

# 國立交通大學

電子工程學系 電子研究所 碩士班

## 碩士論文

以記憶體為基礎的多標準前端錯誤更正

解碼器設計

A Memory Based Multi-Standard FEC Decoder Design

研究生：曾逸晨 (Yi-Chen Tseng)

指導教授：李鎮宜 教授 (Prof. Chen-Yi Lee)

中華民國九十三年六月

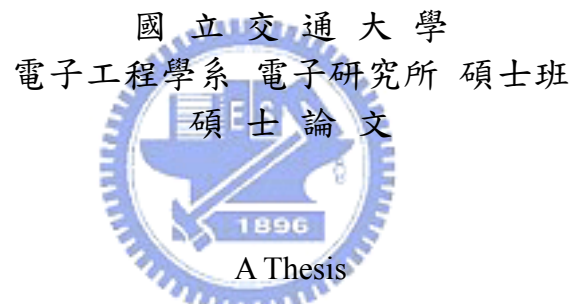
以記憶體為基礎的多標準前端錯誤更正解碼器設計  
A Memory Based Multi-Standard FEC Decoder Design

研究生：曾逸晨

Student : Yi-Chen Tseng

指導教授：李鎮宜

Advisor : Chen-Yi Lee



Submitted to Institute of Electronics  
College of Electrical Engineering and Computer Science  
National Chiao Tung University  
in Partial Fulfillment of the Requirements  
for the Degree of  
Master  
in  
Electronics Engineering

June 2004

Hsinchu, Taiwan, Republic of China

中華民國九十三年六月

# 以記憶體為基礎的多標準前端錯誤更正解碼器設計

學生：曾逸晨

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

前端錯誤更正在通訊系統是一個相當重要的功能，它主要包含攪拌器(scrambler)、里德所羅門編碼(Reed-Solomon coding)、交錯器(interleaver)和迴旋編碼(trellis coding)。對於效能和複雜度的考量，隨著不同的應用而會有不同的設計參數。本論文提出一個高效率節省功率和面積架構的多標準前端錯誤更正解碼器以符合不同系統的要求，而所提出的多標準前端錯誤解碼器可以完全相容於 ITU-T J. 83 的纜線數據系統並且可相容於數位視訊廣播和 ATSC 數位電視等系統。我們所提出的多標準前端錯誤更正解碼器主要包含一以記憶體來儲存及更正資料的多模里德所羅門解碼器和一以記憶體為基礎及位址產生器的泛用型卷積解交錯器(convolutional de-interleaver)，皆具有以最小晶片面積達到最多功能的特色。以 0.18 微米 1P6M CMOS 製程實作的結果大約需要 5 萬 4 千個邏輯閘以及 6 千位元的嵌入式靜態隨機存取記憶體，最快可以達到 83MHz(600Mbps)的工作頻率。平均消耗功率在最複雜的解碼模式及在 83MHz 之工作頻率下大約是 45mW；在滿足系統規範之操作參數環境下，平均消耗功率大約 5.4mW。

# A Memory Based Multi-Standard FEC Decoder Design

Student: Yi-Chen Tseng

Advisor: Chen-Yi Lee

Department of Electronics Engineering  
Institute of Electronics  
National Chiao Tung University

## ABSTRACT

Forward Error Correction (FEC) which mostly contains scrambler, Reed-Solomon coding, interleaving, and trellis coding is a key component in communication system. For the performance and complexity issues, design parameters are different in various applications. In this thesis, a multi-standard FEC decoder is presented to meet different system requirements with a power and area efficient architecture. The proposed multi-standard FEC decoder is fully compliant to ITU-T J.83 cable modem system and is also compatible to DVB-T and ATSC Digital TV, etc. The proposed multi-standard FEC decoder, including a multi-mode Reed-Solomon decoder with memories to store and correct the received data and a memory-based universal convolutional interleaver with a simple address generator, has the advantage of lowest overhead. With 0.18 $\mu$ m 1P6M CMOS technology, the implemented chip shows the FEC decoder can work at 83MHz (600Mbps) while costs 54.5K gate counts and two 376x8 bits embedded dual-port SRAM. The average power consumption in full spec. mode is about 45mW at 83Mhz. While running at 7MHz that meets symbol rate of cable modem, the power dissipation is 5.4mW.

## 誌 謝

光陰似箭，時光荏苒，二年的時光一下就過去了，在這碩士二年的學習時光中，首先我要向指導教授李鎮宜博士表達最誠摯的謝意，感謝老師二年來的諄諄誘導，讓我有明確的研究方向及正確的研究心態，因此能夠順利的完成學業及研究成果。另外，我很高興能夠加入 Si2 實驗室，在這個大家庭中我不只學到了專業技能-IC Design，更了解了團隊合作的重要性，且實驗室的每一個成員都樂於助人，讓我能盡快的解決所遭遇到的問題。再一次謝謝李鎮宜老師的教導和 Si2 每個成員的幫忙。



# Contents

<b>CHAPTER 1 INTRODUCTION.....</b>	<b>1</b>
1.1    MOTIVATION.....	1
1.2    INTRODUCTION TO THE PLATFORM.....	1
1.3    THESIS ORGANIZATION.....	3
<b>CHAPTER 2 ALGORITHM OF FEC.....</b>	<b>5</b>
2.1    SCRAMBLER.....	5
2.2    INTERLEAVING.....	8
2.3    REED-SOLOMON CODES.....	10
2.3.1 <i>Reed-Solomon encoder</i> .....	10
2.3.2 <i>Reed-Solomon decoder</i> .....	12
2.4    TRELLIS CODES.....	18
2.5    SUMMARY.....	20
<b>CHAPTER 3 ALGORITHM AND ARCHITECTURE FOR MULTI - MODE FEC DECODER.....</b>	<b>21</b>
3.1    THE PROPOSED MULTI-MODE FEC DECODER.....	21
3.2    MEMORY-BASED UNIVERSAL CONVOLUTIONAL INTERLEAVER/ DE-INTERLEAVER.....	22
3.2.1 <i>The algorithm and architecture of memory-based universal convolutional interleaving.....</i>	23

3.3	THE MULTI-MODE RS DECODER.....	28
3.3.1	<i>Multi-Mode Finite Field Multiplier</i> .....	29
3.3.2	<i>Syndrome Calculator</i> .....	29
3.3.3	<i>Key Equation Solver</i> .....	31
3.3.4	<i>Chien Search</i> .....	32
3.3.5	<i>Error Value Evaluator</i> .....	33
3.3.6	<i>Memory structure to correct the RS codeword</i> .....	34
3.4	OTHER COMPONENTS .....	35
3.4.1	<i>De-scrambler</i> .....	35
3.4.2	<i>Viterbi Decoder</i> .....	36
3.5	THE MEMORY CONSIDERATION FOR TEST CHIP .....	42
3.6	SUMMARY .....	43
<b>CHAPTER 4 SIMULATION AND IMPLEMENTATION RESULT .....</b>		<b>44</b>
4.1	PLATFORM AND SYSTEM DESIGN.....	44
4.2	CHIP INTEGRATION AND THE RESULTS OF CHIP IMPLEMENTATION.....	48
4.3	SUMMARY .....	52
<b>CHAPTER 5 CONCLUSION AND FUTURE WORK.....</b>		<b>54</b>
5.1	CONCLUSION .....	54
5.2	FUTURE WORK .....	55
<b>BIBLIOGRAPHY.....</b>		<b>57</b>
<b>APPENDIX-A DECODING ALGORITHM OF LDPC CODES .....</b>		<b>60</b>

# List of Figures

Figure 2.1: (a) FEC in ITU-T J.83 annexes A, C and D. (b) FEC in ITU-T J.83 annex B.....	5
Figure 2.2: Scrambler in (a) J.83A, C and DVB-T system. (b) J.83B. (c) J.83D.....	6
Figure 2.3: Structure of (I, J) convolutional interleaving .....	9
Figure 2.4: The output symbols in convolutional interleaver with $I = 12, J = 17$ .....	9
Figure 2.5: The circuit of the systematic feedback shift register RS encoder.....	11
Figure 2.6: RS decoding process .....	13
Figure 2.7: Punctured binary convolutional codes in ITU-T J.83B .....	19
Figure 3.1: The proposed multi-mode FEC decoder .....	22
Figure 3.2: The memory array by rebuilding the FIFO registers of deinterleaver .....	24
Figure 3.3: Behavior of the novel algorithm for (12, 17) convolutional deinterleaver .....	25
Figure 3.4: Pseudo codes of universal convolutional deinterleaver .....	27
Figure 3.5: The architecture of the address generator for convolutional interleaving .....	28
Figure 3.6: Multi-mode FFM over $GF(2^m)$ .....	29
Figure 3.7: Multi-mode syndrome calculator: (a) Basic cell $SC_i$ for $GF(2^8)$ . (b) Basic cell $SC_{2i}$ for dual mode purpose ( $GF(2^8)$ and $GF(2^7)$ ). (c) The overall structure of multi-mode syndrome calculator .....	30
Figure 3.8: Multi-mode key equation solver .....	32
Figure 3.9: Multi-mode chien search. (a) Basic cell $C_i$ for $GF(2^8)$ . (b) Basic cell $C_{2i}$ for dual mode purpose ( $GF(2^8)$ and $GF(2^7)$ ). (c) The overall structure of multi-mode chien search.	



.....	33
Figure 3.10: Multi-mode error value evaluator .....	34
Figure 3.11: The operation of accessing memory in multi-mode RS decoder .....	35
Figure 3.12: The transformed structure of scrambler in J.83A and C .....	36
Figure 3.13: Register contents for register-exchange method .....	37
Figure 3.14: Architecture of register-exchange approach applied in SM unit. (a) Trellis diagram. (b) The connections of registers and multiplexers between each state. ....	37
Figure 3.15: The upper bound of PM difference .....	38
Figure 3.16: Illustration of Modulo Normalization .....	39
Figure 3.17: The ACS module used for Viterbi decoder .....	40
Figure 3.18: Survivor memory and trace back unit .....	40
Figure 3.19: Architecture of trellis decoder .....	41
Figure 3.20: The system platform with memory consideration .....	42
Figure 4.1: The design flow .....	45
Figure 4.2: FEC encoder in J.83 .....	46
Figure 4.3: Simulation environment .....	46
Figure 4.4: The chip connected with external memory .....	49
Figure 4.5: The floor plan of the chip .....	50
Figure 5.1: Turbo encoder .....	55
Figure A.1: The message passing on bipartite graph of LDPC codes .....	60
Figure A.2: The performance of rate 1/2 (1008, 504) irregular LDPC codes by reduced-complexity LLR-SPA over AWGN channel .....	65

# ***List of Tables***

Table 1: Comparison of different specification in FEC of ITU-T J.83 .....	3
Table 2: Summary of CHIP Implementation for J.83 FEC .....	49
Table 3: Gate Count for each module .....	51
Table 4: Comparisons between the proposed architecture and other people's work.....	52



# **Chapter 1**

## **Introduction**

---

### *1.1 Motivation*

In communication system, channel coding which uses various types of error correcting algorithm and interleaving techniques is a key module to minimize the effect of channel noise during data transmission. They can be summarized as the following four parts in most systems: scrambler, Reed-Solomon (RS) coding, interleaving, and trellis coding. And different applications have specific parameters to achieve an optimum system. Due to the similarity in FEC sections, such as ITU-T J.83 [1], DVB-T [2], and Advanced Television Systems Committee (ATSC) Digital TV [3], etc, a multi-mode FEC design will cause a lower cost design and better system integrations. For the RS code, it is not easy to implement a decoder that meets different finite field definitions and generator polynomials. Thus, each application has its own dedicated hardware for RS decoding. Moreover, memory controller of interleaver is also difficult to generate proper addresses for multi-standard. Hence, an efficient algorithm and architecture for multi-mode design is an important issue and challenge to lower down the design cost.

### *1.2 Introduction to the platform*

In ITU-T J.83, DVB-T, and ATSC Digital TV, etc, there are some similarities in FEC sections. However, ITU-T J.83 has the most kinds of modes than the other standards. Besides, the FEC sections in DVB-T and ATSC Digital TV are also included in ITU-J.83. Hence, ITU-T J.83 is chosen as the simulation plat form.

ITU-T J.83 is the digital multi-program systems for television, sound and data services for cable network. There are four annexes (A, B, C, and D), which provide the specification of J.83, including the frame structure, channel coding, and the method of modulation. A comparison of FEC section in different annexes of ITU-T J.83 is listed in table 1. The FEC is based on a concatenated coding approach that produces high coding gain at the moderate complexity and overhead. For main modules, there are three modes in RS codes, various parameters in convolutional interleaving, three kinds of scrambler and one trellis coding.

The main purpose of each FEC module is summarized as follows:

- (1) Reed-Solomon coding-Provide block encoding and decoding to correct up  $t$  symbols within each RS block. It can slightly resist channel burst errors.
- (2) Interleaving-Evenly spread the symbols by disordering the data sequence. It can protect the transmitted data against channel burst errors from being sent to RS decoder.
- (3) Scrambler-Randomize the data to allow effective modulation and prevent high PAPR (Peak-to-Average Power Ratio) after IFFT (inverse fast Fourier transform for OFMD-based systems). High PAPR will lower down the system performance. Scrambler can prevent this situation from happening.
- (4) Trellis coding-It is also called convolutional coding. By using Viterbi decoding algorithm (maximum likelihood decoding on trellis), it has very good ability of error correction. However, it is sensitive to channel burst errors. Hence, convolutional codes usually collaborate with interleaver and RS codes.

The details of FEC algorithm, decoding architecture, and the result of chip implementation will be described in later chapters, respectively.

Table 1: Comparison of different specification in FEC of ITU-T J.83

Item	Annex B	Annex A & C	Annex D
Scrambler	$x^3 + x + \alpha^3$ over $GF(2^7)$	$1 + x^{14} + x^{15}$ for 15-bits polynomial of the PRBS	$1 + x + x^3 + x^6 + x^7 + x^{11} + x^{12} + x^{13} + x^{16}$ for 16-bits polynomial of the PRBS
Reed-Solomon coding	(128,122) extended RS codes over $GF(2^7)$ , $t = 3$	(204,188) RS codes over $GF(2^8)$ , $t = 8$	(207,187) RS codes over $GF(2^8)$ , $t = 10$
Interleaving	Convolutional interleaving depth: I=128,64,32,16,8 J=1,2,3,4,5,6,7,8,16	Convolutional interleaving depth: I=12 J=17	Convolutional interleaving depth: I=52 J=4
Trellis coding	Rate 4/5, G=(25,37octal) with punctured	None	

### 1.3 Thesis Organization

The organization of this thesis is as follows. In chapter 2, the algorithm of FEC will be described, including the algorithm of FEC encoder and decoder. It contains scrambler,

interleaver, Reed-Solomon codes and convolutional codes. And, the proposed algorithms and architectures of FEC for multi-standard will be addressed in chapter 3, which mainly contains a multi-mode RS decoder with memories to store and correct the received data and a memory-based universal convolutional interleaver and de-interleaver with a simple address generator. Chapter 4 will show the result of the chip implementation, the simulation result and will do some comparisons between other reference works and the proposed result. The last chapter is the conclusion and the future work.

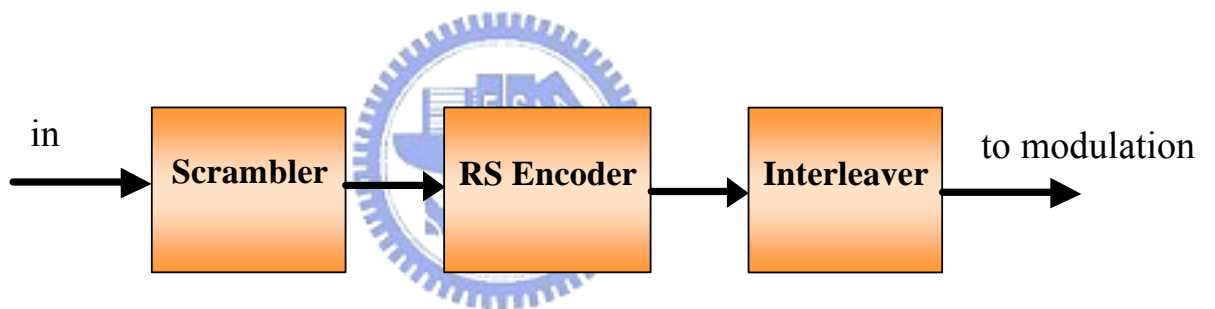


# Chapter 2

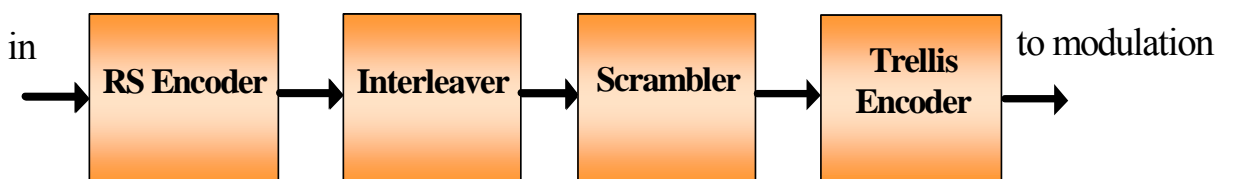
## Algorithm of FEC

---

First of all, the encoder and decoder of FEC will be introduced. In ITU-T J.83, it can be divided into two main parts and composed of three or four processing layers. The first one is shown in figure 2.1(a), including ITU-T J.83 annex A, C, and D. The other is shown in figure 2.1(b), including ITU - T J.83 annex B. The following sections will define and introduce the algorithm of each layer in FEC.



(a)



(b)

Figure 2.1: (a) FEC in ITU-T J.83 annexes A, C and D. (b) FEC in ITU-T J.83 annex B

### 2.1 Scrambler

The basic idea of scrambler is to randomize the transmitted data to provide the even distribution of the symbols in the constellation and to ensure adequate binary transitions for clock recovery.

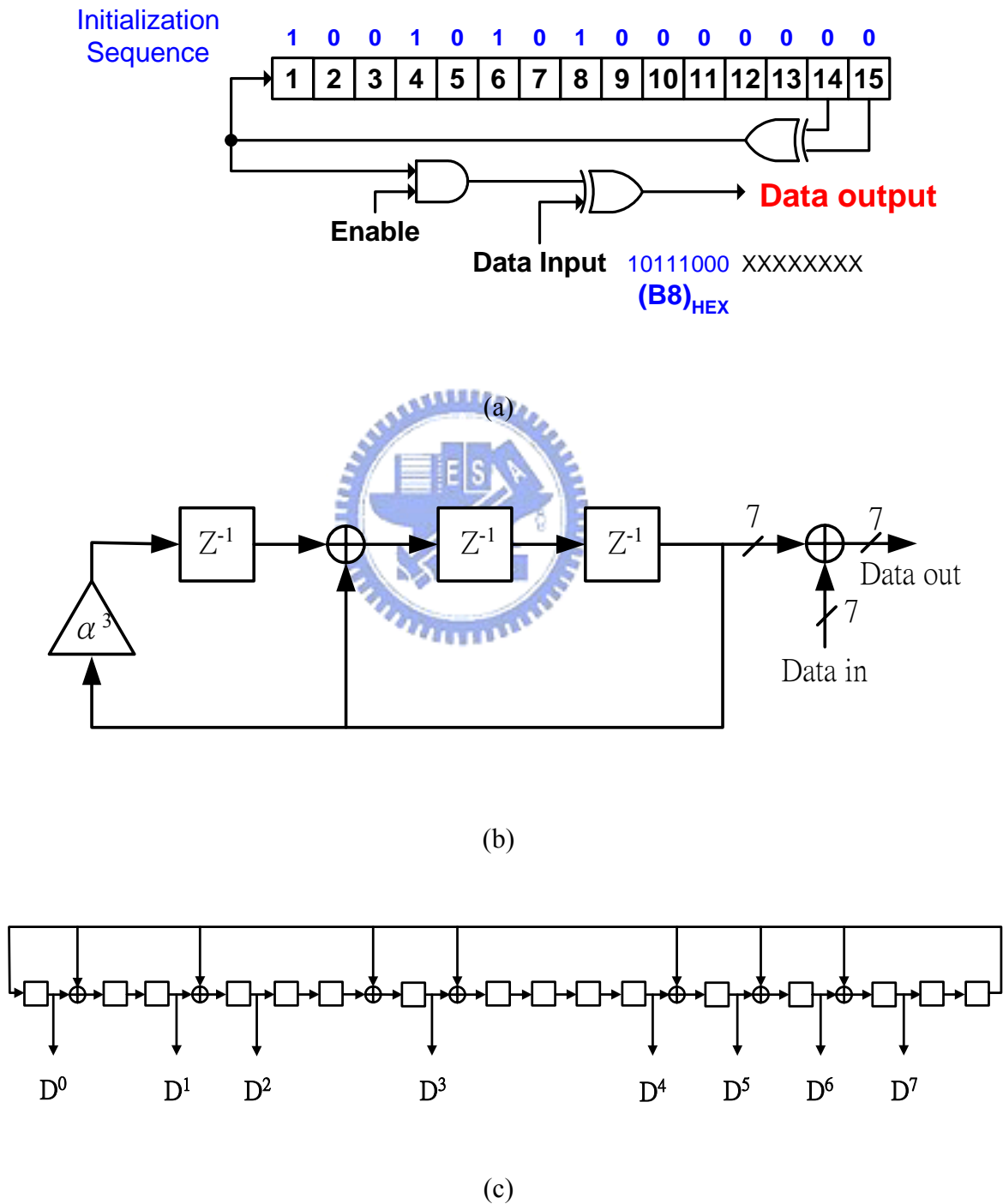


Figure 2.2: Scrambler in (a) J.83A, C and DVB-T system. (b) J.83B. (c) J.83D



Figure 2.2(a) shows the scrambler in J.83 annexes A, C and DVB-T systems. The scrambler adds a Pseudorandom Noise (PN) sequence to input symbols. And, the polynomial for the Pseudo-Random Binary Sequence (PRBS) generator is:

$$x^{15} + x^{14} + 1 \quad (2.1)$$

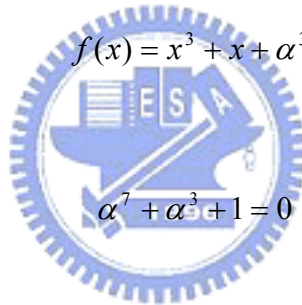
At the start of every eight transport packets, the PRBS registers shall be initiated to the sequence “100101010000000”.

Figure 2.2(b) shows the scrambler in J.83B. The scrambler adds a PN sequence of 7-bit symbols over GF(128) to the input symbols to assure a random transmitted sequence. Initialization is defined as pre-loading to the “all one” state. The scrambler uses a linear feedback shift register specified by a GF(128) polynomial defined as follows:

$$f(x) = x^3 + x + \alpha^3 \quad (2.2)$$

Where:

$$\alpha^7 + \alpha^3 + 1 = 0 \quad (2.3)$$



The scrambler generator polynomial and initialization in J.83D are shown in figure 2.2(c). The PRBS is generated in a 16-bit shift register that has nine feedback taps. Eight of the shift registers outputs are selected as the fixed randomizing byte ( $D^7 D^6 D^5 D^4 D^3 D^2 D^1 D^0$ ), where each bit from this byte is used to individually XOR the corresponding input data bit. The random generator polynomial is denoted as:

$$x^{16} + x^{13} + x^{12} + x^{11} + x^7 + x^6 + x^3 + x + 1 \quad (2.4)$$

Where initialization is defined as pre-load to F180<sub>h</sub>, indicating the registers of  $x^{16}$ ,  $x^{15}$ ,  $x^{14}$ ,  $x^{13}$ ,  $x^9$  and  $x^8$  will be loaded to 1 during the field sync interval.

The structure of de-scrambler is the same as scrambler since PRBS generator is

constructed by shift registers and all operations are XORS.

## 2.2 *Interleaving*

The main purpose of interleaving is to resist burst errors, which are induced in noisy channel. It rearranges the order of input data sequence. Generally, there are two kinds of techniques of interleaving. One is block interleaving, and the other is convolutional interleaving. However, convolutional interleaving has better ability to spread burst errors than block interleaving.

The structure of (I, J) convolutional interleaver and deinterleaver based on Forney approach [8] and Ramsey type III approach [5] is shown in figure 2.3. The parameter I is the interleaving depth and is chosen to be larger than the maximum expected length of burst errors. It also represents that there are I branches in the structure of convolutional interleaving. The parameter J is usually chosen such that  $I \times J$  should be larger than the decoding constraint length for convolutional codes. It also means that branch 0 has zero delays in convolutional interleaver. And, there are J shift registers in branch 1, 2J shift registers in branch 2, and so on,  $(I-1) \times J$  shift registers in branch I-1. Convolutional de-interleaver has the inverse of this property. Hence, the memory requirement is  $J \times I \times (I-1) / 2$  in both convolutional interleaver and deinterleaver. The total end-to-end delay is  $J \times I \times (I-1)$ . This is half the required delay and memory in the block interleaving.

The operation of convolutional interleaving is that at the start of the FEC frame, the input switch is initialized to the top-most branch. Then, the input switch is cyclically connected to the other branches as the valid symbols come in. So does the output switch. And, the input and output switches shall be synchronized.

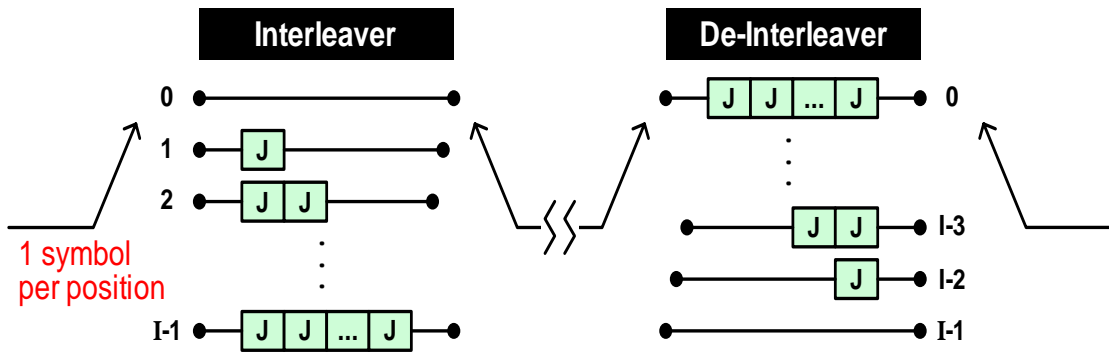


Figure 2.3: Structure of (I, J) convolutional interleaving

Taking (12, 17) convolutional interleaver for an example, this interleaver is adopted in J.83 annexes A, C and DVB-T systems. Assume the input sequence is 0, 1, 2, ..., 204, 205, ..., and so on. Where the number means the input timing index. And the output sequence will be 0, x, x, ..., x, 12, x, x, ..., x, 204, 1, x, x, ..., x, 2244, 2041, ..., 11, and so on, as shown in figure 2.4. Where x means “the don't care symbols” at the beginning transmission. Hence, the burst errors will be spread out as the pseudo noise after deinterleaving. And, the data should be reordered to the original sequence after deinterleaving in receiver part.

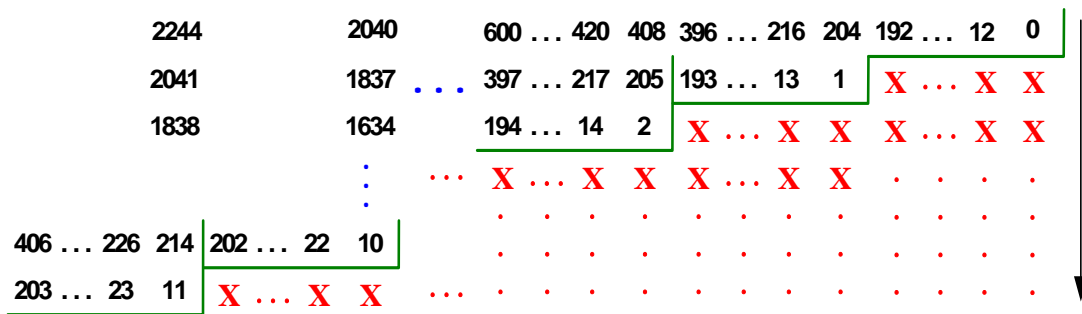


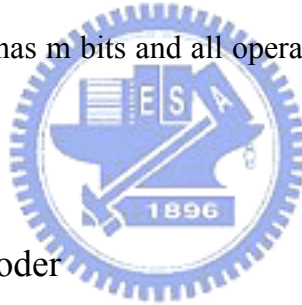
Figure 2.4: The output symbols in convolutional interleaver with I = 12, J = 17

In J.83 annexes A, C, DVB-T and ATSC Digital TV system, the convolutional interleaving is with I = 12 and J = 17. In J.83 annex D, the convolutional interleaving is with I = 52 and J = 4. The upper systems have only one dedicated parameters. However, the convolutional interleaving in J.83 annex B has lots of different modes to be operated. That is,

I can be 128, 64, 32, 16 and 8. J can be 1, 2, 3 ~ 7, 8 and 16. The most critical mode is with I = 128, J = 8. The detail information about specifications is in [1].

## 2.3 Reed-Solomon Codes

Reed-Solomon codes have become the most important code of various types of error-control codes due to its superior capability for burst error correcting and the feasibility for digital implementation. Hence, RS codes are widely adopted in many data communication applications, such as digital TV system, compact disk (CD), and digital versatile disk (DVD). It is adopted in DVB-T, ITU-J.83 cable systems, too. A (N, K) RS codes over  $GF(2^m)$  contain N coded symbols with K message symbols and can correct up to  $t = \lfloor (N-K) / 2 \rfloor$  errors. Note that each symbol over  $GF(2^m)$  has m bits and all operations in RS codes are based on  $GF(2^m)$  [11][12].



### 2.3.1 Reed-Solomon encoder

Let  $(M_{K-1}, M_{K-2}, \dots, M_1, M_0)$  denote K message symbols that are to be transmitted. So the message polynomial:

$$M(x) = M_{K-1}x^{K-1} + M_{K-2}x^{K-2} + \dots + M_1x + M_0 \quad (2.5)$$

And, there is a generator polynomial:

$$g(x) = (x + \alpha^h)(x + \alpha^{h+1}) \dots (x + \alpha^{h+2t-1}) \quad (2.6)$$

Where  $g(x)$  has the degree of  $2t$ ,  $h$  may be 0 or 1, and  $\alpha$  is the primitive  $n$ -th root over  $GF(2^m)$ . Firstly, the message polynomial  $M(x)$  is multiplied by  $x^{2t}$  and then divided by the generator polynomial  $g(x)$  to obtain a remainder polynomial  $R(x)$ :

$$M(x)x^{2t} = q(x)g(x) + R(x) \quad (2.7)$$

$$R(x) = R_{2t-1}x^{2t-1} + R_{2t-2}x^{2t-2} + \dots + R_1x + R_0 \quad (2.8)$$

Then, the codeword polynomial  $C(x)$  with the systematic form can be expressed as:

$$\begin{aligned} C(x) &= q(x)g(x) = M(x)x^{2t} + R(x) \\ &= M_{K-1}x^{2t+K-1} + \dots + M_0x^{2t} + R_{2t-1}x^{2t-1} + \dots + R_1x + R_0 \end{aligned} \quad (2.9)$$

The previous description of RS encoder can be implemented as the systematic feedback shift register encoder as shown in figure 2.5 [11][12], where  $G_0, G_1, \dots, G_{r-1}$  is the coefficient of the generator polynomial. In first  $K$  cycles, it will output the message  $M(x)$ . In last  $N-K$  cycles, it will output  $R(x)$ . This forms the final codeword  $C(x)$ .

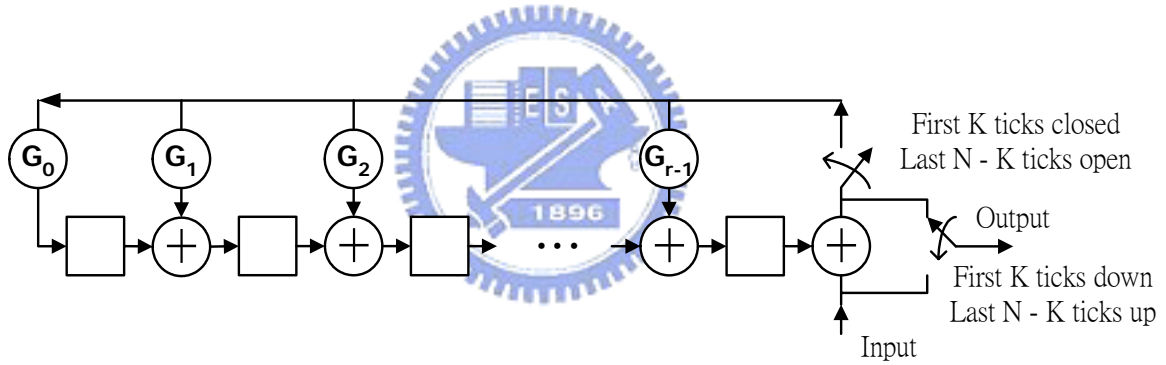


Figure 2.5: The circuit of the systematic feedback shift register RS encoder

For J.83 annex A and C, the  $(204, 188)$  RS codes over  $GF(2^8)$  are utilized for correcting 8 errors. The code generator polynomial is denoted as:

$$g(x) = (x + \alpha^0)(x + \alpha^1) \dots (x + \alpha^{15}) \quad (2.10)$$

Where  $\alpha$  represents the primitive element for the primitive polynomial:

$$p(x) = x^8 + x^4 + x^3 + x^2 + 1 \quad (2.11)$$

For J.83 annex B, the RS encoder is utilized to implement a  $t = 3$ , (128, 122) extended RS codes over  $GF(2^7)$ . The primitive polynomial used to form the field over  $GF(2^7)$  is:

$$p(x) = x^7 + x^3 + 1 \quad (2.12)$$

And the generator polynomial is:

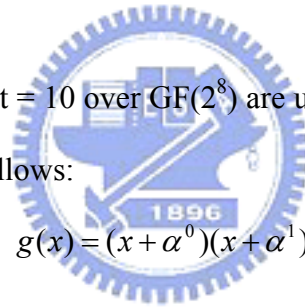
$$g(x) = (x + \alpha^1)(x + \alpha^2) \cdots (x + \alpha^5) \quad (2.13)$$

After  $C(x)$  is generated from equation (2.9), an extended parity symbol  $C_-$  is generated by evaluating the codeword at the sixth power of  $\alpha$  and denoted as  $C_- = C(\alpha^6)$ . This extended symbol is used to form the last symbol of a transmitted RS codeword. The extended codeword polynomial  $\hat{C}(x)$  is then as follows:

$$\hat{C}(x) = xC(x) + C_- = M_{121}x^{127} + \cdots + M_0x^6 + R_4x^5 + R_0x + C_- \quad (2.14)$$

(207, 187) RS codes with  $t = 10$  over  $GF(2^8)$  are utilized for J.83 annex D. The generator polynomial  $g(x)$  is shown as follows:

$$g(x) = (x + \alpha^0)(x + \alpha^1) \cdots (x + \alpha^{19}) \quad (2.15)$$



### 2.3.2 Reed-Solomon decoder

Assume the received data polynomial is  $r(x)$ , and error polynomial is  $e(x)$ . That is:

$$r(x) = c(x) + e(x) \quad (2.16)$$

And,

$$e(x) = e_1X_1 + e_2X_2 + \cdots + e_vX_v, 0 \leq v \leq t \quad (2.17)$$

Where  $e_i$  is the error value,  $X_i$  is the error location, and  $v$  is the error numbers. And, Reed-Solomon decoding process can be divided into four steps [4]: (1) syndrome calculator, (2) Key equation solver, (3) chien Search, and (4) error value evaluator, as shown in figure 2.6.

The syndrome calculator calculates a set of syndromes from the received codewords. The key equation solver produces the error locator polynomial  $\sigma(x)$  and the error value evaluator polynomial  $\Omega(x)$  from the syndromes. By the Chien search and the error value evaluator, we can get the error locations and error values respectively.

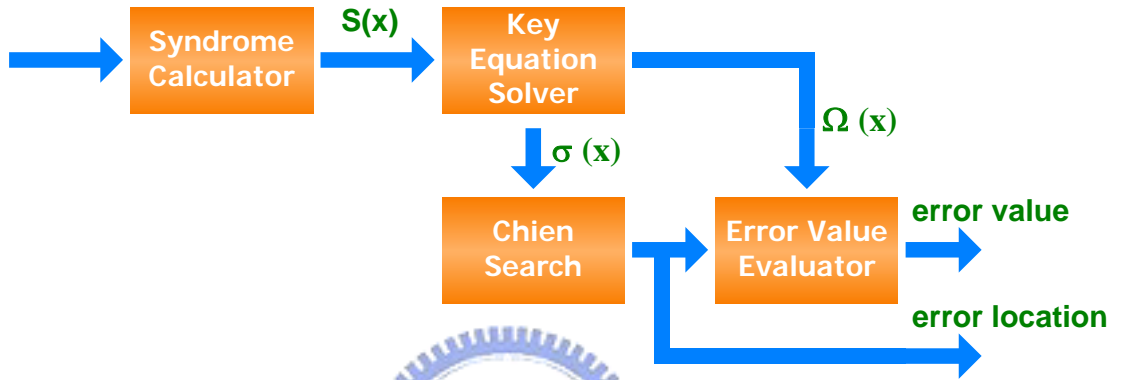


Figure 2.6: RS decoding process

In syndrome calculator, the syndromes are calculated as follows:

$$S_1 = r(\alpha^h) = e(\alpha^h) = e_1 X_1^h + e_2 X_2^h + \dots + e_v X_v^h$$

$$S_2 = r(\alpha^{h+1}) = e(\alpha^{h+1}) = e_1 X_1^{h+1} + e_2 X_2^{h+1} + \dots + e_v X_v^{h+1}$$

⋮

$$S_{2t} = r(\alpha^{h+2t-1}) = e(\alpha^{h+2t-1}) = e_1 X_1^{h+2t-1} + e_2 X_2^{h+2t-1} + \dots + e_v X_v^{h+2t-1} \quad (2.18)$$

Since  $C(\alpha^h) = C(\alpha^{h+1}) = \dots = C(\alpha^{h+2t-1}) = 0$ . Hence, the syndrome polynomial can be defined as:

$$\begin{aligned}
S(x) &= S_1 + S_2x + \dots + S_{2t}x^{2t-1} \\
&= e_1X_1^h(1 + X_1x + X_1^2x^2 + \dots + X_1^{2t-1}x^{2t-1}) \\
&+ e_2X_2^h(1 + X_2x + X_2^2x^2 + \dots + X_2^{2t-1}x^{2t-1}) \\
&\quad \vdots \\
&+ e_vX_v^h(1 + X_vx + X_v^2x^2 + \dots + X_v^{2t-1}x^{2t-1}) \\
&= \frac{e_1X_1^h(1-(X_1x)^{2t})}{1-X_1x} + \frac{e_2X_2^h(1-(X_2x)^{2t})}{1-X_2x} + \dots + \frac{e_vX_v^h(1-(X_vx)^{2t})}{1-X_vx} \quad (2.19)
\end{aligned}$$

The equation (2.19) will be used later for calculating the error values  $e_i$ ,  $0 \leq i \leq t$ .

For the extended RS codes in J.83B, the syndrome should be modified. Recall  $r(x) = \hat{C}(x) + e(x) = xC(x) + C_{-} + e(x)$ ; there are two cases discussed individually as follows:

(1)  $r_0$  is not an error, meaning  $r_0 = C_{-}$ .

The decoding procedure is the same as the normal cases.

(2)  $r_0$  is an error, meaning  $r_0 = C_{-} + e_{-}$ . Then,

$$\begin{aligned}
S_1 &= r(\alpha^h) = e(\alpha^h) = e_1X_1^h + e_2X_2^h + \dots + e_vX_v^h + e_{-} \\
S_2 &= r(\alpha^{h+1}) = e(\alpha^{h+1}) = e_1X_1^{h+1} + e_2X_2^{h+1} + \dots + e_vX_v^{h+1} + e_{-} \\
&\quad \vdots \\
S_{2t} &= r(\alpha^{h+2t-1}) = e(\alpha^{h+2t-1}) = e_1X_1^{h+2t-1} + e_2X_2^{h+2t-1} + \dots + e_vX_v^{h+2t-1} + e_{-} \quad (2.20)
\end{aligned}$$

While  $v \leq t$ , the error value  $e_{-}$  can be calculated to let the discrepancy  $\Delta^{(2t-1)} = 0$  during solving key equation by Berlekamp-Massey algorithm that will be introduced later.

The key equation is defined as follows:

$$\Omega(x) = S(x)\sigma(x) \bmod x^{2t} \quad (2.21)$$

Where  $\Omega(x)$  is the error evaluator polynomial, and  $\sigma(x)$  is the error locator polynomial.



The key equation can be solved by Euclidean algorithm or Berlekamp-Massey (BM) algorithm [11][12][19].

An inversionless BM algorithm which is a  $2t$ -step iterative algorithm is shown as follows [4]:

Initial condition:

$$D^{(-1)} = 0; \delta = 1; \sigma^{(-1)}(x) = \tau^{(-1)}(x) = 1; \Delta^{(0)} = S_1 \quad (2.22)$$

For ( $i = 0$  to  $2t-1$ )

$$\begin{cases} \sigma^{(i)}(x) = \delta \cdot \sigma^{(i-1)}(x) + \Delta^{(i)} x \tau^{(i-1)}(x) \\ \Delta^{(i+1)} = S_{i+2} \cdot \sigma_0^{(i)} + S_{i+1} \sigma_1^{(i)} + \dots + S_{i+2-t} \sigma_t^{(i)} \end{cases} \quad (2.23)$$

If ( $\Delta^{(i)} = 0$  or  $2D^{(i-1)} \geq i+1$ )

$$D^{(i)} = D^{(i-1)}, \quad \tau^{(i)}(x) = x \tau^{(i-1)}(x) \quad (2.24)$$

else

$$D^{(i)} = i+1 - D^{(i-1)}, \quad \delta = \Delta^{(i)}, \quad \tau^{(i)}(x) = \sigma^{(i-1)}(x) \quad (2.25)$$

Where  $\sigma^{(i)}(x)$  is the  $i$ -th step error locator polynomial and  $\sigma_j^{(i)}$  is the coefficient of  $\sigma^{(i)}(x)$ .  $\Delta^{(i)}$  is the  $i$ -th step discrepancy and  $\delta$  is the previous discrepancy.  $\tau^{(i)}(x)$  is an assisting polynomial and  $D^{(i)}$  is an assisting degree variable in  $i$ -th step.

And the modified inversionless BM algorithm with some differences in initial conditions can be shown as follows [4]:

$$\sigma_j^{(i)} = \begin{cases} \delta \cdot \sigma_0^{(i-1)}, & \text{for } j = 0 \\ \delta \cdot \sigma_j^{(i-1)} + \Delta^{(i)} \tau_{j-1}^{(i-1)}, & \text{for } 1 \leq j \leq t \end{cases} \quad (2.26)$$

$$\Delta_j^{(i+1)} = \begin{cases} 0, & \text{for } j = 0 \\ \Delta_{j-1}^{(i+1)} + S_{i-j+3} \cdot \sigma_{j-1}^{(i)}, & \text{for } 1 \leq j \leq t \end{cases} \quad (2.27)$$

Where  $\tau_j^{(i)}$  is the coefficient of  $\tau^{(i)}(x)$  and  $\Delta_j^{(i)}$  is the partial results in computing  $\Delta^{(i)}$ .

Besides, if  $\sigma(x)$  is first obtained, from the key equation and the Newton's identity we could derive  $\Omega(x)$  as follows:

$$\Omega_j^{(i)} = \begin{cases} S_{i+1} \cdot \sigma_0, & \text{for } j = 0 \\ \Omega_{j-1}^{(i)} + S_{i-j+1} \cdot \sigma_j, & \text{for } 1 \leq j \leq i \end{cases} \quad (2.28)$$

These modified inversionless BM equation will be adopted in our proposed multi-mode key equation solver because of its regularity.

The alternative algorithm of key equation solver is Euclidean algorithm. It can be summarized as follows:

Initial condition:

$$\begin{cases} \sigma^{-1}(x) = 0, \sigma^0(x) = 1 \\ \Omega^{-1}(x) = x^{2t}, \Omega^0(x) = S(x) \end{cases} \quad (2.29)$$

Do the following operation until  $\deg\{\sigma(x)\} > \deg\{\Omega(x)\}$ :

$$\begin{cases} \sigma^i(x) = \sigma^{i-2}(x) + q^{i-1}(x)\sigma^{i-1}(x) \\ \Omega^i(x) = \Omega^{i-2}(x) + q^{i-1}(x)\Omega^{i-1}(x) \end{cases} \quad (2.30)$$

Where  $\sigma^i(x)$  is the i-th step error locator polynomial,  $\Omega^i(x)$  is the i-th step error evaluator polynomial, and  $q^i(x)$  is the i-th quotient polynomial generated in key equation.

After solving the key equation, we find the roots of  $\sigma(x)$  for error location  $X_1, X_2, \dots, X_v$  in chien search, where the roots of  $\sigma(x)$  are  $X_1^{-1}, X_2^{-1}, \dots, X_v^{-1}$ . Hence,  $\sigma(x)$  can be represented as:

$$\sigma(x) = (1 - X_1x)(1 - X_2x) \cdots (1 - X_vx) \quad (2.31)$$

Then, using Forney algorithm to calculate error values in error value evaluator and together key equation and equation (2.19), we can get:

$$\begin{aligned}
\Omega(x) &= S(x)\sigma(x) \bmod x^{2t} \\
&= e_1 X_1^h (1 - X_2 x)(1 - X_3 x) \cdots (1 - X_v x) \\
&\quad + e_2 X_2^h (1 - X_1 x)(1 - X_3 x) \cdots (1 - X_v x) \\
&\quad \vdots \\
&\quad + e_v X_v^h (1 - X_1 x)(1 - X_2 x) \cdots (1 - X_{v-1} x)
\end{aligned} \tag{2.32}$$

And, the former derivative of  $\sigma(x)$  can be represented as:

$$\begin{aligned}
\frac{d\sigma(x)}{dx} &= X_1 (1 - X_2 x)(1 - X_3 x) \cdots (1 - X_v x) \\
&\quad + X_2 (1 - X_1 x)(1 - X_3 x) \cdots (1 - X_v x) \\
&\quad \vdots \\
&\quad + X_v (1 - X_1 x)(1 - X_2 x) \cdots (1 - X_{v-1} x) \\
&= \frac{\sigma_{odd}(x)}{x}
\end{aligned} \tag{2.33}$$

Where  $\sigma_{odd}(x)$  is the odd parts of  $\sigma(x)$ . Hence, combine the equations (2.32) and (2.33), the error values can be calculated as follows:

$$e_i = \frac{\Omega(X_i^{-1})}{\sigma'(X_i^{-1}) X_i^{h-1}} \tag{2.34}$$

Where  $\sigma'(x)$  is the formal derivative of  $\sigma(x)$  over  $\text{GF}(2^m)$ .

According to the error locations and error values solved from previous algorithm, we can correct the channel induced errors in received data and get the correct codeword. Unfortunately, if error numbers in one codeword are larger than  $t$ , we could not correct the received data.

For more information about RS decoding process, please see [4] [11][12][19].

## 2.4 Trellis Codes

In a  $(n, k, m)$  Trellis code (or called convolutional code), the coded  $n$ -bit output block depends not only on the corresponding  $k$ -bit input message block, but also on the  $m$  previous message blocks. It can be implemented with an  $n$ -output,  $k$ -input linear sequential circuit with an input memory of  $m$  words. The advantage of the convolutional codes is that it allows the introduction of redundancy to improve the threshold Signal-to-Noise Ratio.

Only J.83 annex B contains the trellis code. This trellis-coded modulator is a 16-state non-systematic rate  $1/2$  encoder with the generator:

$$(G_1, G_2) = (25, 37_{\text{octal}})$$

The punctured matrix proposed in [13] essentially converts the rate  $1/2$  encoder to rate  $5/4$ .

The punctured matrix is defined as:

$$(P_1, P_2) = (0001, 1111)$$

The internal structure of the punctured convolutional encoder is illustrated in figure 2.7.

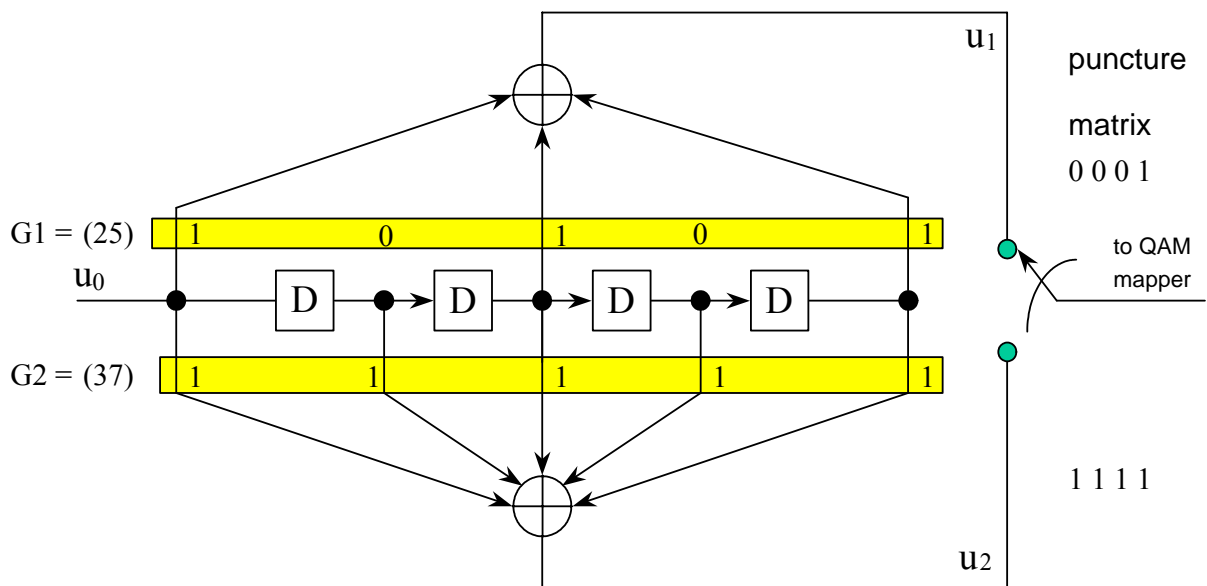


Figure 2.7: Punctured binary convolutional codes in ITU-T J.83B

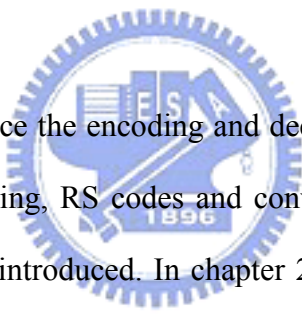
The Viterbi algorithm proposed in 1967 is a straightforward implementation of the maximum likelihood (ML) decoder and is the most powerful and popular algorithm for decoding convolutional codes [14][15]. The following four steps are Viterbi algorithm, which can be applied to find the ML path:

- (1) According to the current received input datum, we calculate the transition metrics (TM) to the next transition states.
- (2) Sum the previous path metrics (PM) with the calculated TM and compare two paths, which come from different states but merge at the same current state. Then, we select the path with the smallest distance. This operation is called ACS (Add-Compare-Select), and we use ACS unit for each state.
- (3) The output of the select branch in each state is stored into the memory, which is called “survivor path”.
- (4) Repeat (1), (2) and (3) until the memory of survivor path is full, then the output decision begins to trace-back the survivor path to find the output of the smallest path metrics (the

ML path).

In practice, the register-exchange approach and trace-back approach are useful methods for survivor path storage management in Viterbi decoder architecture. The former one takes more area but less time than the latter one. We will use register-exchange method to implement the survivor path storage management in Viterbi decoder since the convolutional codes in J.83B has only 16 states and thus the number of registers required for this decoder is not quite large. The detail architecture of Viterbi decoder for J.83B will be introduced in chapter 3.4.2.

## 2.5 Summary



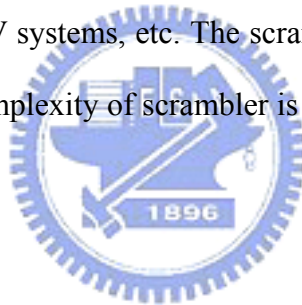
In this chapter, we introduce the encoding and decoding algorithm of each FEC section. It includes scrambler, interleaving, RS codes and convolutional codes. In chapter 2.1, three kinds of scrambler of J.83 are introduced. In chapter 2.2, both convolutional interleaver and deinterleaver are introduced. It has more advantage than block interleaving. In chapter 2.3, the encoding and decoding algorithm of RS codes is introduced. In RS decoding algorithm, two kinds of key equation solver are presented. One is BM algorithm, and the other is Euclidean algorithm. We also introduce three kinds of RS codes among J.83, one is over  $GF(2^7)$  with  $t = 3$ , the others are in  $GF(2^8)$  with  $t = 8$  and  $10$ , respectively. In chapter 2.4, we introduce the convolutional codes and Viterbi algorithm. Fortunately, it has only one mode in J.83, that is, a 16-state non-systematic rate 1/2 encoder with the generator:  $(G_1, G_2) = (25, 37_{\text{octal}})$ .

# **Chapter 3**

## **Algorithm and Architecture for Multi - Mode FEC Decoder**

---

The algorithm and architecture of a multi-mode RS decoder with memories to store and correct the received data and a memory-based universal convolutional interleaver/de-interleaver will be proposed in this chapter. These two modules are compatible for ITU-T J.83, DVB-T, ATSC Digital TV systems, etc. The scrambler and Viterbi decoder will be only mentioned briefly since the complexity of scrambler is so simple and there is only one kind of convolutional codes.



### ***3.1 The proposed multi-mode FEC decoder***

Figure 3.1 shows the block diagram of the proposed multi-mode FEC decoder. It integrates all systems from figure 2.1 into one system. The symbols A/B/C/D represent the annex A/B/C/D in ITU-T J.83. The different data paths between J.83 annex B and annex A/C/D are decided by multiplexer.

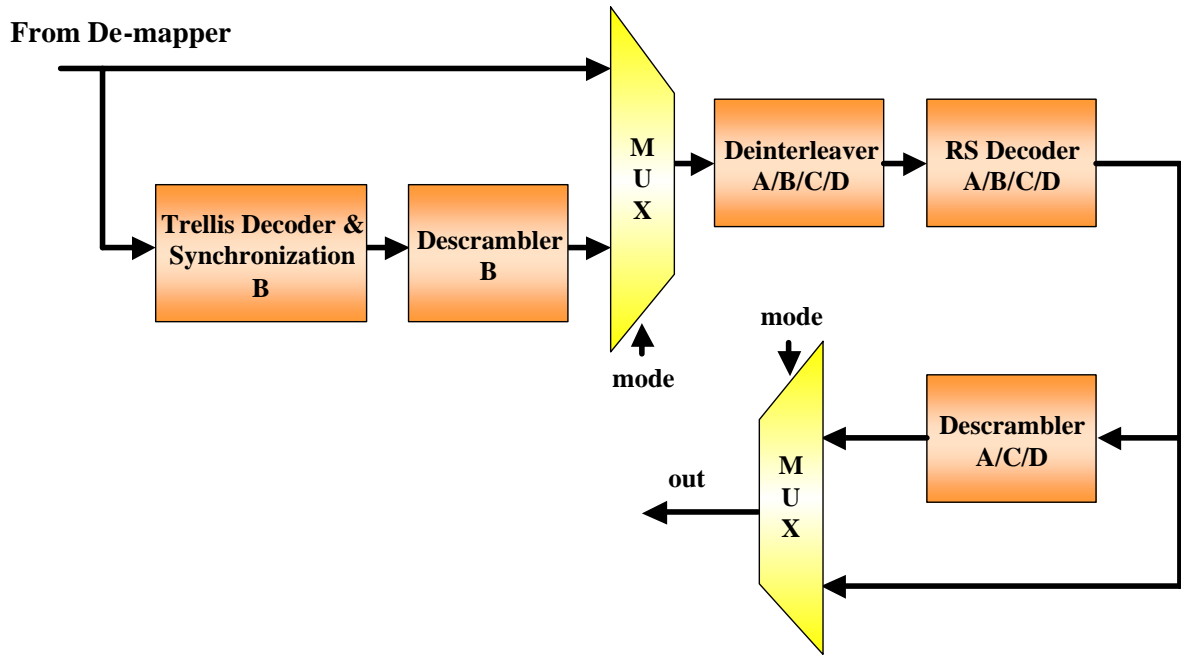


Figure 3.1: The proposed multi-mode FEC decoder

### 3.2 *Memory-based universal convolutional interleaver/de-interleaver*



It is not efficient for implementing so many pieces of FIFO in convolutional interleaver or deinterleaver since it consumes lots of power, area and induces routing difficulty in APR (Auto Placement and Route). Hence, a better solution is to use SRAM to solve these problems. The key issue becomes how to generate the correct address of SRAM for each input and output data. As a result, a novel, low complexity, high flexibility and memory-based method to implement the multi-mode convolutional interleaver and deinterleaver is proposed, which is induced from [6][7].



### 3.2.1 The algorithm and architecture of memory-based universal convolutional interleaving

The idea is that we rebuilt the FIFO registers of convolutional deinterleaver as a memory array. Assume the FIFO registers in first branch are put in somewhere of the memory array, and the FIFO registers in second branch are appended latter, and so on, until the last FIFO registers are appended. Hence, the memory array is as shown in figure 3.2. For writing, we realize that after writing first symbol into the head of the memory array, the next symbol should be written into the head of the second branch, i.e., the address distance of memory between first symbol and second symbol is  $(I-1) \times J$ . The values are the same as the numbers of the FIFO in first branch. Hence, we call this “branch address”. And the address for first symbol is called intra-initial address. For the third symbol, the address distance of memory between second symbol and third symbol is  $(I-2) \times J$ . And so on, the address distance of memory between  $(I-2)$ -th symbol and  $(I-1)$ -th symbol is  $2J$ . In contrast to write, the first readout symbol should be in the end of the first branch in memory array. The second readout symbol should be in the end of the second branch, i.e., the address distance between first symbol and second symbol is  $(I-2) \times J$ . Similarly, the address distance between second symbol and third symbol is  $(I-3) \times J$ . And so on, the address distance between  $(I-2)$ -th symbol and  $(I-1)$ -th symbol is  $J$ . For the coincidence of writing and reading direction, the initial address pointer should be decreased by 1 for the next  $I$  symbols. Then, do the previous operation again. In addition, the memory size should be defined. If the memory address is out of the memory size, it should modulo the address by the memory size.

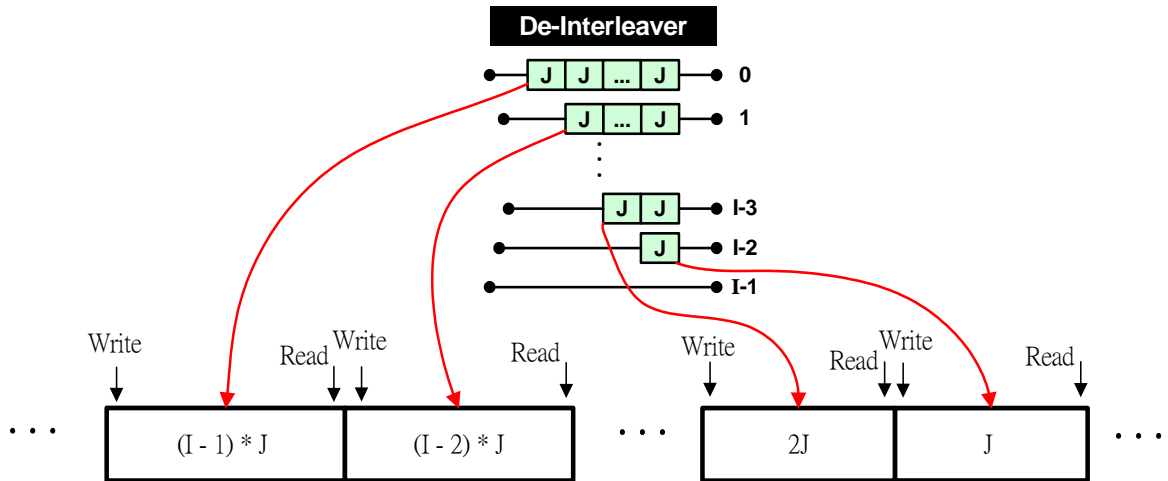


Figure 3.2: The memory array by rebuilding the FIFO registers of deinterleaver

A (12, 17) convolutional deinterleaver which is adopted in ITU-T J.83A, C and DVB-T system will be taken for an example to show how it works. Assume the datum we received are 0, x, x, ..., x, 12, x, x, ..., x, 204, 1, x, x, ..., x, 2244, 2041, ..., 11, ..., as shown in figure 2.4. Where the number means the input indexes from interleaver, and x means “don’t care symbols” at the beginning. When deinterleaving, after writing 0 to memory, the interval between 0 and the next writing address is  $(I-1) \times J = 187$  as shown in figure 3.3(a). The interval between previous address and the next address is  $(I-2) \times J = 170$ , and so on, until to  $2J = 34$ . These numbers are the same as the numbers of FIFO on branches of convolutional deinterleaver. When writing 12 to the memory, it needs go back to the address of “initial writing address-1” and does the previous operation again. After writing 202 into the memory, the data stored in memory is like in figure 3.3(b). Then we can see that the distance between 0 and 1 is  $(I-2) \times J = 170$ . The distance between 1 and 2 is  $(I-3) \times J = 153$ , and so on. The distance between 9 and 10 is  $J = 17$ . At this time, the memory size in figure 3.3(b) is  $J \times I \times (I-1) / 2$ , just the same as the minimum memory requirement in figure 2.3. Because there is no more space to write 2244 into memory, so it must increase more memory sizes. Or it will violate the rules. By the observation, it needs more  $J$  memory size. As shown in figure 3.3(c),

when 0 is read out from memory, 2244 is written into memory. And, 1 is read out, 2041 is written to the original position of 0. Then, do the previous operation again. In addition, when the address is out of the memory size, it must modulo the address by the memory size. Hence, the required memory size is  $J \times I \times (I-1) / 2 + J$ . The maximum size is 65032 bytes for (128, 8) convolutional deinterleaver in J.83B. We realize that it just needs more 8 bytes than the original structure and has the advantage of low cost and high flexibility for multi-mode design.

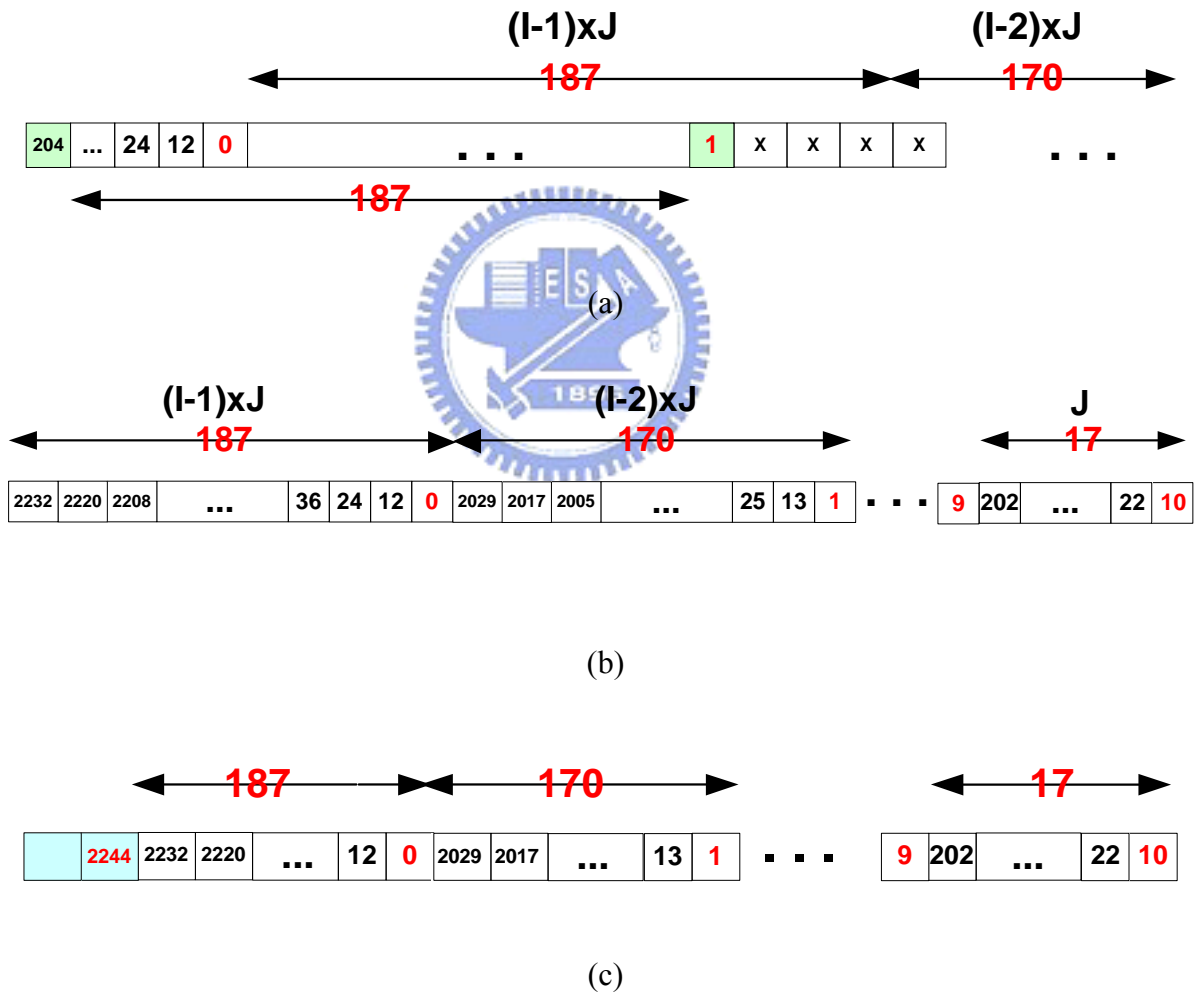


Figure 3.3: Behavior of the novel algorithm for (12, 17) convolutional deinterleaver

The detail operations of universal convolutional deinterleaver are described as pseudo codes in figure 3.4, where there are 12 parameters that we used:

- (1) I: Interleaver depth
- (2) J: The difference delays between each neighboring branch
- (3) in: data input.
- (4) out: data output.
- (5) w\_addr: The writing address for memory input.
- (6) r\_addr: The reading address from memory to output
- (7) w\_ini\_addr: The intra-initial address of w\_addr.
- (8) r\_ini\_addr: The intra-initial address of r\_addr.
- (9) branch\_addr: This is the address between 2 neighboring data.
- (10) counter: For determining when to output directly and reset w\_addr and r\_addr.
- (11) mem\_bound: Maximum size of memory
- (12) mem[ ]: It represent the memory and the size is mem\_bound.

Convolutional interleaver which is the inverse of convolutional deinterleaver can be easily formulated, too.

```

Initial condition:
w_addr = w_ini_addr = 0;      branch_addr = (I-1)*J;
r_addr = r_ini_addr = (I-1)*J;  counter = 1;
mem_bound = J*I*(I-1)/2 + J;

While ( in != NULL)
{
  if (counter == I)      /* In last branch, input will pass to output directly*/
  {
    out = in;
    branch_addr = (I-1)*J;      /* branch_addr goes back to initial condition */
    counter = 1;                /* reset the counter */
    w_ini_addr = w_ini_addr - 1; /* reset the writing and reading address */
    r_ini_addr = r_ini_addr - 1;
    w_ini_addr = w_ini_addr mod mem_bound;      /* mod the address */
    r_ini_addr = r_ini_addr mod mem_bound;
    w_addr = w_ini_addr;      /* set writing address to w_ini_addr */
    r_addr = r_ini_addr;
  }
  else
  {
    out = mem[r_addr];      /* read out from memory */
    mem[w_addr] = in;      /* write data into memory */
    w_addr = w_addr + branch_addr;
    branch_addr = branch_addr - J;
    r_addr = r_addr + branch_addr;
    w_addr = w_addr mod mem_bound;      /* mod the address */
    r_addr = r_addr mod mem_bound;
    counter = counter + 1;
  }
}

```

Figure 3.4: Pseudo codes of universal convolutional deinterleaver

The architecture of the proposed algorithm for convolutional interleaving is depicted in figure 3.5. FSM controls the branch address generator and the intra-initial address generator. Combining the branch address and intra-initial address together forms the final address for memory.

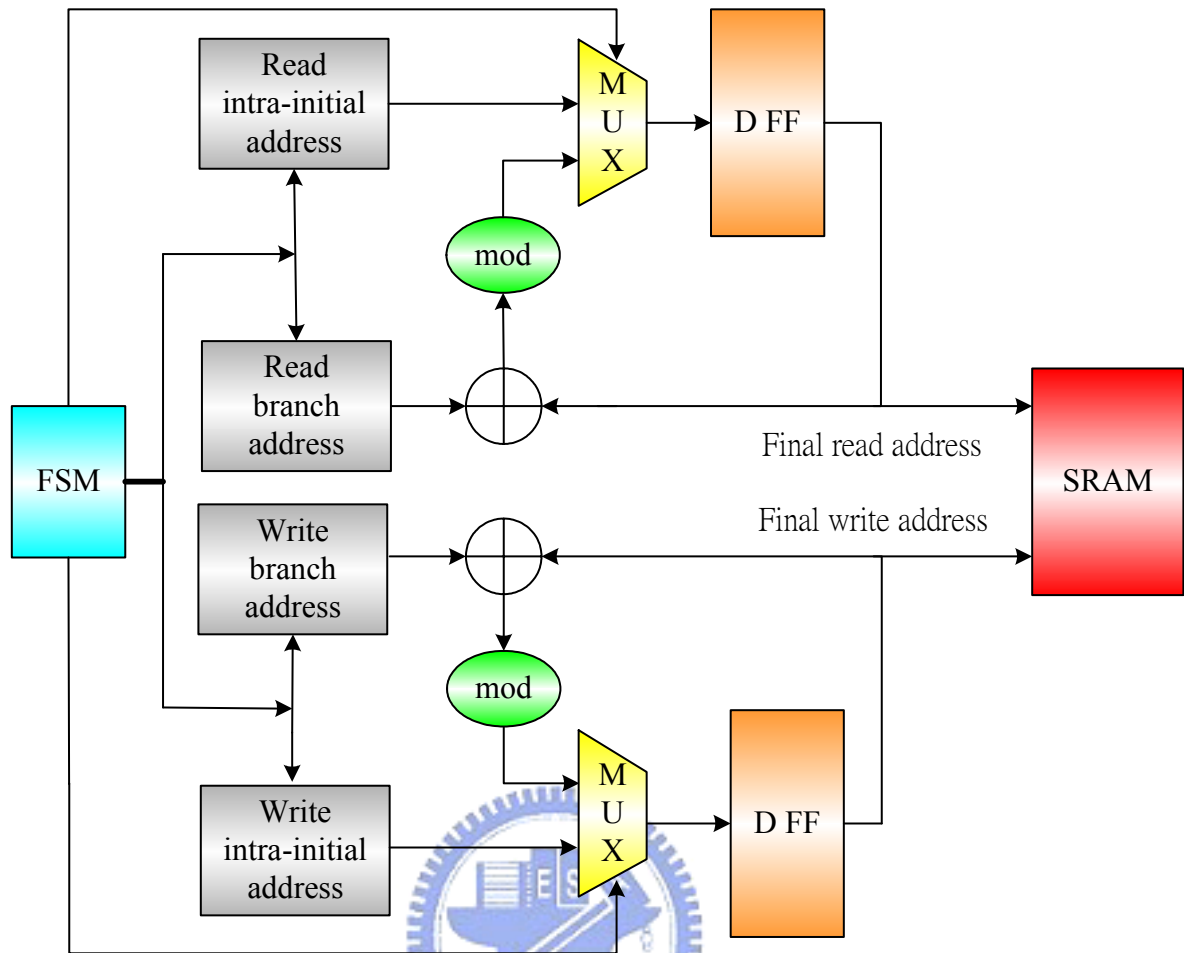


Figure 3.5: The architecture of the address generator for convolutional interleaving

### 3.3 The multi-mode RS decoder

To design a multi-mode RS decoder, at first, a finite field multiplier (FFM) for different finite field definition should be designed. Then, the four steps of RS decoding process [4] can be proceeded. As a result, in the sub-section, the multi-mode FFM will be proposed in the first. Then, the multi-mode syndrome calculator, key equation solver, chien search and error value evaluator will be proposed, respectively. The multi-mode RS decoder can be used in many applications, such as ITU-T J.83, DVB-T systems, etc.

### 3.3.1 Multi-Mode Finite Field Multiplier

For different RS codes, the different primitive polynomial will cause a challenge to design a FFM. However, FFM can be split into multiply and modular operation respectively. The primitive polynomial only has an impact on modular operation. Therefore, the complexity of programmable design just lies in the modular operation. So, a multi-mode FFM is proposed as shown in figure 3.6, where  $p_i(x)$  and  $p_j(x)$  are different primitive polynomial over  $GF(2^m)$  respectively.

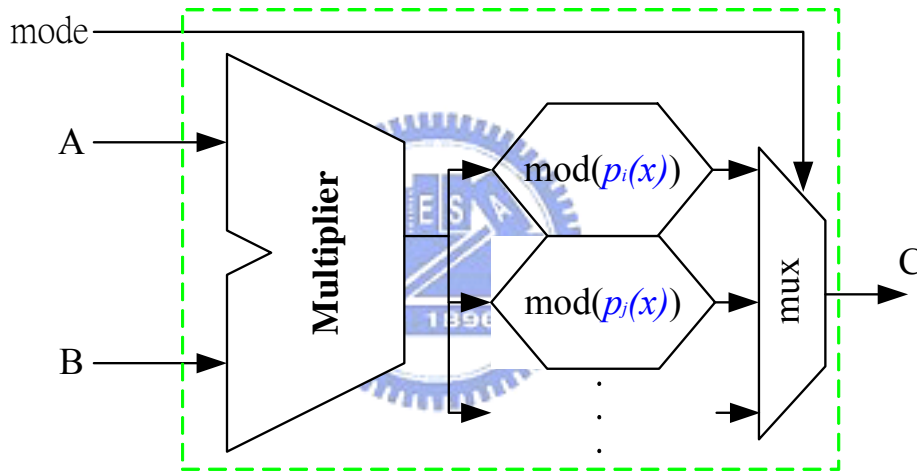


Figure 3.6: Multi-mode FFM over  $GF(2^m)$

### 3.3.2 Syndrome Calculator

To calculate the syndromes, we can use Horner's Rule:

$$\begin{aligned}
 u(x) &= u_n x^n + u_{n-1} x^{n-1} + \cdots + u_1 x + u_0 \\
 &= (\cdots(((u_n x + u_{n-1})x + u_{n-2})x + u_{n-3})\cdots)x + u_0
 \end{aligned} \tag{3.1}$$

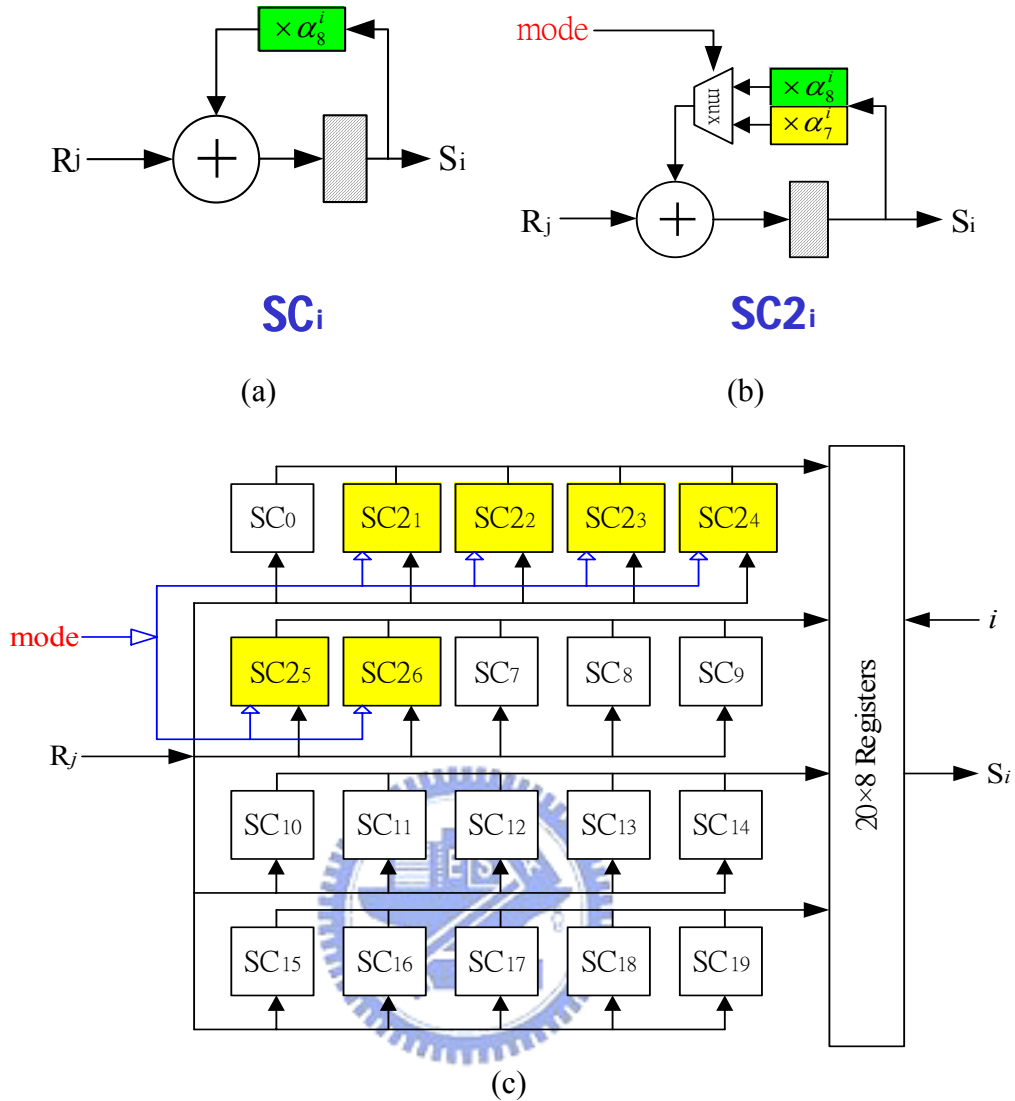


Figure 3.7: Multi-mode syndrome calculator: (a) Basic cell  $SC_i$  for  $GF(2^8)$ . (b) Basic cell  $SC_{2i}$  for dual mode purpose ( $GF(2^8)$  and  $GF(2^7)$ ). (c) The overall structure of multi-mode syndrome calculator

Hence, the basic cell to calculate the syndrome based on Horner's Rule should be proposed at first. In the simulation platform of J.83, there are two kinds of finite field, the one is  $GF(2^8)$ , the other is  $GF(2^7)$ . Besides, the roots of the generator polynomial are from  $\alpha^0$  to  $\alpha^{2^t-1}$  in J.83A, C and D. But in J.83B, the roots of the generator polynomial are from  $\alpha^1$  to  $\alpha^{2^t-1}$ . Hence, the two kinds of different basic cells  $SC_i$  and  $SC_{2i}$  are proposed as shown in figure 3.7(a) and (b).  $SC_i$  is for  $GF(2^8)$ ;  $SC_{2i}$  is for  $GF(2^8)$  and  $GF(2^7)$  which are decided by



the current mode. The architecture of multi-mode syndrome calculator is shown in figure 3.7(c). For different specification, a specific group of cells will be chosen. For J.83 A and C,  $SC_0, SC_1, \dots, SC_{15}$  will be chosen.  $SC_{21}, SC_{22}, \dots, SC_{26}$  will be chosen in J.83B. All basic cells will be chosen for J.83D.

Based on [9], moreover, the first  $t$  syndromes are equal to zeros implies all syndromes are zeros, which can simplify the error detection procedure. It not only improves the power consumption, but also reduces the complexity.

### 3.3.3 Key Equation Solver

To solve the key equation, Berlekamp-Massey algorithm is used due to its regular operation. For different  $t$ , it needs  $2t$  iterations to find error locator polynomial  $\sigma(x)$ . Base on the proposed multi-mode FFM and modified decomposed algorithm [4][9] mentioned in chapter 2.3.2, the architecture of multi-mode key equation solver is proposed as shown in figure 3.8. The computation of  $\Omega(x)$  after  $\sigma(x)$  results in fewer multiplications and additions than the original BM algorithm. It includes only one key equation solver with three proposed multi-mode FFMs to calculate  $\sigma(x)$  and  $\Omega(x)$  respectively. Hence, the hardware complexity is reduced.

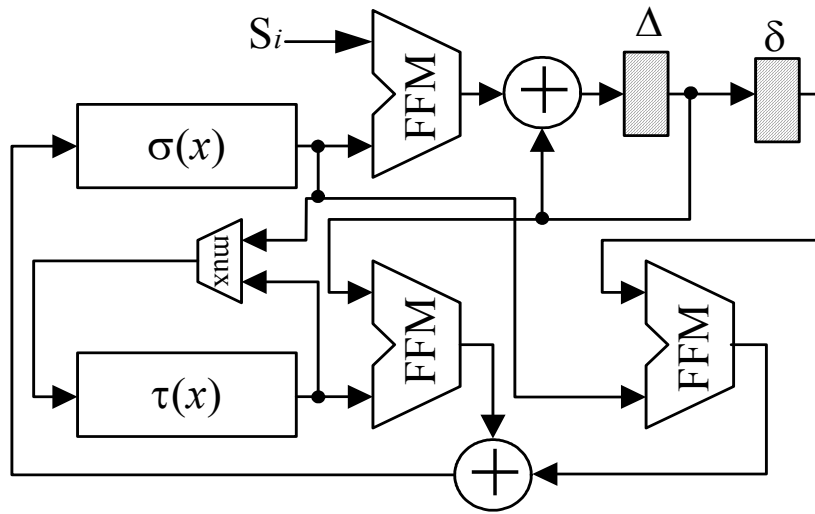


Figure 3.8: Multi-mode key equation solver

### 3.3.4 Chien Search

Similar to syndrome calculator, for the different finite field ( $GF(2^7)$  and  $GF(2^8)$ ) and the capability of error correction  $t$ , the two kinds of basic cells  $C_i$  and  $C_{2i}$  are proposed for multi-mode chien search as shown in figure 3.9(a) and (b).  $C_i$  is designed only for  $GF(2^8)$ .  $C_{2i}$  is designed for  $GF(2^8)$  and  $GF(2^7)$ . And the architecture of multi-mode chien search is depicted in figure 3.9(c). For different specifications, the sums of proper cells will be chosen. The sums of  $C_{2_0}$ ,  $C_{2_1}$ ,  $C_{2_2}$  and  $C_{2_3}$  are chosen for J.83B. The sums of  $C_{2_0}$ ,  $C_{2_1}$ ,  $C_{2_2}$ ,  $C_{2_3}$ ,  $C_4$ ,  $C_5$ , ...,  $C_8$  are chosen for J.83A and C. The sums of  $C_{2_0}$ ,  $C_{2_1}$ ,  $C_{2_2}$ ,  $C_{2_3}$ ,  $C_4$ ,  $C_5$ , ...,  $C_{10}$  are chosen for J.83D. And the cell of  $C_{2_L}$  calculates the current calculating location. If the sums are equal to zero, the location will be stored in the registers.

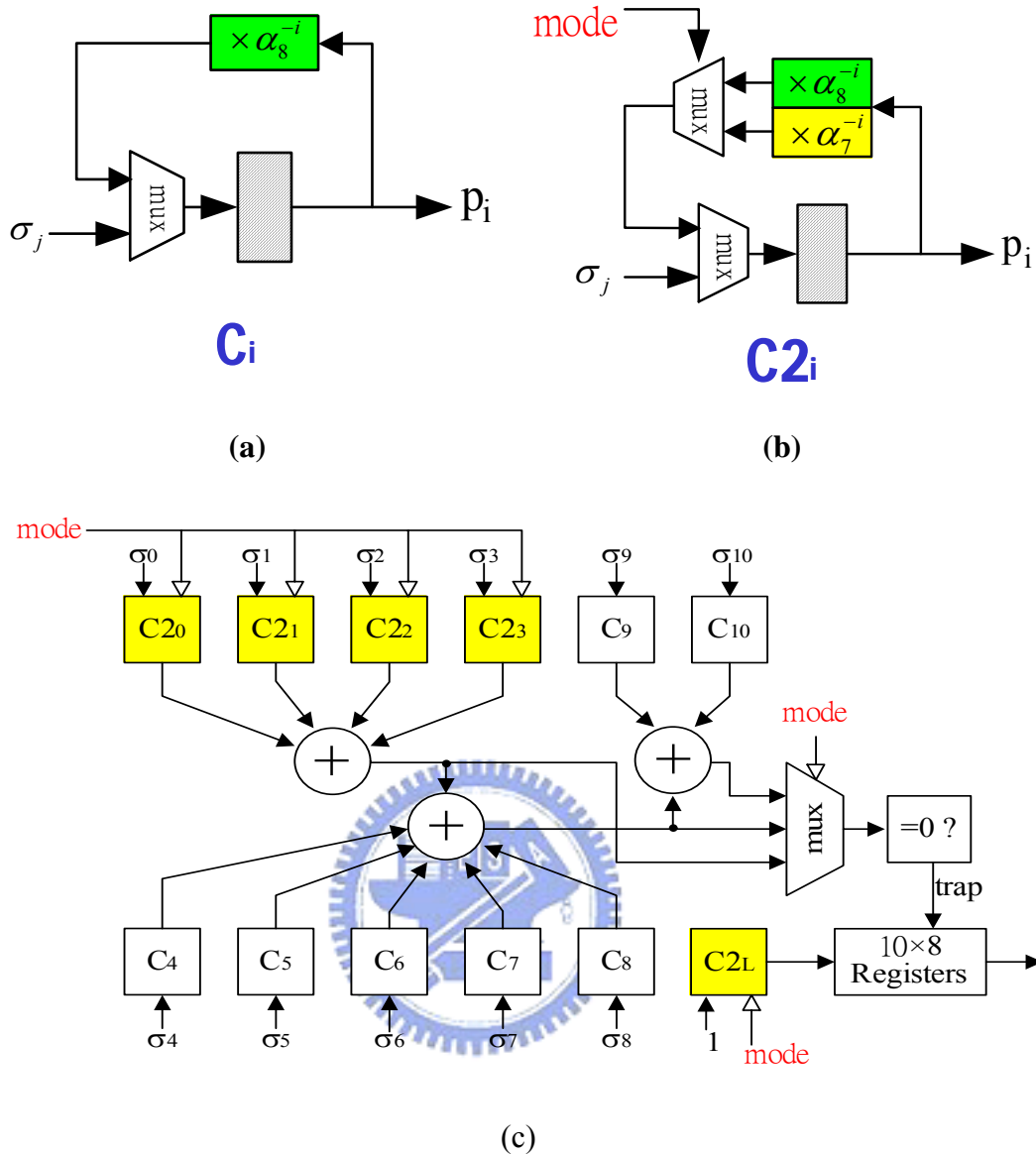


Figure 3.9: Multi-mode chien search. (a) Basic cell  $C_i$  for  $GF(2^8)$ . (b) Basic cell  $C_{2i}$  for dual mode purpose ( $GF(2^8)$  and  $GF(2^7)$ ). (c) The overall structure of multi-mode chien search.

### 3.3.5 Error Value Evaluator

Based on Forney algorithm and assume  $\beta_j$  is the  $j$ -th root of error locator polynomial. For J.83A, C and D, the error value:

$$e_i = \frac{\Omega(\beta_j)}{\beta_j \sigma'(\beta_j)} \quad (3.2)$$

For J.83B, the error value:

$$e_i = \frac{\Omega(\beta_j)}{\sigma'(\beta_j)} \quad (3.3)$$

Based on the previous equations, the architecture of multi-mode error value evaluator is proposed as shown in figure 3.10. It will calculate  $\sigma'(\beta_j)$  and  $\Omega(\beta_j)$  at the same time while the left multiplexer will choose  $\beta_j^2$ , the bottom multiplexer will choose  $\beta_j$ . After calculating  $\sigma'(\beta_j)$ ,  $\sigma'(\beta_j)$  will multiply  $\beta_j$  for J.83A,C and D. The block of “ $( )^{-1}$ ” is implemented by a table. In order to calculate the final error value, the bottom multiplexer will choose the upper path.

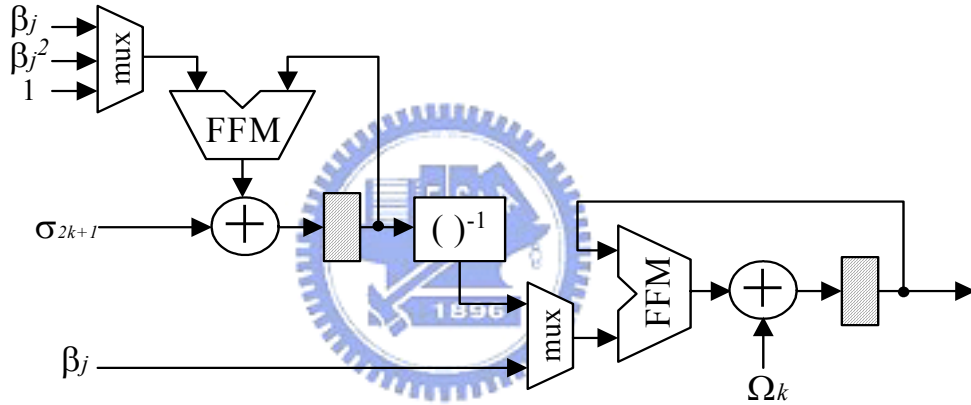


Figure 3.10: Multi-mode error value evaluator

### 3.3.6 Memory structure to correct the RS codeword

Based on the proposed architecture, the memory requirement is four times the codeword length because of the output latency. And, because of the output latency, memory structure is built as two interleaved structure to avoid accessing the same bank of memory in writing the current RS codeword and correcting the previous RS codeword at the same time, as shown in figure 3.11. The interleaved structure of memory is that packet 0 of RS codeword is written into bank 0 of memory, packet 1 is written into bank 1, packet 2 is written into bank 0, and

packet 3 is written into bank 1. Due to the output latency, we will know the error location and error value of RS codeword 0 until writing packet 3 into the memory. When correcting packet 1 in bank 1, the packet 4 is written into bank 0, and so on. Hence, it avoids accessing the same memory bank at the same time. Based on this interleaved structure of memory, the memory requirement for multi-mode RS decoder is 752 bytes (two 188x2 bytes) since the maximum K is 188.

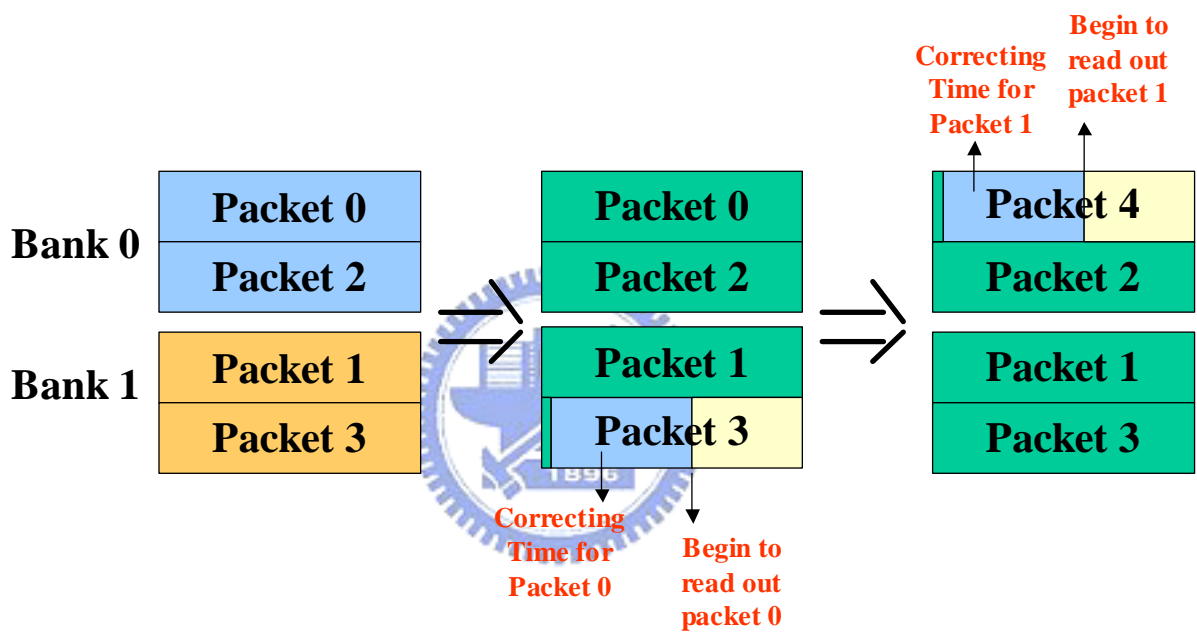


Figure 3.11: The operation of accessing memory in multi-mode RS decoder

### 3.4 Other Components

#### 3.4.1 De-scrambler

The circuit complexity of scrambler is so simple that it is suggested to use dedicated hardware for different annexes. Because of the property of “serial in serial out” in J.83A and C, we can transform the structure of scrambler into the one as shown in figure 3.12. The

transformed structure has the property of “symbol in symbol out”. Hence, the serial to parallel converter or parallel to serial converter can be omitted. The other annexes can be implemented as the original structure.

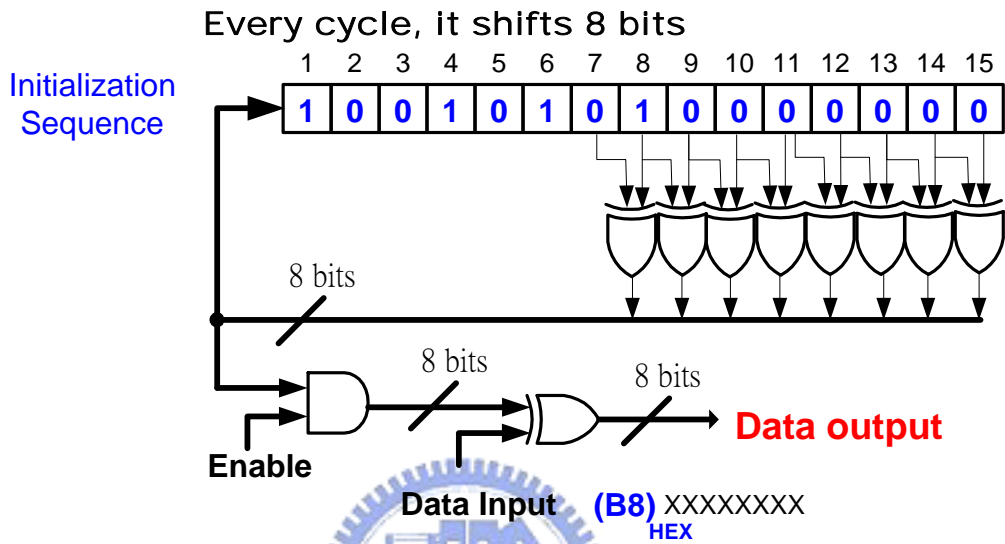


Figure 3.12: The transformed structure of scrambler in J.83A and C

### 3.4.2 Viterbi Decoder

Trellis coding is only included in J.83B. Using hard-decision Viterbi decoder with 16 states can fit the requirement of ITU-T J.83B.

We take register-exchange method as the architecture of survivor path storage management to realize the Trellis decoder since the convolutional codes in J.83B has only 16 states and thus the number of registers required for this decoder is not quite large. According to this approach, we assign one register to each state. Each register records the decoded output sequence along the survivor path for each state, as shown in figure 3.13 [16]. The decoded output sequence stored in survival memory (SM) depends on the path of minimum sum of the

coming TM and the previous PM. At the last stage, we select the sequence content stored in the register of the state with minimum PM.

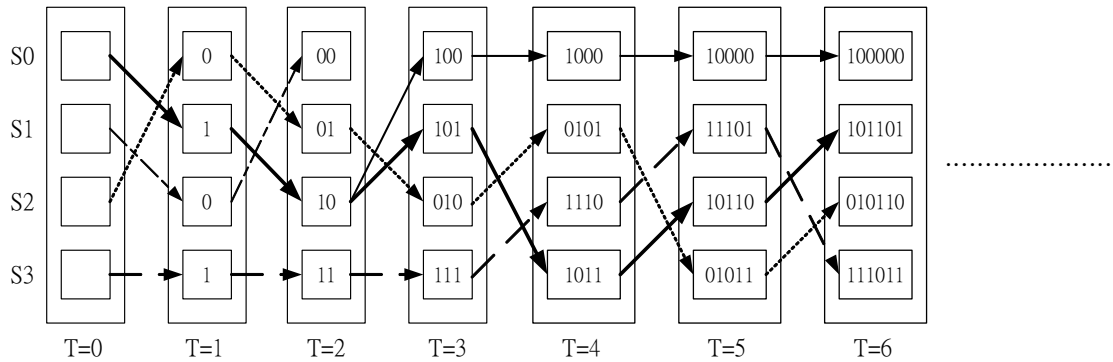


Figure 3.13: Register contents for register-exchange method

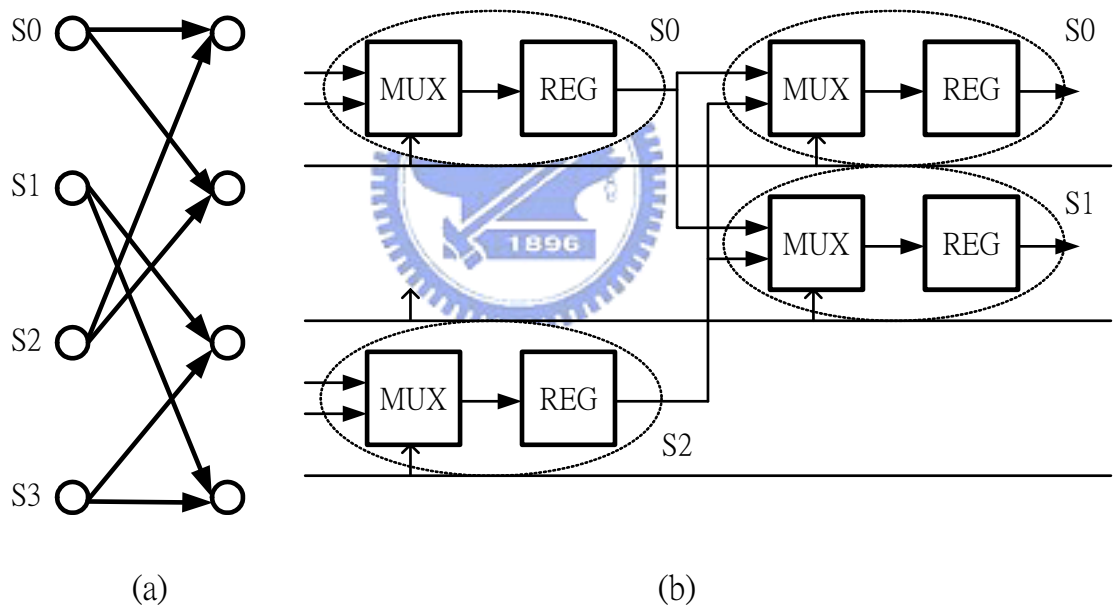


Figure 3.14: Architecture of register-exchange approach applied in SM unit. (a) Trellis diagram. (b) The connections of registers and multiplexers between each state.

The implementation of register-exchange method is really simple. The connections of registers and multiplexers between each state are decided by Trellis diagram. Using the property of the structure of trellis diagram as shown in figure 3.14(a), it is shown that there are always two states having the same previous two states. And, the current SM will select the

minimum path of sum of the coming TM and the previous PM as the new decoded output sequence. Thus, decoding sequence of  $S_0$  and  $S_1$  must come from  $S_0$  and  $S_2$ , and the connections can be represented by figure 3.14(b) [17].

The other issue in Viterbi decoder is metric rescaling. In Viterbi algorithm, the path metric is unboundedly increasing as time goes by. To implement a trellis decoder, we have to limit the path metric within a finite numerical range so that it can be expressed with finite bits. There are several approaches to do rescaling, such as “Reset”, “Rescaling Subtraction”, “Shift”, and “Modulo Normalization”. Among these approaches, “Modulo Normalization”, which is also called “Two’s Complement Arithmetic Approach” [18], is much more efficient than the other approaches and can be implemented by two’s complement arithmetic.

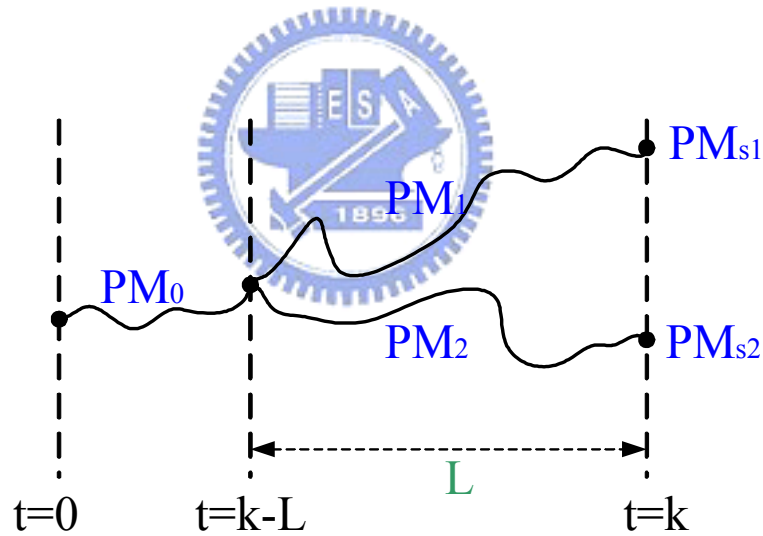


Figure 3.15: The upper bound of PM difference

Before describing how two’s complement arithmetic approach works, we should know the upper bound of PM difference at first. Assume all survivor paths selected at time unit  $k$  come from the same state at time unit “ $k-L$ ” as shown in figure 3.15. Then, the difference between any two PM must less than  $B \times L$ , where  $B$  and  $L$  are maximum value of TM and truncation length respectively.



The key idea of the “Modulo Normalization” approach is not to avoid overflow, but to accommodate overflow. Even the overflow occurs; the PM differences are also preserved. This concept can be represented by figure 3.16. Suppose both  $M_1$  and  $M_2$  are positive real number and  $|M_1 - M_2| < 2^{c-1}$ , where  $c$  is the bit number of PM value, then  $m_1 = M_1 \bmod 2^c$  and  $m_2 = M_2 \bmod 2^c$ . Thus,  $m_1$  and  $m_2$  can be presented on half cycle without confusing their difference relationship. If  $m_1 - m_2 \geq 0$ , then  $M_1 > M_2$  and we can select the suitable survivor path. Note that both the add operation “ $PM_{\text{new}} = (PM+TM) \bmod 2^c$ ” and subtract operation “ $m_1 - m_2$ ” can be realized with 2’s complement components. In principle, we achieve metric rescaling at the cost of one-bit penalty. However, such a method can avoid redundant rescaling operations or performance degradation due to metric overflow.

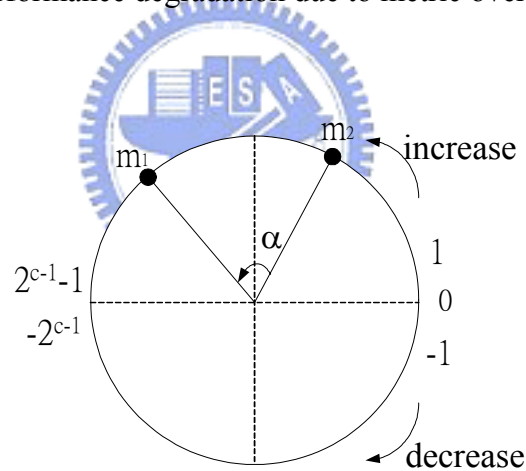


Figure 3.16: Illustration of Modulo Normalization

After summarizing the architecture of register-exchange and the “Modulo Normalization” approaches, we can implement the trellis decoder by combining the following components:

- (1) TM: Compute all branch metrics from the received symbols.
- (2) ACS: Perform the “add-compare-select” operation for each state to update their path metrics, respectively. The block diagram is shown in figure 3.17.

- (3) Metric Rescaling Unit: Confine all PM values to a finite range without losing their difference relationship.
- (4) SM: Record all decision result according to the choice of ACS unit and trace back the survivor path to find the oldest data as decoded bits. The block diagram is shown in figure 3.18.

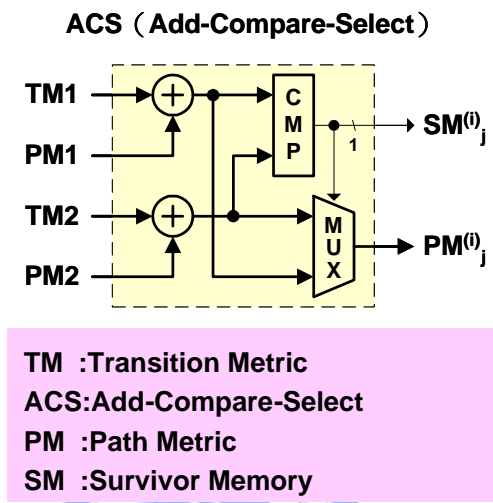


Figure 3.17: The ACS module used for Viterbi decoder

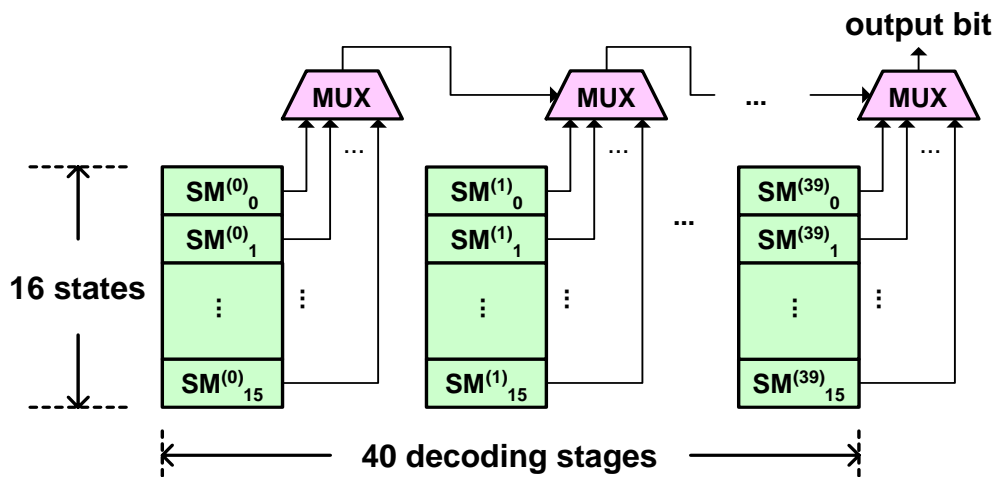


Figure 3.18: Survivor memory and trace back unit

The overall implementation architecture of Viterbi decoding algorithm is shown in figure

3.19. Note that although the hard decision-making method is applied here, the soft demodulator decisions can result in a performance advantage over hard decision decoding, so the TM units and Metric Rescaling units could be re-designed according to what kind of demodulator used in this system. In parallel Viterbi decoder there are  $2^m$  ACS units and PM units for each state. The truncation length is an important issue to obtain a small probability of error. Typically, the required truncation length is approximately  $10xm$  for  $4/5$  coding rate. In our case, 40 stages are required. The work of this part is referred to our lab's IP which is implemented by Chia-Cho Wu.

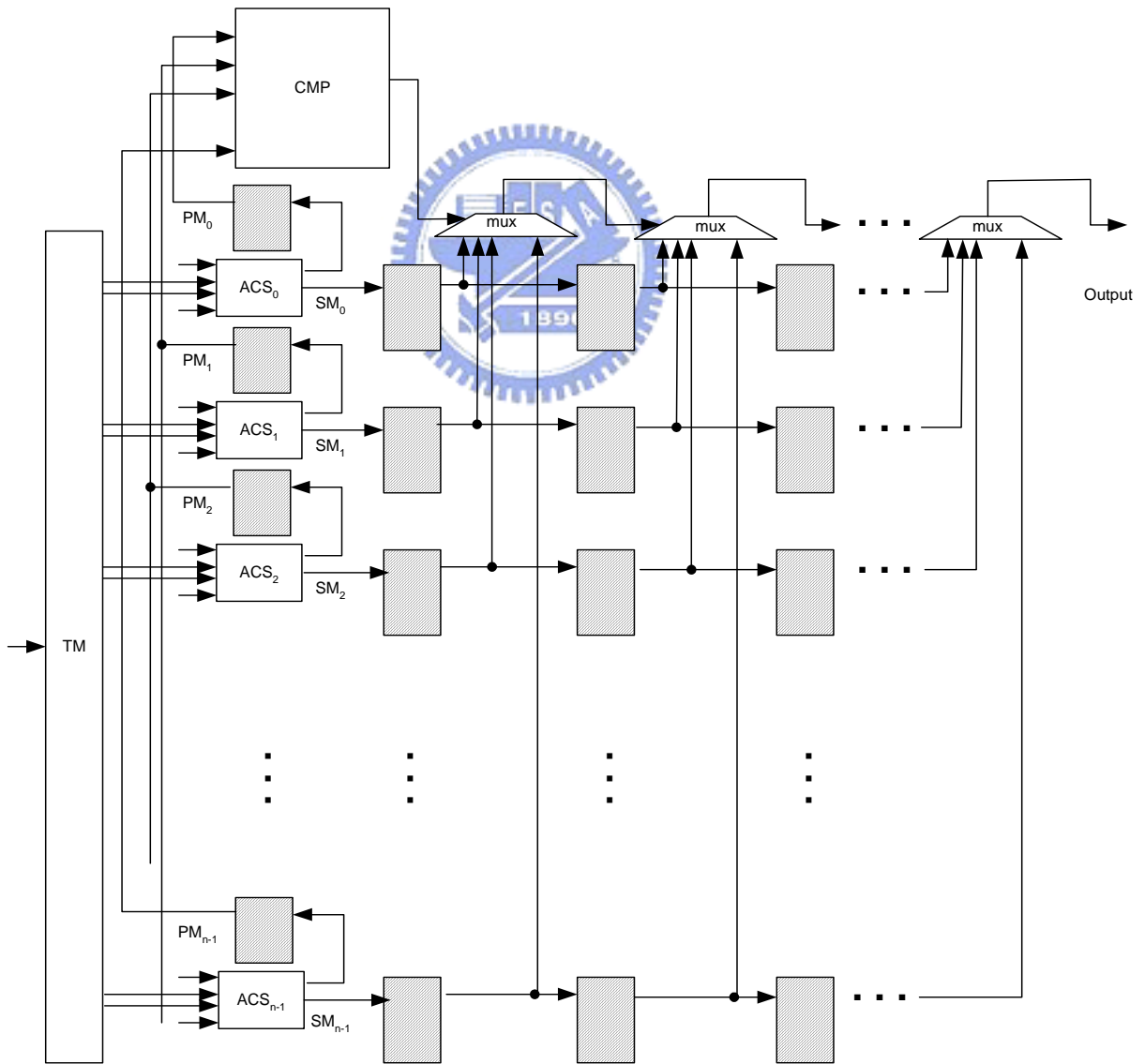


Figure 3.19: Architecture of trellis decoder.

### 3.5 The memory consideration for test chip

As mentioned above, the memory requirement of universal convolutional interleaving for ITU-T J.83B is about 64K bytes. And, the memory requirement for multi-mode RS decoder is 752 bytes. Although 64K bytes memory is not so large, we still cannot embed the 64K bytes SRAM in the test chip due to the constraint of test chip area for academic research purpose. Hence, the 64K bytes memory will be taken as the external memory and only 752 bytes SRAM for RS decoder are embedded in the test chip. So, the system platform is modified as figure 3.20.

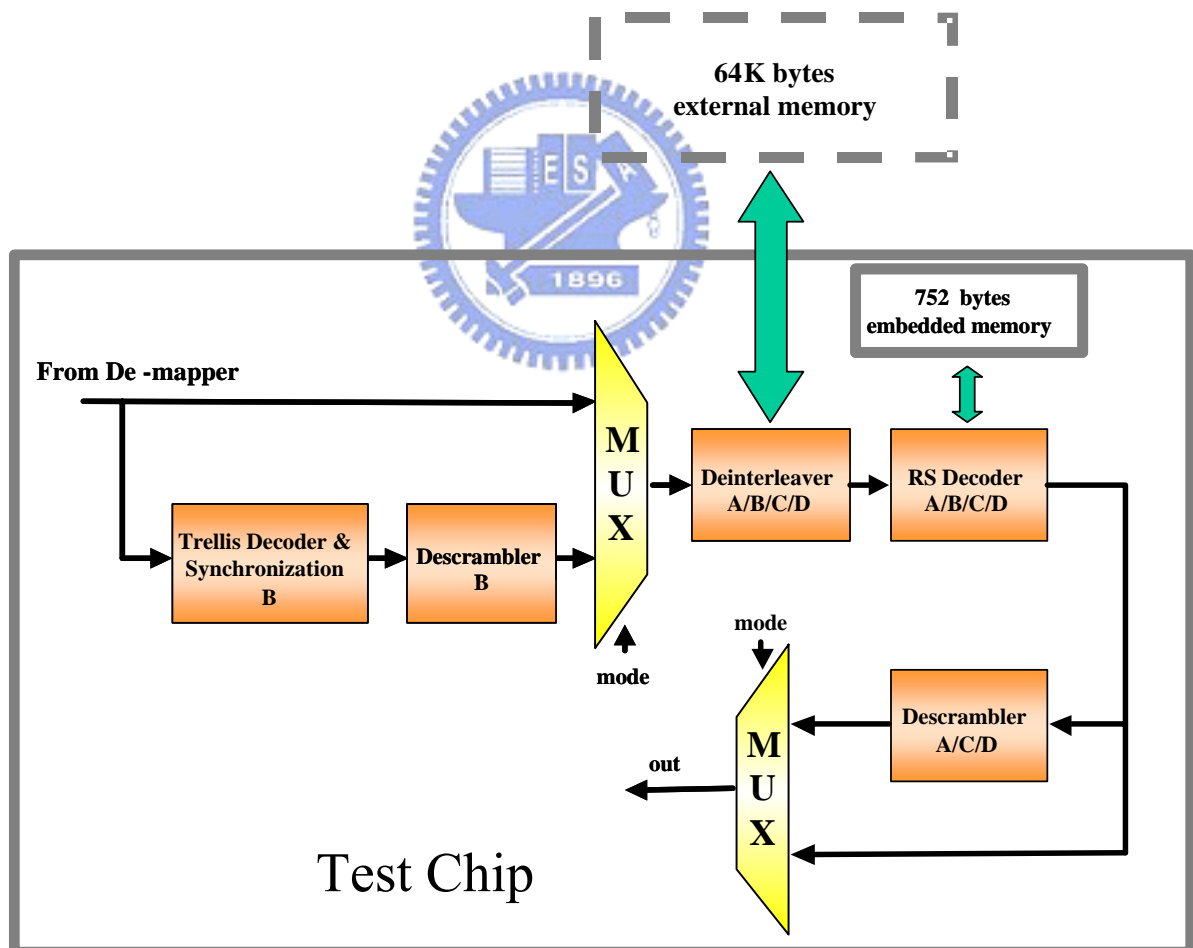


Figure 3.20: The system platform with memory consideration

### 3.6 *Summary*

In this chapter, a multi-mode RS decoder with memories to store and correct received data and a memory-based universal convolutional interleaver and deinterleaver are proposed. Both of them have the advantage of low overhead, high flexibility to achieve multi-mode design and can be compatible with the standard of J.83, DVB-T and ATSC Digital TV, etc. The proposed multi-mode RS decoder can support the error correction capability with  $t = 3, 8$  and  $10$  over  $GF(2^7)$  and  $GF(2^8)$  respectively. BM algorithm is adopted for key equation due to its regularity instead of Euclidean algorithm. And, the proposed multi-mode RS decoder can be easily modified to meet the different requirement of different applications. In addition, the proposed universal convolutional interleaver and deinterleaver can support all kinds of parameters of convolutional interleaving. The parameter design in convolutional interleaving has the advantage for time-to-market. We also mention the implementing method for single mode scrambler and Viterbi decoder. Viterbi decoder takes register-exchange method as the architecture of survivor path storage management since the convolutional codes in J.83B has only 16 states and thus the number of registers required for this decoder is not quite large. For the memory consideration, due to the constraint of test chip area for academic research purpose, the 64K bytes memory for universal convolutional interleaving are taken as the external memory.

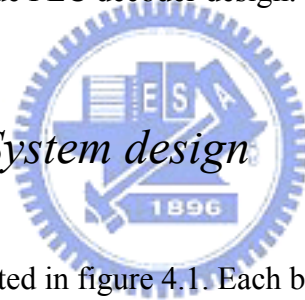
# Chapter 4

## Simulation and Implementation Result

---

The environment of simulation platform and the result of chip implementation will be shown in this chapter. And, the result of our proposed architecture and chip implementation will do some comparisons with other reference works. By the comparisons, it shows that our proposed architecture and chip has the advantages of low overhead, low power, and high flexibility to achieve multi-mode FEC decoder design.

### 4.1 Platform and System design



The design flow is illustrated in figure 4.1. Each block is defined as follows:

#### (1) System platform simulation:

At first, the system platform based on high-level language will do the simulation to verify the proposed algorithm. High-level simulation is very important to guarantee the functionality of the whole system before the hardware design. And, Matlab is chosen as the simulation environment since it has the advantages of simple usage and powerful functionality. In addition to the functional block of multi-mode FEC decoder for J.83 as shown in figure 3.1, the functional block of multi-mode FEC encoder for J.83 is included in the Matlab platform as shown in figure 4.2. The relationship between the multi-mode FEC encoder and decoder is depicted in figure 4.3. After encoding the test pattern, the noises will be added to the encoded data, where the noises should be within the capability of error correction. Furthermore, the

noisy encoded pattern should be recorded to a file for RTL simulation later. Then, the noisy encoded data are going to be fed to the multi-mode FEC decoder. The output data from the FEC decoder will compare to the original uncoded pattern. After verification of the proposed algorithm and, we can design the architecture and write the RTL code to do the RTL behavior simulation.

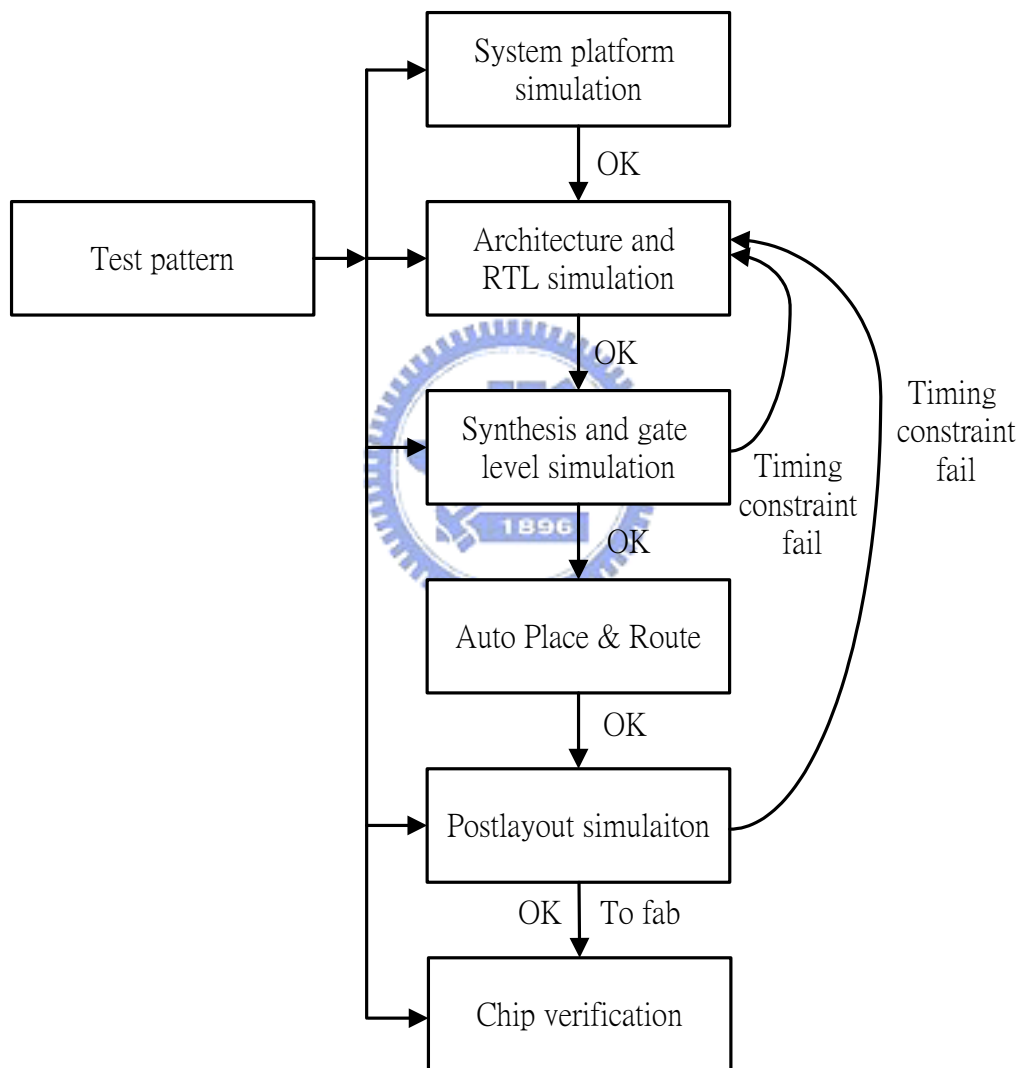


Figure 4.1: The design flow

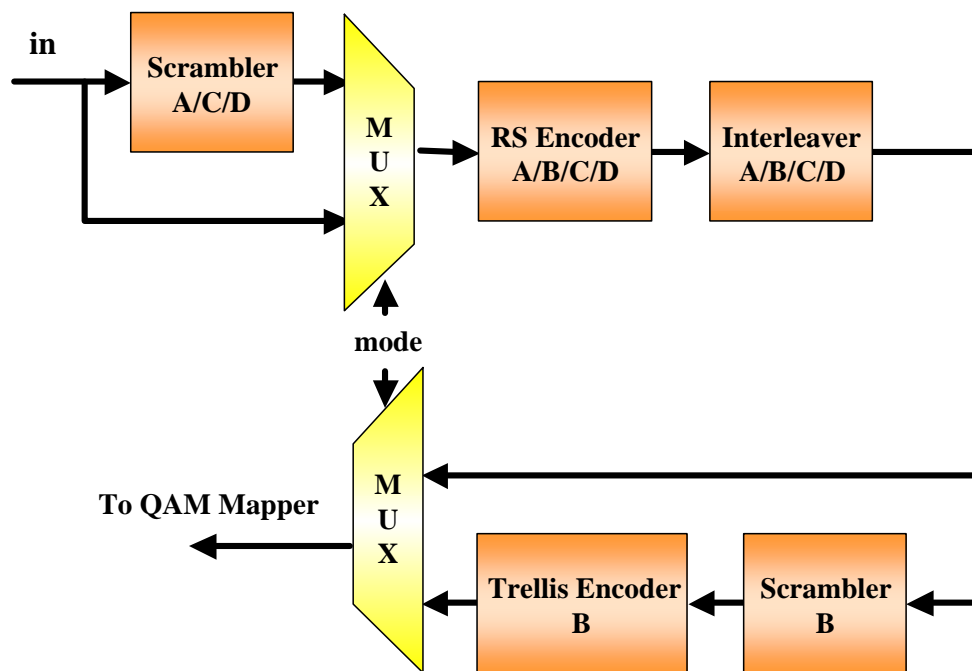


Figure 4.2: FEC encoder in J.83

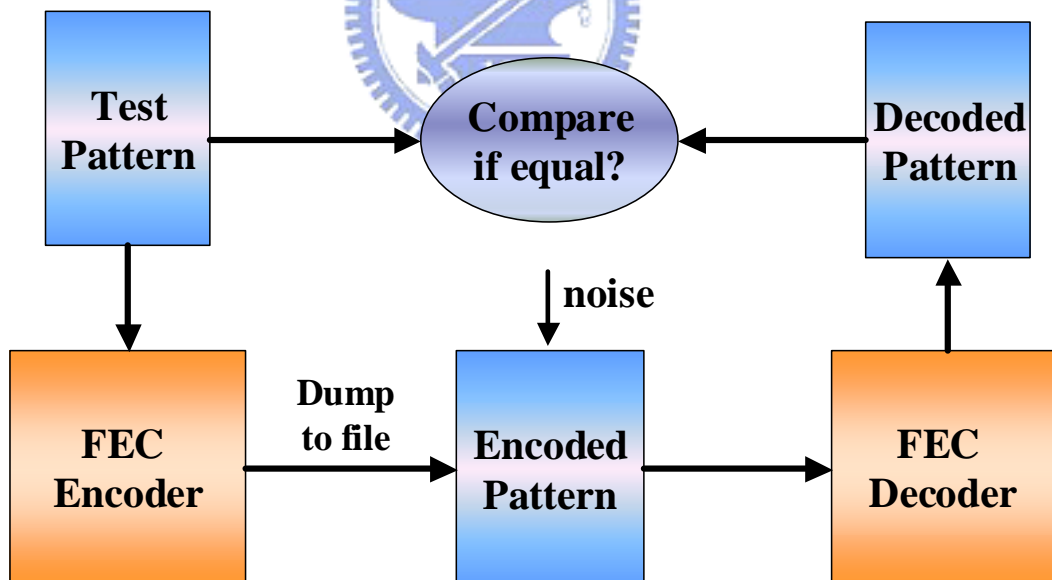


Figure 4.3: Simulation environment

(2) Architecture and RTL simulation:

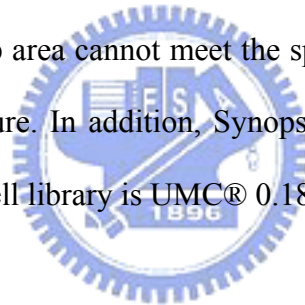
The RTL code describes the system in hardware level. The architecture and circuit



should be defined according to the timing constraint at first before writing the RTL code. The architecture is mentioned in chapter 4. Here, Verilog is chosen as the hardware description language (HDL). The test bench stored from the Matlab platform should be used to check if the functionality of RTL coding is correct or not. We should note that the RTL simulation takes logic circuit as the ideal behavior. Hence, the gate level simulation is required after synthesis.

(3) Synthesis and gate level simulation:

After checking the RTL behavior simulation, we can do the synthesis and do the gate level simulation. By the aid of synthesis and gate level simulation, we can know the almost real logic gate delay and area of chip. Thus, the chip performance and area complexity can be estimated. If the timing or chip area cannot meet the specification requirement, we should go back to redesign the architecture. In addition, Synopsys® Design Analyzer is our synthesis CAD tool. And, the standard cell library is UMC® 0.18μm 1P6M CMOS technology.



(5) Auto place and route (APR):

After succeeding the gate level simulation, we should place and route the logic gate to layout. Because there are more problems in deep-submicron process, such as signal integrity, IR drop, wire delay, and so on, we should use the new CAD tool “Cadence® SOC Encounter” to handle the deep-submicron problem. After APR, we should use “Calibre® DRC/LVS” CAD tools to verify DRC (design rule check) and LVS (layout versus schematic) errors. Then, the postlayout-gate level simulation can be taken to simulate the prototype of the chip. If timing cannot meet the requirement of specification, we should go back to (2) to redesign the architecture.

(6) Postlayout simulation:

Use “Calibre® LPE (Layout parameter extraction)” CAD tool to extract the parameters from layout, such as transistors, capacitances, resistors, and so on. After extraction, we can use “Nanosim” CAD tool to do postlayout simulation. The target of Nanosim is between SPICE and Verilog. It is a transistor-level timing simulator and power dissipation analysis tool for digital circuit design. Thus, it handles current, voltages simulations and timing checks. After verification of postlayout simulation, we can tape out the chip to fab.

(7) Chip verification:

Using IMS100 to verify the chip in CIC®. The test pattern is generated from the system platform and gate level or postlayout simulation. In addition to verification, the power consumption of the chip will be measured at the same time.

## 4.2 *Chip integration and the results of chip implementation*



As mentioned in chapter 3, the memory requirement for universal convolutional deinterleaver is 65032 bytes. The chip area is limited due to academic research purpose. Hence, using it as the external-memory is a good solution. So, the simulation environment in gate level simulation and postlayout simulation will become the one as shown in figure 4.4. The 65032 bytes memory is used as the behavior model and does the simulation with the chip. Only 752 bytes memory for RS decoder are embedded in the test chip. As a result, for the chip verification, we will feed the data from the simulated external memory to chip instead of real external memory for convenience.

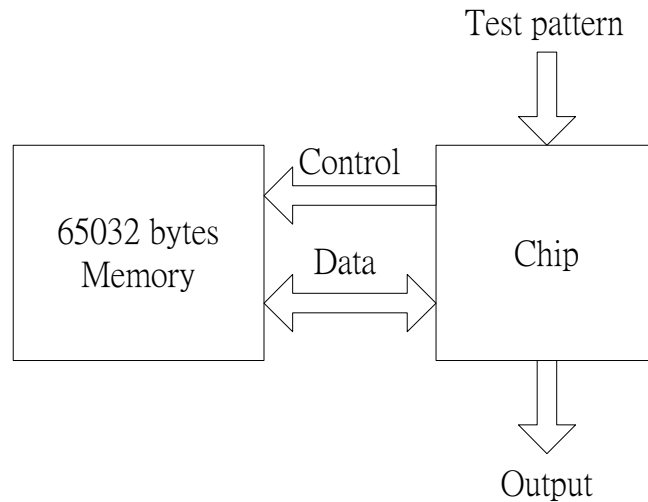


Figure 4.4: The chip connected with external memory

Table 2: Summary of CHIP Implementation for J.83 FEC

Technology	UMC® 0.18 $\mu\text{m}$ 1P6M CMOS process	
Chip size	1.89mm x 1.89mm	
Core size	1.28mm x 1.28mm	
Gate count	54.5K	
Embedded SRAM	752Bytes	
Supply voltage	1.8V	
Max operating frequency	83MHz (600Mbps)	
Average Power	J.83 Annex A&C	25.2mW @83MHz 3.6mW @7MHz
	J.83 Annex B in 64QAM	43.2mW @83MHz 5.4mW @7MHz
	J.83 Annex B in 256QAM	45mW @83MHz 5.4mW @7MHz
	J.83 Annex D	30.6mW @83MHz 4.5mW @7MHz

Table 2 shows the result and the measurement of the chip implementation. By implementing with UMC® 0.18 $\mu\text{m}$  1P6M CMOS technology, the chip shows that the proposed multi-mode FEC decoder can work at 83MHz (600Mbps) while costs 54.5K logic gate counts, two 376x8 bits embedded dual-port SRAM and 65032 bytes external memory for de-interleaver with only 8 bytes overhead. In fact, 7 MHz has met the requirement of specification. And, the chip size is 1892 x 1892  $\mu\text{m}^2$ . The floor plan of chip is shown in figure 4.5. The maximum power consumption is 45mW at 83MHz (5.4mW at 7MHz) with the supply voltage 1.8 volts for J.83B in 256QAM. For more detail about power consumption, please see table 2. It shows that our chip has the advantage of low power requirement.

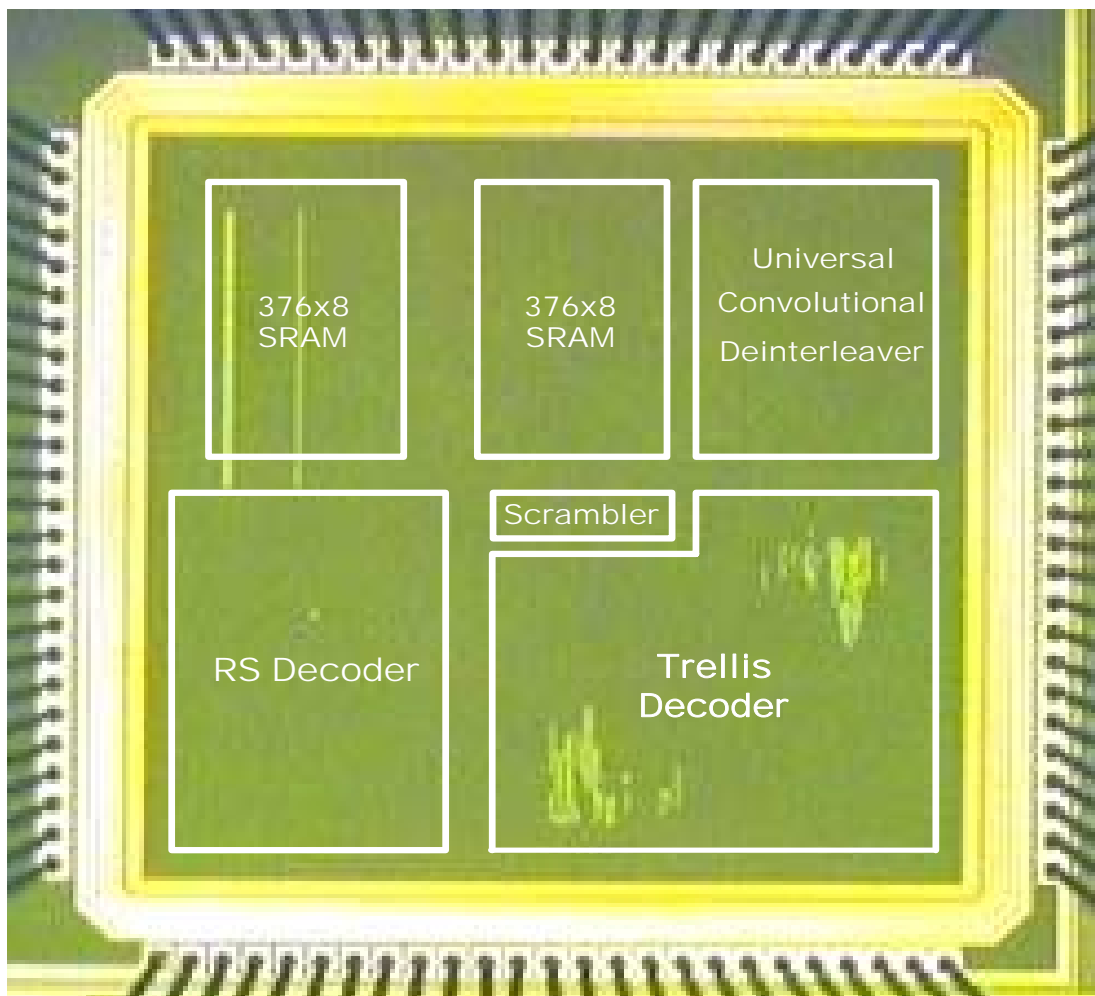


Figure 4.5: The floor plan of the chip

The detailed gate counts of each module are listed in table 3, where trellis decoder contains two Viterbi decoders and the circuit of synchronization for FEC frame[1]. Table 3 also shows the logic gate counts of RS Decoder in ITU-T J.83D which is the most complex RS code in ITU-T J.83. It shows that the proposed multi-mode RS decoder is only larger about 1.1K gate counts than that specified in J.83D. In other words, the proposed multi-mode RS decoder has only the overhead of 6% compare to the most critical mode.

Table 3: Gate Count for each module

Module	Logic gate count
Multi-Modes RS Decoder	19051
Universal deinterleaver	8306
Viterbi Decoder	9883
Trellis decoder (contains 2 Viterbi Decoder)	24632
Scrambler	1190
Overall FEC Decoder	54542
J.83D RS Decoder	17963

Compare the proposed architecture for multi-mode RS decoder with other reference works as shown in table 4, although [10], [21] and [29] support only one mode, their gate counts or throughput rate are not better than the proposed work. Besides, compare the proposed memory-based universal convolutional deinterleaver with other people's works, (12, 17) convolutional deinterleaver in [10] requires memory size of 1280 bytes with two 128-byte RAM and four 256-byte RAM, that is, overhead is 158 bytes. In [21], (15, 17) convolutional deinterleaver needs 1829 bytes with 44 bytes overhead. For the proposed algorithm and architecture in the same convolutional deinterleaver, we only have the overhead of 17 bytes memory and a low complexity controller. Furthermore, in [21], and [10], they can only meet

for suitable standard using the same component, but the proposed multi-mode FEC decoder can be used in many standards, such as ITU-T J.83, DVB-T, ATSC Digital TV, etc. Hence, the proposed architecture has the advantage of low-overhead, high throughput rate and high flexibility to achieve multi-mode design.

Table 4: Comparisons between the proposed architecture and other reference works

		Proposed	[21]	[10]	[29]
Technology		0.18 $\mu$ m	0.6 $\mu$ m	FPGA	0.25 $\mu$ m
RS decoder	Mode	Multi-mode	Single-mode	Single-mode	Single-mode
	m	7, 8	8	8	8
	t	3, 8, 10	16	8	8
	Gate counts	19K			55K
Convolutional deinterleaving	Mode	Universal	Single-mode (15, 17)	Single-mode (12, 17)	
	Memory overhead	$J, 1 \leq J \leq 17$	44 bytes	158 bytes	
Throughput		600Mbps	73Mbps		600Mbps

### 4.3 Summary

The chip implementation of the proposed multi-mode FEC decoder is introduced in this chapter. With 0.18 $\mu$ m 1P6M CMOS technology, the implemented chip shows that the FEC decoder can work at 83MHz (600Mbps) while costs 54.5K gate counts and two 376x8 bits embedded dual-port SRAM. The chip size is 1.89mm x 1.89mm. And the average power consumption in full spec. mode is about 45mW at 83MHz. While running at 7MHz that meets

symbol rate of cable modem, the power dissipation is 5.4mW. Compare to other people's work, the proposed architecture shows that it has the advantage of low-overhead, high throughput rate requirement and high flexibility to achieve multi-mode design.



# Chapter 5

## Conclusion and Future Work

---

### 5.1 Conclusion

In this thesis, a solution to design a multi-mode FEC decoder is proposed. It mainly contains a multi-mode RS decoder for different finite field and different capability of error correction with memories to store and correct received data and a memory-based universal convolutional interleaver/deinterleaver. Both of them have the advantage of low-overhead, and high flexibility to achieve multi-mode FEC design. And, this multi-mode FEC decoder can be adopted in J.83 cable modem system, DVB-T system, and so on.

To design the multi-mode FEC decoder systematically, we began from the system view and built a high-level simulation platform by Matlab to verify the proposed algorithm and architecture at first. J.83 cable system is chosen as the simulated platform since it is the most complex system among those communication systems with the similar modules, such as DVB-T, J.83, ATSC Digital TV, and so on. Then, we construct the hardware architecture in RTL-level by Verilog. By implementing with UMC® 0.18 $\mu$ m 1P6M CMOS technology, the chip shows that the proposed multi-mode FEC decoder can work at 83MHz (600Mbps) while costs 54.5K logic gate counts, two 376x8 bits embedded dual-port SRAM and 65032 bytes external memory for de-interleaver with only 8 bytes overhead. And, chip size is 1.89mm x 1.89mm. Compare to other related works, our proposed architecture has the advantage of high throughput rate, low-overhead and high flexibility to achieve multi-mode design to reduce the design cost.



## 5.2 Future Work

As mentioned in chapter one, channel coding is a key module to minimize the effect of channel noise during data transmission, especially in wireless communications. For wireless communications in the future, designing an error control code to achieve Shannon bounds is more and more important. Those concatenated codes, such as FEC in J.83, will not meet the requirement of future wireless communications.

Thus, the iterative decoding algorithm[22] is used to achieve Shannon limits more close. Both turbo codes[26] and LDPC (Low Density Parity Check) codes[23] adopt this idea of iterative decoding. Turbo codes were proposed in 1993. Turbo encoder usually comprises the parallel concatenation of two RSC (recursive systematic convolutional codes) and one interleaver to encode the information as shown in figure 5.1, where  $\pi$  means interleaver and  $x_0$ ,  $x_1$  and  $x_2$  are encoded datum. As the block length of interleaving increases, the performance of turbo codes is more close to Shannon bounds. Turbo codes has been adopted in third generation mobile systems, such as 3gpp and 3gpp2 systems since CDMA system needs a powerful error correction codes to increase the channel capacity.

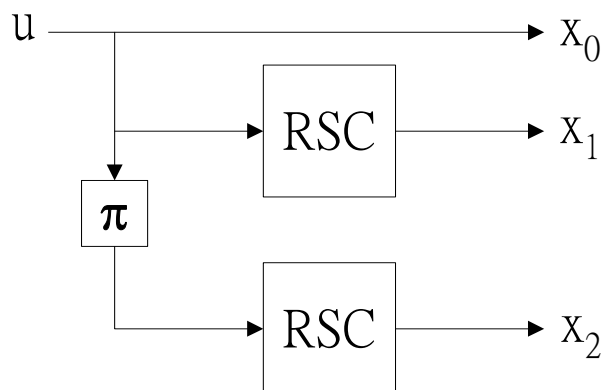


Figure 5.1: Turbo encoder

On the other hand, LDPC codes were created in 1962 by Gallager, but were rediscovered in 1995, 1996[24][25]. LDPC codes are one kind of block codes but the parity check matrix is sparse compared to the traditional block codes. Same as the turbo codes, as the block length increases, the iterative decoding algorithm can achieve more near Shannon limits. And, the advantage of LDPC codes over turbo codes contains:

- (1) They do not require a long interleaver.
- (2) They have better block error performance.
- (3) Their error floor occurs at a much lower BER (Bit Error Rate).
- (4) Their decoding is not trellis based, so they are suitable for high throughput rate requirement due to its parallel characteristic.

Due to the above advantage, the next generation wireless communications are considering using LDPC codes as their error control codes instead of turbo codes, such as UWB (ultra wide band) system and DVB-S2 for high reliability and high throughput rate requirement. The decoding algorithm and the performance of LDPC codes can be seen in Appendix-A.

# Bibliography

---

- [1] ITU-T, Telecommunication Standardization Sector of ITU, "Digital multi-programme systems for television sound and data services for cable distribution"-Digital transmission of television signals, ITU-T Recommendation J.83, Apr. 1997.
- [2] ETSI, "Digital Video Broadcasting (DVB); Framing structure, channel coding and modulation for digital terrestrial television"-EN 300 744 V1.1.2, Nov. 1998.
- [3] ATSC Digital Television Standard, Sep. 1995.
- [4] H. C. Chang, C. B. Shung, and C. Y. Lee, "A Reed-Solomon Product-Code (RS-PC) Decoder Chip for DVD Applications," *IEEE J. Solid-State Circuits*, Vol. 36, No. 2, pp. 229-238, Feb. 2001.
- [5] J. L. Ramsey, "Realization of Optimum Interleavers," *IEEE Trans. on Inform. Theory*, vol. IT-16, no. 3, May 1970.
- [6] Y. X. You, J. X. Wang, and X. R. Piao, "Design and Implementation of Concatenated Encoder," in *Int. Conf. ASIC*, Oct. 2001.
- [7] H. Yang, Y. Zhong, and L. Yang, "An FPGA Prototype of A Forward Error Correction (FEC) Decoder For ATSC Digital TV," *IEEE Trans. on Consumer Electron*, vol. 45, no. 2, pp. 387-395, May 1999.
- [8] G. D. Forney, Jr., "Burst-Correcting Codes for the Classic Bursty Channel", *IEEE Trans. on Communications*, vol. 19, no. 5, pp. 772-781, Oct. 1971.
- [9] H. C. Chang, C. C. Lin, and C. Y. Lee, "A Low-Power Reed-Solomon Decoder For STM-16 Optical Communications," in *IEEE Asia-Pacific Conf. ASIC*, Aug. 2002.
- [10] J. B. Kim, Y. J. Lim, and M. H. Lee, "A Low Complexity FEC Design for DAB," in *ISCAS*, May 2001.

- [11] R. J. McEliece, *The Theory of Information and Coding, 2nd ed.* Cambridge, UK: Cambridge University Press, 2002.
- [12] S. Lin and D. J. Costello, Jr., *Error Control Coding, Fundamentals and Applications.* Englewood Cliffs, NJ: Prentice-Hall, 1983.
- [13] J. B. Cain, G. C. Clark, and J. M. Geist, "Punctured convolutional codes of rate  $(n-1)/n$  and simplified maximum likelihood decoding," *IEEE Trans. on Inform. Theory*, vol. IT-25, No. 1, pp. 97-101, Jan. 1979.
- [14] A. J. Viterbi, "Error bounds for convolutional codes and an asymptotically optimum decoding algorithm," *IEEE Trans. on Inform. Theory*, vol. IT-13, pp. 260-269, April. 1967.
- [15] G. D. Forney, Jr., "Convolutional Code II: Maximum likelihood decoding," *Information and Control*, 25, pp. 222-266, July 1974.
- [16] Dalia A. F. El-Dib and Mohamed I. Elmasry, "Low-Power Register-Exchange Viterbi Decoder For High-Speed Wireless Communications," *IEEE International Symposium on Circuits and Systems*, vol. 5, pp. 737-740, 2002
- [17] S. R. Meier, M. Steinert, S. Buch, "Testability of Path History Memories with Register-Exchange Architecture Used in Viterbi-Decoders," *IEEE International Symposium on Circuits and Systems*, vol. 3, pp. 165-168, 2002
- [18] Andries P. Hekstra, "An Alternative to Metric Rescaling in Viterbi Decoders," *IEEE Trans. on Communications*, vol. 37, NO. 11, pp 1220-1222, Nov. 1989.
- [19] Richard E. Blahut, *Theory and Practice of Error Control Codes*, Addison-Wesley Publishing Company, 1983
- [20] Gennady Feygin, and P. G. Gulak, "Architectural Tradeoffs for Survivor Sequence Memory Management in Viterbi Decoders," *IEEE Trans. on Communications*, vol. 41, NO. 3, pp. 425-429, March 1993.
- [21] Daniel A. Luthi, Advait Mogre, Nadav Ben-Efraim, Alok Gupta, "A single-chip concatenated FEC decoder," *IEEE custom integrated circuits conference*, pp. 285-288, May 1995.

- [22] Joachim Hagenauer, Elke Offer, and Lutz Papke, "Iterative Decoding of Binary Block and Convolutional Codes," *IEEE Trans. Inform. Theory*, vol. 42, No. 2, pp. 429-445, March 1996.
- [23] R. G. Gallager, "Low Density Parity Check Codes," *IRE Trans. Inform. Theory*, vol. IT-8, pp. 21-28, Jan. 1962.
- [24] Niclas Wiberg, Hans-Andrea Loeliger, and Ralph Kotter, "Codes and Iterative Decoding on General Graphs," *IEEE International Symposium on Information Theory*, pp. 468, Sept. 1995.
- [25] D. J. C. MacKay and R. M. Neal, "Near Shannon limit performance of low density parity check codes," *IEE Electronics Letters*, vol. 32, Issue: 18, pp. 1645, Aug. 1996.
- [26] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near Shannon Limit Error-Correcting Coding and Decoding: turbo-codes," *IEEE Int. conf. Communications (ICC)*, pp. 1064-1070, May 1993.
- [27] D. J. C. MacKay, "Good error-correcting codes based on very sparse matrices," *IEEE Trans. Inform. Theory*, vol. 45, pp. 399-431, Mar. 1999.
- [28] Xiao-Yu Hu, Evangelos Eleftheriou, Dieter-Michael Arnold, and Ajay Dholakia, "Efficient Implementations of the Sum-Product Algorithm for Decoding LDPC Codes," *IEEE Global Telecommunications Conference*, vol. 2, 25-29, pp. 1036 - 1041, Nov. 2001.
- [29] H. Lee, M. L. Yu, and L. Song, "VLSI Design of Reed-Solomon Decoder Architecture," *IEEE ISCAS*, May 2000.

# Appendix-A

## Decoding algorithm of LDPC codes

The decoding algorithm of LDPC codes is called iterative Sum-Product Algorithm (SPA), or message passing (MP) algorithm, belief propagation (BP) algorithm. The behavior of MP algorithm for LDPC codes can be expressed a bipartite graph for parity check matrix H as shown in figure A.1. The message of variable node and function node pass to each other iteratively.

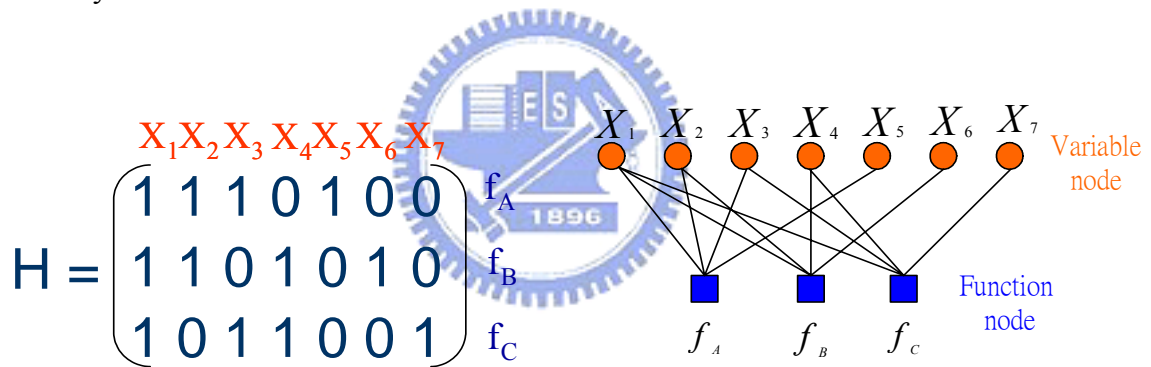


Figure A.1: The message passing on bipartite graph of LDPC codes

To explain SPA to decode LDPC codes, we have some notations defined for parity check matrix H at first. We denote the set of bits  $n$  that participate in check  $m$  by  $N(m) \equiv \{n : H_{mn} = 1\}$ . Similarly, we define the set of checks  $m$  in which bit  $n$  participates,  $M(n) \equiv \{m : H_{mn} = 1\}$ . We denote a set  $N(m)$  with bit  $n$  excluded by  $N(m) \setminus n$ . And, the algorithm has two parts, in which quantities  $q_{n \rightarrow m}$  and  $r_{m \rightarrow n}$  associated with each nonzero element in the H matrix are iteratively updated. The quantity  $q_{n \rightarrow m}^x$  is meant to be the probability that bit  $n$  has the value  $x$ , given the information obtained via checks other than check  $m$ . The quantity  $r_{m \rightarrow n}^x$  is meant

to be the probability of check  $m$  being satisfied if bit  $n$  is considered fixed at  $x$  and other bits have a separable distribution given by the probabilities  $\{q_{n' \rightarrow m} : n' \in N(m) \setminus n\}$ . SPA is summarized as follows.

**Initialization:**

$$p_n^0 = P(x_n = 0 | r_n), \quad p_n^1 = P(x_n = 1 | r_n) = 1 - p_n^0 \quad (\text{A.1})$$

$q_{n \rightarrow m}^0$  and  $q_{n \rightarrow m}^1$  are initialized to the values  $p_n^0$  and  $p_n^1$ , respectively, such that  $H_{mn}=1$

$r_{m \rightarrow n}^0$  and  $r_{m \rightarrow n}^1$  are initialized to zeros.

**Horizontal step (check-node update):**

We run through the checks  $m$  and compute for each  $n \in N(m)$  probability  $r_{m \rightarrow n}^x$ . It is the probability of the observed value of  $z_m$  arising when  $x_n = x$ , given that the other bits  $\{x_{n'} : n' \neq n\}$  have a separable distribution given by the probabilities  $q_{n' \rightarrow m}^x$ .

$$r_{m \rightarrow n}^0 = \sum_{\{x_{n'} : n' \in N(m) \setminus n\}} P(z_m | x_n = 0, \{x_{n'} : n' \in N(m) \setminus n\}) \times \prod_{n' \in N(m) \setminus n} q_{n' \rightarrow m}^{x_{n'}} \quad (\text{A.2})$$

$$r_{m \rightarrow n}^1 = \sum_{\{x_{n'} : n' \in N(m) \setminus n\}} P(z_m | x_n = 1, \{x_{n'} : n' \in N(m) \setminus n\}) \times \prod_{n' \in N(m) \setminus n} q_{n' \rightarrow m}^{x_{n'}} \quad (\text{A.3})$$

A particularly convenient implementation of (A.2) and (A.3) is described in [27].

**Vertical step (variable-node update):**

The vertical step takes the computed values of  $r_{m \rightarrow n}^x$  and updates the values of the probabilities  $q_{n \rightarrow m}^x$ .

$$q_{n \rightarrow m}^0 = \alpha_{mn} p_n^0 \prod_{m' \in M(n) \setminus m} r_{m' \rightarrow n}^0 \quad (\text{A.4})$$

$$q_{n \rightarrow m}^1 = \alpha_{mn} p_n^1 \prod_{m' \in M(n) \setminus m} r_{m' \rightarrow n}^1 \quad (\text{A.5})$$

where  $\alpha_{mn}$  is chosen such that  $q_{n \rightarrow m}^0 + q_{n \rightarrow m}^1 = 1$ , we also compute the ‘‘pseudo-posterior

probabilities”, given by

$$q_n^0 = \alpha_n p_n^0 \prod_{m \in M(n)} r_{m \rightarrow n}^0 \quad (\text{A.6})$$

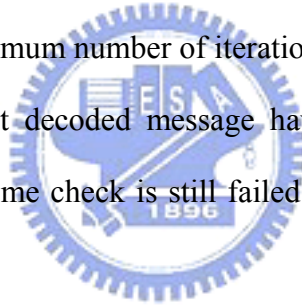
$$q_n^1 = \alpha_n p_n^1 \prod_{m \in M(n)} r_{m \rightarrow n}^1 \quad (\text{A.7})$$

where  $\alpha_n$  is chosen such that  $q_n^0 + q_n^1 = 1$ . These quantities are used to create a tentative decoding  $\hat{x}$ .

**Decision:**

If  $q_n^1 > 0.5$ ,  $\hat{x}_n = 1$ , else  $\hat{x}_n = 0$ . Then, check: if  $H \times \hat{x}^T = 0$ , terminate the decoder and  $\hat{x}$  is the decoding result. Otherwise, go to the horizontal step and do decoding iteratively. A failure is declared if some maximum number of iteration occurs.

Thus, we can realize that decoded message have errors when maximum number of iteration is reached and syndrome check is still failed. Convolutional codes and turbo codes do not have this characteristic.



In SPA described above, there are so many multiplier operations. To reduce the complexity, we can use log domain to represent the received information. By doing this, the operation of multiplier after log domain transformation will become addition. Hence, define our message as posteriori log-likelihood-ratio (LLR):

$$L(x_n) = \ln \frac{P(x_n = 0 | r_n)}{P(x_n = 1 | r_n)} \quad (\text{A.8})$$

For AWGN channel, the value of  $L(x_n)$  is equal to  $-2 r_n / \sigma^2$  when 0 mapping to  $-1$  and 1 mapping to  $+1$ , where  $\sigma^2$  is the noise variance. Furthermore, define the LLRs:

$$\lambda_{n \rightarrow m}(x_n) = \ln \frac{q_{n \rightarrow m}^0}{q_{n \rightarrow m}^1}, \quad \Lambda_{m \rightarrow n}(x_n) = \ln \frac{r_{m \rightarrow n}^0}{r_{m \rightarrow n}^1} \quad (\text{A.9})$$

We call “SPA in the LLR domain” as LLR-SPA, and it is summarized as follows.



**Initialization:**

For every position  $(m, n)$  such that  $H_{mn} = 1$ ,

$$\lambda_{n \rightarrow m}(x_n) = L(x_n)$$

$$\Lambda_{m \rightarrow n}(x_n) = 0$$

**Horizontal step (check-node update):**

From [22] and [28], we can translate equations (A.2), (A.3) and (A.9) together into the following equation,

$$\Lambda_{m \rightarrow n}(x_n) = 2 \tanh^{-1} \left\{ \prod_{n' \in N(m) \setminus n} \tanh[\lambda_{n' \rightarrow m}(x_{n'}) / 2] \right\} \quad (\text{A.10})$$

Obviously, the complexity of equation (A.10) is too high. Hence, we can use the following approximation to reduce the complexity.

Use the parity-check node constraint  $(x_{n_1} \oplus x_{n_2} \oplus \dots \oplus x_{n_{dc}}) = 0$ , where  $N(m) = (n_1, n_2, \dots, n_{dc})$ . And we obtain the simplified expression of equation (A.10),

$$\Lambda_{m \rightarrow n_i}(x_{n_i}) = L(\dots \oplus x_{n_{i-1}} \oplus x_{n_{i+1}} \oplus \dots), \quad i = 1, 2, 3, \dots, dc \quad (\text{A.11})$$

Now, the core operation becomes  $L(U \oplus V)$ . It can be simplified by using the following equation [22]:

$$L(U \oplus V) = \ln \frac{1 + e^{L(U)+L(V)}}{e^{L(U)} + e^{L(V)}} = \text{sign}[L(U)]\text{sign}[L(V)] \cdot \min[|L(U)|, |L(V)|] + \ln[1 + e^{-|L(U)+L(V)|}] - \ln[1 + e^{-|L(U)-L(V)|}] \quad (\text{A.12})$$

where  $\text{sign}[L(U)]\text{sign}[L(V)] \cdot \min[|L(U)|, |L(V)|]$  is the dominant factor. We can discard the remaining part with the penalty of the performance degradation or do some approximation to lower the loss of performance. The details in different kinds of approximation can be seen in [28].

**Vertical step (variable-node update):**

Equations (A.4) and (A.5) can be translated into the following equation in LLR domain,

$$\lambda_{n \rightarrow m}(x_n) = L(x_n) + \sum_{m' \in M(n) \setminus m} \Lambda_{m' \rightarrow n}(x_n) \quad (\text{A.13})$$

Besides, compute the LLR of the ‘‘pseudo-posterior probabilities’’:

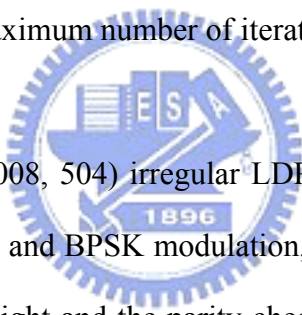
$$\lambda_n(x_n) = L(x_n) + \sum_{m \in M(n)} \Lambda_{m \rightarrow n}(x_n) \quad (\text{A.14})$$

(A.13) can be rewritten as

$$\lambda_{n \rightarrow m}(x_n) = \lambda_n(x_n) - \Lambda_{m \rightarrow n}(x_n) \quad (\text{A.15})$$

**Decision:**

If  $\lambda_n(x_n) \geq 0, \hat{x}_n = 0$ , else  $\hat{x}_n = 1$ . Then, check: if  $H \times \hat{x} = 0$ , terminate the decoder and  $\hat{x}$  is the decoding result. Otherwise, go to the horizontal step and do decoding iteratively. A failure is declared if some maximum number of iteration occurs.



We simulate a rate 1/2 (1008, 504) irregular LDPC codes with the reduced complexity LLR-SPA over AWGN channel and BPSK modulation, where ‘‘irregular’’ means all rows and columns are not of uniform weight and the parity check matrix is referenced from [28]. The performance is shown in figure A.2 with different iterations. We can see that the performance is saturated at about 16 iterations. The larger the block length is, the more iteration numbers for performance saturation is required.

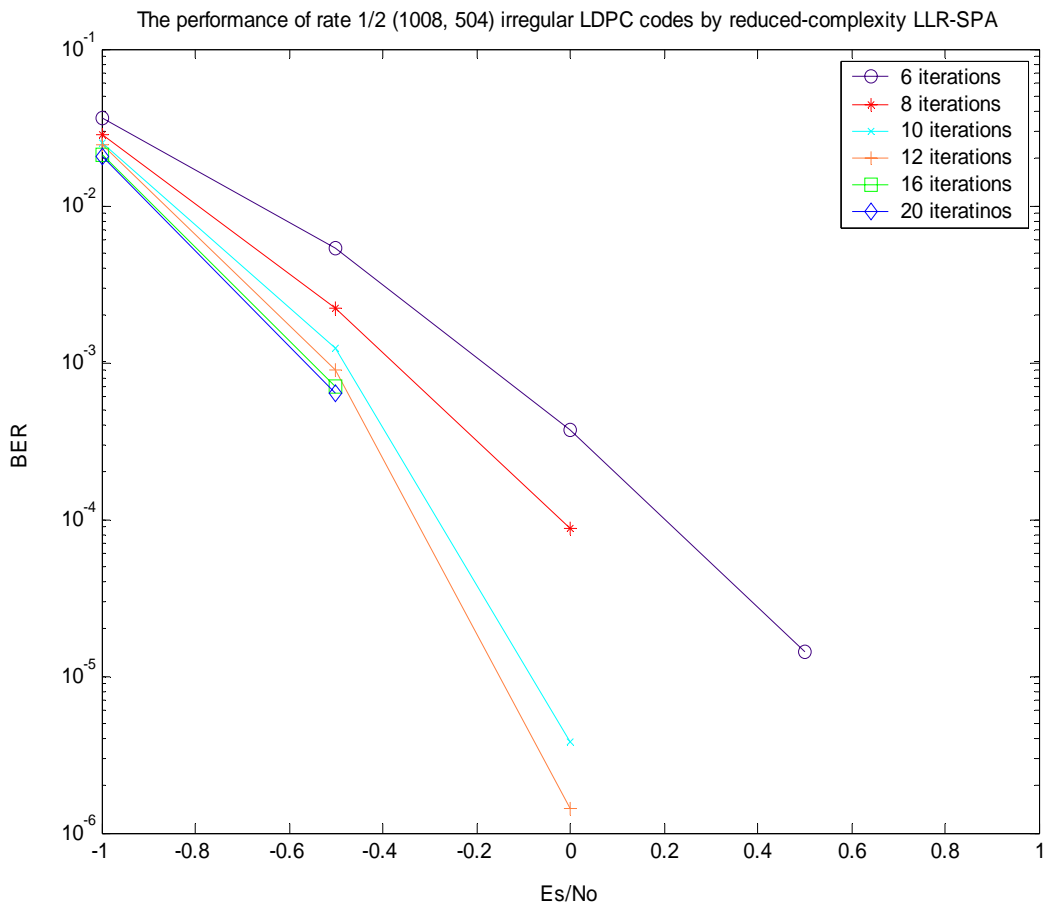


Figure A.2: The performance of rate 1/2 (1008, 504) irregular LDPC codes by reduced-complexity LLR-SPA over AWGN channel

## 作者簡歷

姓名：曾逸晨

出生地：台灣省宜蘭縣

出生日期：1980.2.19.

學歷：2002.9~2004.6 國立交通大學 電子研究所 系統組 碩士 (Si2 Lab)

1998.9~2002.6 國立交通大學 電子工程系 學士

1995.9~1998.6 國立宜蘭高級中學

1992.9~1995.6 宜蘭縣立羅東國民中學

1986.9~1992.6 宜蘭縣立南安國民小學

## 得獎事績

九十二學年度全國大專院校 FPGA 系統設計競賽 Xilinx 研究所組優等

## 發表演文

Yi-Chen Tseng, Chien-Ching Lin, Hsie-Chia Chang, Chen-Yi Lee, “**A Power and Area Efficient Multi-Mode FEC Processor**,” in *IEEE ISCAS*, May 2004.

Keng-Khai Ong, Wei-Hsin Chang, Yi-Cheng Tseng, Yew-San Lee, Chen-Yi Lee, “**A High-Throughput Low Cost Context-based Adaptive Arithmetic Codec for Multiple Standards**,” in *IEEE ICIP*, Sep. 2002.

Keng-Khai Ong, Wei-Hsin Chang, Yi-Cheng Tseng, Yew-San Lee, Chen-Yi Lee, “**A High-Throughput Context-based Adaptive Arithmetic Codec for JPEG2000**,” in *IEEE ISCAS*, May. 2002.