S_{m'}(t+1)$  at time  $t+1$ , we can obtain a joint probability:

$$\begin{aligned} & \Pr\{S_m(t), S_{m'}(t+1), r\} \\ &= \Pr\{S_m(t), S_{m'}(t+1), r_0^{t-1}, r_t^t, r_{t+1}^{N-1}\}. \end{aligned} \quad (2.5)$$

Using joint probability property

$$\Pr(A, B) = \Pr(A) \Pr(B|A) \quad (2.6)$$

or

$$\Pr(A, B) = \Pr(B) \Pr(A|B), \quad (2.7)$$

we can rewrite (2.5) as follows

$$\begin{aligned} & \Pr\{S_m(t), S_{m'}(t+1), r\} \\ &= \Pr\{S_m(t), S_{m'}(t+1), r_0^{t-1}, r_t^t\} \Pr\{r_{t+1}^{N-1} | S_m(t), S_{m'}(t+1), r_0^{t-1}, r_t^t\} \\ &= \Pr\{S_m(t), r_0^{t-1}\} \Pr\{S_{m'}(t+1), r_t^t | S_m(t), r_0^{t-1}\} \Pr\{r_{t+1}^{N-1} | S_m(t), S_{m'}(t+1), r_0^{t-1}, r_t^t\} \\ &= \Pr\{S_m(t), r_0^{t-1}\} \Pr\{S_{m'}(t+1), r_t^t | S_m(t)\} \Pr\{r_{t+1}^{N-1} | S_{m'}(t+1)\}, \end{aligned} \quad (2.8)$$

where  $r_a^b$ , means that receive signals from time instance  $a$  to time instance  $b$ .

From (2.8), let us define three functions for description convenience, i.e.

$$\alpha(S_m(t)) = \Pr\{S_m(t), r_0^{t-1}\}, \quad (2.9)$$

$$\beta(S_{m'}(t+1)) = \Pr\{r_{t+1}^{N-1} | S_{m'}(t+1)\}, \quad (2.10)$$

and

$$\gamma(S_m(t), S_m(t+1)) = \Pr\{S_m(t+1), r_t | S_m(t)\}. \quad (2.11)$$

Hence (2.8) can be rewritten as:

$$\Pr\{S_m(t), S_m(t+1), r\} = \alpha(S_m(t))\gamma(S_m(t), S_m(t+1))\beta(S_m(t+1)). \quad (2.12)$$

Define  $S$  be the set of all the states at time  $t$ . We can further extend (2.9) as

$$\begin{aligned} \alpha(S_m(t+1)) &= \Pr\{S_m(t+1), r_0^t\} \\ &= \sum_{S_m(t) \in S} \Pr\{S_m(t), S_m(t+1), r_0^t\} \\ &= \sum_{S_m(t) \in S} \Pr\{S_m(t), r_0^{t-1}\} \Pr\{S_m(t+1), r^t | S_m(t), r_0^{t-1}\} \\ &= \sum_{S_m(t) \in S} \Pr\{S_m(t), r_0^{t-1}\} \Pr\{S_m(t+1), r^t | S_m(t)\} \\ &= \sum_{S_m(t) \in S} \alpha(S_m(t))\gamma(S_m(t), S_m(t+1)). \end{aligned} \quad (2.13)$$

Similarly, define  $S'$  be the set of all the states at time  $t+1$ . We can further extend (2.10) as

$$\begin{aligned} \beta(S_m(t)) &= \Pr\{r_t^{N-1} | S_m(t)\} \\ &= \sum_{S_m(t+1) \in S'} \Pr\{S_m(t+1), r_t^{N-1} | S_m(t)\} \\ &= \sum_{S_m(t+1) \in S'} \Pr\{r_{t+1}^{N-1} | S_m(t+1), r_t, S_m(t)\} \Pr\{S_m(t+1), r_t | S_m(t)\} \\ &= \sum_{S_m(t+1) \in S'} \Pr\{r_{t+1}^{N-1} | S_m(t+1), r_t\} \Pr\{S_m(t+1), r_t | S_m(t)\} \\ &= \sum_{S_m(t+1) \in S'} \beta(S_m(t+1))\gamma(S_m(t), S_m(t+1)). \end{aligned} \quad (2.14)$$

Finally, the branch metric can be rewritten as

$$\gamma(S_m(t), S_m(t+1)) = \Pr\{S_m(t+1), r_t | S_m(t)\}. \quad (2.15)$$

From the Bayes' rule

$$\Pr(A|B) = \frac{\Pr(A, B)}{\Pr(B)},$$



we can rewrite (2.15) as

$$\begin{aligned}
\gamma(S_m(t), S_m(t+1)) &= \frac{\Pr\{S_m(t+1), S_m(t), r_t\}}{\Pr\{S_m(t)\}} \\
&= \frac{\Pr\{S_m(t+1), S_m(t)\} \Pr\{S_m(t+1), S_m(t), r_t\}}{\Pr\{S_m(t)\} \Pr\{S_m(t+1), S_m(t)\}} \\
&= \Pr\{S_m(t+1)|S_m(t)\} \Pr\{r_t|S_m(t+1), S_m(t)\} \\
&= \Pr(c_t) \Pr(r_t|w_t),
\end{aligned} \tag{2.16}$$

where  $c_t$  is encoder input that make the state change from  $S_m(t)$  to  $S_m(t+1)$ , and  $w_t$  is the corresponding codeword.

Consider the log-likelihood ratio (LLR)

$$\begin{aligned}
L(c_t) &\equiv \ln \frac{\Pr\{c_t = +1|r\}}{\Pr\{c_t = -1|r\}} \\
&= \ln \frac{\sum_{S(m,m') \in c_t^{+1}} \Pr\{S_m(t), S_m(t+1)|r\}}{\sum_{S(m,m') \in c_t^{-1}} \Pr\{S_m(t), S_m(t+1)|r\}},
\end{aligned} \tag{2.17}$$

where  $S(m, m') \in c_t^{\pm 1}$  indicates that all the states from state  $m$  to state  $m'$  result in transmitting  $c_t = +1$  and  $c_t = -1$  respectively. From (2.12), we can rearrange (2.17) as

$$L(c_t) \equiv \ln \frac{\sum_{S(m,m') \in c_t^{+1}} \alpha(S_m(t)) \gamma(S_m(t), S_m(t+1)) \beta(S_m(t+1))}{\sum_{S(m,m') \in c_t^{-1}} \alpha(S_m(t)) \gamma(S_m(t), S_m(t+1)) \beta(S_m(t+1))}. \tag{2.18}$$

From (2.18) is the LLR of turbo code. In VLSI design it is difficult to implement the LLR as in (2.18), because the LLR of natural log function in hardware demand memory for look-up table. To overcome this, we will change the MAP criteria to Log-MAP criteria and then apply the Log-MAP to the LLR.

### 2.2.2 Log-MAP and Max-Log-MAP criteria

In this section we derive the Log-MAP algorithm (2.18) can be rewritten to the following

$$L(c_t) = \ln \frac{\sum_{S(m,m') \in c_t^{+1}} \exp[\bar{\alpha}(S_m(t)) + \bar{\gamma}(S_m(t), S_m(t+1)) + \bar{\beta}(S_m(t+1))]}{\sum_{S(m,m') \in c_t^{-1}} \exp[\bar{\alpha}(S_m(t)) + \bar{\gamma}(S_m(t), S_m(t+1)) + \bar{\beta}(S_m(t+1))]}, \tag{2.19}$$

where

$$\begin{aligned}\bar{\alpha}(S_m(t+1)) &= \ln(\bar{\alpha}(S_m(t+1))) \\ &= \ln \left\{ \sum_{S_m(t) \in S} \exp[\bar{\alpha}(S_m(t)) + \bar{\gamma}(S_m(t), S_m(t+1))] \right\},\end{aligned}\quad (2.20)$$

$$\begin{aligned}\bar{\beta}(S_m(t)) &= \ln(\beta(S_m(t))) \\ &= \ln \left\{ \sum_{S_m(t+1) \in S'} \exp[\bar{\beta}(S_m(t+1)) + \bar{\gamma}(S_m(t), S_m(t+1))] \right\},\end{aligned}\quad (2.21)$$

and

$$\bar{\gamma}(S_m(t), S_m(t+1)) = \ln(\gamma(S_m(t), S_m(t+1))).\quad (2.22)$$

According to the Jacobian function [14], we have

$$\ln(\exp(x) + \exp(y)) \triangleq \max^*(x, y) = \max(x, y) + \ln(1 + \exp(-|x - y|)).\quad (2.23)$$

Referring to Fig. 2.2, a more general form is given by

$$\begin{aligned}\ln\left(\sum_{i=1}^d \exp(x_i)\right) &\triangleq \widehat{\max}(x_1, x_2, \dots, x_d) \\ &= \max^*(\dots, \max^*(\max^*(x_1, x_2), \max^*(x_3, x_4)), \dots \\ &\quad, \max^*(\max^*(x_{d-3}, x_{d-2}), \max^*(x_{d-1}, x_d)), \dots).\end{aligned}\quad (2.24)$$

Thus (2.19) can be rewritten as

$$\begin{aligned}L(c_t) &= \widehat{\max}_{S(m,m) \in c_t^+} [\bar{\alpha}(S_m(t)) + \bar{\gamma}(S_m(t), S_m(t+1)) + \bar{\beta}(S_m(t+1))] \\ &\quad - \widehat{\max}_{S(m,m) \in c_t^-} [\bar{\alpha}(S_m(t)) + \bar{\gamma}(S_m(t), S_m(t+1)) + \bar{\beta}(S_m(t+1))].\end{aligned}\quad (2.25)$$

From (2.20), (2.21) and (2.24),  $\bar{\alpha}(S_m(t+1))$  and  $\bar{\beta}(S_m(t))$  can be written as

$$\bar{\alpha}(S_m(t+1)) = \widehat{\max}_{S_m(t) \in S} [\bar{\alpha}(S_m(t)) + \bar{\gamma}(S_m(t), S_m(t+1))],\quad (2.26)$$

$$\bar{\beta}(S_m(t)) = \widehat{\max}_{S_m(t+1) \in S'} [\bar{\beta}(S_m(t+1)) + \bar{\gamma}(S_m(t), S_m(t+1))].\quad (2.27)$$

Because the encoder starts at zero state and terminates at zero state,  $\alpha$  and  $\beta$  satisfied the following initial conditions:

$$\alpha(S_0(t=0)) = 1, \quad \alpha(S_m(t=0)) = 0 \text{ for } m \neq 0,$$

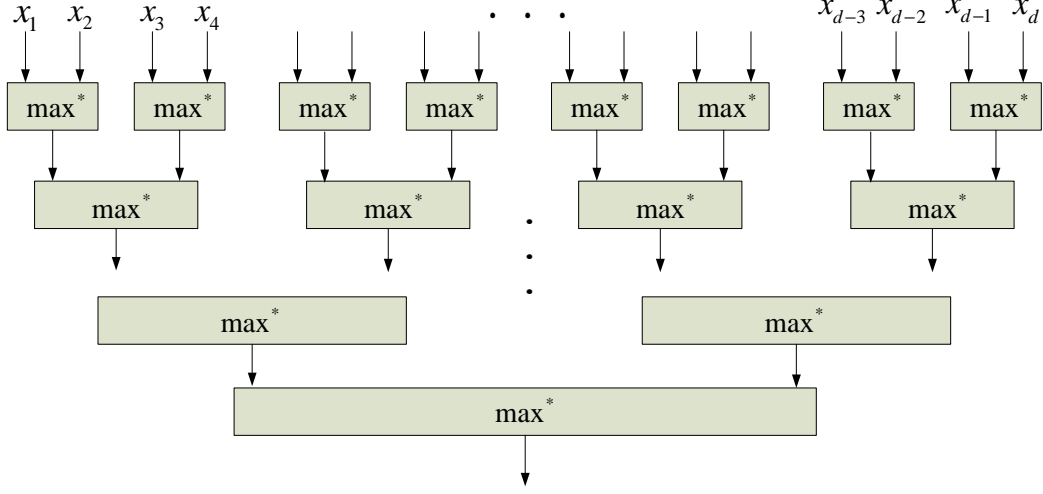


Figure 2.2: A general form of  $\max^*(\cdot)$ .

and

$$\beta(S_0(t=N)) = 1, \beta(S_m(t=N)) = 0 \text{ for } m \neq 0.$$

Using natural log conditions:

$$\bar{\alpha}(S_0(t=0)) = 0, \bar{\alpha}(S_m(t=0)) = -\infty \text{ for } m \neq 0,$$

and

$$\bar{\beta}(S_0(t=N)) = 0, \bar{\beta}(S_m(t=N)) = -\infty \text{ for } m \neq 0.$$

Let us define a priori information:

$$L_a(c_t) \triangleq \ln \frac{\Pr(c_t = +1)}{\Pr(c_t = -1)} \quad (2.28)$$

Taking the exponential operation on both sides in (2.28) and employing  $\Pr(c_t = -1) = 1 - \Pr(c_t = +1)$ , we have

$$\Pr(c_t = +1) = \frac{\exp(L_a(c_t))}{1 + \exp(L_a(c_t))} = \frac{1}{1 + \exp(-L_a(c_t))} \quad (2.29)$$

and

$$\Pr(c_t = -1) = \frac{\exp(-L_a(c_t))}{1 + \exp(-L_a(c_t))}, \quad (2.30)$$

thus we can rewrite (2.29) and (2.30) as

$$\Pr(c_t = \pm 1) = \frac{\exp^{-L_a(c_t)/2}}{1 + \exp^{-L_a(c_t)}} \cdot \exp^{c_t L_a(c_t)/2} = A_t \cdot \exp^{c_t L_a(c_t)/2}, \quad (2.31)$$

where

$$A_t = \frac{\exp^{-L_a(c_t)/2}}{1 + \exp^{-L_a(c_t)}}.$$

According to (2.16), the probability  $P(r_t|w_t)$  in AWGN channel is

$$\begin{aligned} \Pr(r_t(i)|w_t(i)) &= \left( \frac{1}{\sqrt{2\pi\sigma^2}} \right)^n \cdot \exp \left( -\frac{\sum_{i=0}^{n-1} (r_t(i) - w_t(i))^2}{2\sigma^2} \right) \\ &= \left( \frac{1}{\sqrt{2\pi\sigma^2}} \right)^n \cdot \exp \left( -\frac{\sum_{i=0}^{n-1} (r_t^2(i) + w_t^2(i))}{2\sigma^2} \right) \cdot \exp \left( \frac{\sum_{i=0}^{n-1} r_t(i) \cdot w_t(i)}{\sigma^2} \right) \\ &= V_t \cdot \exp \left( \sum_{i=0}^{n-1} (L_c \cdot r_t(i) \cdot w_t(i)/2) \right), \end{aligned} \quad (2.32)$$

where  $n$  is inverse of the code rate,  $L_c$  is channel reliable value defined as  $4\frac{E_S}{N_0}$ ,  $\sigma^2$  is the noise variance and

$$V_t = \left( \frac{1}{\sqrt{2\pi\sigma^2}} \right)^n \cdot \exp \left( -\frac{\sum_{i=0}^{n-1} (r_t^2(i) + w_t^2(i))}{2\sigma^2} \right).$$

Substitute (2.16), (2.31) and (2.32) into (2.19), we find  $A_t$  and  $V_t$  are cancelled, thus branch metric can be obtained as following

$$\bar{\gamma}(S_m(t), S_m(t+1)) = \frac{1}{2} (c_t \cdot L_a(c_t) + \sum_{i=0}^{n-1} L_c \cdot r_t(i) \cdot w_t(i)), \quad (2.33)$$

where  $r_t(i)$  are received signal and  $w_t(i)$  are  $\pm 1$ . Initially, the  $L_a(c_t)$  is unknown. In this case we assume that  $\Pr(c_t = +1) = \Pr(c_t = -1)$ , therefore  $L_a(c_t)$  is zero at the beginning. Combining (2.26), (2.27) and (2.33) to calculate the LLR in (2.25), the received data can be decoded as follows

$$c_t = \begin{cases} 1 & \text{if } L(c_t) \geq 0 \\ 0 & \text{if } L(c_t) < 0 \end{cases} \quad (2.34)$$

From (2.23) and (2.24), we can further simplify the  $\max^*(\cdot)$  and  $\widehat{\max}(\cdot)$  functions as follows

$$\begin{aligned} \widehat{\max}(x_1, x_2, \dots, x_d) &\approx \max(\dots, \max(\max(x_1, x_2), \max(x_3, x_4)), \dots \\ &\quad, \max(\max(x_{d-3}, x_{d-2}), \max(x_{d-1}, x_d)), \dots). \end{aligned} \quad (2.35)$$

Eq. (2.35) is obtained by removing the correction term and becomes  $\max(\cdot)$  maximum function. Replacing  $\max^*(\cdot)$  by  $\max(\cdot)$  in (2.25), (2.26) and (2.27), we have the LLR with Max-Log-MAP. There is a small performance degradation by using LLR with MAX-Log-MAP instead of LLR with MAP. The degradation is more pronounced in low SNR region.

## 2.3 Decoding Algorithm

### 2.3.1 Radix-2 algorithm

In this section we use the trellis diagram to explain how  $\bar{\alpha}$ ,  $\bar{\beta}$  and  $\bar{\gamma}$  are calculated in the radix-2 and the radix-4 algorithm. Fig. 2.3 and Fig. 2.4 show the trellis of  $\bar{\alpha}$  and  $\bar{\beta}$  respectively, where the dotted lines and solid lines stand for  $c_t = 0$  and  $c_t = 1$ . An example will help understand.

**Example 1:** Calculation of  $\bar{\alpha}$  in radix-2 algorithm.

Referring to Fig 2.3 to obtain  $\bar{\alpha}(S_0(t+1))$ , we know that there are two paths connected to it. One path is  $S_0(t) \xrightarrow{c_t=0} S_0(t+1)$  and the another path is  $S_1(t) \xrightarrow{c_t=1} S_0(t+1)$ . Therefore  $\bar{\gamma}(S_0(t), S_0(t+1))$  and  $\bar{\gamma}(S_1(t), S_0(t+1))$  from (2.33) can be expressed as

$$\bar{\gamma}(S_0(t), S_0(t+1)) = \frac{1}{2} \cdot [(-1) \cdot L_a(c_t = -1) + L_c \cdot (r_t(0) \cdot (-1) + r_t(1) \cdot (-1))], \quad (2.36)$$

and

$$\bar{\gamma}(S_1(t), S_0(t+1)) = \frac{1}{2} \cdot [(+1) \cdot L_a(c_t = +1) + L_c \cdot (r_t(0) \cdot (+1) + r_t(1) \cdot (+1))]. \quad (2.37)$$

Substituting (2.36) and (2.37) into (2.26) leads to  $\bar{\alpha}(S_0(t+1))$  as follows

$$\bar{\alpha}(S_0(t+1)) = \max^*[\bar{\alpha}(S_0(t)) + \bar{\gamma}(S_0(t), S_0(t+1)), \bar{\alpha}(S_1(t)) + \bar{\gamma}(S_1(t), S_0(t+1))]. \quad (2.38)$$

**Example 2:** Calculation of  $\bar{\beta}$  in radix-2 algorithm.

Referring to Fig 2.4 to obtain  $\bar{\beta}(S_m(t))$ , we know that there are two paths connected to it. One path is  $S_0(t+1) \xrightarrow{c_t=0} S_0(t)$  and another path is  $S_4(t+1) \xrightarrow{c_t=1} S_0(t)$ . Therefore  $\bar{\gamma}(S_0(t), S_0(t+1))$  and  $\bar{\gamma}(S_0(t), S_4(t+1))$  from (2.33) can be

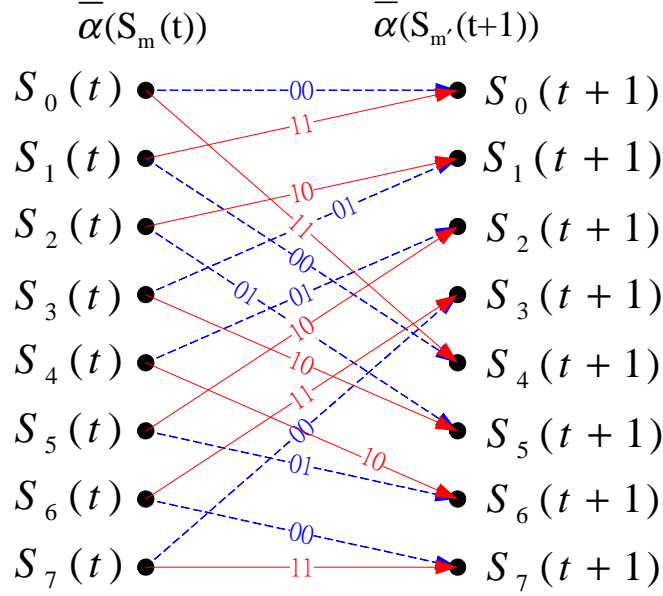


Figure 2.3: The trellis diagram of  $\alpha$ .

expressed as

$$\bar{\gamma}(S_0(t), S_0(t+1)) = \frac{1}{2} \cdot [(-1) \cdot L_a(c_t = -1) + L_c \cdot (r_t(0) \cdot (-1) + r_t(1) \cdot (-1))], \quad (2.39)$$

and

$$\bar{\gamma}(S_0(t), S_4(t+1)) = \frac{1}{2} \cdot [(-1) \cdot L_a(c_t = +1) + L_c \cdot (r_t(0) \cdot (+1) + r_t(1) \cdot (+1))]. \quad (2.40)$$

Substituting (2.39) and (2.40) into (2.27) leads to  $\bar{\beta}(S_0(t))$  as follows

$$\bar{\beta}(S_0(t)) = \max^* [\bar{\beta}(S_0(t+1)) + \bar{\gamma}(S_0(t), S_0(t+1)), \bar{\beta}(S_4(t+1)) + \bar{\gamma}(S_0(t), S_4(t+1))]. \quad (2.41)$$

### 2.3.2 Radix-4 algorithm

In 2003, the radix-4 algorithm was proposed by *M. Bickerstaff* [15] and has commonly used in hardware implementation, since the Radix-4 Log-MAP architecture doubling the throughput for a given clock rate over the radix-2 architecture. In

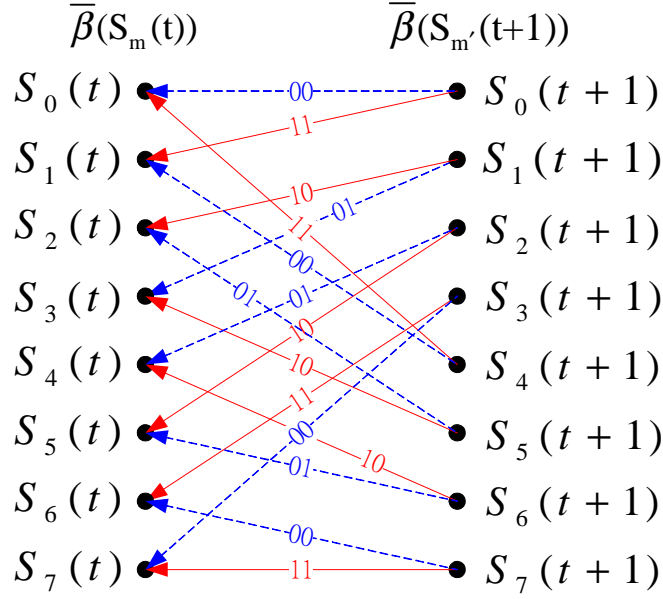


Figure 2.4: The trellis diagram of  $\bar{\beta}$ .

additional, it only calculate the even time stages as shown in Fig. 2.5 (with out calculating stages  $t + 1, t + 3, \dots$ ). Thus it can further reduce the memory.

For convenience, we define the symbols as follows

$$\bar{\alpha}_{t+k}^{m'} \triangleq \bar{\alpha}(S_m(t+k)),$$

$$\bar{\gamma}_{t+u}^{t+u+1}(m, m') \triangleq \bar{\gamma}(S_m(t+u), S_{m'}(t+u+1)),$$

and

$$\bar{\beta}_{t+k}^m \triangleq \bar{\beta}(S_m(t+k)).$$

Let us give an example to derive the recursive units  $\bar{\alpha}$  and  $\bar{\beta}$  for Radix-4 algorithm.

**Example 3:** Calculating  $\bar{\alpha}_{t+2}^0$  and  $\bar{\beta}_t^0$  in Radix-4 algorithm.

Referring to Fig. 2.5,  $\bar{\alpha}_{t+2}^0$  and  $\bar{\beta}_t^0$  are derived as follows

$$\begin{aligned}
\bar{\alpha}_{t+2}^0 &= \max^*[\bar{\alpha}_{t+1}^0 + \bar{\gamma}_{t+1}^{t+2}(0, 0), \bar{\alpha}_{t+1}^1 + \bar{\gamma}_{t+1}^{t+2}(1, 0)] \\
&= \max^*\{\max^*[\bar{\alpha}_t^0 + \bar{\gamma}_t^{t+1}(0, 0), \bar{\alpha}_t^1 + \bar{\gamma}_t^{t+1}(1, 0)] + \bar{\gamma}_{t+1}^{t+2}(0, 0), \\
&\quad \max^*[\bar{\alpha}_t^3 + \bar{\gamma}_t^{t+1}(3, 1), \bar{\alpha}_t^2 + \bar{\gamma}_t^{t+1}(2, 1)] + \bar{\gamma}_{t+1}^{t+2}(1, 0)\} \\
&= \max^*\{\ln[\exp(\bar{\alpha}_t^0 + \bar{\gamma}_t^{t+1}(0, 0)) + \exp(\bar{\alpha}_t^1 + \bar{\gamma}_t^{t+1}(1, 0))] + \ln[\exp(\bar{\gamma}_{t+1}^{t+2}(0, 0))], \\
&\quad \ln[\exp(\bar{\alpha}_t^3 + \bar{\gamma}_t^{t+1}(3, 1)) + \exp(\bar{\alpha}_t^2 + \bar{\gamma}_t^{t+1}(2, 1))] + \ln[\exp(\bar{\gamma}_{t+1}^{t+2}(1, 0))]\} \\
&= \max^*\{\ln[\exp(\bar{\alpha}_t^0 + \bar{\gamma}_t^{t+1}(0, 0) + \bar{\gamma}_{t+1}^{t+2}(0, 0)) + \exp(\bar{\alpha}_t^1 + \bar{\gamma}_t^{t+1}(1, 0) + \bar{\gamma}_{t+1}^{t+2}(0, 0))], \\
&\quad \ln[\exp(\bar{\alpha}_t^3 + \bar{\gamma}_t^{t+1}(3, 1) + \bar{\gamma}_{t+1}^{t+2}(1, 0)) + \exp(\bar{\alpha}_t^2 + \bar{\gamma}_t^{t+1}(2, 1) + \bar{\gamma}_{t+1}^{t+2}(1, 0))]\} \\
&= \max^*\{\max^*[\bar{\alpha}_t^0 + \bar{\gamma}_t^{t+1}(0, 0) + \bar{\gamma}_{t+1}^{t+2}(0, 0), \bar{\alpha}_t^1 + \bar{\gamma}_t^{t+1}(1, 0) + \bar{\gamma}_{t+1}^{t+2}(0, 0)], \\
&\quad \max^*[\bar{\alpha}_t^3 + \bar{\gamma}_t^{t+1}(3, 1) + \bar{\gamma}_{t+1}^{t+2}(1, 0), \bar{\alpha}_t^2 + \bar{\gamma}_t^{t+1}(2, 1) + \bar{\gamma}_{t+1}^{t+2}(1, 0)]\}, \\
&\hspace{15em} (2.42)
\end{aligned}$$

and

$$\begin{aligned}
\bar{\beta}_t^0 &= \max^*[\bar{\gamma}_t^{t+1}(0, 4) + \bar{\beta}_{t+1}^4, \bar{\gamma}_t^{t+1}(0, 0) + \bar{\beta}_{t+1}^0] \\
&= \max^*\{\bar{\gamma}_t^{t+1}(0, 4) + \max^*[\bar{\gamma}_{t+1}^{t+2}(4, 2) + \bar{\beta}_{t+2}^2, \bar{\gamma}_{t+1}^{t+2}(4, 6) + \bar{\beta}_{t+2}^6], \\
&\quad \bar{\gamma}_t^{t+1}(0, 0) + \max^*[\bar{\gamma}_{t+1}^{t+2}(0, 0) + \bar{\beta}_{t+2}^0, \bar{\gamma}_{t+1}^{t+2}(0, 4) + \bar{\beta}_{t+2}^4]\} \\
&= \max^*\{\ln[\exp(\bar{\gamma}_t^{t+1}(0, 4))] + \ln[\exp(\bar{\gamma}_{t+1}^{t+2}(4, 2) + \bar{\beta}_{t+2}^2) + \exp(\bar{\gamma}_{t+1}^{t+2}(4, 6) + \bar{\beta}_{t+2}^6)], \\
&\quad \ln[\exp(\bar{\gamma}_t^{t+1}(0, 0))] + \ln[\exp(\bar{\gamma}_{t+1}^{t+2}(0, 0) + \bar{\beta}_{t+2}^0) + \exp(\bar{\gamma}_{t+1}^{t+2}(0, 4) + \bar{\beta}_{t+2}^4)]\} \\
&= \max^*\{\ln[\exp(\bar{\gamma}_t^{t+1}(0, 4) + \bar{\gamma}_{t+1}^{t+2}(4, 2) + \bar{\beta}_{t+2}^2)] + \ln[\exp(\bar{\gamma}_t^{t+1}(0, 4) + \bar{\gamma}_{t+1}^{t+2}(4, 6) + \bar{\beta}_{t+2}^6)], \\
&\quad \ln[\exp(\bar{\gamma}_t^{t+1}(0, 0) + \bar{\gamma}_{t+1}^{t+2}(0, 0) + \bar{\beta}_{t+2}^0)] + \ln[\exp(\bar{\gamma}_t^{t+1}(0, 0) + \bar{\gamma}_{t+1}^{t+2}(0, 4) + \bar{\beta}_{t+2}^4)]\} \\
&= \max^*\{\max^*[\bar{\gamma}_t^{t+1}(0, 4) + \bar{\gamma}_{t+1}^{t+2}(4, 2) + \bar{\beta}_{t+2}^2, \bar{\gamma}_t^{t+1}(0, 4) + \bar{\gamma}_{t+1}^{t+2}(4, 6) + \bar{\beta}_{t+2}^6], \\
&\quad \max^*[\bar{\gamma}_t^{t+1}(0, 0) + \bar{\gamma}_{t+1}^{t+2}(0, 0) + \bar{\beta}_{t+2}^0, \bar{\gamma}_t^{t+1}(0, 0) + \bar{\gamma}_{t+1}^{t+2}(0, 4) + \bar{\beta}_{t+2}^4]\}. \\
&\hspace{15em} (2.43)
\end{aligned}$$

### 2.3.3 Deriving LLR for Radix-4

Because we use radix-4 algorithm to decode two stages of soft information in one clock cycle, we need to use two LLR units. Let us derive LLR for stage  $t$  and stage  $t + 1$ . Define  $LLR_t^b$  as the LLR for  $c_t = b$ , where  $b \in (0, 1)$ . For instance,



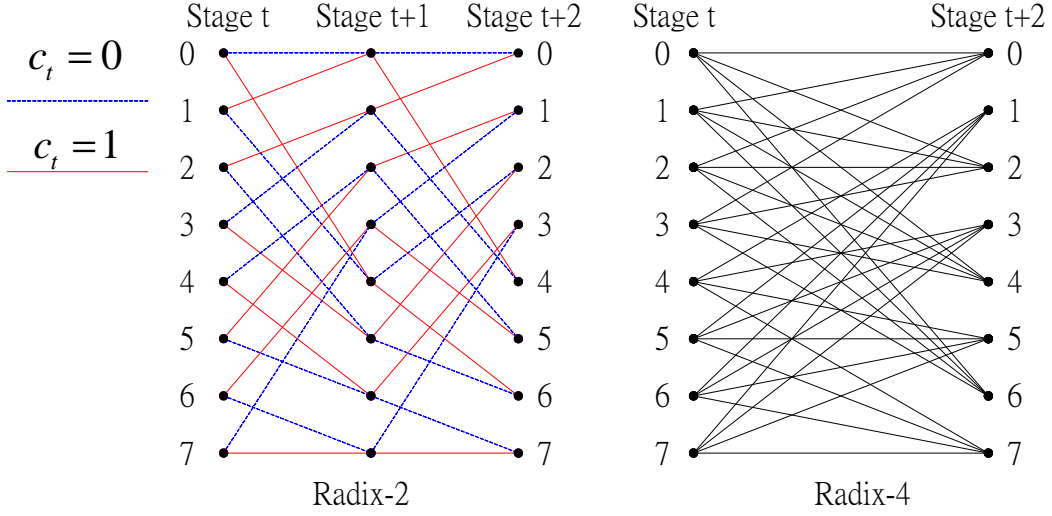


Figure 2.5: The Radix-2 and Radix-4 trellis diagram.

$LLR_t^1$  denotes the LLR for  $c_t = 1$ , and  $LLR_{t+1}^0$  denotes the LLR for  $c_{t+1} = 0$ . Below, we derive the LLR for stage  $t$ . Referring to Fig. 2.6 and Fig. 2.7, we have

$$\begin{aligned}
LLR_t^1 = \widehat{max}(\bar{\alpha}_t^0 + \bar{\gamma}_t^{t+1}(0, 4) + \bar{\beta}_{t+1}^4, & \bar{\alpha}_t^1 + \bar{\gamma}_t^{t+1}(1, 0) + \bar{\beta}_{t+1}^0, \\
& \bar{\alpha}_t^2 + \bar{\gamma}_t^{t+1}(2, 1) + \bar{\beta}_{t+1}^1, \bar{\alpha}_t^3 + \bar{\gamma}_t^{t+1}(3, 5) + \bar{\beta}_{t+1}^5, \\
& \bar{\alpha}_t^4 + \bar{\gamma}_t^{t+1}(4, 6) + \bar{\beta}_{t+1}^6, \bar{\alpha}_t^5 + \bar{\gamma}_t^{t+1}(5, 2) + \bar{\beta}_{t+1}^2, \\
& \bar{\alpha}_t^6 + \bar{\gamma}_t^{t+1}(6, 3) + \bar{\beta}_{t+1}^3, \bar{\alpha}_t^7 + \bar{\gamma}_t^{t+1}(7, 7) + \bar{\beta}_{t+1}^7), \quad (2.44)
\end{aligned}$$

Because in radix-4 algorithm we do not calculate  $\bar{\beta}_{t+1}$ , we should replace  $\bar{\beta}_{t+1}^i$ ,  $0 \leq i \leq 7$ , by the trace-back values from stage  $t+2$  and rewrite (2.44) as follows

$$\begin{aligned}
LLR_t^1 = \widehat{max}(\bar{\alpha}_t^0 + \bar{\gamma}_t^{t+1}(0, 4) + \max^*(\bar{\gamma}_{t+1}^{t+2}(4, 2) + \bar{\beta}_{t+2}^2, \bar{\gamma}_{t+1}^{t+2}(4, 6) + \bar{\beta}_{t+2}^6), \\
\bar{\alpha}_t^1 + \bar{\gamma}_t^{t+1}(1, 0) + \max^*(\bar{\gamma}_{t+1}^{t+2}(0, 0) + \bar{\beta}_{t+2}^0, \bar{\gamma}_{t+1}^{t+2}(0, 4) + \bar{\beta}_{t+2}^4), \\
\bar{\alpha}_t^2 + \bar{\gamma}_t^{t+1}(2, 1) + \max^*(\bar{\gamma}_{t+1}^{t+2}(1, 0) + \bar{\beta}_{t+2}^0, \bar{\gamma}_{t+1}^{t+2}(1, 4) + \bar{\beta}_{t+2}^4), \\
\bar{\alpha}_t^3 + \bar{\gamma}_t^{t+1}(3, 5) + \max^*(\bar{\gamma}_{t+1}^{t+2}(5, 2) + \bar{\beta}_{t+2}^2, \bar{\gamma}_{t+1}^{t+2}(5, 6) + \bar{\beta}_{t+2}^6), \\
\bar{\alpha}_t^4 + \bar{\gamma}_t^{t+1}(4, 6) + \max^*(\bar{\gamma}_{t+1}^{t+2}(6, 3) + \bar{\beta}_{t+2}^3, \bar{\gamma}_{t+1}^{t+2}(6, 7) + \bar{\beta}_{t+2}^7), \\
\bar{\alpha}_t^5 + \bar{\gamma}_t^{t+1}(5, 2) + \max^*(\bar{\gamma}_{t+1}^{t+2}(2, 1) + \bar{\beta}_{t+2}^1, \bar{\gamma}_{t+1}^{t+2}(2, 5) + \bar{\beta}_{t+2}^5), \\
\bar{\alpha}_t^6 + \bar{\gamma}_t^{t+1}(6, 3) + \max^*(\bar{\gamma}_{t+1}^{t+2}(3, 1) + \bar{\beta}_{t+2}^1, \bar{\gamma}_{t+1}^{t+2}(3, 5) + \bar{\beta}_{t+2}^5), \\
\bar{\alpha}_t^7 + \bar{\gamma}_t^{t+1}(7, 7) + \max^*(\bar{\gamma}_{t+1}^{t+2}(7, 3) + \bar{\beta}_{t+2}^3, \bar{\gamma}_{t+1}^{t+2}(7, 7) + \bar{\beta}_{t+2}^7)).
\end{aligned} \tag{2.45}$$

Similarly  $LLR_t^1$  and  $LLR_t^0$  can be shown as follows

$$\begin{aligned}
LLR_t^0 = \widehat{max}(\bar{\alpha}_t^0 + \bar{\gamma}_t^{t+1}(0, 0) + \max^*(\bar{\gamma}_{t+1}^{t+2}(0, 0) + \bar{\beta}_{t+2}^0, \bar{\gamma}_{t+1}^{t+2}(0, 4) + \bar{\beta}_{t+2}^4), \\
\bar{\alpha}_t^1 + \bar{\gamma}_t^{t+1}(1, 4) + \max^*(\bar{\gamma}_{t+1}^{t+2}(4, 2) + \bar{\beta}_{t+2}^2, \bar{\gamma}_{t+1}^{t+2}(4, 6) + \bar{\beta}_{t+2}^6), \\
\bar{\alpha}_t^2 + \bar{\gamma}_t^{t+1}(2, 5) + \max^*(\bar{\gamma}_{t+1}^{t+2}(5, 2) + \bar{\beta}_{t+2}^2, \bar{\gamma}_{t+1}^{t+2}(5, 6) + \bar{\beta}_{t+2}^6), \\
\bar{\alpha}_t^3 + \bar{\gamma}_t^{t+1}(3, 1) + \max^*(\bar{\gamma}_{t+1}^{t+2}(1, 0) + \bar{\beta}_{t+2}^0, \bar{\gamma}_{t+1}^{t+2}(1, 4) + \bar{\beta}_{t+2}^4), \\
\bar{\alpha}_t^4 + \bar{\gamma}_t^{t+1}(4, 2) + \max^*(\bar{\gamma}_{t+1}^{t+2}(2, 1) + \bar{\beta}_{t+2}^1, \bar{\gamma}_{t+1}^{t+2}(2, 5) + \bar{\beta}_{t+2}^5), \\
\bar{\alpha}_t^5 + \bar{\gamma}_t^{t+1}(5, 6) + \max^*(\bar{\gamma}_{t+1}^{t+2}(6, 3) + \bar{\beta}_{t+2}^3, \bar{\gamma}_{t+1}^{t+2}(6, 7) + \bar{\beta}_{t+2}^7), \\
\bar{\alpha}_t^6 + \bar{\gamma}_t^{t+1}(6, 7) + \max^*(\bar{\gamma}_{t+1}^{t+2}(7, 3) + \bar{\beta}_{t+2}^3, \bar{\gamma}_{t+1}^{t+2}(7, 7) + \bar{\beta}_{t+2}^7), \\
\bar{\alpha}_t^7 + \bar{\gamma}_t^{t+1}(7, 3) + \max^*(\bar{\gamma}_{t+1}^{t+2}(3, 1) + \bar{\beta}_{t+2}^1, \bar{\gamma}_{t+1}^{t+2}(3, 5) + \bar{\beta}_{t+2}^5).
\end{aligned} \tag{2.46}$$

Finally, the LLR at stage  $t$  can be expressed as

$$LLR_t = LLR_t^1 - LLR_t^0. \tag{2.47}$$

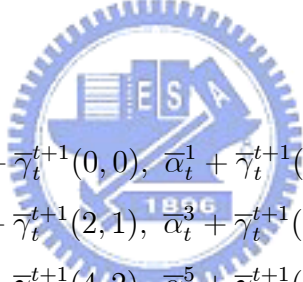
Similarly, the LLR at stage  $t+1$  can be expressed as

$$LLR_{t+1} = LLR_{t+1}^1 - LLR_{t+1}^0, \tag{2.48}$$

where

$$\begin{aligned}
LLR_{t+1}^1 = \widehat{max}( & max^*(\bar{\alpha}_t^0 + \bar{\gamma}_t^{t+1}(0, 0), \bar{\alpha}_t^1 + \bar{\gamma}_t^{t+1}(1, 0)) + \bar{\gamma}_{t+1}^{t+2}(0, 4) + \bar{\beta}_{t+2}^4, \\
& max^*(\bar{\alpha}_t^2 + \bar{\gamma}_t^{t+1}(2, 1), \bar{\alpha}_t^3 + \bar{\gamma}_t^{t+1}(3, 1)) + \bar{\gamma}_{t+1}^{t+2}(1, 0) + \bar{\beta}_{t+2}^0, \\
& max^*(\bar{\alpha}_t^4 + \bar{\gamma}_t^{t+1}(4, 2), \bar{\alpha}_t^5 + \bar{\gamma}_t^{t+1}(5, 2)) + \bar{\gamma}_{t+1}^{t+2}(2, 1) + \bar{\beta}_{t+2}^1, \\
& max^*(\bar{\alpha}_t^6 + \bar{\gamma}_t^{t+1}(6, 3), \bar{\alpha}_t^7 + \bar{\gamma}_t^{t+1}(7, 3)) + \bar{\gamma}_{t+1}^{t+2}(3, 5) + \bar{\beta}_{t+2}^5, \\
& max^*(\bar{\alpha}_t^0 + \bar{\gamma}_t^{t+1}(0, 4), \bar{\alpha}_t^1 + \bar{\gamma}_t^{t+1}(1, 4)) + \bar{\gamma}_{t+1}^{t+2}(4, 6) + \bar{\beta}_{t+2}^6, \\
& max^*(\bar{\alpha}_t^2 + \bar{\gamma}_t^{t+1}(2, 5), \bar{\alpha}_t^3 + \bar{\gamma}_t^{t+1}(3, 5)) + \bar{\gamma}_{t+1}^{t+2}(5, 2) + \bar{\beta}_{t+2}^2, \\
& max^*(\bar{\alpha}_t^4 + \bar{\gamma}_t^{t+1}(4, 6), \bar{\alpha}_t^5 + \bar{\gamma}_t^{t+1}(5, 6)) + \bar{\gamma}_{t+1}^{t+2}(6, 3) + \bar{\beta}_{t+2}^3, \\
& max^*(\bar{\alpha}_t^6 + \bar{\gamma}_t^{t+1}(6, 7), \bar{\alpha}_t^7 + \bar{\gamma}_t^{t+1}(7, 7)) + \bar{\gamma}_{t+1}^{t+2}(7, 7) + \bar{\beta}_{t+2}^7),
\end{aligned} \tag{2.49}$$

and



$$\begin{aligned}
LLR_{t+1}^0 = \widehat{max}( & max^*(\bar{\alpha}_t^0 + \bar{\gamma}_t^{t+1}(0, 0), \bar{\alpha}_t^1 + \bar{\gamma}_t^{t+1}(1, 0)) + \bar{\gamma}_{t+1}^{t+2}(0, 0) + \bar{\beta}_{t+2}^0, \\
& max^*(\bar{\alpha}_t^2 + \bar{\gamma}_t^{t+1}(2, 1), \bar{\alpha}_t^3 + \bar{\gamma}_t^{t+1}(3, 1)) + \bar{\gamma}_{t+1}^{t+2}(1, 4) + \bar{\beta}_{t+2}^4, \\
& max^*(\bar{\alpha}_t^4 + \bar{\gamma}_t^{t+1}(4, 2), \bar{\alpha}_t^5 + \bar{\gamma}_t^{t+1}(5, 2)) + \bar{\gamma}_{t+1}^{t+2}(2, 5) + \bar{\beta}_{t+2}^5, \\
& max^*(\bar{\alpha}_t^6 + \bar{\gamma}_t^{t+1}(6, 3), \bar{\alpha}_t^7 + \bar{\gamma}_t^{t+1}(7, 3)) + \bar{\gamma}_{t+1}^{t+2}(3, 1) + \bar{\beta}_{t+2}^1, \\
& max^*(\bar{\alpha}_t^0 + \bar{\gamma}_t^{t+1}(0, 4), \bar{\alpha}_t^1 + \bar{\gamma}_t^{t+1}(1, 4)) + \bar{\gamma}_{t+1}^{t+2}(4, 2) + \bar{\beta}_{t+2}^2, \\
& max^*(\bar{\alpha}_t^2 + \bar{\gamma}_t^{t+1}(2, 5), \bar{\alpha}_t^3 + \bar{\gamma}_t^{t+1}(3, 5)) + \bar{\gamma}_{t+1}^{t+2}(5, 6) + \bar{\beta}_{t+2}^6, \\
& max^*(\bar{\alpha}_t^4 + \bar{\gamma}_t^{t+1}(4, 6), \bar{\alpha}_t^5 + \bar{\gamma}_t^{t+1}(5, 6)) + \bar{\gamma}_{t+1}^{t+2}(6, 7) + \bar{\beta}_{t+2}^7, \\
& max^*(\bar{\alpha}_t^6 + \bar{\gamma}_t^{t+1}(6, 7), \bar{\alpha}_t^7 + \bar{\gamma}_t^{t+1}(7, 7)) + \bar{\gamma}_{t+1}^{t+2}(7, 3) + \bar{\beta}_{t+2}^3).
\end{aligned} \tag{2.50}$$

## 2.4 Decoding Architecture

The decoder consists of two identical SISO decoders, interleavers and de-interleavers. The architecture is shown in Fig. 2.8. Applying (2.19) and (2.33), the SISO decoder can be derived as follows

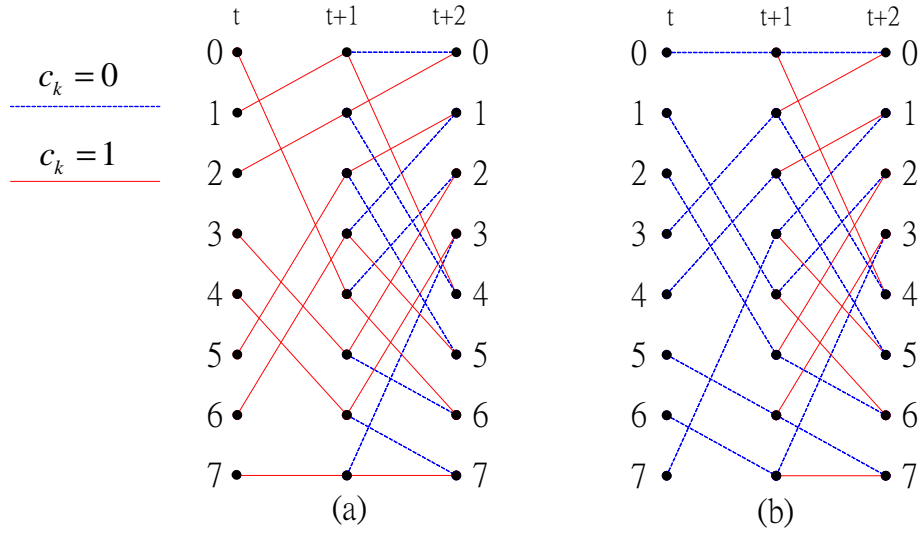


Figure 2.6: The trellis diagram of LLR unit for stage  $t$  : (a)  $LLR_t^1$  and (b)  $LLR_t^0$ .

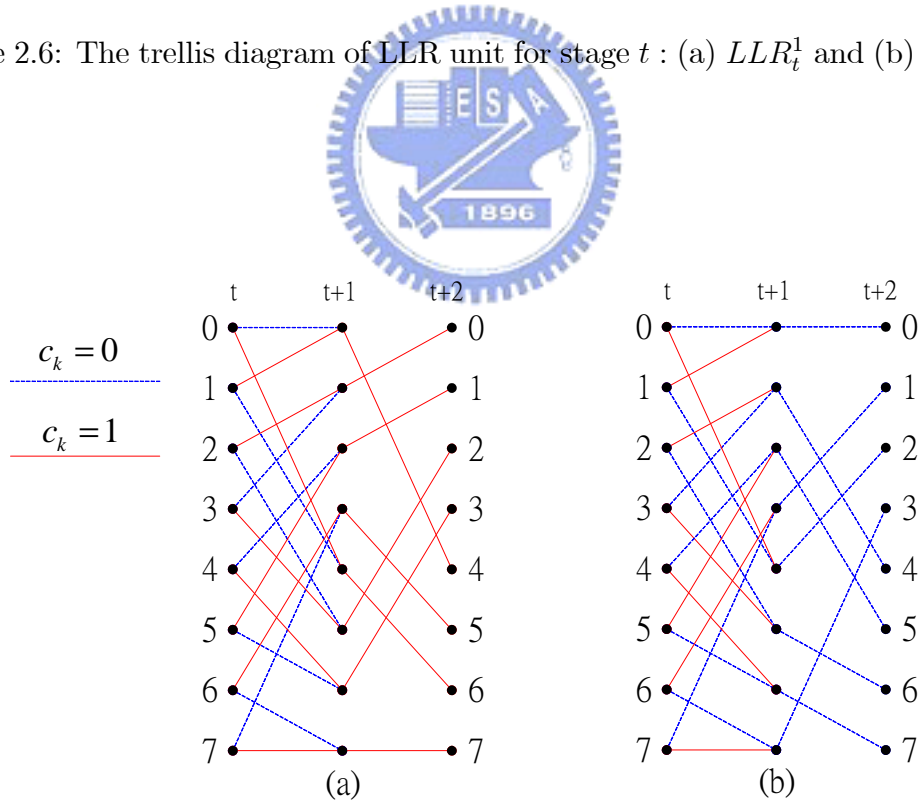


Figure 2.7: The trellis diagram of LLR unit for stage  $t + 1$  : (a)  $LLR_{t+1}^1$  and (b)  $LLR_{t+1}^0$ .

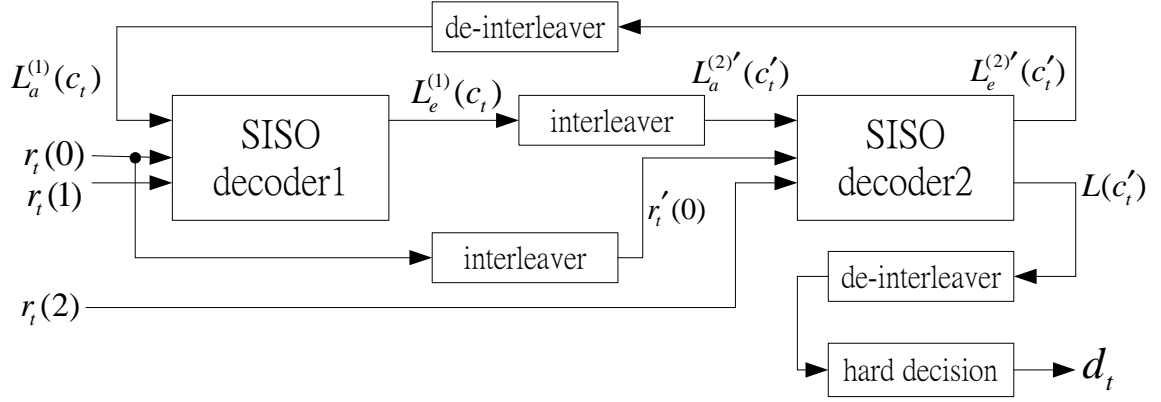


Figure 2.8: An architecture of the turbo decoder.

$$\begin{aligned}
L(c_t) &= \ln \frac{\sum_{S(m,m) \in c_t^{+1}} e^{\bar{\alpha}(S_m(t)) + \bar{\gamma}(S_m(t), S_m(t+1)) + \bar{\beta}(S_m(t+1))}}{\sum_{S(m,m) \in c_t^{-1}} e^{\bar{\alpha}(S_m(t)) + \bar{\gamma}(S_m(t), S_m(t+1)) + \bar{\beta}(S_m(t+1))}} \\
&= \ln \frac{\sum_{S(m,m) \in c_t^{+1}} e^{\frac{1}{2} \cdot [(+1) \cdot L_a(c_t) + L_c \cdot r_t(0) \cdot (1)]} \cdot e^{\bar{\alpha}(S_m(t)) + \frac{1}{2} \cdot \sum_{i=1}^{n-1} L_c \cdot r_t(i) w_t(i) + \bar{\beta}(S_m(t+1))}}{\sum_{S(m,m) \in c_t^{-1}} e^{\frac{1}{2} \cdot [(-1) \cdot L_a(c_t) + L_c \cdot r_t(0) \cdot (-1)]} \cdot e^{\bar{\alpha}(S_m(t)) + \frac{1}{2} \cdot \sum_{i=1}^{n-1} L_c \cdot r_t(i) w_t(i) + \bar{\beta}(S_m(t+1))}} \\
&= L_a(c_t) + L_c r_t(0) + \ln \frac{\sum_{S(m,m) \in c_t^{+1}} e^{\bar{\alpha}(S_m(t)) + \frac{1}{2} \cdot \sum_{i=1}^{n-1} L_c \cdot r_t(i) w_t(i) + \bar{\beta}(S_m(t+1))}}{\sum_{S(m,m) \in c_t^{-1}} e^{\bar{\alpha}(S_m(t)) + \frac{1}{2} \cdot \sum_{i=1}^{n-1} L_c \cdot r_t(i) w_t(i) + \bar{\beta}(S_m(t+1))}} \\
&= L_a(c_t) + L_c r_t(0) + L_e(c_t), \tag{2.51}
\end{aligned}$$

where the  $L_e(c_t)$  are called extrinsic information corresponding to  $c_t$ . The  $L_e(c_t)$  can be obtained from (2.51) as follows

$$L_e(c_t) = L(c_t) - [L_a(c_t) + L_c r_t(0)]. \tag{2.52}$$

$r_t(0)$ ,  $r_t(1)$  and  $r_t(2)$  are the transmitter signals  $x_t$ ,  $z_t$  and  $z'_t$  after passing AWGN channel respectively. Initially, we set the a priori information  $L_a^{(1)}(c_t)$  for the first SISO decoder to zero. Then  $r_t(0)$ ,  $r_t(1)$ , and  $L_a^{(1)}(c_t)$  are passed into the first SISO decoder and obtain the extrinsic information  $L_e^{(1)}(c_t)$  that can offer the next SISO decoder more accurate information for  $L_a^{(2)}(c_t)$ . After  $L_e^{(1)}(c_t)$  and  $r_0(t)$  pass the interleaver, we obtain the signal  $L_a^{(2)'}(c'_t)$  and  $r_0'(t)$  respectively. Similarly, the second SISO decoder can generate  $L_e^{(2)}(c'_t)$  and  $L(c'_t)$  after the  $L_a^{(2)'}(c'_t)$ ,  $r_t'(0)$

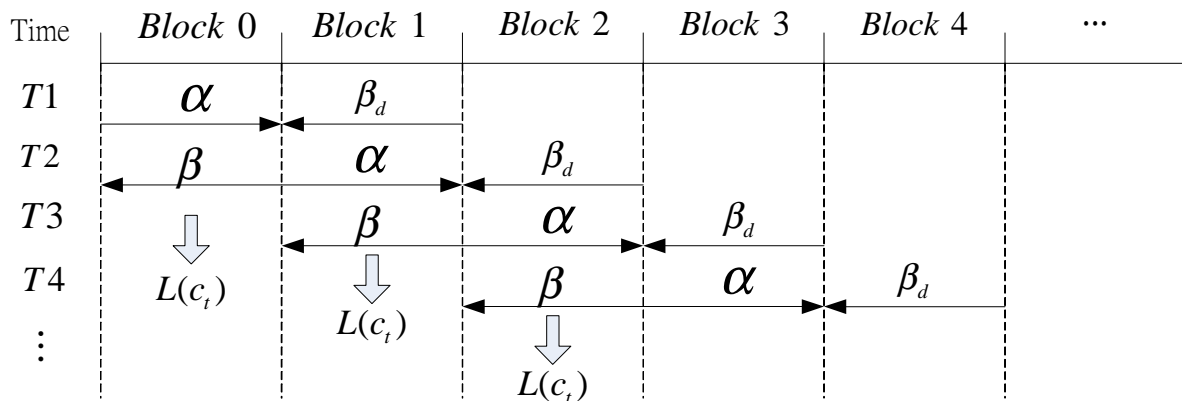


Figure 2.9: The sliding window diagram.

and  $r_t(2)$  pass it. When the maximum allowable number of iterations is reached,  $d_t$  can be obtained by de-interleaving  $L(\hat{c}_t)$  and then take hard decision.

### 2.4.1 Sliding Window

Theoretically, we need to calculate the LLR according to the whole received data in a block. However, when  $N$  is large, it is impractical to implement this ideal in a hardware since we need large memory and latency in this case. To reduce the memory and latency, a sliding window can be adopted [16]. In Fig. 2.9, the received data stream is divided into  $n$  blocks, the dummy backward recursion  $\beta_d$  set initial value equal to  $\log(\frac{1}{\text{Number of the states}})$ . When a block is computed, the value  $\beta_d$  is fed to  $\beta$  for initial boundary value. The larger the block length is, the more accurate  $\beta$  we will obtain. As soon as  $\beta_d$  is ready for a specified received data, we can obtain its corresponding  $L(c_t)$ . At the same time, we can calculate  $\alpha$  for the next blocks. Repeat the same procedure, all of the data can be decoded.

# Chapter 3

## Radix-4 Recursive Architecture

### 3.1 Conventional Architecture

We derived  $\bar{\alpha}_{t+2}^0$  in (2.42), however we must take recursive value of  $(\bar{\alpha}, \bar{\beta})$  approximately (i.e. replace some  $\max^*\{\cdot\}$  with  $\max\{\cdot\}$ ) in hardware implementation. In [15], Lucent Bell Labs proposed the following approximation

$$\begin{aligned} \bar{\alpha}_{t+2}^0 \approx \max^* \{ & \max[\bar{\alpha}_t^0 + \bar{\gamma}_t^{t+1}(0, 0) + \bar{\gamma}_{t+1}^{t+2}(0, 0), \bar{\alpha}_t^1 + \bar{\gamma}_t^{t+1}(1, 0) + \bar{\gamma}_{t+1}^{t+2}(0, 0)], \\ & \max[\bar{\alpha}_t^3 + \bar{\gamma}_t^{t+1}(3, 1) + \bar{\gamma}_{t+1}^{t+2}(1, 0), \bar{\alpha}_t^2 + \bar{\gamma}_t^{t+1}(2, 1) + \bar{\gamma}_{t+1}^{t+2}(1, 0)] \}. \end{aligned} \quad (3.1)$$

Fig. 3.1 is a radix-4 architecture for (3.1), the critical path (with dash line) consists of four multi-bit additions, one 2-to-1 MUX and one LUT, where the LUT is implemented using look-up table for correction term. In next section, we proposed an architecture for LUT which can reduce the critical path.

### 3.2 Proposed Architecture

Because the hardware of the recursive unit in radix-4 is more complicated than that of radix-2, the critical path is too long so that it cannot achieve exactly twice of the throughput over radix-2. Thus our goal is to reach twice of the throughput while the area is as small as possible. From [20], Z. Wang proposed an architecture for high speed recursion and approximation for  $\bar{\alpha}$  and  $\bar{\beta}$  shown

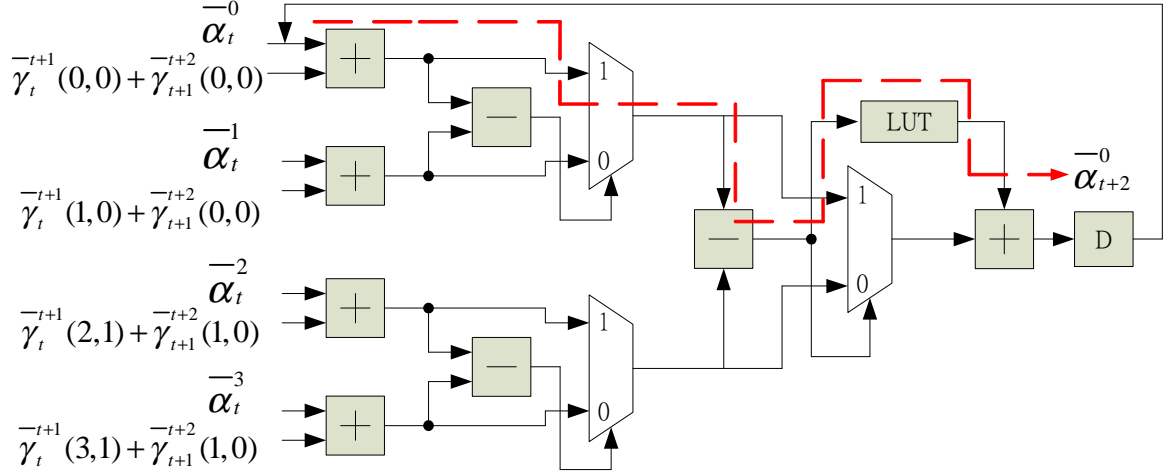


Figure 3.1: Conventional Radix-4 Architecture.

as follows

$$\begin{aligned} \bar{\alpha}_{t+2}^0 \approx \max\{ & \max^*[\bar{\alpha}_t^0 + \bar{\gamma}_t^{t+1}(0,0) + \bar{\gamma}_{t+1}^{t+2}(0,0), \bar{\alpha}_t^1 + \bar{\gamma}_t^{t+1}(1,0) + \bar{\gamma}_{t+1}^{t+2}(0,0)], \\ & \max^*[\bar{\alpha}_t^3 + \bar{\gamma}_t^{t+1}(3,1) + \bar{\gamma}_{t+1}^{t+2}(1,0), \bar{\alpha}_t^2 + \bar{\gamma}_t^{t+1}(2,1) + \bar{\gamma}_{t+1}^{t+2}(1,0)]\}, \end{aligned} \quad (3.2)$$

and

$$\begin{aligned} \bar{\beta}_t^0 = \max^*\{ & \max^*[\bar{\gamma}_t^{t+1}(0,4) + \bar{\gamma}_{t+1}^{t+2}(4,2) + \bar{\beta}_{t+2}^2], \bar{\gamma}_t^{t+1}(0,4) + \bar{\gamma}_{t+1}^{t+2}(4,6) + \bar{\beta}_{t+2}^6], \\ & \max^*[\bar{\gamma}_t^{t+1}(0,0) + \bar{\gamma}_{t+1}^{t+2}(0,0) + \bar{\beta}_{t+2}^0, \bar{\gamma}_t^{t+1}(0,0) + \bar{\gamma}_{t+1}^{t+2}(0,4) + \bar{\beta}_{t+2}^4]\}. \end{aligned} \quad (3.3)$$

Applying (3.2) to the hardware in Fig. 3.2. At the first stage we use carry save adder (CSA) to convert three additions to two additions. At last stage of Fig. 3.1,  $\bar{\alpha}_{t+2}^0$  is divided into two parts as in Fig. 3.2. One is  $\max(\cdot)$  value and the other is the correction term that both are removed to the first stage adder, which is called Offset-Add-Compare-Select (OACS) operation [19]. Conventionally, in LUT block we need to take a absolute value and then take table look-up. Therefore the computation time of the LUT is larger than the MUX marked with dash line. Hence the LUT dominates the critical path. In [20] a method was proposed to reduce computation time of the LUT. The computation



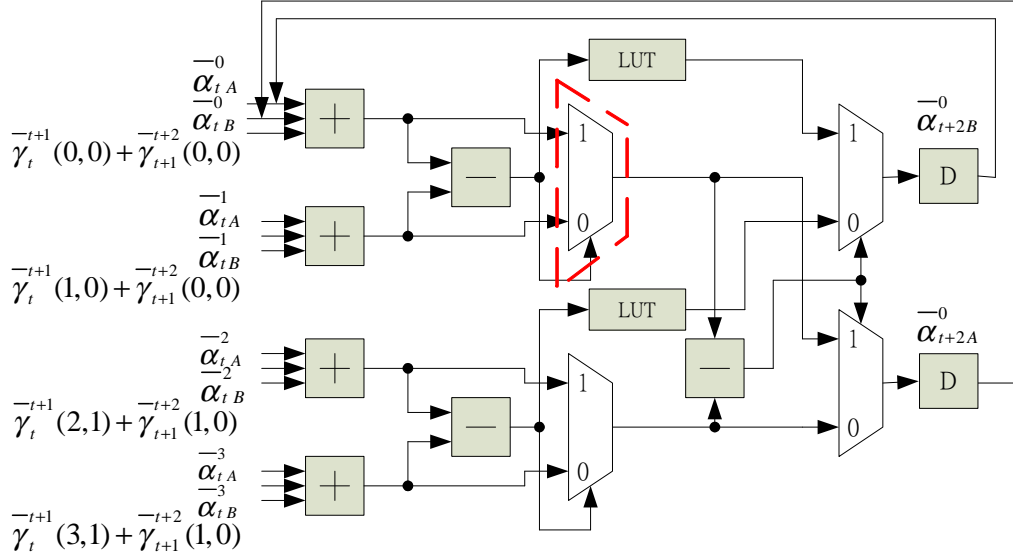


Figure 3.2: Radix-4 recursion architecture of [20]

time is reduced since it does not need to perform absolute operation. Let us explain as follows.

In Fig. 3.3, assume the input of the LUT is a  $n$ -bits sign number, the  $g(\cdot)$  function is used to detect the absolute value of the input which is less than 2.0, i.e.  $z = 1$  if the input is less than 2.0. The ELUT block is a small LUT with 3-bits input and 2-bits output. It is used to simplify the logic design. Table 3.1 shows the LUT approximation where  $x$  and  $g(x)$  are the quantized input and output of ELUT. The final output of the LUT is  $d_1$  and  $d_0$ . The general form of  $z$  can be derived as follows:

$$z = \overline{b_{n-1}} \cdot \overline{b_{n-2} + b_{n-3}, \dots, +b_3} + f(b_{n-1} = 1, b_{n-2}, \dots, b_1, b_0),$$

$$d_1 = z \cdot c_1$$

and

$$d_0 = z \cdot c_0,$$

where  $f(\cdot)$  is a combination circuit that consists of  $b_{n-2}, \dots, b_1, b_0$  when  $b_{n-1} = 1$ .

The proposed structure of LUT is in Fig. 3.4. The inputs of the LUT are quantized to integer number and then use simple logic design to obtain the output.

$ x $	0.0	0.5	1.0	1.5	$\geq 2$
$g(x)$	0.75	0.5	0.25	0.25	0

Table 3.1: Approximation of [20].

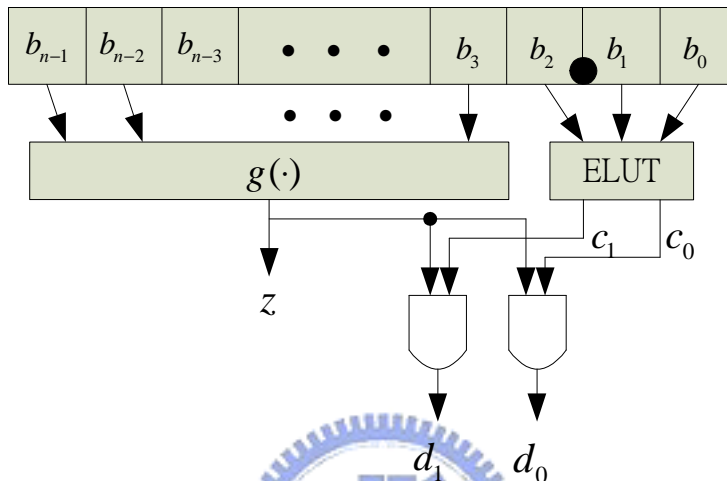


Figure 3.3: Architecture of the proposed LUT used in [20].

In Table 3.2, we observe the dynamic range of the input can be divided as  $[-1 -0.25]$ ,  $[-2 -1.25]$ ,  $[0 0.75]$  and  $[1 1.75]$  and use combination logic to obtain the output  $p_1$  and  $p_0$ , which is independent of  $b_1$  and  $b_0$ . Therefore we do not need to take care of part of the input signal, and the logic gate can somewhat be. The performance comparison for various algorithm is as shown in Fig. 3.5, we see the proposed method and Arch-Z achieve nearly the same performance. The approximated values of the proposed method are given by

$$u(v) = \begin{cases} 0.5, & -1 \leq v < 1 \\ 0.25, & -2 \leq v < -1 \text{ or } 1 \leq v < 2 \\ 0, & \text{otherwise} \end{cases}, \quad (3.4)$$

where  $v$  and  $u(v)$  are the input and output of the proposed LUT. In (3.4), we eliminate that  $u(v)$  equal to 0.75 (compared to Table 3.1), if we consider the case that  $u(v)$  equal to 0.75, we find the output  $p_0$  is dependent of  $b_1$  and  $b_0$ . Thus we ignore the value 0.75 of  $u(v)$ . In section 3.3, the comparison of the proposed LUT and the LUT is shown in Table 3.1. The BER performance with the two

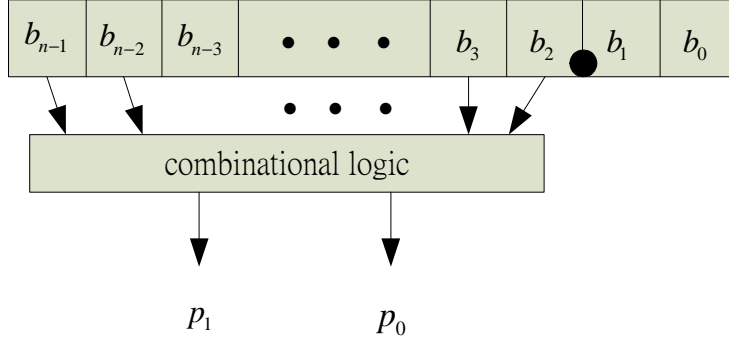


Figure 3.4: Architecture of the proposed LUT.

LUT are nearly same. In Fig. 3.4,  $p_1$  and  $p_0$  are used to simplify the combination logic shown as follows

$$p_1 = b_{n-1} \cdot \dots \cdot b_3 \cdot b_2 + \overline{b_{n-1}} \cdot \dots \cdot \overline{b_3} \cdot \overline{b_2}$$

and

$$p_0 = b_{n-1} \cdot \dots \cdot b_3 \cdot \overline{b_2} + \overline{b_{n-1}} \cdot \dots \cdot \overline{b_3} \cdot b_2.$$

We give an example to compare the two LUTs as follows

**Example 4:** Refer to Table. 3.2, Fig. 3.3 and Fig. 3.4, assume the input bit-length of the LUT is 13, and that of the output is 2 . We define the notations as follows

$b[12 : 0]$  : input of the LUT which is 13-bit with the 2 LSBs be the fractional bits.

$[c_1 c_0]$  : output of the ELUT consisting of simple combination.

$[d_1 d_0]$  : output of the LUT in [20].

$[p_1 p_0]$  : output of the LUT in the proposed method.

(1). The method in [20]:  $z$  is given by

$$z = \overline{b_{12}} \cdot \overline{b_{11}} \cdot \dots \cdot \overline{b_4} \cdot b_3 + b_{12} \cdot b_{11} \cdot \dots \cdot b_4 \cdot b_3 \cdot (b_2 + b_1 + b_0),$$

$$c_1 = \overline{b_{12} + b_2} + b_{13} \cdot b_2 \cdot (b_1 + b_0),$$

and

$$c_0 = (b_{12} + b_2) \cdot \overline{b_{12} \cdot b_2} \cdot (b_1 \oplus b_0).$$

The out results of the LUT is

$$d_1 = z \cdot c_1,$$

and

$$d_0 = z \cdot c_0.$$

(2). The proposed method: the output  $[p_1 \ p_0]$  of the LUT is as follows

$$p_1 = b_{12} \cdot b_{11} \cdot \dots \cdot b_3 \cdot b_2 + \overline{b_{12}} \cdot \overline{b_{11}} \cdot \dots \cdot \overline{b_3} \cdot \overline{b_2}, \quad (3.5)$$

and

$$p_0 = b_{12} \cdot b_{11} \cdot \dots \cdot b_3 \cdot \overline{b_2} + \overline{b_{12}} \cdot \overline{b_{11}} \cdot \dots \cdot \overline{b_3} \cdot b_2. \quad (3.6)$$

From (3.5) and (3.6), we find that  $p_1$  and  $p_0$  have common terms and the logic can share the common terms to reduce complexity. Define the following two terms:

$$COMB_1 = b_{12} \cdot b_{11} \cdot \dots \cdot b_3$$

and

$$COMB_2 = \overline{b_{12}} \cdot \overline{b_{11}} \cdot \dots \cdot \overline{b_3} = \overline{b_{12} + b_{11} + \dots + b_3}.$$

Rearrange (3.5) and (3.6), we have

$$p_1 = COMB_1 \cdot b_2 + COMB_2 \cdot \overline{b_2},$$

and

$$p_0 = COMB_1 \cdot \overline{b_2} + COMB_2 \cdot b_2.$$

### 3.2.1 Performance Comparison

We compare five recursive architectures (including the proposed one) in terms of their critical path, area and throughput as shown in Table. 3.3. The architectures used for comparison are explained as follows:

Binary of input $b[12 : 0]$	Decimal	$q(x)$	$d_1$	$d_0$	$u(v)$	$p_1$	$p_0$
$\vdots$	$\vdots$	0	0	0	0	0	0
1111111110111	-2.25	0	0	0	0	0	0
1111111111000	-2	0	0	0	0.25	0	1
1111111111001	-1.75	0.25	0	1	0.25	0	1
1111111111010	-1.5	0.25	0	1	0.25	0	1
1111111111011	-1.25	0.25	0	1	0.25	0	1
1111111111100	-1	0.25	0	1	0.5	1	0
1111111111101	-0.75	0.5	1	0	0.5	1	0
1111111111110	-0.5	0.5	1	0	0.5	1	0
1111111111111	-0.25	0.75	1	1	0.5	1	0
0000000000000	0	0.75	1	1	0.5	1	0
0000000000001	0.25	0.75	1	1	0.5	1	0
0000000000010	0.5	0.5	1	0	0.5	1	0
0000000000011	0.75	0.5	0	0	0.5	1	0
0000000000100	1	0.25	0	1	0.25	0	1
0000000000101	1.25	0.25	0	1	0.25	0	1
0000000000110	1.5	0.25	0	1	0.25	0	1
0000000000111	1.75	0.25	0	1	0.25	0	1
0000000001000	2	0	0	0	0	0	0
0000000001001	2.25	0	0	0	0	0	0
$\vdots$	$\vdots$	0	0	0	0	0	0

Table 3.2: The values of  $g(x)$ ,  $u(v)$ ,  $d_1$ ,  $d_0$  and  $p_1$ ,  $p_0$ .

- (1). Arch-O: traditional radix-2 architecture.
- (2). Arch-L: the radix-4 architecture proposed by Lucent [15].
- (3). Arch-Y: the radix-4 architecture proposed by [21].
- (4). Arch-Z: the radix-4 architecture proposed by [20].

We find that Arch-Y has the largest area and the fastest throughput rate, its approximation is the same as in (3.1). Thus the performance degradation is large. In Example 4, we compared Arch-Z and the proposed scheme, Assume that AND, OR, XOR and NOT gates have the same delay time (one unit time), the delay time in Arch-Z is about *five* unit times and that of the proposed scheme

Architecture	Maximum Clock Freq.	Relative Area	Relative Throughput	Power Consumption
Arch-O	286	1	1	3.5478 mW
Arch-L [15]	217	1.53	1.52	4.4691 mW
Arch-Y [21]	240	3.08	1.68	8.5570 mW
Arch-Z [20]	231	1.83	1.62	5.2784 mW
Proposed	232	1.80	1.62	5.2839 mW

Table 3.3: Comparison of various recursive architectures.

is about *four* unit times. Therefore our propose method can somewhat reduce the critical path. As a result the clock rate can be somewhat increased, and the power consumption and BER performance (see Fig. 3.5) are nearly the same.

### 3.3 Fixed point Analysis

Table 3.4 shows the quantization format of the Proposed scheme. Fig. 3.5 also shows the fixed-point performance of the proposed scheme. We see that the Max-Log-MAP degrade the performance about 0.4dB of Log-MAP. In Arch-L, because his method only use an LUT, which leads to less accuracy, its performance is worst than Arch-Z and the proposed one. In addition, the proposed scheme has worse performance than the Log-MAP by only 0.025dB. The fix-point simulation, we find the performance is smaller than 0.1dB compared with log-MAP.

Functions \ Word Length	Integer Parts (include sign bit)	Fraction Parts
Received Bits	2	2
Channel Reliable ( $L_c$ )	2	2
State Metrics ( $\bar{\alpha}, \bar{\beta}$ )	8	2
Branch Metrics ( $\bar{\gamma}$ )	8	2
Extrinsic ( $L_{ex}$ )	6	2
LLR	10	2

Table 3.4: Quantization format for the proposed Turbo Decoder.

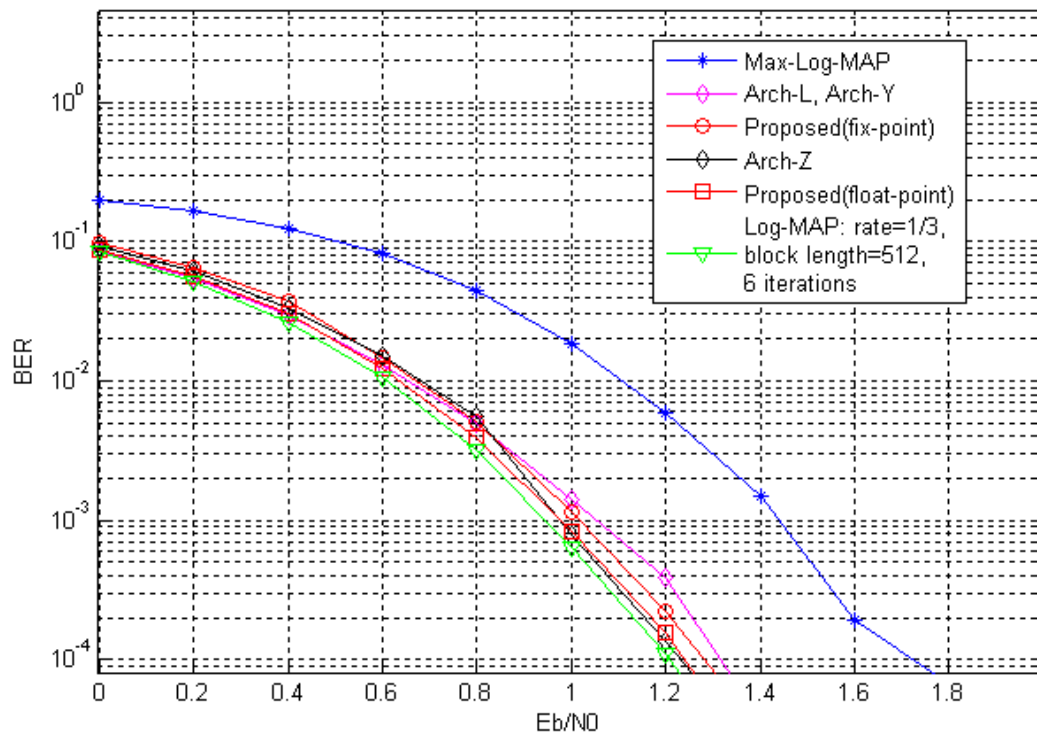


Figure 3.5: Performance comparison among the Log-MAP and four approximated algorithms.

# Chapter 4

## VLSI Implementation

In this chapter, we describe the decoding flow with hardware architecture and show the chip layout as well as the corresponding chip performance comparison.

### 4.1 Hardware Design for 3GPP

Fig. 4.1 is the overall turbo decoder architecture. Because we use radix-4 algorithm to decode data, we can deal with two stages of data per clock cycle. In order to achieve this goal, we can use Dual-RAM which can either read or write two samples of data per clock cycle. However using Dual-RAM doubles the memory area as well. In the proposed VLSI scheme, we will use one Single-RAM which can either read or write one sample of data per clock cycle, we divide this Single-RAM into two smaller RAMs to save even indexed data and odd indexed data. Below, we describe the subblock of decoder operation.

#### 4.1.1 Input Buffer

In Fig. 4.2, there are two paths, path 1 is for SISO Decoder 1 and path 2 is for SISO Decoder 2. In the beginning, the input data ( $r_t(0)$ ,  $r_t(1)$  and  $r_t(2)$ ) are saved in RAM1, RAM3 and RAM4, and  $r_t(0)$  is fed to Interleaver and then the output of the interleaver are saved in RAM2. We proposed a solution to save memory. In the proposed scheme, we divide one Single-RAM into two smaller RAMs to save even indexed data and odd indexed data. Also, the ROM in interleaver is divided



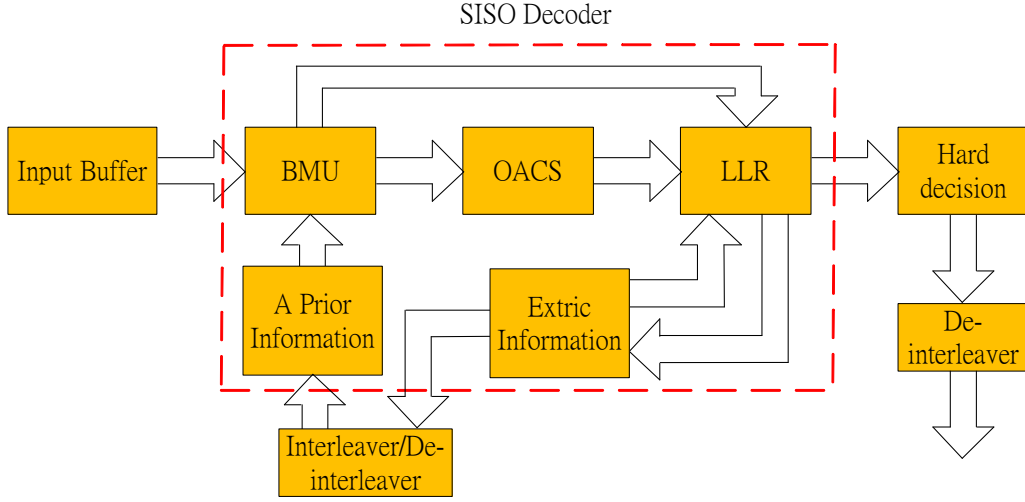


Figure 4.1: The turbo decoder architecture with a single SISO decoder.

into smaller ROMs to save even indexed address and odd indexed address. We take two address (Address 1 and Address 2) from two smaller ROMs (Sub-ROM (E) and Sub-ROM (O), where E for even and O for odd) as shown in Fig. 4.3. Referring to Fig. 4.4, the data corresponding to  $\lfloor \frac{Address\ 1}{2} \rfloor$  and  $\lfloor \frac{Address\ 2}{2} \rfloor$  is saved in Sub-RAM2 (E) and Sub-RAM2 (O), and according to Address 1 and Address 2 within two clock cycles, we have the following four cases

- 1. Case 1:** Address 1 is even number and Address 2 is odd number.

Address 1 is equal to 500 at  $T_0$  and it corresponds to the address 250 ( $\lfloor \frac{500}{2} \rfloor$ ) in Sub-RAM2 (E). Thus it enables sub-RAM2 (E) and disables sub-RAM2 (O). Then  $d_0$  is saved to sub-RAM2 (E) at address 250. Address 2 is equal to 201 at  $T_1$  and it corresponds to the address 100 ( $\lfloor \frac{201}{2} \rfloor$ ) in Sub-RAM2 (O). Thus it disables sub-RAM2 (E) and enables sub-RAM2 (O). Then  $d_1$  is saved to sub-RAM2 (O) at address 100.

- 2. Case 2:** Address 1 is odd number and Address 2 is even number.

Address 1 is equal to 211 at  $T_2$  and it corresponds to the address 105 in Sub-RAM2 (O). Thus it enables sub-RAM2 (O) and disables sub-RAM2 (E). Then  $d_2$  is saved to sub-RAM2 (O) at address 105. Address 2 is equal

		$T_{(0, 2, \dots)}$	
Number of Address1	Number of Address2	Input of Sub-RAM2(E)	Input of Sub-RAM2(O)
even	odd	$\lfloor \frac{Address\ 1}{2} \rfloor$	$\times$
odd	even	$\times$	$\lfloor \frac{Address\ 1}{2} \rfloor$
even	even	$\lfloor \frac{Address\ 1}{2} \rfloor$	$\times$
odd	odd	$\times$	$\lfloor \frac{Address\ 1}{2} \rfloor$
		$T_{(1, 3, \dots)}$	
even	even	$\times$	$\lfloor \frac{Address\ 2}{2} \rfloor$
odd	odd	$\lfloor \frac{Address\ 2}{2} \rfloor$	$\times$
even	even	$\lfloor \frac{Address\ 2}{2} \rfloor$	$\times$
odd	odd	$\times$	$\lfloor \frac{Address\ 2}{2} \rfloor$

Table 4.1: Summary of interleaver process with four case.

to 510 at  $T_3$  and it corresponds to the address 255 in Sub-RAM2 (E). Thus it disables sub-RAM2 (O) and enables sub-RAM2 (E). Then  $d_3$  is saved to sub-RAM2 (E) at address 255.

**3. Case 3:** both of Address 1 and Address 2 are even number.

At  $T_4$  and  $T_5$ , Address 1 is 520 and Address 2 is 530. Therefore  $d_4$  and  $d_5$  will be saved to sub-RAM2 (E). Thus we enable sub-RAM2 (E) and disable sub-RAM2 (O). Then,  $d_4$  and  $d_5$  are saved in sub-RAM2 (E) at address 260 and 265.

**4. Case 4:** both of Address 1 and Address 2 are odd number.

At  $T_6$  and  $T_7$ , Address 1 is 221 and Address 2 is 301. Therefore  $d_6$  and  $d_7$  will be saved to sub-RAM2 (O). Thus we enable sub-RAM2 (O) and disable sub-RAM2 (E). Then,  $d_6$  and  $d_7$  are saved in sub-RAM2 (O) at address 110 and 150.

We summary the interleaver process with four cases as in Table 4.1, where the notation  $\times$  denotes don't care.

Table 4.2, is a comparison of input buffer implemented by Single-RAM and Dual-RAM. From the table, if we use the Dual-RAM the area is larger than the

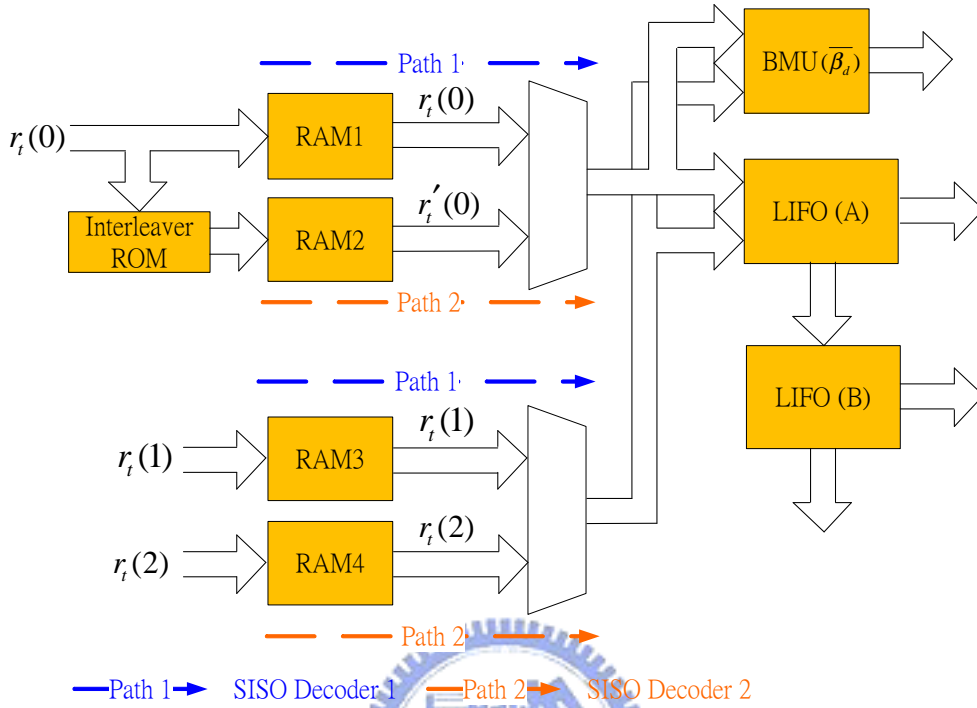


Figure 4.2: The input data flow.

	Relative area	Relative power
Single-RAM (proposed)	1	1
Dual-RAM	2.37	3.55

Table 4.2: Comparison of area and power for implementing the input buffer by dual-RAM and singl-RAM.

Single-RAM by 2.37 times, and the power is larger than the Single-RAM by 3.55 times. Thus our proposed method reduced the area and power significant.

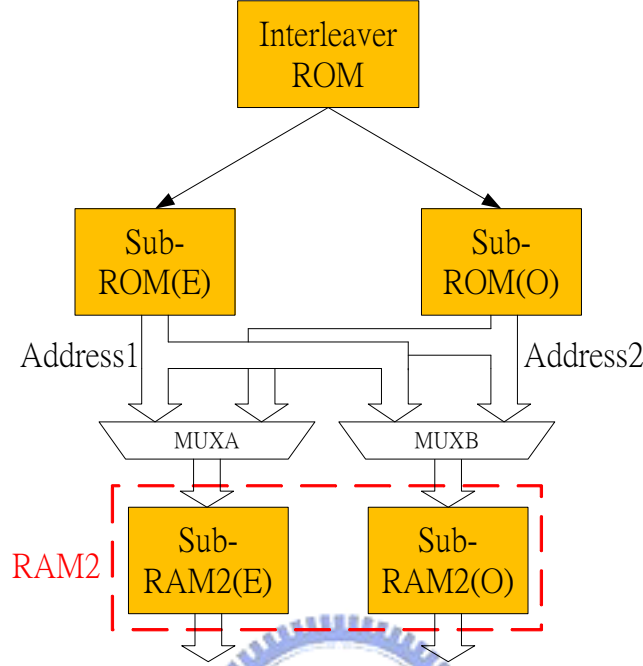


Figure 4.3: The proposed ROM and RAM scheme to achieve two-read and two-write in one clock cycle.

#### 4.1.2 BMU(branch metric unit)

The BMU is used to compute the branch metrics  $\bar{\gamma}$ . The BMU correspond to equation is (2.33). In 3GPP std.,  $\bar{\gamma}$  can as shows as follows

$$\begin{aligned}\bar{\gamma}_t^{t+1}(0, 4) &= \bar{\gamma}_t^{t+1}(1, 0) = \bar{\gamma}_t^{t+1}(6, 3) = \bar{\gamma}_t^{t+1}(7, 7) = \frac{1}{2}[L_a(c_t) - (r_t(0) + r_t(1))], \\ \bar{\gamma}_t^{t+1}(2, 1) &= \bar{\gamma}_t^{t+1}(3, 5) = \bar{\gamma}_t^{t+1}(6, 4) = \bar{\gamma}_t^{t+1}(5, 2) = \frac{1}{2}[L_a(c_t) + (r_t(0) - r_t(1))],\end{aligned}\quad (4.1)$$

and

$$\begin{aligned}\bar{\gamma}_t^{t+1}(0, 0) &= \bar{\gamma}_t^{t+1}(1, 4) = \bar{\gamma}_t^{t+1}(6, 7) = \bar{\gamma}_t^{t+1}(7, 3) = -\bar{\gamma}_t^{t+1}(0, 4), \\ \bar{\gamma}_t^{t+1}(2, 5) &= \bar{\gamma}_t^{t+1}(3, 1) = \bar{\gamma}_t^{t+1}(4, 2) = \bar{\gamma}_t^{t+1}(5, 6) = -\bar{\gamma}_t^{t+1}(2, 1).\end{aligned}\quad (4.2)$$

In (4.1) and (4.2), we only calculate  $\bar{\gamma}_t^{t+1}(0, 4)$ ,  $\bar{\gamma}_t^{t+1}(2, 1)$ ,  $\bar{\gamma}_t^{t+1}(0, 0)$  and  $\bar{\gamma}_t^{t+1}(2, 5)$  and the other  $\bar{\gamma}$  can obtained from these four values. Fig. 4.5 shows a hardware architecture for  $\bar{\gamma}$ . Here we ignore divide-by-2, because in VLSI implementation divide-by-2 operation is just a shift operation.

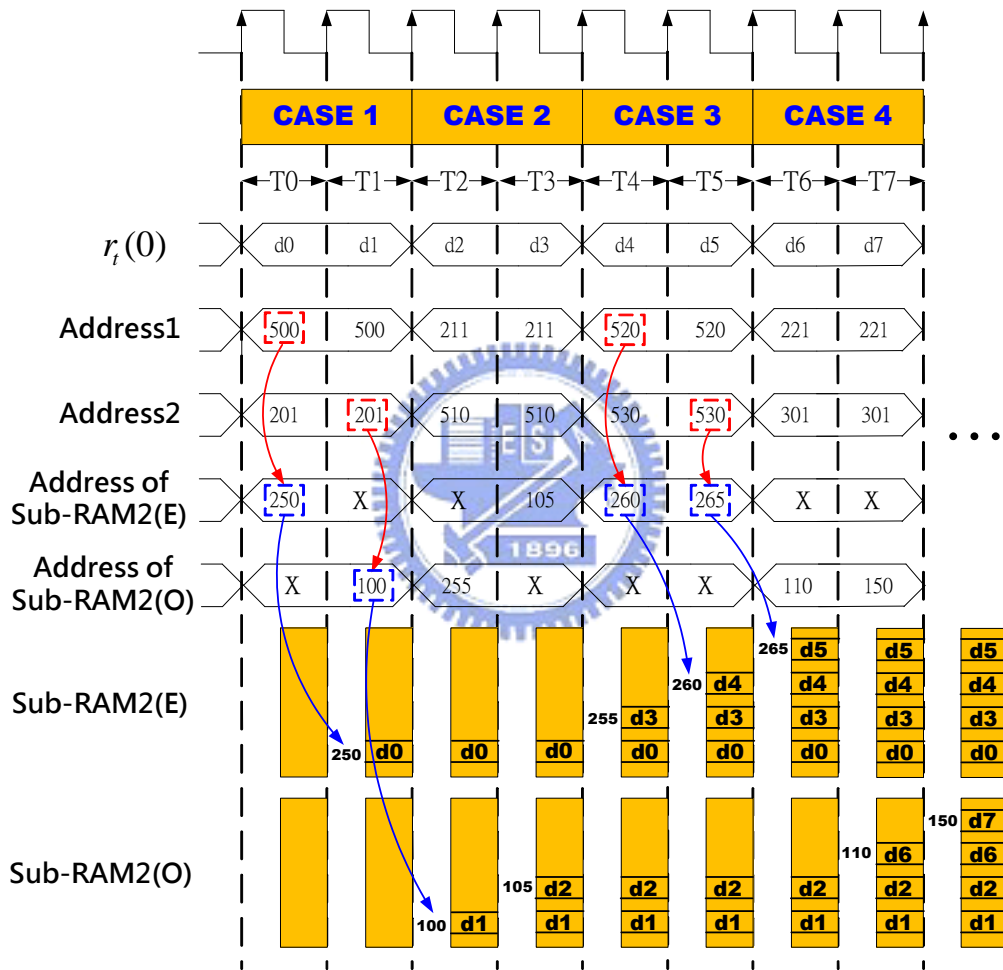


Figure 4.4: Timing diagram for the proposed RAM and ROM schemes.

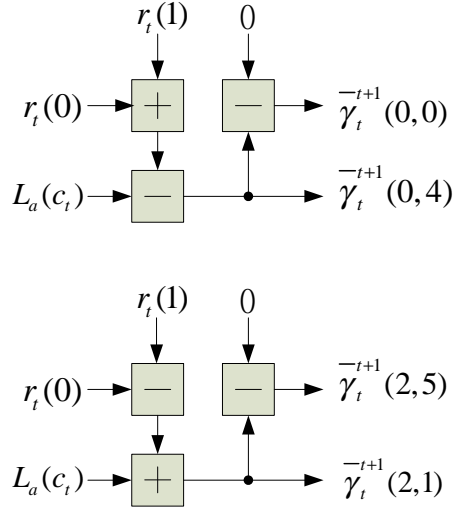


Figure 4.5: The architecture of  $\bar{\gamma}$ .

### 4.1.3 OACS(Offset-Add-Compare-Select)

The OACS is used to calculate  $\bar{\beta}_a$ ,  $\bar{\alpha}$  and  $\bar{\beta}$ . The architecture is shown in Fig. 3.2. Since the OACS is a recursive unit, its computation result will increase after each iteration. Hence the final result may saturate. In order to overcome this situation, we adopt the normalization scheme proposed in [21]. Fig. 4.6 shows an example for  $\bar{\alpha}_{t+2}^0$  with normalization. When one of the values,  $\bar{\alpha}_{t+2, A}^0 \sim \bar{\alpha}_{t+2, A}^7$ , are large than or equal to  $2^{L-2}$ , where  $L$  is the word length of the state metrics ( $\bar{\alpha}_{t+2, A}^{(0\sim7)}$  and  $\bar{\beta}_{t+2, A}^{(0\sim7)}$ ), we subtract  $2^{L-2}$  from all of the state metrics to avoid saturation.

### 4.1.4 LLR (Log-Likelihood Ratio )

The *LLR* output for the radix-4 turbo decoder can be calculated according to (2.48)-(2.50) and the corresponding hardware is shown in Fig. 4.7, where  $max^*$  (see (2.23)) can be express as hardware shown in Fig. 4.8, and the LUT is the proposed structure in section 3.2. In LLR unit, we used pipeline skill to reduce the critical path with the penalty of increasing 28 registers. The processing time for each SISO decoding is three clock cycles.

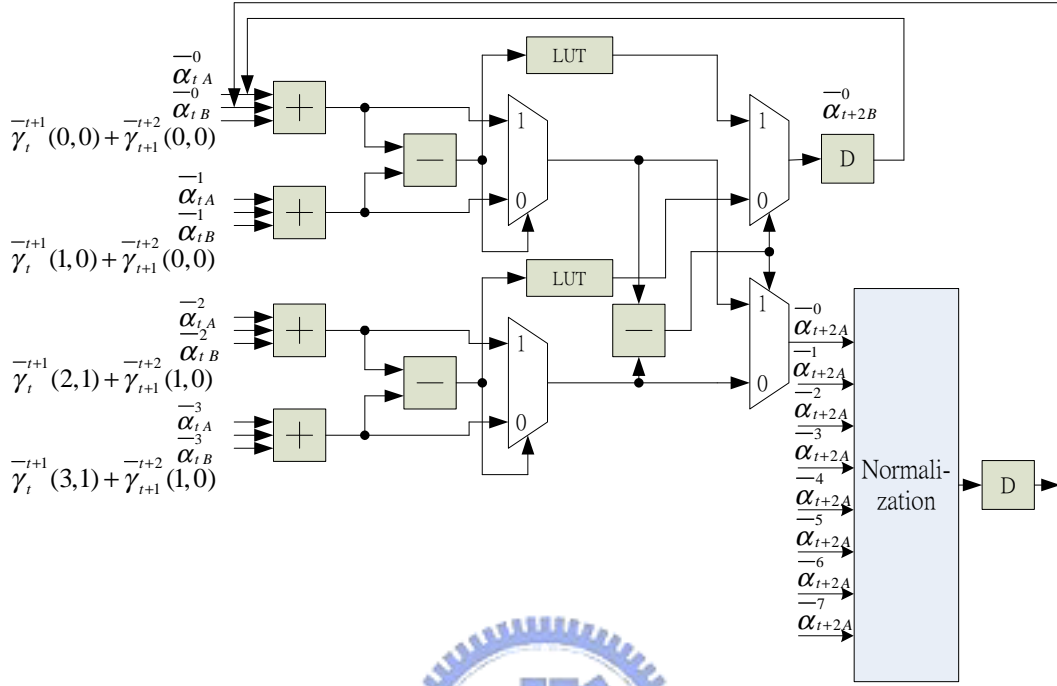


Figure 4.6: The normalization of OACS.

#### 4.1.5 Extrinsic Information and a Priori Information

When the computation of LLR is completed, we calculate  $L_e(c_t)$  refer to (2.52).  $L_e(c_t)$  is to be sent to interleaver/de-interleaver and then saved as the a Priori information. The a Priori information is saved in two RAMs, where we use two Dual-RAMs to achieve this, because adopting one Dual-RAM will lead to data hazard. Fig. 4.9 is the timing diagram for read/write situation in Dual-RAM 1 and Dual-RAM 2. Assume the data block length is  $N_D$  and the window length is  $N_L$ , operating at SISO decoder 1. At SISO decoder 1 period, the Dual-RAM 1 is in write-mode and the Dual-RAM 2 is in read-mode. At  $T_0$ , a priori information is read from Dual-RAM 2 to calculate the extrinsic information and then obtain extrinsic information  $e_4$  and  $e_5$ . The  $e_4$  and  $e_5$  are passed into interleaver and then are saved to dual-RAM 1 at Address 1 and Address 0. Here if we adopt one Dual-RAM at  $T_2$ , we will extract a priori information at Address 0 and Address 1, however this a priori information has been updated at  $T_0$ . Therefore we need to

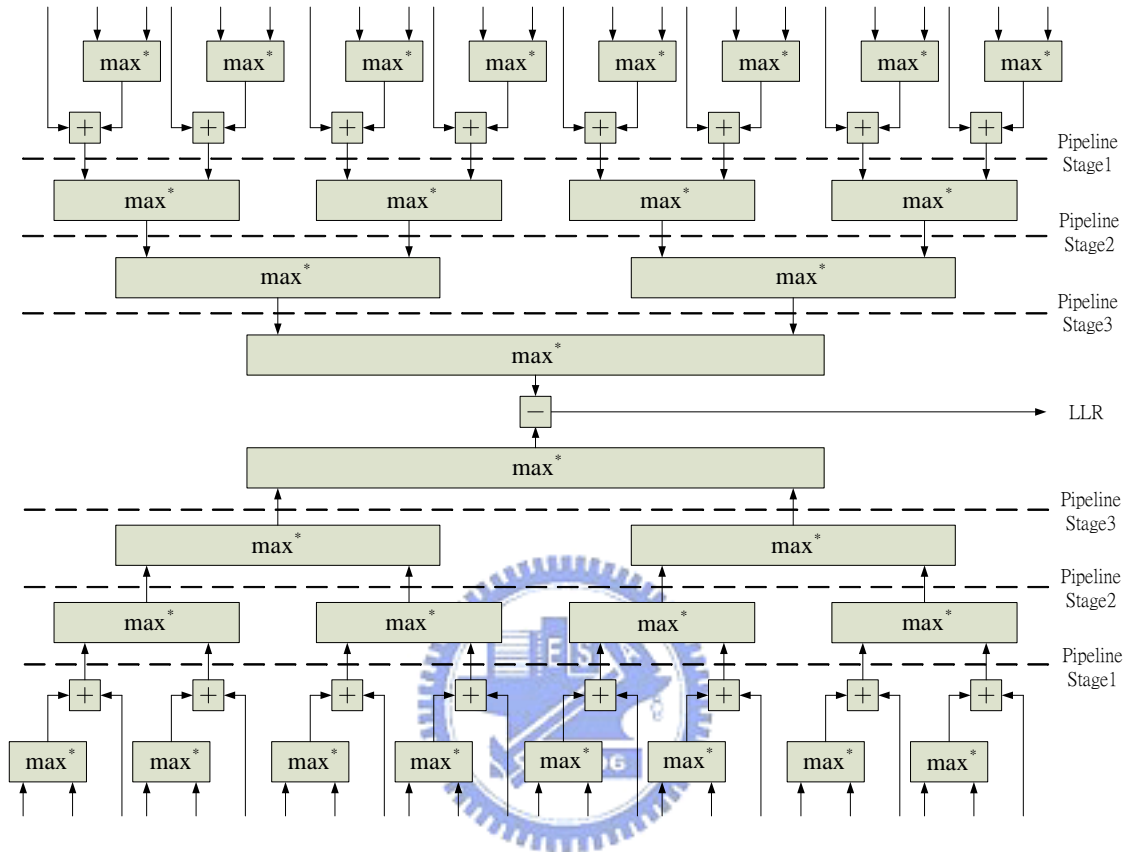


Figure 4.7: The Architecture of LLR.

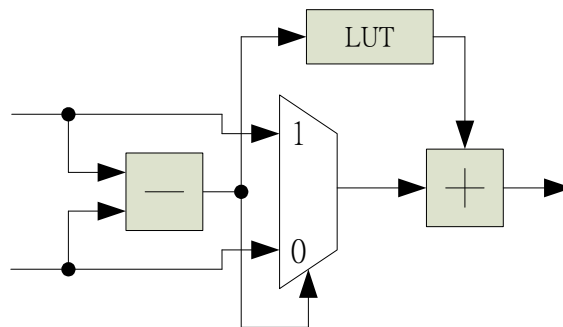


Figure 4.8: The hardware architecture of  $max^*(\cdot)$ .



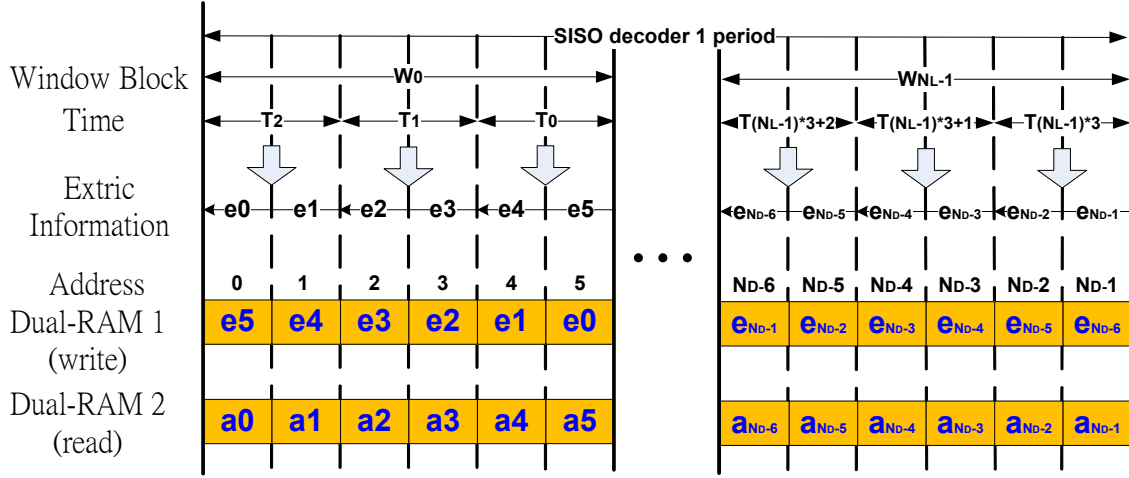


Figure 4.9: Timing diagram of a priori information for two Dual-RAMs.

save extrinsic information  $e_4$  and  $e_5$  to Dual-RAM 2 to avoid data hazard. Operating at SISO decoder 2 period, the Dual-RAM 1 is in read-mode and the Dual-RAM 2 is in write-mode, its operation is similar with that in SISO decode 1 period.

#### 4.1.6 Interleaver and De-interleaver

The Interleaver has two purposes. One is to interleave  $r_t(0)$  mentioned in *Input Buffer*. The other is to interleave  $L_e(c_t)$  mentioned in *Extrinsic Information* and *a Priori information*. De-interleaver is used to de-interleave  $L_e(c_t)$ . In *Input Buffer*, we only use interleaver once. Then the interleaver is used to interleave  $L_e(c_t)$  in iteration procedure. Hence we only need one interleaver. Since we need to use it several times in iteration process and only once to de-interleaver the output the LLR. Also, we only need one de-interleaver.

#### 4.1.7 Hard Decision

The number of iteration is in general limited, when the number of limitation iteration is reached, the output of the LLR will take hard decision (refer to (2.34)) to decode two information bits. Fig. 4.10 is an architecture for the hard decision, assuming the LLR has  $n$  bits. We only consider the sign bit of the LLR. If the

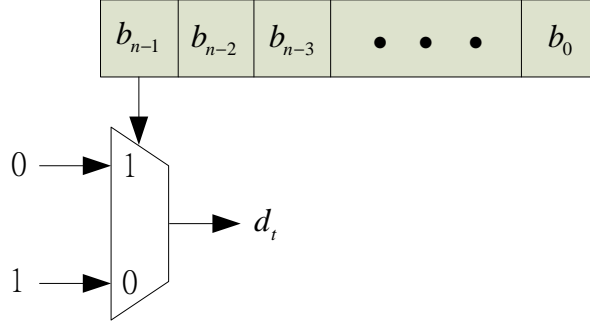


Figure 4.10: The architecture of hard decision.

sign bit is 1, the output is 0; otherwise the output is 1.

#### 4.1.8 Sliding Window

Here we describe the sliding window approach. Referring to Fig. 4.11 and Fig 4.12, after all of the data are saved to RAM in *Input Buffer*. Then the data are taken from RAM1 and RAM3 /RAM2 and RAM4 and saved in LIFO (A) for SISO Decoder 1/SISO Decoder 2. The data LIFO (A) are then shift to LIFO (B). At the beginning, we delay  $k/2$  ( $k$  is window length) clock cycles to take data from RAM with data address from  $k/2 - 1$  to 0 and save them in LIFO(A). In second  $k/2$  clock cycles, we take data from RAM with data address from  $k - 1$  to  $k/2$ , and save them in LIFO(A). At this duration, take data from the input buffer and LIFO(A) and then fed to BMU ( $\overline{\beta}_d$ ) and BMU ( $\overline{\alpha}$ ) to calculate branch metrics  $\overline{\beta}_d$  and  $\overline{\alpha}$ . In the third  $k/2$  clock cycles, the data in LIFO (B) is fed to BMU ( $\overline{\beta}$ ) to calculate branch metric  $\overline{\beta}$ . The OACS ( $\overline{\beta}_d$ ) and OACS ( $\overline{\alpha}$ ) is calculated  $\overline{\beta}_d$  and  $\overline{\alpha}$  at second the  $k/2$  clock cycles, and the calculated  $\overline{\alpha}$  is saved in buffer ( $\overline{\alpha}$ ). At the end of the second  $k/2$  clock cycles,  $\overline{\beta}_d$  is ready to calculate  $\overline{\beta}$  by OACS ( $\overline{\beta}$ ). In the third  $k/2$  clock cycles, we obtain a boundary value  $\overline{\beta}_d$  to begin to calculate  $\overline{\beta}$ , and output  $\overline{\beta}$ . At the third  $k/2$  clock cycles, all of the data (i.e.  $\overline{\alpha}$ ,  $\overline{\beta}$  and  $\overline{\gamma}$ ) are fed to LLR to decide the soft information. If the number of iteration limitation is reached, we stop the process and fed the output of the LLR to hard decision and then perform De-interleaving for the decoded information bits.

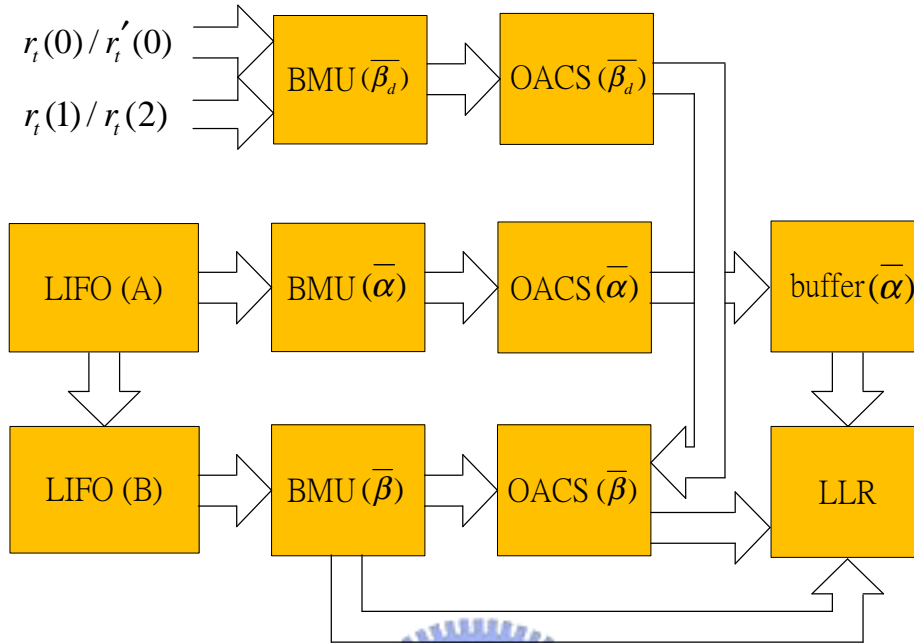


Figure 4.11: Calculating BMU, OACS and LLR.

## 4.2 Design flow

In the section, we will introduce the design flow for the proposed turbo decoder. The cell-based design flow is as shown in Fig. 4.13.

### 4.2.1 System model

First the encoder is created according to the 3GPP standard, and we use the proposed architecture to decoder. The simulation platform was built on Matlab. In VLSI impelmentation, we quantize the floating-point to fixed-point and adjust the bit length so that the BER performance is close to floating-point simulation result. When the bit length is decided, we can generate input and output test patterns for RTL.

Using tool: Matlab.

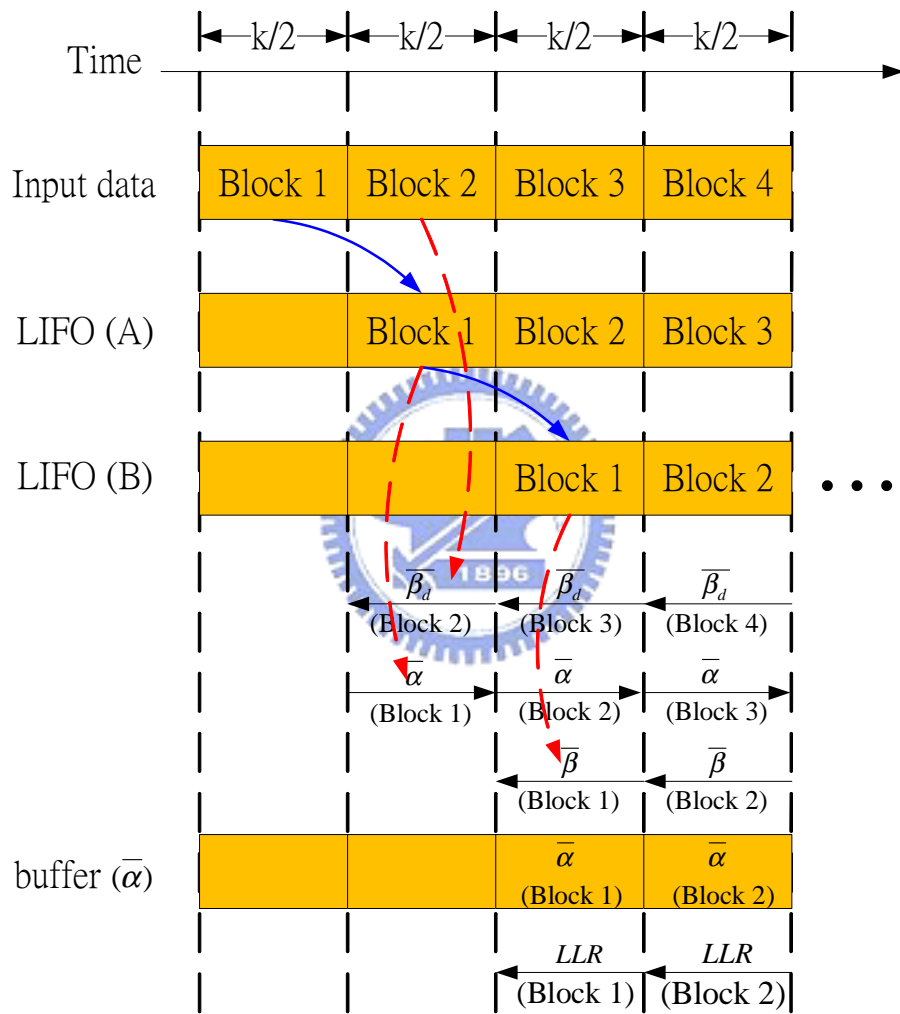


Figure 4.12: Timing diagram of Sliding Window.

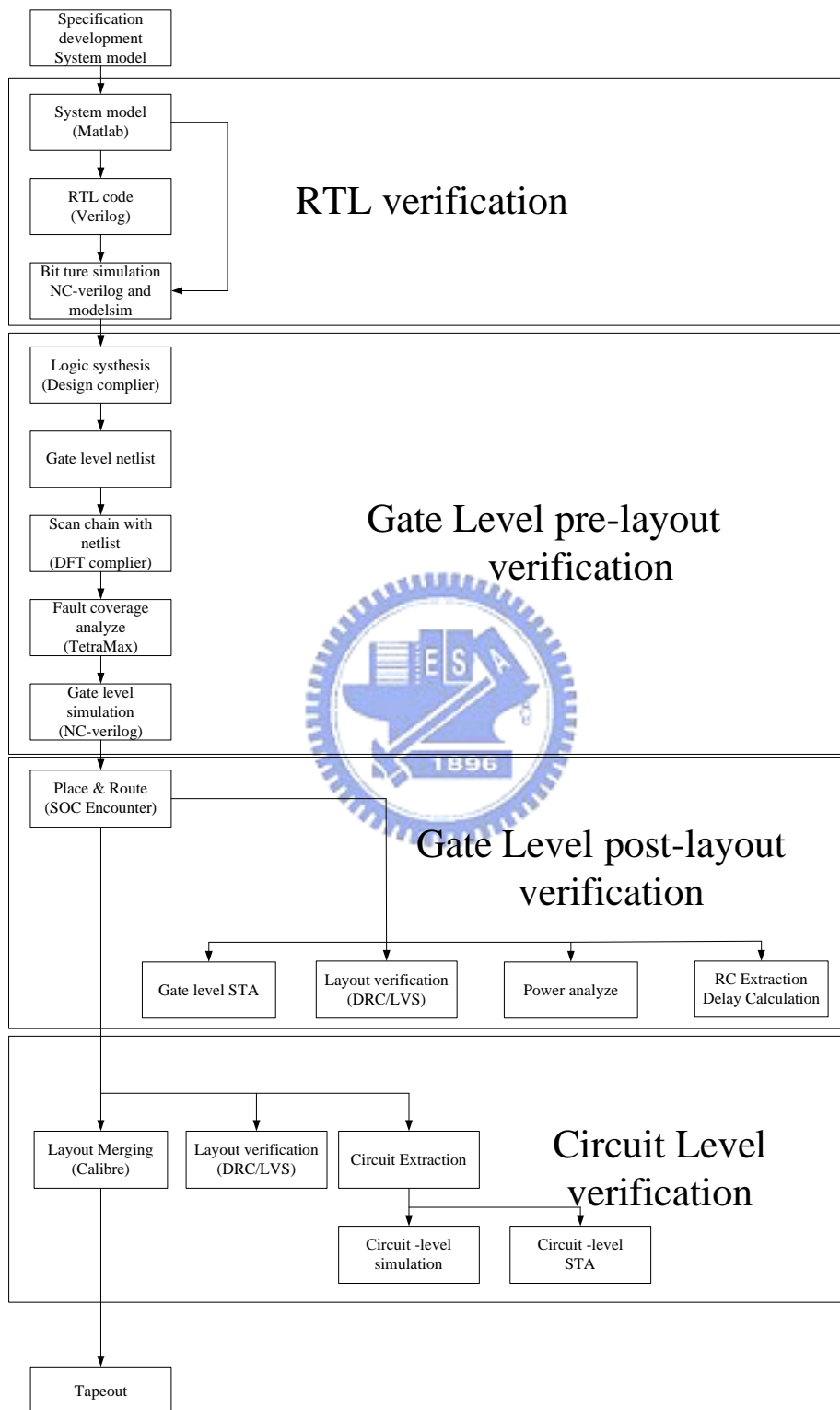


Figure 4.13: IC design flow.

## 4.2.2 RTL code

We use Verilog-HDL to describe the hardware architecture. The general design method is hierarchically method. Hence we need to divide the overall design into several basic modules first. Then, connecting among the basic modules to complete the rough structure. Finally we need to perform bit true in order to make sure the output signals of RTL code and Matlab are same with same input signals. In addition, we have using memory in our architecture, so we use the memory compiler to generate.

Using tools: memory compiler, NC-verilog, modelsim, and Debussy nWave.

## 4.2.3 BIST

Because there are memory in our architecture, we need to add BIST circuit on memory control for the testability of IC. After adding BIST circuit, there are two mode in circuit, i.e. function mode and test mode. Function mode means that normal Turbo decoding can be performed, and test mode can be used test that there are have any error in memory.

Using tool: TurboBIST.

## 4.2.4 Synthesis

In this step, we start to synthesize our circuit. Before this step, our program is just hardware language, is not real gate. By using Synopsys Design Compiler to do the synthesis, our program can be translate as real gate. And we can get the rough area and some timing information of the gate. In our decoder design, all modules except the one port and two port register files are synthesized with TSMC 0.18 $\mu$ m CMOS process technology.

Using tool: Design Compiler.

## 4.2.5 Gate-level simulation

After synthesis, we can get timing information of gate. So we can perform our circuit to check have any error with real time. We use NC-Verilog to do the

gate-level simulation and use Debussy nWave to check waveform. By checking waveform, we can observe function exactitude with our predetermined clock period.

Using tools: NC-Verilog, and Debussy nWave.

#### **4.2.6 DFT**

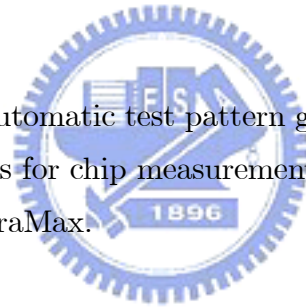
For IC testing, we need to add mux in front of Flip-Flop and scan chains for the testability of IC. After adding mux, we can get there is any error between Flip-Flop and Flip-Flop by passing mux input signal. We use to Synopsys DFT Compiler to do scan chain insertion.

Using tool: DFT compiler.

#### **4.2.7 ATPG**

In the step, we use ATPG (automatic test pattern generator) of Synopsys TetraMax to generate test patterns for chip measurement.

Using tool: Synopsys TetraMax.



#### **4.2.8 APR**

We use SOC encounter to do automatical placement and routing (APR). Before placing and routing, we need to add power I/O and core I/O on Gate-level netlist and arrange location of input, output, I/O power, and core power on pad CIC supported. We need to consider core utilization, location of one port and two port register files, number of power ring, location and number of stripe to meet timing constraints from SDC file.

Using tool: SOC encounter.

#### **4.2.9 DRC and LVS**

In general, we usually have consider DRC (design rule checking) and LVS (layout V.S. schematic) in APR. But there is just rough check result in SOC encounter. So we need to do detail verification. We use the Calibre DRC to check whether

there is any error with design rule and use the Calibre LVS to make sure that whether the layout and the schematic are identical or not.

Using tool: Calibre.

#### **4.2.10 Post-layout level**

In order to check function, we take the netlist and file of timing information generated by SOC encounter to run NC-Verilog. We can observe wave to find whether is any error by Debussy nWave. This is the last step to check function on myself work.

Using tools: NC-Verilog, and Debussy nWave.





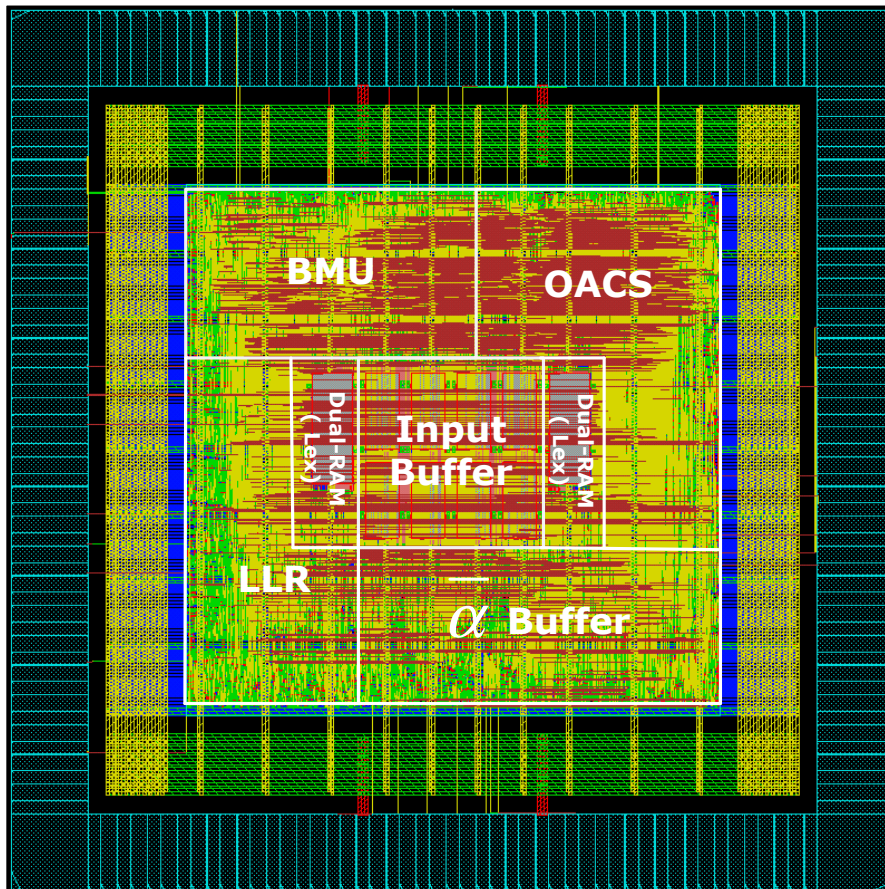


Figure 4.14: Chip layout of the proposed Radix-4 Turbo Decoder for 3GPP.

### 4.3 Chip Layout and Comparison

The turbo decoder is implemented by using the TSMC 0.18  $\mu\text{m}$  1P6M CMOS process. It achieves the maximum clock rate of 167MHz. The chip layout is shown in Fig. 4.14 and the chip summary is also listed in Table 4.3. Comparing to [15], [23] and [24] as shown in Table 4.4, the core size and area of the proposed scheme is relative high. However the proposed scheme can achieve higher clock rate. In our proposed, the throughput is worst than the [15], but faster than the [23] and [24].

Technology	TSMC 0.18 $\mu m$ 1P6M CMOS
Chip size	7.28 $mm^2$
Core size	2.65 $mm^2$
Gate count	200K
Embedded SRAM	28K bits
Embedded ROM	9K bits
Clock rate	167 MHz
Power consumption	135mW

Table 4.3: The expected turbo decoder chip summary.



	[15]	[23]	[24]	Proposed design
Technology	0.18 $\mu m$	0.25 $\mu m$	0.18 $\mu m$	0.18 $\mu m$
Block Length	5114	5114	5114	512
Core Size ( $mm^2$ )	14.5	9	9	2.65
Clock Rate (MHz)	145	135	88	167
Throughput (Mb/s)	24	5.48	2	22
Number of iteration	6	6	10	6
Energy efficiency (nJ/b/iter.)	10	6.98	14.60	1.02

Table 4.4: Chip comparison.

# Chapter 5

## Conclusion

In this thesis, we proposed a LUT architecture, so the speed of MUX and LUT are nearly the same. As a result, the critical path is reduced. Because the decoder uses the Radix-4 algorithm, which deals with 2 stages of data in one clock cycle, we proposed a ROM and RAM read/write scheme to avoid the use of Dual-RAM. In chip implementation, the chip is fabricated in TSMC 0.18  $\mu m$  CMOS process, operating at 167MHz clock rate with voltage supply 1.62V. The power consumption is 135mW at decoding rate 22Mb/s with code rate 1/3 for 3GPP standard. The core area is 2.65  $mm^2$ , contains 200K gate counts.

# Reference

- [1] C. Berrou, A. Glavieux, P. Thitimajshima, “Near Shannon limit error-correcting coding and decoding: Turbo-Codes,” *Proc. of IEEE ICC'93*, Geneva, pp. 1064-1070, Volume 2, May 1993.
- [2] TIA/EIA/CDMA2000, “Physical layer standard for CDMA-2000 standards for spread spectrum systems,” June, 2000.
- [3] “Technical Specification Group Radio Access Network, Multiplexing and channel coding (FDD) (TS 25.212 V8.2.0)” 3rd Generation Partnership Project (3GPP).
- [4] J. Hagenauer and P. Hoeher, “A Viterbi algorithm with soft-decision outputs and its applications,” in *Proc. IEEE GLOBECOM*, Dallas, TX, pp. 47.1.1–47.1.7, Nov. 1989.
- [5] J. Hagenauer et al., “Decoding turbo codes with the soft-output Viterbi algorithm (SOVA),” in *Proc. IEEE Int. Symp. Information Theory*, Trondheim, Norway, pp. 164, 1994.
- [6] L. Papke and P. Robertson, “Improved decoding with the SOVA in a parallel concatenated (turbo-code) scheme,” in *Proc. IEEE Int. Conf. Communications*, pp. 102–106, 1996.
- [7] P. Robertson, E. Villebrun, and P. Hoeher, “A comparison of optimal and suboptimal MAP decoding algorithms operating in the log domain,” in *Proc. IEEE Int. Conf. Communications*, pp. 1009–1013, 1995.

- [8] J. Vogt and A. Finger, "Improving the MAX-LOG-MAP turbo decoder," *Electron. Lett.*, vol. 36, pp. 1937–1939, Nov. 2000.
- [9] J. Hagenauer et al., "Iterative (turbo) decoding of systematic convolutional codes with the MAP and SOVA algorithms," in *Proc. ITG*, Munich, Germany, pp. 21–29, Oct. 1994.
- [10] M. Shin and I.-C. Park, "Processor-based turbo interleaver for multiple third-generation wireless standards," *IEEE Commun. Lett.*, vol. 7, no. 5, pp. 210–12, May 2003.
- [11] P. Ampadu and K. Kornegay, "An efficient hardware interleaver for 3G turbo decoding," *Proc. RAWCON'03*, pp. 199–201, Aug. 2003.
- [12] Z. Wang and Q. Li, "Very low-complexity hardware interleaver for turbo decoding," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 54, no. 7, pp. 636–640, Jul. 2007.
- [13] 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.
- [14] 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.
- [15] M. Bickerstaff, L. Davis, C. Thomas, D. Garrett, C. Nicol, "A 24 Mb/s Radix-4 LogMAP Turbo Decoder for 3GPP-HSDPA Mobile Wireless," in *Proc. IEEE Int. Solid-State Circuit Conf.*, pp. 1-10, 2003.
- [16] 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.
- [17] S. A. Barbulescu, "Iterative decoding of turbo codes and other concatenated codes," Ph.D. dissertation, Univ. South Australia, 1996.

- [18] S. Benedetto and G. Montorsi, "Design of parallel concatenated convolutional codes," *IEEE Trans. Commun.*, vol. 44, no. 5, pp. 591–600, May 1996.
- [19] E. Boutillon, W. Gross, and P. Gulak, "VLSI architectures for the MAP algorithm," *IEEE Trans. Commun.*, vol. 51, no. 2, pp. 175–185, Feb. 2003.
- [20] Z. Wang, "High-speed recursion architectures for MAP-based turbo decoders," *IEEE Trans. on VLSI Syst.*, vol. 15, no. 4, pp. 470–474, Apr. 2007.
- [21] Y. Zhang and K.K. Parhi, "High-Throughput Radix-4 LogMAP Turbo Decoder Architecture," *Proc. of 40th Asilomar Conf. on Signals, Systems and Computers*, pp. 1711–1715, Oct. 2006.
- [22] Z. Wang, H. Suzuki and K. K. Parhi, "VLSI Implementation Issues of Turbo Decoder Design for Wireless Applications", *Proc. of 1999 IEEE Workshop on Signal Processing Systems: Design and Implementation*, Taipei, Oct. 1999.
- [23] M.-C. Shin and I.-C. Park, "A programmable turbo decoder for multiple 3G wireless standards," in *IEEE Int. Solid-State Circuits Conf. Dig. Tech. Papers*, pp. 154–155, Feb. 2003.
- [24] M. Bickerstaff, D. Garrett, T. Prokop, C. Thomas, B. Widdup, G. Zhou, C. Nicol, and R.-H. Yan, "A unified Turbo/Viterbi channel decoder for 3GPP mobile wireless in 0.18  $\mu$ m CMOS," in *IEEE Int. Solid-State Circuits Conf. Dig. Tech. Papers*, pp. 90–91, Feb. 2002.