里德所羅門碼之運算分析

研究生：林詩倩　　指導教授：張文鐘 博士

國立交通大學

電信工程研究所

中文摘要

里德所羅門碼因具有良好更正叢集錯誤之能力而成為目前許多通訊系統所選用的錯誤更正碼。本論文將由演算法的基礎理論到運算電路分析來探討里德所羅門碼的編解碼機制，解碼端主要以有效率的代數硬式決策解碼演算法為研究對象，因為在硬體設計複雜度的考量下，硬式決策解碼器仍是最佳選擇，也是至今常被大家採用的解碼器。解碼程序繁複，於計算徵狀值流程後，在此我們以最廣為人知的Berlekamp-Massey演算法、Chien's search 演算法和 Forney 演算法為分析要點。本論文將以詳細分析編解碼器之各個演算法單元的運算電路為主軸，再搭配使用 C 語言來進行編解碼器的運算設計與驗證，並利用 Verilog 硬體描述語言來進行硬體設計，最後再透過 ModelSim 軟體來驗證模擬結果之正確性。

關鍵字：里德所羅門碼，Berlekamp-Massey，Chien's search，Forney。

Operational Analysis of Reed-Solomon Codes

Student: Shih-Chien Lin Advisor: Dr. Wen-Thong Chang

Institute of Communications Engineering,

National Chiao Tung University

Hsinchu, Taiwan

Abstract

Because of the capability to correct burst errors, Reed-Solomon codes are known as one of the widespread error-correction codes in many communication systems currently. In this thesis, we research Reed-Solomon codec from basic theory of algorithms to analysis of operational circuit. For the decoding, the efficient algebraic hard-decision decoding algorithms are our main subject, since the hard-decision decoder is still the best and widely-used one so far in consideration of hardware complexity. We primarily analyze well-known Berlekamp-Massey Algorithm, Chien's search Algorithm, and Forney Algorithm after syndrome computation for the complicated decoding procedure. The object of this thesis is the detailed analysis for the operational circuits of each algorithm module in the codec. We use C language and Verilog HDL to design and verify the codec function respectively. Finally, some ModelSim simulations are performed in order to validate the correctness of each module.

Keywords: Reed-Solomon Code, Berlekamp-Massey, Chien's search, Forney.

誌謝

韶光荏苒，碩士生涯終要在此畫下句點，首先最感謝我的指導教授張文鐘教授，老師對於研究的專業，總能在我學習遇到瓶頸時，幫我尋找資源和給予我最精闢的指導，並總是非常有耐心的協助我洞悉學問，讓我學習到無數寶貴的知識；另外，於生活上老師亦適時的給予關心，分享人生經驗，謝謝老師的教誨，這些我將會銘記在心。同時也感謝口試委員林大衛教授、林信標教授以及尤信程教授，各位口試委員都以精闢的角度和專業能力給予論文上的建議，讓我能更清楚分析與探討論文中的每個環節，終使論文更加完整。

對於 Lab821 的所有成員我滿懷感謝，博班學長家豪不時傳授學習經驗及幫我加油打氣，讓我於學習上更得心應手且信心加倍；97 級學長姐明穎、雅嵐帶領我到羽球的世界，開心運動之餘又因此認識許多朋友；耀葦、怡如則不嫌麻煩總讓我請教課業問題；98 級學長姐維哲、耀駿、舒評、信妤也常與我們分享研究的心路歷程和生活的趣事；99 級同學任恆讓我見識到運動健將熱血的精神；奕廷幽默的話語總能讓我開懷大笑；至中規律的步調則是我學習的好榜樣，也是我一起打拼和打球的好夥伴；有你們的陪伴，讓我的研究生活精彩萬分，希望未來大家還能時常相聚。另外要感謝每當我陷入困難時總是伸出援手的俊宇學長，讓我覺得我好像擁有一隻哆啦 a 夢；還有要感謝這一路以來曾經幫助過我的所有人及身旁所有關心我的朋友們，有你們的拔刀相助才能讓我順利渡過難關，你們總讓我有幸福的感覺。

最後也是最重要的，我要感謝我的家人，你們是我最大的精神支柱，父母總是打電話前來關心我的生活狀況，並偶爾來新竹探望我，讓我在異鄉念書也能感受到家裡的溫暖；哥哥則經常開車專送我來回宜蘭與新竹，讓我免於塞車與轉車之苦。還有要感謝交大土地公保佑，讓我可以在交大兩年都很平順。這一路以來，沒有你們的支持，就沒有現在的我，衷心感謝大家，希望未來大家都能平安健康、事事順心。

誌於　　2012.夏　新竹。交大

詩倩

目錄

表目錄

圖目錄

第1章　　緒論

1.1　研究動機與方向

錯誤更正碼(error correction code)主要是藉由在欲傳輸的訊息符號元(symbol)序列之中加入額外具有數學結構特性的校驗符號元，或是將其轉化為長度較長的訊息符號元序列，以求得編碼增益(coding gain)，利用此冗餘(redundancy)資訊，在接收端依原有之數學結構進行錯誤偵測(error detection)，進而將傳輸過程所造成的錯誤訊息加以更正，以期能達到降低位元之錯誤率(bit-error rate, BER)並有效節省傳輸所需之功率。目前於寬頻通訊系統應用上，經常使用區塊碼(block code)，此編碼方式是以一個固定大小的訊息區塊為單位，每個碼字的 $n-k$ 個奇偶查核元(parity-check bit)僅與本碼字的 k 個資訊元有關，而與其他碼字無關。一般而言，為了達到一定的糾錯能力和編碼效率，區塊碼長度會比較大，而編解碼時必須儲存整個碼字，故由此產生的時間延遲將隨之增加。

在多種的區塊碼中以里德所羅門碼(Reed-Solomon codes, RS codes)的應用最為廣泛，由 I. S. Reed 和 G. Solomon 在 1960 年提出[1]，為一非二元碼(nonbinary codes)，它有效的糾錯能力，使其被運用於眾多通訊與資料儲存系統當中，例如光碟、衛星傳輸以及數位電視地面廣播國際標準都選用了 RS 碼的編碼技術。RS 碼也常作為連接碼(concatenated code)的外部碼(outer code)並結合簡單的二元碼(例如：迴旋碼(convolutional code))作為內部碼(inner code)運用於通道編碼中以增強資料保護的效能。此外，(n, k) RS 碼的最小距離(minimum distance)恰為 $n-k+1$，僅需由 n 和 k 兩個參數決定，故此優點非常便於系統之設計，也是 RS 碼易被採用的原因。

由於 RS 碼之解碼程序繁複，故一直是研究者探討之議題，最早由 Peterson、Gorenstein 和 Zierler 所提出[2][3]，此演算法是基於伴隨矩陣和基本對稱函數(elementary symmetric function)之代數解碼方式，藉由求解反矩陣(matrix inversion)來確定錯誤位置多項式(error-location polynomial)，再以解線性方程式求得錯誤值大小，此為最直觀的解法，但反矩陣運算及求解線性方程式對於錯誤數量較多時效率會變很差，故此方法較少被人提及，雖然如此，但實際上仍對於 RS 碼解碼演算法的發展極其重要。上述方法後來再經由 Chien[4]、Forney[5]、Berlekamp[6]與 Massey[7]改善，其中 Berlekamp 針對求解錯誤位置多項式提出更有效的方法，它以疊代(iterative)方式取代反矩陣的運算，之後由 Massey 以線性反饋移位暫存器(Linear Feedback Shift Register, LFSR)電路實現，後來

被稱為 Berlekamp-Massey 演算法；另外，Sugiyama 等人也發現錯誤位置多項式可利用最大公因數(greatest common divisor, GCD)求解之輾轉相除法得到，此為 Euclidean 演算法[8]，至今這兩種演算法皆較易實現而仍常被大家所採用。後來亦有許多學者進一步又發現於硬體實作時有限場(finite field)的反相器相當複雜，因此有人於 1991 年相繼提出不需使用反相器的方法[9][10]克服了此項難題。然而，以上所提的方法皆需大量的有限場乘法運算及反相運算，故電路設計較複雜且成本較高，直到 1999 年 Chang[11]針對 Berlekamp-Massey 演算法進行了改進，它提出了序列解碼演算法以大大減少實現的複雜度與其所占用的資源。

RS 碼解碼演算法不斷的改進，上述皆屬於硬式決策解碼(hard-decision decoding)演算法，然而，近期亦有許多學者為了達到更精確的解碼效果而提出了軟式決策解碼(soft-decision decoding)演算法，例如[12][13]，雖然其相較於傳統的硬式決策解碼能提供明顯的效能改善，但由於考量硬體實現上的複雜度及成本，目前大部分的系統仍然使用傳統的硬式決策解碼器。有鑑於此，我們將針對 RS 碼的整體演算流程進行詳細的運算分析，並於解碼端選擇基本的硬式決策解碼器作為研究的目標。

1.2 章節概要

第 2 章介紹 RS 碼的理論基礎，包含伽羅瓦場的建立與其四則運算，以及編解碼的演算流程，並說明解碼端所使用的代數解碼演算法。第 3 章介紹第 2 章的數學理論與實現其電路的關係，分別探討每個數學式子適用於硬體電路的系統化運算，另外 Berlekamp-Massey 演算法的疊代步驟亦一併討論。第 4 章則介紹利用硬體描述語言實現 RS 碼的電路架構，包含伽羅瓦場的乘法器設計與元素查表規則，及每個編解碼方塊的運算電路模組，並針對每個模組做詳細的電路分析。第 5 章進行實驗模擬與分析。第 6 章則為結論與未來展望。

第2章　　里德所羅門碼

此章節介紹里德所羅門碼的基本數學理論以及演算流程，其編碼方式主要是透過抽象代數中的伽羅瓦場 GF(2^m)(Galois Field, GF)來進行。首先簡介伽羅瓦場的構成原理及場中之加乘運算；接著介紹 RS 碼之編解碼方式，編碼主要分為系統式編碼與非系統式編碼；而解碼過程較為複雜，在歷年文獻探討中已提供許多演算法，其中仍以 Berlekamp－Massey 演算法及 Euclidean 演算法為目前較常被使用於硬體實現的演算法，本篇則以介紹 Berlekamp－Massey 演算法為主。

2.1　伽羅瓦場GF(2^m)

由有限個元素所構成的場，稱之為有限場，又稱為伽羅瓦場。一般而言，數位訊號在傳輸和儲存系統中普遍為二進制碼，因此我們在此只討論二進制場 GF(2)或它的擴充場 GF(2^m)。首先，若一多項式無法再被因式分解(即它的因式只有 1 與本身)，則稱為不可化簡多項式(irreducible polynomial)。而又假設當不可化簡多項式 $p(x)$ 可整除 X^n+1 且 n 的最小值為 2^m-1，則稱它為 m 階本質多項式(primitive polynomial)。

在此，令 $p(x)$ 為 GF(2^m)的本質多項式，本質元素 α 定義為本質多項式的根，$p(\alpha)=0$ 且 $\alpha^{2^m-1}=1$，則可構成一包含 2^m 個元素之有限場 $F^*=\{0,1,\alpha,\alpha^2,\cdots,\alpha^{2^m-2}\}$，其中每個元素皆有三種表示法，分別為 Power 表示法、Polynomial 表示法及 m-Tuple 向量表示法。表 2.1 為本質多項式 $p(X)=1+X+X^4$ 所產生的有限場 GF(2^4)。

Power representation	Polynomial representation	4-Tuple representation
0	0	(0 0 0 0)
1	1	(1 0 0 0)
α	α	(0 1 0 0)
α^2	α^2	(0 0 1 0)
α^3	α^3	(0 0 0 1)
α^4	$1+\alpha$	(1 1 0 0)
α^5	$\alpha+\alpha^2$	(0 1 1 0)
α^6	$\alpha^2+\alpha^3$	(0 0 1 1)

α^7	$1+\alpha \quad +\alpha^3$	(1 1 0 1)
α^8	$1 \quad +\alpha^2$	(1 0 1 0)
α^9	$\alpha \quad +\alpha^3$	(0 1 0 1)
α^{10}	$1+\alpha+\alpha^2$	(1 1 1 0)
α^{11}	$\alpha+\alpha^2+\alpha^3$	(0 1 1 1)
α^{12}	$1+\alpha+\alpha^2+\alpha^3$	(1 1 1 1)
α^{13}	$1 \quad +\alpha^2+\alpha^3$	(1 0 1 1)
α^{14}	$1 \quad +\alpha^3$	(1 0 0 1)

表 2.1　由 $p(X)=1+X+X^4$ 所產生之 GF(2^4)

其構成原理為：對於每個 Power 表示法的元素來說，其相對應的 Polynomial 表示法即為此元素 modular 本質多項式的結果。如(1)式。

$$\alpha^i \bmod p(\alpha) \tag{1}$$

伽羅瓦場中主要以二進制加法與乘法運算為主，減法同加法運算，除法則可藉由反元素的乘法運算完成。

(1) 加法運算：

假設任意兩元素 α^a 、 α^b 其對應之二進位值(即 m-Tuple 向量表示法)分別為 $(a_0a_1\cdots a_{m-1})$ 與 $(b_0b_1\cdots b_{m-1})$，則

$$\alpha^a+\alpha^b=(a_0a_1\cdots a_{m-1})\oplus(b_0b_1\cdots b_{m-1}) \tag{2}$$

其中 $a_i+b_i, 0<i<m-1$ 為 modular 2 的加法運算，可以利用互斥或閘(XOR)於硬體中實現。

(2) 乘法運算：

假設任意兩元素分別為 Power 表示法 α^a 與 α^b ，則

$$\alpha^a\cdot\alpha^b=\alpha^{(a+b)\bmod(2^m-1)} (\because \alpha^{2^m-1}=1) \tag{3}$$

(3) 反元素(Inverse)：

假設有一元素 α^a ，其反元素為

$$\alpha^{-a}=\alpha^{-a}\alpha^{2^m-1}=\alpha^{2^m-1-a} \tag{4}$$

2.2 RS碼之編碼

RS 碼為一線性區塊碼(linear block code)，是非二元 BCH 碼(nonbinary BCH codes)的一種特例。主要是將每一訊息(message)區塊加入額外資料以達到訊息保護的作用，這些額外資料稱為奇偶查核元/符號元，方便解碼端利用此訊息來做錯誤更正的動作。

(n, k, t)RS 碼表示將長度為 k 的訊息符號元序列編碼成長度為 n 的碼字(codeword)符號元序列，最多可改正 t 個錯誤符號元，而其中每一個訊息符號元及碼字符號元皆為 GF(2^m)中的元素，而奇偶查核符號元個數為 $n-k$ 個，且需滿足 $n-k=2t$ 的條件。由此得知，每個符號元都包含了 m 個位元(bit)訊息，故此特性使它可有效的更正叢集錯誤(burst errors)，因為無論一個符號元中有多少個位元發生錯誤，都僅表示為一個符號元發生錯誤，而對於隨機錯誤(random errors)而言，RS 碼的性能較不明顯。

令 α 為有限場 GF(2^m)的本質元素，則可改正 t 個錯誤符號元的生成多項式(generator polynomial) $g(X)$ 包含了 $\alpha,\alpha^2,\cdots,\alpha^{2t}$ 這些根，因此 $g(X)$ 為

$$g(X)=(X-\alpha)(X-\alpha^2)\cdots(X-\alpha^{2t}) \tag{5}$$

$$=g_0+g_1X+g_2X^2+\cdots+g_{2t-1}X^{2t-1}+X^{2t}, \tag{6}$$

$$g_{i,0\le i\le 2t-1}\in GF\left(2^m\right)$$

很明顯的，當 $g(X)$ 的階數設計愈高，即增加的奇偶查核符號元愈多，代表糾錯能力愈高。

然而，一般編碼方式分為非系統化(nonsystematic)編碼與系統化(systematic)編碼，以下將大略分析其優缺點。假設長度 k 的訊息符號元序列 $u=(u_0,u_1,\cdots u_{k-1})$，它的訊息多項式 $u(X)$ 為

$$u(X)=u_0+u_1X+\cdots+u_{k-1}X^{k-1},\ u_{i,0\le i\le k-1}\in GF\left(2^m\right) \tag{7}$$

非系統化編碼是直接將生成多項式和訊息多項式相乘，但其得到的碼字是不規則的，原本的訊息訊號會被改變，解碼時還要將收到的訊號除以生成多項式才能得到原來的訊息，因此，會使得解碼端在電路設計上困難度增加，故較少被拿來作應用。

系統式編碼則是直接將產生的奇偶查核符號元序列與訊息符號元序列合併即得到碼字，它的優點主要是原本的訊息訊號不會因編碼而改變，所以解碼時，可以很快速地得到原本的訊息，提升解碼的效率。編碼過程分為三個步驟：

步驟一：將訊息多項式 $u(X)$ 乘上 X^{n-k}

$$X^{n-k}u(X) = u_0X^{n-k} + u_1X^{n-k+1} + \cdots + u_{k-1}X^{n-1} \tag{8}$$

步驟二：將 $X^{n-k}u(X)$ 除以生成多項式 $g(X)$ 以得到奇偶查核多項式 $b(X)$

$$X^{n-k}u(X) = a(X)g(X) + b(X) \tag{9}$$

$$b(X) + X^{n-k}u(X) = a(X)g(X) \tag{10}$$

其中 $a(X)$ 和 $b(X)$ 分別為商式和餘式。

而 $b(X) = b_0 + b_1X + \cdots + b_{n-k-1}X^{n-k-1}$, $b_{i,0\le i\le n-k-1} \in GF\left(2^m\right)$。

步驟三：將 $b(X)$ 和 $X^{n-k}u(X)$ 相加，得到碼字多項式 $v(X)$

$$v(X) = b(X) + X^{n-k}u(X) \tag{11}$$

$$= b_0 + b_1X + \cdots + b_{n-k-1}X^{n-k-1} + u_0X^{n-k} + u_1X^{n-k+1} + \cdots + u_{k-1}X^{n-1} \tag{12}$$

因此，碼字為 $(b_0, b_1, \cdots, b_{n-k-1}, u_0, u_1, \cdots u_{k-1})$。由(10)式當中得知，傳送訊號的碼字多項式含有 $g(X)$ 之因式。同理可推，若接收端收到的訊號發生錯誤，接收訊號多項式必不含 $g(X)$ 之因式。

2.3 RS碼之解碼

由於 RS 碼是非二元碼，所以需要決定符號元錯誤的位置(error locations)和其錯誤的值(error values)。相較於編碼端來說，解碼過程複雜許多，因此如何設計一個精確且有效率的解碼器，一直是大家熱烈探討的問題。一般來說，較常使用代數解碼演算法。其中，1965 年由 Forney 提出求錯誤值的演算法；而 Berlekamp 則於 1968 年提出基於疊代的方式找尋錯誤位置多項式，並於 1969 年由 Massey 發現 Berlekamp 演算法等同於尋找出一個能產生給定序列的最短 LFSR 之電路問題，故可透過 LFSR 電路來加以實現，此即為 Berlekamp-Massey 演算法。接著我們將詳細介紹整個解碼流程。

假設傳送訊號多項式 $v(X)$ 為

$$v(X) = v_0 + v_1X + \cdots + v_{n-1}X^{n-1}, \; v_{i,0\le i\le n-1} \in GF\left(2^m\right) \tag{13}$$

而接收到的訊號多項式 $r(X)$ 為

$$r(X)=r_0+r_1X+\cdots+r_{n-1}X^{n-1},\ r_{i,0\le i\le n-1}\in GF\left(2^m\right) \tag{14}$$

經過通道傳輸，傳送訊號有可能受到通道雜訊的干擾，故接收訊號包含了傳送的資訊與雜訊，可表示為 $r(X)=v(X)+e(X)$。我們假設錯誤多項式 $e(X)$ 為

$$e(X)=r(X)-v(X) \tag{15}$$

$$=e_0+e_1X+\cdots+e_{n-1}X^{n-1},\ e_{i,0\le i\le n-1}\in GF\left(2^m\right) \tag{16}$$

若 $e(X)=0$，則接收訊號為正確的。然而，要如何將錯誤訊號更正回來，解碼流程大致分為下列四個部分：

(1) 計算徵狀值(Syndromes)。
(2) 決定錯誤位置多項式 $\sigma(X)$。
(3) 找出錯誤位置(即解出錯誤位置多項式的根)。
(4) 計算出相對應於錯誤位置的錯誤值。

最後再利用解出的錯誤位置及對應的錯誤值將接收到的錯誤訊號更正回來。

2.3.1 計算徵狀值

首先，我們要確認接收到的訊號是否正確，故定義徵狀值來檢查。由於傳送訊號多項式 $v(X)$ 是藉由生成多項式 $g(X)$ 來進行編碼，因此，若 $r(X)=v(X)$，則 $r(X)$ 必定含有 $g(X)$ 之因式(即可被 $g(X)$ 整除)，也就是包含 $g(X)$ 中所有獨立的因式，所以 $g(X)$ 的根必為 $r(X)$ 的根。反之，則表示 $v(X)$ 在傳輸過程中受到雜訊干擾而導致錯誤，即有非零徵狀值出現。

徵狀值的計算原理是將 $g(X)$ 包含的所有根 $\alpha,\alpha^2,\cdots,\alpha^{2t}$ 代入 $r(X)$ 中，因此定義徵狀值 $S=(S_1,S_2,\cdots,S_{2t}),\ S_{i,1\le i\le 2t}\in GF\left(2^m\right)$，更正能力為 t 個錯誤之 RS 碼會產生 $2t$ 個徵狀值。我們必須算出所有 $2t$ 個徵狀值，藉由這些徵狀值來判斷訊號有無錯誤，並可藉此訊息加以做錯誤更正的動作，而只要其中任一 $S_i\neq 0,\ 1\le i\le 2t$，就代表接收訊號錯誤；相反地，所有徵狀值皆為零，接收訊號即為正確的。數學式子表示為

$$S_i=r(\alpha^i)=v(\alpha^i)+e(\alpha^i)=e(\alpha^i),\ 1\le i\le 2t \tag{17}$$

式子(17)中可看出，錯誤多項式會導致 $S_i\neq 0,\ 1\le i\le 2t$，故假設其中 $e(X)$ 包含了 v 個錯誤

符號元，則

$$e(X) = e_{j_1} X^{j_1} + e_{j_2} X^{j_2} + \cdots + e_{j_v} X^{j_v}, \tag{18}$$

$$0 \le j_1 < j_2 < \cdots < j_v \le n-1$$

其中 $X^{j_1}, X^{j_2}, \cdots, X^{j_v}$ 為錯誤位置，而 $e_{j_1}, e_{j_2}, \cdots, e_{j_v}$ 為錯誤的值。

由(17)和(18)式得知

$$\begin{aligned} S_1 &= e_{j_1}\alpha^{j_1} + e_{j_2}\alpha^{j_2} + \cdots + e_{j_v}\alpha^{j_v} \\ S_2 &= e_{j_1}\alpha^{2j_1} + e_{j_2}\alpha^{2j_2} + \cdots + e_{j_v}\alpha^{2j_v} \\ &\vdots \\ S_{2t} &= e_{j_1}\alpha^{2tj_1} + e_{j_2}\alpha^{2tj_2} + \cdots + e_{j_v}\alpha^{2tj_v} \end{aligned} \tag{19}$$

因此，很明顯地看出徵狀值包含了錯誤位置和錯誤值這兩項資訊，若我們想要將訊號更正，就必須解出(19)式，從中找出符號元的錯誤位置和其對應之錯誤值。徵狀值的多項式表示式為

$$S(X) = S_1 + S_2 X + \cdots + S_{2t} X^{2t-1} + S_{2t+1} X^{2t} + \cdots \tag{20}$$

$$= \sum_{j=1}^{\infty} S_j X^{j-1} \tag{21}$$

2.3.2 Berlekamp演算法

Berlekamp 演算法主要是透過疊代的方式來找尋錯誤位置多項式。首先，為了簡化數學運算，先將(19)式整理為

令 $\beta_i \triangleq \alpha^{j_i}$ 且 $\delta_i \triangleq e_{j_i}$ ，$1 \le i \le v$ ，

$$\begin{aligned} S_1 &= \delta_1\beta_1 + \delta_2\beta_2 + \cdots + \delta_v\beta_v \\ S_2 &= \delta_1\beta_1^2 + \delta_2\beta_2^2 + \cdots + \delta_v\beta_v^2 \\ &\vdots \\ S_{2t} &= \delta_1\beta_1^{2t} + \delta_2\beta_2^{2t} + \cdots + \delta_v\beta_v^{2t} \end{aligned} \tag{22}$$

(22)式中，等號左邊是已知的徵狀值，而等號右邊錯誤位置 β_i 和錯誤值 δ_i 則是欲求的未知變數，此時，錯誤個數 v 亦是未知變數。定義錯誤位置多項式 $\sigma(X)$ 為

$$\sigma(X) = (1-\beta_1 X)(1-\beta_2 X)\cdots(1-\beta_v X) \tag{23}$$

$$=\sigma_0+\sigma_1 X+\sigma_2 X^2+\cdots+\sigma_v X^v \tag{24}$$

其中

$$\begin{aligned}
&\sigma_0=1\\
&\sigma_1=-\left(\beta_1+\beta_2+\cdots+\beta_v\right)\\
&\sigma_2=\beta_1\beta_2+\beta_2\beta_3+\cdots+\beta_{v-1}\beta_v\\
&\quad\vdots\\
&\sigma_v=\left(-1\right)^v\beta_1\beta_2\cdots\beta_v
\end{aligned} \tag{25}$$

然而，經由數學推導，可以從(22)和(25)中得到$\sigma(X)$的係數σ_i與徵狀值S_i之關係式：

$$S_{i+v}+\sigma_1 S_{i+v-1}+\cdots+\sigma_v S_i=0,\ i=1,\cdots,2t-v \tag{26}$$

(26)即為廣義牛頓恆等式(generalized Newton's identities)，因此，我們的目標就是要找出可以滿足(26)式之最小階數多項式$\sigma(X)$，亦表示滿足此關係式的情況下，此錯誤位置多項式$\sigma(X)$可找到的錯誤樣本是錯誤個數最少的。因而 Berlekamp 根據此關係式提出一個較有效率的演算法，同時亦可解決錯誤發生的個數v為未知數的問題，其採用反覆疊代的技巧，從滿足$\{S_1\}$的$\sigma(X)$找起，再找滿足$\{S_1,S_2\}$的$\sigma(X)$，不斷疊代直到$\sigma(X)$滿足所有$\{S_1,S_2,\cdots,S_\mu\}$為止，故為了必須滿足$2t$個徵狀值，所以總共要經過$2t$次的疊代。此外，對於最多可更正t個錯誤的 RS 碼來說，必須滿足$v\le t$。若$v>t$，則解碼端將無法正確有效解碼。

我們假設第μ次疊代滿足$\{S_1,S_2,\cdots,S_\mu\}$之錯誤位置多項式為$\sigma^{(\mu)}(X)$，L_μ為$\sigma^{(\mu)}(X)$的階數。(26)式可改寫成

$$S_{j+L_\mu}+\sigma_1^{(\mu)}S_{j+L_\mu-1}+\cdots+\sigma_{L_\mu}^{(\mu)}S_j=0,\ j=1,\cdots,\mu-L_\mu \tag{27}$$

接著檢驗$\sigma^{(\mu)}(X)$是否可滿足下一個徵狀值$S_{\mu+1}$時會產生差異值(discrepancy)

$$d_\mu=S_{\mu+1}+\sigma_1^{(\mu)}S_\mu+\cdots+\sigma_{L_\mu}^{(\mu)}S_{\mu+1-L_\mu} \tag{28}$$

若$d_\mu=0$，則可繼續檢查下一個徵狀值，不需修改錯誤位置多項式，故

$$\sigma^{(\mu+1)}(X)=\sigma^{(\mu)}(X) \tag{29}$$

$$L_{\mu+1}=L_\mu \tag{30}$$

若$d_\mu\neq 0$，則需要修正錯誤位置多項式來滿足$\{S_1,S_2,\cdots,S_{\mu+1}\}$。我們需要藉由先前已找到

的$\sigma^{(\rho)}(X)$來作為輔助修正多項式，其中，$d_\rho \neq 0$且$\rho - L_\rho$為最大值，即ρ亦可選擇為當下錯誤位置多項式最後一次改變階數的前一次疊代。因此，$\sigma^{(\mu+1)}(X)$修正為

$$\sigma^{(\mu+1)}(X) = \sigma^{(\mu)}(X) - d_\mu d_\rho^{-1} X^{(\mu-\rho)} \sigma^{(\rho)}(X) \tag{31}$$

其階數隨之改變為

$$L_{\mu+1} = \max\left(L_\mu, L_\rho + \mu - \rho\right) \tag{32}$$

經過$2t$次疊代完後，得到$\sigma(X) = \sigma^{(2t)}(X)$，即是我們要的錯誤位置多項式，其中階數$L = L_{2t}$即表示錯誤之個數$v$。而 Massey 將以 LFSR 電路實現此演算法，詳細過程於下章節中介紹。

2.3.3 Chien Search演算法

在(23)式定義當中得知，若解出錯誤位置多項式的根，其反元素即可找到錯誤位置。由於無法直接從$\sigma(X)$多項式分解出其因式來得到所有的β值，所以在此利用 Chien's search 演算法，其原理很簡單，就是將伽羅瓦場 GF(2^m)中的所有元素一個個代入$\sigma(X)$中，若可使$\sigma(X) = 0$，則表示此元素即是它的根。而$\sigma(X)$的根之反元素就是β值，$\beta_{i,1\le i\le v} \in GF\left(2^m\right)$，又$\beta_i \triangleq \alpha^{j_i}$，故$j_i$就是代表其錯誤位置。

2.3.4 Forney演算法

經過 Chien search 找到錯誤位置後，即可從(22)式中解出每個錯誤位置β_i相對應的錯誤值δ_i，但當錯誤發生越多時，此聯立方程式會越複雜。在此 Forney 提出了一個更有效的演算法，它利用了錯誤位置多項式$\sigma(X)$與徵狀值多項式$S(X)$的關係，經由數學推導找出錯誤值計算方程式，因而可免於解聯立方程式的運算。

首先，定義了關鍵方程式(key equation)

$$\Omega(X) = \sigma(X) S(X) \mod X^{2t} \tag{33}$$

又從(21)和(22)得知

$$S(X) = \sum_{l=1}^{v} \frac{\delta_l \beta_l}{1 - \beta_l X} \tag{34}$$

故可從(23)、(33)及(34)推得

$$\Omega(X) \triangleq \sum_{l=1}^{v} \delta_l \beta_l \prod_{i=1, i\neq l}^{v} (1-\beta_i X) \tag{35}$$

再將 β_k 的反元素 β_k^{-1} 代進(35)得到

$$\Omega(\beta_k^{-1}) = \delta_k \beta_k \prod_{i=1, i\neq k}^{v} (1-\beta_i \beta_k^{-1}) \tag{36}$$

接著將 $\sigma(X)$ 微分

$$\sigma'(X) = -\sum_{l=1}^{v} \beta_l \prod_{i=1, i\neq k}^{v} (1-\beta_i X) \tag{37}$$

且將 β_k^{-1} 代進(37)可得

$$\sigma'(\beta_k^{-1}) = -\beta_k \prod_{i=1, i\neq k}^{v} (1-\beta_i \beta_k^{-1}) \tag{38}$$

因此，我們可以利用 $\Omega(X)$ 和 $\sigma(X)$ 來求得錯誤值。而由(36)及(38)可計算出錯誤位置 β_k 相對應的錯誤值 δ_k 為

$$\delta_k = \frac{-\Omega(\beta_k^{-1})}{\sigma'(\beta_k^{-1})},\ \delta_{k,1\le k\le v} \in GF(2^m) \tag{39}$$

當我們有了錯誤位置和其相對應之錯誤值後，就可以將錯誤的接收訊號更正回來，圖 2.1 為 RS 碼的整體解碼流程。

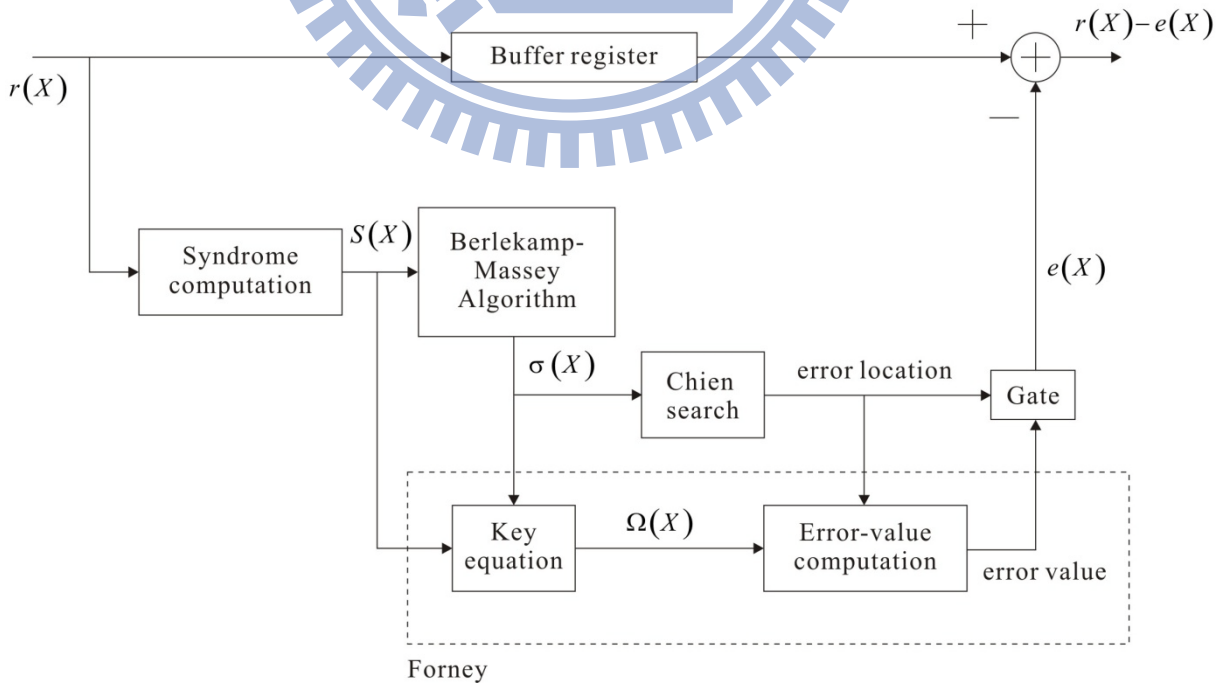

圖 2.1　　RS 解碼端流程圖

接收到的訊號輸入至徵狀值計算方塊檢查訊號是否正確而開始進行解碼的動作；再將徵狀值輸入 Berlekamp-Massey 方塊找出錯誤位置多項式;接著藉由徵狀值多項式與錯誤位置多項式即可找出關鍵方程式；然後將錯誤位置多項式輸入 Chien search 方塊中找出錯誤位置，而關鍵方程式輸入錯誤值計算方塊中，計算出錯誤值。另外，利用信號緩衝器將接收到的訊號暫時儲存起來，使其能於找出錯誤位置及錯誤值後做錯誤更正的動作。

第3章 RS編解碼數學分析

此章節我們將從硬體實現的角度去深入探討 RS 編解碼數學運算流程，分析第二章中的理論數學要如何以實際電路來實現，其中包括伽羅瓦場的乘法器，以及編解碼過程中多項式運算的系統化設計。

3.1 伽羅瓦場乘法器

實際上，伽羅瓦場的乘法運算相較於加法運算複雜許多，我們所處理的數位訊號通常為m-Tuple向量型態，若要如同(3)式以Power型態做乘法運算，則需經過一個轉換的動作。然而，場中加法運算卻是以m-Tuple向量型態完成，因此為了方便運算，所以在硬體中伽羅瓦場的乘法運算仍是同以m-Tuple向量型態來實現。而Polynomial表示法多項式係數即為m-Tuple向量型態，故我們以Polynomial表示法來解釋乘法規則：若場內任意兩元素，皆以冪次為 $m-1$ 的多項式做相乘時，其會產生一冪次為 $2m-2$ 的多項式，但此多項式的係數並不屬於GF(2^m)的有效元素，所以需要再modular本質多項式使它成為有限場內的元素。但硬體要實現modular多項式的除法並不容易，因此可以運用建立好的伽羅瓦場元素表，透過不同型態直接代換更加迅速。我們以GF(2^4)為例，假設場中兩個元素分別為 $a=a_0+a_1\alpha+a_2\alpha^2+a_3\alpha^3$ 及 $b=b_0+b_1\alpha+b_2\alpha^2+b_3\alpha^3$，其中 $a_i, b_i \in \{0,1\}$，而兩個三階多項式相乘結果為六階多項式，故相乘動作分為兩個階段，第一階段先把多項式乘開，第二階段再將結果依照GF(2^4)中的規則轉換為三階以下的多項式。

第一階段：

$$a\cdot b=\left(a_0+a_1\alpha+a_2\alpha^2+a_3\alpha^3\right)\left(b_0+b_1\alpha+b_2\alpha^2+b_3\alpha^3\right) \tag{40}$$

$$\begin{aligned}&=\left(a_0b_0\right)+\\&\left(a_0b_1+a_1b_0\right)\alpha+\\&\left(a_0b_2+a_1b_1+a_2b_0\right)\alpha^2+\\&\left(a_0b_3+a_1b_2+a_2b_1+a_3b_0\right)\alpha^3+\end{aligned} \tag{41}$$

$$(a_1b_3 + a_2b_2 + a_3b_1)\alpha^4 +$$

$$(a_2b_3 + a_3b_2)\alpha^5 +$$

$$(a_3b_3)\alpha^6$$

其中，多項式α^0至α^6的係數分別以t_0、t_1、t_2、t_3、t_4、t_5、t_6表示。

第二階段：將 Power 型態 Power 大於 3 的元素，代換為相對應的 Polynomial 型態，經整理後可得到三階以下的多項式，而此多項式係數即為 GF(2^4)中的有效元素。

$$a \cdot b = t_0 + t_1\alpha + t_2\alpha^2 + t_3\alpha^3 + t_4\alpha^4 + t_5\alpha^5 + t_6\alpha^6 \quad (42)$$

$$= t_0 + t_1\alpha + t_2\alpha^2 + t_3\alpha^3 +$$

$$t_4(1+\alpha) + t_5(\alpha + \alpha^2) + t_6(\alpha^2 + \alpha^3) \quad (43)$$

$$= (t_0 + t_4) + (t_1 + t_4 + t_5)\alpha + (t_2 + t_5 + t_6)\alpha^2 +$$

$$(t_3 + t_6)\alpha^3 \quad (44)$$

因此，同一伽羅瓦場 GF(2^4)的任意兩元素皆可依照此規則來做乘法的動作，其中，係數相乘可利用及閘(AND)於硬體中實現。

然而，上述是探討對於場中任何兩個變數的乘法規則。接下來探討場中一個變數與一個常數彼此相乘之乘法運算，我們亦可推導出其規則，假設要將一個變數元素$c = c_0 + c_1\alpha + c_2\alpha^2 + c_3\alpha^3$乘上常數$\alpha$，其中$c_i \in \{0,1\}$，則

$$c \cdot \alpha = c_0\alpha + c_1\alpha^2 + c_2\alpha^3 + c_3\alpha^4 \quad (45)$$

$$= c_0\alpha + c_1\alpha^2 + c_2\alpha^3 + c_3(1+\alpha) \quad (46)$$

$$= c_3 + (c_0 + c_3)\alpha + c_1\alpha^2 + c_2\alpha^3 \quad (47)$$

由此可知，若有一元素為常數，則乘法動作可直接透過原本係數的加法運算完成。同理可推，亦可將常數α^i的乘法規則建立好，以加速乘法運算。

因此，我們可針對不同的伽羅瓦場，推導出不同的乘法運算規則，而此規則對於同一個伽羅瓦場內的元素皆適用。

3.2 RS碼編碼器

RS 碼系統化編碼主要目的需找到餘式 $b(X)$，將(9)式重新改寫為

$$\frac{X^{2t}u(X)}{g(X)} = a(X) + \frac{b(X)}{g(X)} \tag{48}$$

其中，$a(X) = a_0 + a_1X + \cdots + a_{k-1}X^{k-1}$，所產生的碼字多項式為

$$v(X) = b(X) + X^{2t}u(X) \tag{49}$$

$$= b_0 + b_1X + \cdots + b_{2t-1}X^{2t-1} +$$

$$u_0X^{2t} + u_1X^{2t+1} + \cdots + u_{k-1}X^{n-1} \tag{50}$$

根據(48)、(49)式，編碼動作就是將訊息多項式升 $2t$ 次冪後除以生成多項式，再將所得餘式置於升冪後的訊息多項式後端，由此可見，編碼即藉由多項式除法找出餘式 $b(X)$，故編碼器同除法器，可利用線性反饋移位暫存器電路來實現，其電路原理即為直式長除法過程。RS 碼系統化編碼電路如圖 3.1 所示，初始狀態暫存器 b_i 都是 0，啟動編碼時，Gate 為 ON，依序將被除式係數 $u_{k-1},\cdots,u_1,u_0$ 輸入，而此時 feedback 亦依序為商式中的係數 $a_{k-1},\cdots,a_1,a_0$，經過 k 個循環(即當輸入所有訊息後)，暫存器內容就是餘式係數，即產生奇偶查核符號元 $(b_0,b_1,\cdots,b_{2t-1})$，接著 Gate 為 OFF，feedback 歸零，將 $b_{2t-1},\cdots,b_1,b_0$ 依序輸出加到訊息後面即產生碼字 $(b_0,b_1,\cdots,b_{2t-1},u_0,u_1,\cdots u_{k-1})$，完成編碼。

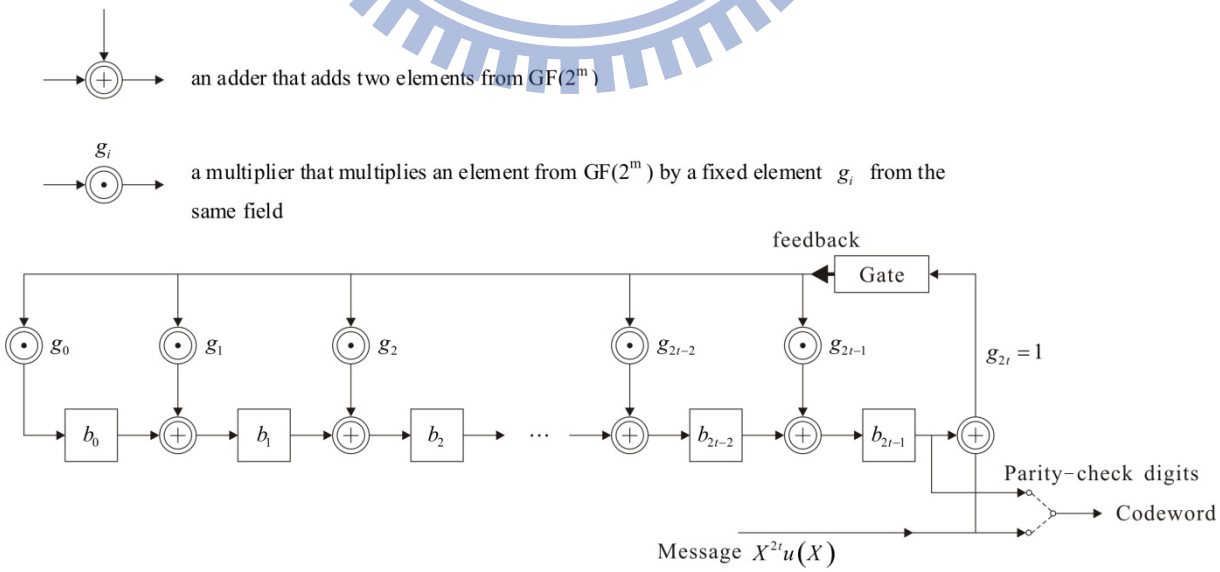

圖 3.1 RS 碼編碼器電路

編碼器電路中需 $2t$ 個乘法器來滿足 $g(X)$ 的係數，然而，如同上節所提到，乘法器

可依乘法規則設計，又欲編碼的生成多項式可依照我們期望的更正錯誤能力來設計，因此，當我們決定生成多項式後，即可依照伽羅瓦場中的乘法規則設計出此$2t$個常數乘法器，使得乘法運算更有效率。

3.3 徵狀值運算分析

由於我們接收到的訊息符號元長度為n，故接收到的訊息多項式是一個$n-1$階的多項式，如(14)式。徵狀值計算需將生成多項式的每個根代入此$n-1$階的多項式中，如(17)式，其展開為$2t$個式子。

$$S_i = r\left(\alpha^i\right) = r_0 + r_1\alpha^i + r_2\left(\alpha^i\right)^2 + \cdots + r_{n-1}\left(\alpha^i\right)^{n-1} \tag{51}$$

其中$1 \le i \le 2t$。此即為多項式求值問題，一般而言，最直觀的求法是分別計算每一項的值之後，再把它們全部累加起來，但此種方法效率很低，對於(51)式來說，它需要進行$1+2+\cdots+(n-1) = n(n-1)/2$次乘法運算和$n-1$次加法運算，因此，假設多項式的數據規模很大時，這種算法顯然不是最佳選擇。然而，若把(51)式更換表示為

$$S_i = r\left(\alpha^i\right) = \left(\cdots\left(\left(r_{n-1}\right)\alpha^i + r_{n-2}\right)\alpha^i + r_{n-3}\right)\alpha^i + \cdots\Big)\alpha^i + r_0 \tag{52}$$

其中$1 \le i \le 2t$。(52)式具有相當良好的遞迴關係，在每一階段的運算皆只要乘上同一個元素α^i，非常適合於電路中實現，且其僅需要$n-1$次乘法運算和$n-1$次加法運算，減少了相當多的乘法運算次數，因此很明顯的提升不少計算效率，而此多項式求值方式亦稱為Horner's Rule。

3.4 Berlekamp-Massey演算法

Berlekamp-Massey演算法主要即藉由(26)這個關係式，並利用LFSR電路實現，如圖3.2，故亦可解釋成：找出LFSR的最小長度使得LFSR前$2t$個輸出徵狀值順序是$S_1, S_2, \cdots, S_{2t}$，而此暫存器所儲存的值即為錯誤位置多項式$\sigma(X)$的係數。尋找方式是從滿足第一個徵狀值$\{S_1\}$的LFSR開始找起，接著修正反饋移位暫存器(即$\sigma(X)$)使其滿足$\{S_1, S_2\}$，反覆執行同樣的動作，經過$2t$次循環，直到反饋移位暫存器滿足$\{S_1, S_2, \cdots, S_{2t}\}$為止，透過這種疊代方式找出$\sigma(X)$的所有係數。

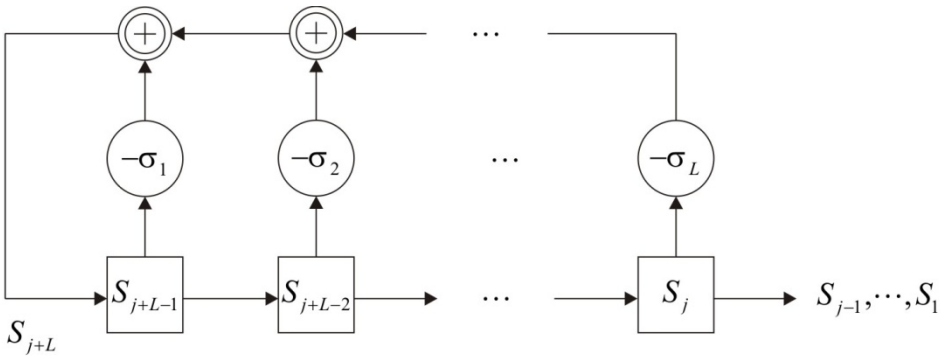

圖 3.2　　L 階的 LFSR 電路

我們假設$\left(L_\mu,\sigma^{(\mu)}(X)\right)$為滿足$\left\{S_1,S_2,\cdots,S_\mu\right\}$之 LFSR，$L_\mu$表示此 LFSR 之長度，$\sigma^{(\mu)}(X)$表示此 LFSR 暫存器中的值，而它會在下一個徵狀值輸入時產生差異值 d_μ，其數學表示式如(28)式。

若$d_\mu=0$，則可繼續輸入下一個徵狀值，不需修改反饋移位暫存器(即錯誤位置多項式)。

若$d_\mu\neq 0$，則需要修正反饋移位暫存器來滿足$\left\{S_1,S_2,\cdots,S_{\mu+1}\right\}$。

然而，參照[7]，符合最小長度的 LFSR 條件為

$$L_{\mu+1}=\max\left(L_\mu,\ \mu+1-L_\mu\right) \tag{53}$$

如同 2.3.2 節所提到，L_ρ為最小長度暫存器當下最後一次改變長度之前一次長度，所以可以假設為

$$L_\rho<L_\mu,\ L_{\rho+1}=L_\mu \tag{54}$$

又因為滿足(53)式，可以得到

$$L_{\rho+1}=L_\mu=\max\left(L_\rho,\ \rho+1-L_\rho\right) \tag{55}$$

$$\therefore L_\mu=\rho+1-L_\rho \tag{56}$$

故可推得

$$\mu+1-L_\mu=\mu+1-\left(\rho+1-\mathrm{L}_\rho\right)=\mu-\rho+\mathrm{L}_\rho \tag{57}$$

因此

$$L_{\mu+1}=\max\left(\mathrm{L}_\mu,\ \mu+1-\mathrm{L}_\mu\right)=\max\left(\mathrm{L}_\mu,\ \mu-\rho+\mathrm{L}_\rho\right) \tag{58}$$

上式成立。

接下來將介紹 Berlekamp-Massey 疊代解碼的演算流程：

令 $B^{(\mu)}(X)$ 為第 μ 個徵狀值的輔助多項式。

(1) Initialize： $\mu=0,\ \sigma^{(\mu)}(X)=1,\ L_\mu=0,\ B^{(\mu)}(X)=1$ 。($d_\rho^{-1}\sigma^{(\rho)}(X)=1,\ \rho=-1$)

(2) For $\mu=0$ to $\mu=2t-1$, compute d_μ.

(3) If $d_\mu=0$, then

$\sigma^{(\mu+1)}(X)=\sigma^{(\mu)}(X)$

$L_{\mu+1}=L_\mu$

$B^{(\mu+1)}(X)=X\cdot B^{(\mu)}(X)$

and go to (6).

(4) If $d_\mu\neq 0$ and $2L_\mu>\mu$, then

$\sigma^{(\mu+1)}(X)=\sigma^{(\mu)}(X)-d_\mu XB^{(\mu)}(X)$

$L_{\mu+1}=L_\mu$

$B^{(\mu+1)}(X)=X\cdot B^{(\mu)}(X)$

and go to (6).

(5) If $d_\mu\neq 0$ and $2L_\mu\leq\mu$, then

$\sigma^{(\mu+1)}(X)=\sigma^{(\mu)}(X)-d_\mu XB^{(\mu)}(X)$

$L_{\mu+1}=\mu+1-L_\mu$

$B^{(\mu+1)}(X)=d_\mu^{-1}\sigma^{(\mu)}(X)$

(6) $\mu=\mu+1$ and return to (2).

故 $\sigma(X)=\sigma^{(2t)}(X)$ 即為所求。

Berlekamp-Massey演算法在修正輔助多項式時會運用到反元素的計算，同乘法原理，數位訊號通常表示為m-Tuple向量型態，其反元素運算較不容易，必須先轉換為Power型態，經由(4)式運算，再轉換回m-Tuple向量型態；因此，我們通常會預先對GF(2^m)中的每個元素，建立一個相對應的反元素表，方便運算時可藉由查表的方式快速搜尋出反元素，提升電路實現的運算效率。

3.5 Chien search電路運算

經由Berlekamp-Massey找到 $\sigma(X)$ 的所有係數後， $\sigma(X)$ 即為一個 $v(v\leq t)$ 階的多項

式，接著Chien's search演算法要將GF(2^m)中的每個元素代進(24)式求解。

$$\sigma\left(\alpha^{l}\right)=\sigma_0+\sigma_1\alpha^{l}+\sigma_2\left(\alpha^{l}\right)^2+\cdots+\sigma_v\left(\alpha^{l}\right)^v \tag{59}$$

$$\triangleq r_{0,l}+r_{1,l}+r_{2,l}+\cdots+r_{v,l} \tag{60}$$

其中$0\le l\le 2^m-2$。並且將(59)式定義為(60)式。同理，若將α^{l+1}代入式子中則為

$$\sigma\left(\alpha^{l+1}\right)=\sigma_0+\sigma_1\left(\alpha^{l+1}\right)+\sigma_2\left(\alpha^{l+1}\right)^2+\cdots+\sigma_v\left(\alpha^{l+1}\right)^v \tag{61}$$

$$=\sigma_0+\sigma_1\left(\alpha^{l}\right)\alpha+\sigma_2\left(\alpha^{l}\right)^2\alpha^2+\cdots+\sigma_v\left(\alpha^{l}\right)^v\alpha^v \tag{62}$$

$$\triangleq r_{0,l+1}+r_{1,l+1}+r_{2,l+1}+\cdots+r_{v,l+1} \tag{63}$$

由上面的式子得知，對於加總的每一項皆符合以下關係式：

$$r_{i,l+1}=r_{i,l}\alpha^{i} \tag{64}$$

其中$0\le i\le v$。因此，我們可以從$l=0$開始反覆計算至$l=2^m-2$，假設

$$\sum_{i=0}^{v} r_{i,l}=0 \tag{65}$$

則α^{l}就是此$\sigma(X)$多項式的根，故錯誤位置即為2^m-1-l。

若將此數學計算方式實現於電路中，其可減低電路複雜度，每一項的乘法運算都簡化為一個變數與一個常數的運算，但因為v為未知數，所以將此電路設計為滿足最大值t之$t+1$個σ暫存器以及t個常數乘法器。

Chien's search演算法電路如圖 3.3，首先，σ暫存器初始值為σ_0, σ_1, σ_2, …, σ_t，當$v<t$，則$\sigma_{v+1}=\sigma_{v+2}=\cdots=\sigma_t=0$。每一冪次項係數$\sigma_1$, σ_2, …, σ_t之常數乘法器分別為α, α^2, …, α^t。電路啟動時，乘法器於每個循環皆分別對σ暫存器中每個冪次項做一次元素累乘的動作再存回σ暫存器，接著將σ暫存器中的值全部加總輸出，最後可從輸出端判別總和是否為零而找到錯誤位置。

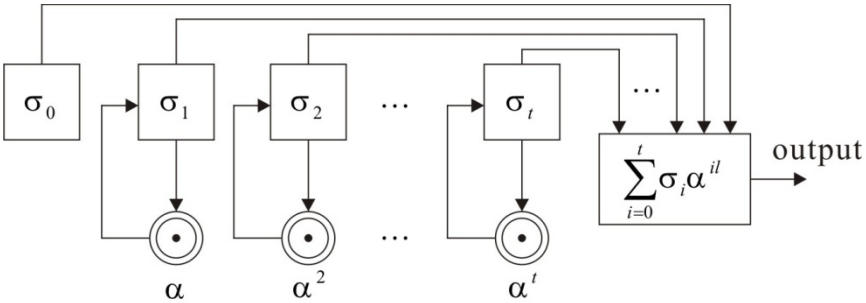

圖 3.3　　Chien's search 演算法電路

Chien's search搜尋法在硬體電路上雖然很耗費時間，但卻是一種很系統化的架構。

3.6 Forney電路運算

由2.3.4節得知，我們可利用(39)式Forney演算法求得錯誤位置相對應之錯誤值。首先將(33)式展開

$$\Omega(X)=\left(\left(\sigma_0+\sigma_1X+\cdots+\sigma_vX^v\right)\cdot\right.$$

$$\left.\left(S_1+S_2X+\cdots+S_{2t}X^{2t-1}\right)\right) \bmod X^{2t} \tag{66}$$

$$=S_1+\left(S_2+\sigma_1S_1\right)X+\left(S_3+\sigma_1S_2+\sigma_2S_1\right)X^2$$

$$+\cdots+\left(S_v+\sigma_1S_{v-1}+\sigma_{v-1}S_1\right)X^{v-1} \tag{67}$$

$$\triangleq\Omega_0+\Omega_1X+\Omega_2X^2+\cdots+\Omega_{v-1}X^{v-1} \tag{68}$$

其中$\Omega_{i,0\le i\le v-1}$為$\Omega(X)$之係數。若把(67)式$\Omega(X)$的每項係數皆設為零，就與(26)廣義牛頓恆等式相同，因此可使用相同的電路來計算。

接下來介紹$\sigma(X)$在硬體電路中的微分運算，由於$\sigma(X)$的係數皆是屬於伽羅瓦場中的元素，場中兩個相同元素相加為零(相當於XOR運算)，亦代表場中任何元素的偶數倍即為零，而奇數倍即等於自己本身，因此，實現$\sigma(X)$的微分運算相當簡單，第一步先把偶數次項的係數設為零，第二步再將所有係數向低冪次方位移一階即可。如(71)式所示。

$$\sigma'(X)=\sigma_1+2\sigma_2X+3\sigma_3X^2+\cdots+v\sigma_vX^{v-1} \tag{69}$$

$$=\sigma_1+\left(\sigma_2+\sigma_2\right)X+\left(\sigma_3+\sigma_3+\sigma_3\right)X^2+$$

$$\cdots+\left(\underbrace{\sigma_v+\sigma_v+\cdots+\sigma_v}_{v}\right)X^{v-1} \tag{70}$$

$$=\sigma_1+\sigma_3X^2+\cdots \tag{71}$$

當我們找到$\Omega(X)$與$\sigma'(X)$後，需再將錯誤位置之反元素β_k^{-1}分別代入多項式(68)與(71)求解，然而，β_k^{-1}即為Chien search演算法中找到的根，在此定義$\beta_k^{-1}\triangleq R_{(k)}$。代入

下列兩式：

$$\Omega\left(R_{(k)}\right)=\Omega_0+\Omega_1\left(R_{(k)}\right)+\Omega_2\left(R_{(k)}\right)^2+\cdots+\Omega_{\text{v-1}}\left(R_{(k)}\right)^{v-1} \tag{72}$$

$$\sigma'\left(R_{(k)}\right)=\sigma_1+0\cdot\left(R_{(k)}\right)+\sigma_3\left(R_{(k)}\right)^2+\cdots \tag{73}$$

經運算後，$\Omega\left(R_{(k)}\right)\in GF\left(2^m\right)$且$\sigma'\left(R_{(k)}\right)\in GF\left(2^m\right)$，最後將這兩個元素相除求得錯誤值，電路中此處的除法運算透過乘法反元素完成。

第4章 RS碼運算硬體電路分析

此章節我們介紹較常被應用於通訊系統的(255, 223, 16)RS 碼，並將編解碼過程於 Verilog HDL 具體實現，再進一步針對每塊模組進行詳細的運算電路分析。整個編解碼流程大致上分為下列五個模組來運作：編碼端之系統化編碼器以及解碼端之徵狀值計算器、Berlekamp-Massey 錯誤位置多項式產生器、Chien's search 錯誤位置運算器與 Forney 錯誤值運算器；而此 RS 碼建立在 GF(2^8)場中，即為於每個編碼區塊內，將 223 bytes 的訊息編碼為 255 bytes 的碼字，並且更正錯誤能力為 16 bytes。

4.1 伽羅瓦場運算電路分析

在此編解碼過程之所有運算皆建立在由本質多項式 $p(X)=1+X^2+X^3+X^4+X^8$ 所產生之 GF(2^8)場中，其元素表為表 4.1，表中的值皆以十六進制表示。而我們將在 Forney 模組電路運算中會使用到此元素表之查表，其查表方式為：假設元素表之記憶體為 gf_alpha_rom[256]，我們知道場中某一元素之 power 為 i，即 α^i，則

$$\alpha^i = \text{gf\_alpha\_rom}[i] \tag{74}$$

因此，其相對之記憶體位址即為元素之 power，故當我們知道元素的 power 即可藉由查表找到其相對應之元素值。

$i=$ 0 ~ 31	$i=$ 32 ~ 63	$i=$ 64 ~ 95	$i=$ 96 ~ 127	$i=$ 128 ~ 159	$i=$ 160 ~ 191	$i=$ 192 ~ 223	$i=$ 224 ~ 255
1	9d	5f	d9	85	e6	82	12
2	27	be	af	17	d1	19	24
4	4e	61	43	2e	bf	32	48
8	9c	c2	86	5c	63	64	90
10	25	99	11	b8	c6	c8	3d
20	4a	2f	22	6d	91	8d	7a
40	94	5e	44	da	3f	7	f4
80	35	bc	88	a9	7e	e	f5
1d	6a	65	d	4f	fc	1c	f7
3a	d4	ca	1a	9e	e5	38	f3
74	b5	89	34	21	d7	70	fb
e8	77	f	68	42	b3	e0	eb
cd	ee	1e	d0	84	7b	dd	cb

87	c1	3c	bd	15	f6	a7	8b
13	9f	78	67	2a	f1	53	b
26	23	f0	ce	54	ff	a6	16
4c	46	fd	81	a8	e3	51	2c
98	8c	e7	1f	4d	db	a2	58
2d	5	d3	3e	9a	ab	59	b0
5a	a	bb	7c	29	4b	b2	7d
b4	14	6b	f8	52	96	79	fa
75	28	d6	ed	a4	31	f2	e9
ea	50	b1	c7	55	62	f9	cf
c9	a0	7f	93	aa	c4	ef	83
8f	5d	fe	3b	49	95	c3	1b
3	ba	e1	76	92	37	9b	36
6	69	df	ec	39	6e	2b	6c
c	d2	a3	c5	72	dc	56	d8
18	b9	5b	97	e4	a5	ac	ad
30	6f	b6	33	d5	57	45	47
60	de	71	66	b7	ae	8a	8e
c0	a1	e2	cc	73	41	9	1

表 4.1 $GF(2^8)$之元素表

然而，於硬體電路中實現查表以唯讀記憶體(ROM)的形式完成；另外，我們將伽羅瓦場中的乘法運算以建立函式的方式實現。

4.1.1 GF(2^8)乘法器電路

如同 3.1 節所分析，我們可以將 $GF(2^8)$中之乘法器依乘法規則設計。首先介紹變數-變數乘法器，$GF(2^8)$中元素為 8-Tuple 向量型態(即 8 位元)，由於場中任意兩個元素其對應之 Polynomial 型態冪次各為 7，彼此相乘展開後冪次為 14(即 15 位元，即不屬於 $GF(2^8)$中之有效元素)，因此我們乘法過程需要一個長度至少為 15 位元的暫存器 t，將乘開後的每個冪次項係數先儲存到 t，接著再遵循推導出之場中規則將結果降回冪次為 7 的有效元素(即為 modular 本質多項式)。假設 $GF(2^8)$中任意兩個 8 位元元素 a 與 b，其每個位元分別以 a[0]~a[7]及 b[0]~b[7]表示，其中 $a[i], b[i] \in \{0,1\}$，經由數學推導，它們的乘法關係如圖 4.1，然而，Polynomial 型態是方便於我們的數學推導，但實際運作卻僅需以 8-Tuple 向量型態計算其係數關係即可得到乘法結果。

```
function[7:0] gfmul_ab  ;
 input[7:0] a ;
 input[7:0] b ;
 reg[15:0] t ;
begin
 t[0] = (a[0]&b[0]) ;
 t[1] = (a[0]&b[1]) ^ (a[1]&b[0]) ;
 t[2] = (a[0]&b[2]) ^ (a[1]&b[1]) ^ (a[2]&b[0]) ;
 t[3] = (a[0]&b[3]) ^ (a[1]&b[2]) ^ (a[2]&b[1]) ^ (a[3]&b[0]) ;
 t[4] = (a[0]&b[4]) ^ (a[1]&b[3]) ^ (a[2]&b[2]) ^ (a[3]&b[1]) ^ (a[4]&b[0]) ;
 t[5] = (a[0]&b[5]) ^ (a[1]&b[4]) ^ (a[2]&b[3]) ^ (a[3]&b[2]) ^ (a[4]&b[1]) ^ (a[5]&b[0]) ;
 t[6] = (a[0]&b[6]) ^ (a[1]&b[5]) ^ (a[2]&b[4]) ^ (a[3]&b[3]) ^ (a[4]&b[2]) ^ (a[5]&b[1]) ^ (a[6]&b[0]) ;
 t[7] = (a[0]&b[7]) ^ (a[1]&b[6]) ^ (a[2]&b[5]) ^ (a[3]&b[4]) ^ (a[4]&b[3]) ^ (a[5]&b[2]) ^ (a[6]&b[1]) ^ (a[7]&b[0]) ;
 t[8] = (a[1]&b[7]) ^ (a[2]&b[6]) ^ (a[3]&b[5]) ^ (a[4]&b[4]) ^ (a[5]&b[3]) ^ (a[6]&b[2]) ^ (a[7]&b[1]) ;
 t[9] = (a[2]&b[7]) ^ (a[3]&b[6]) ^ (a[4]&b[5]) ^ (a[5]&b[4]) ^ (a[6]&b[3]) ^ (a[7]&b[2]) ;
 t[10] = (a[3]&b[7]) ^ (a[4]&b[6]) ^ (a[5]&b[5]) ^ (a[6]&b[4]) ^ (a[7]&b[3]) ;
 t[11] = (a[4]&b[7]) ^ (a[5]&b[6]) ^ (a[6]&b[5]) ^ (a[7]&b[4]) ;
 t[12] = (a[5]&b[7]) ^ (a[6]&b[6]) ^ (a[7]&b[5]) ;
 t[13] = (a[6]&b[7]) ^ (a[7]&b[6]) ;
 t[14] = (a[7]&b[7]) ;
 gfmul_ab[0] = t[0] ^ t[8] ^ t[12] ^ t[13] ^ t[14] ;
 gfmul_ab[1] = t[1] ^ t[9] ^ t[13] ^ t[14] ;
 gfmul_ab[2] = t[2] ^ t[8] ^ t[10] ^ t[12] ^ t[13] ;
 gfmul_ab[3] = t[3] ^ t[8] ^ t[9] ^ t[11] ^ t[12] ;
 gfmul_ab[4] = t[4] ^ t[8] ^ t[9] ^ t[10] ^ t[14] ;
 gfmul_ab[5] = t[5] ^ t[9] ^ t[10] ^ t[11] ;
 gfmul_ab[6] = t[6] ^ t[10] ^ t[11] ^ t[12] ;
 gfmul_ab[7] = t[7] ^ t[11] ^ t[12] ^ t[13] ;
end
endfunction
```

16 bits 暫存器　XOR　AND　Part 1　Part 2

圖 4.1　GF(2^8)中任兩元素之乘法之程式碼

Part 1：

t[0]~t[14]分別儲存多項式乘開後冪次為 0~14 項的係數，$t[i]\in\{0,1\}$，它們係數關係的運算規則是經由多項式的數學推導所得到的。其中，位元相乘以 AND 實現，位元相加以 XOR 實現。

Part 2：

gfmul_ab[0]~ gfmul_ab[7]即為最後相乘結果冪次為 0~7 項的係數。

同理，若將常數元素與變數元素相乘依規則設計為變數-常數乘法器，則可簡化乘法過程。例如 $c\cdot\alpha$，等同將一個 8 位元變數元素 c[0]~c[7]，其中 $c[i]\in\{0,1\}$，輸入常數 α 乘法器，其規則如圖 4.2。

```
function[7:0] gfmul_1 ;
 input[7:0] c ;
begin
 gfmul_1[0] = c[7] ;
 gfmul_1[1] = c[0] ;
 gfmul_1[2] = c[1] ^ c[7] ;
 gfmul_1[3] = c[2] ^ c[7] ;
 gfmul_1[4] = c[3] ^ c[7] ;
 gfmul_1[5] = c[4] ;
 gfmul_1[6] = c[5] ;
 gfmul_1[7] = c[6] ;
end
endfunction
```

圖 4.2　GF(2^8)中 α 乘法器之程式碼

gfmul_1[0]~ gfmul_1[7]即為相乘結果冪次為 0~7 項的係數。

其運算可透過簡單的反饋移位暫存器電路完成。如圖 4.3。

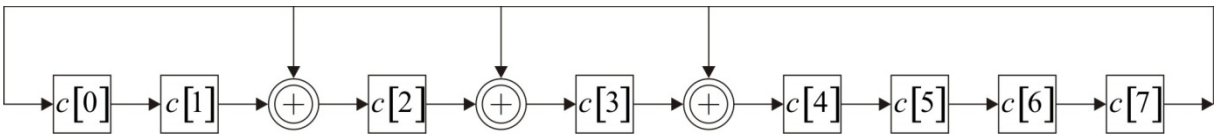

圖 4.3　　GF(2^8)中α乘法器電路

然而，變數-變數乘法器其電路設計包含了許多 AND 閘，尤其當 m 越大(位元數越多)時，需使用的 AND 閘數量越多，導致電路複雜度增加且成本高。因此，在編解碼電路設計過程中，若我們要相乘的其中一個數為已知的常數，相較之下，顯然是直接採用變數-常數乘法器會比採用變數-變數乘法器來的有效率，不僅可省略暫存器的使用，亦可簡化電路設計。

在 Verilog 硬體描述語言中，我們會將所有會用到的乘法運算規則寫成函式之形式，方便於編解碼時直接呼叫過來做運算，而我們實現此 RS 碼之乘法函式總共有 58 個，其中，1 個為變數-變數乘法器，其餘 57 個為變數-常數乘法器。在此，我們的變數-變數乘法器共包含 64 個 AND 閘與 77 個 XOR 閘；而各個變數-常數乘法器包含之 XOR 閘個數如表 4.2 所示。

constant	XOR gate count	constant	XOR gate count
α^1	3	α^{251}	13
α^2	6	α^{215}	26
α^3	9	α^{80}	37
α^4	12	α^{107}	26
α^5	16	α^{248}	22
α^6	19	α^{53}	21
α^7	21	α^{84}	35
α^8	23	α^{194}	17
α^9	22	α^{91}	28
α^{10}	21	α^{59}	31
α^{11}	21	α^{176}	25
α^{12}	21	α^{99}	12
α^{13}	20	α^{203}	24
α^{14}	21	α^{137}	15
α^{15}	23	α^{43}	24
α^{16}	24	α^{104}	23

α^{17}	26	α^{0}	0
α^{18}	25	α^{44}	20
α^{19}	23	α^{149}	21
α^{20}	21	α^{148}	20
α^{21}	19	α^{218}	16
α^{22}	16	α^{75}	30
α^{23}	13	α^{173}	35
α^{24}	11	α^{254}	3
α^{25}	11	α^{109}	30
α^{26}	13		
α^{27}	15		
α^{28}	17		
α^{29}	18		
α^{30}	19		
α^{31}	20		
α^{32}	22		

表 4.2　變數-常數乘法器包含之 XOR 閘個數

由此得知，於解碼器中所需之α^{1}~α^{32}變數-常數乘法器共使用了 571 個 XOR 閘。

4.1.2　GF(2^8)之乘法反元素查表

由於運算有限場反元素的反向器於實際電路設計上較為複雜，故在此硬體實現伽羅瓦場的反元素運算是透過查表的方式來找出其相對應的反元素，接下來在我們的編解碼過程中，會有兩個模組使用到反元素查表運算，分別是 Berlekamp- Massey 以及 Forney。然而，其查表方式為：假設在 GF(2^8)中，反元素表之記憶體為gf_inv_rom$[256]$，對於一元素a而言，其反元素

$$a^{-1} = \text{gf\_inv\_rom}[a] \tag{75}$$

換句話說，反元素記憶體的位址即為其元素本身的值。表 4.3 為基於本質多項式$p(X)=1+X^2+X^3+X^4+X^8$所建立出的相對應反元素表gf_inv_rom$[i]$，表中的值皆以十六進制表示。

i = 0 ~ 31	*i* = 32 ~ 63	*i* = 64 ~ 95	*i* = 96 ~ 127	*i* = 128 ~ 159	*i* = 160 ~ 191	*i* = 192 ~ 223	*i* = 224 ~ 255
1	6c	36	24	1b	1c	12	14
1	ed	5f	57	54	82	59	3f
8e	39	f8	ca	a1	9f	a5	e6
f4	51	d5	5b	1d	c6	35	f0
47	60	92	b9	7c	34	65	86
a7	56	4e	c4	cc	c2	b8	b1
7a	2c	a6	17	e4	46	a3	e2
ba	8a	4	4d	b0	5	9e	f1
ad	70	30	52	49	ce	d2	fa
9d	d0	88	8d	31	3b	f7	74
dd	1f	2b	ef	27	d	62	f3
98	4a	1e	b3	2d	3c	5a	b4
3d	26	16	20	53	9c	85	6d
aa	8b	67	ec	69	8	7d	21
5d	33	45	2f	2	be	a8	b2
96	6e	93	32	f5	b7	3a	6a
d8	48	38	28	18	87	29	e3
72	89	23	d1	df	e5	71	e7
c0	6f	68	11	44	ee	c8	b5
58	2e	8c	d9	4f	6b	f6	ea
e0	a4	81	e9	9b	eb	f9	3
3e	c3	1a	fb	bc	f2	43	8f
4c	40	25	da	f	bf	d7	d3
66	5e	61	79	5c	af	d6	c9
90	50	13	db	b	c5	10	42
de	22	c1	77	dc	64	73	d4
55	cf	cb	6	bd	7	76	e8
80	a9	63	bb	94	7b	78	75
a0	ab	97	84	ac	95	99	7f
83	c	e	cd	9	9a	a	ff
4b	15	37	fe	c7	ae	19	7e
2a	e1	41	fc	a2	b6	91	fd

表 4.3 GF(2^8)之反元素表

4.2 RS編碼電路分析

(255, 223, 16) RS 碼的糾錯能力為 16 個錯誤，根據 RS 碼定義，需要將訊息符號元序列後面加上 32 個奇偶查核符號元才能滿足，因此其生成多項式為

$$g(X)=\prod_{i=1}^{32}\left(X-\alpha^{i}\right) \tag{76}$$

$$=\alpha^{18}+\alpha^{251}X+\alpha^{215}X^{2}+\alpha^{28}X^{3}+\alpha^{80}X^{4}+$$

$$\alpha^{107}X^{5}+\alpha^{248}X^{6}+\alpha^{53}X^{7}+\alpha^{84}X^{8}+\alpha^{194}X^{9}+$$

$$\alpha^{91}X^{10}+\alpha^{59}X^{11}+\alpha^{176}X^{12}+\alpha^{99}X^{13}+\alpha^{203}X^{14}+$$

$$\alpha^{137}X^{15}+\alpha^{43}X^{16}+\alpha^{104}X^{17}+\alpha^{137}X^{18}+\alpha^{0}X^{19}+$$

$$\alpha^{44}X^{20}+\alpha^{149}X^{21}+\alpha^{148}X^{22}+\alpha^{218}X^{23}+\alpha^{75}X^{24}+$$

$$\alpha^{11}X^{25}+\alpha^{173}X^{26}+\alpha^{254}X^{27}+\alpha^{194}X^{28}+$$

$$\alpha^{109}X^{29}+\alpha^{8}X^{30}+\alpha^{11}X^{31}+X^{32} \tag{77}$$

同圖 3.1，此編碼電路之反饋移位暫存器長度為 32，如圖 4.4 所示。圖中變數-常數乘法器為滿足(77)式中 $g(X)$ 的係數，此外，電路包含了兩個二選一多工器，它們用來控制反饋值與電路輸出的資料。

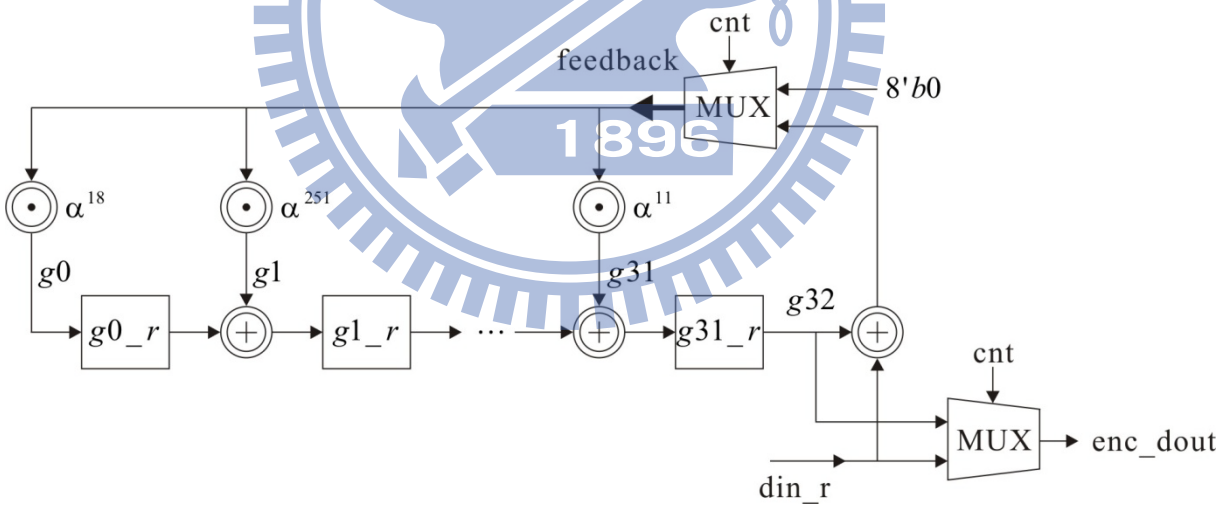

圖 4.4　　(255, 223, 16) RS 碼之編碼電路

編碼電路流程：

(一) 初始化：暫存器 $g0\_r=\cdots=g31\_r=0$ 、循環次數 $\text{cnt}=0$ 。

(二) 當 $0\le \text{cnt} < k$ ：將訊息符號元由 din_r 輸入反饋電路，其每個時刻輸入 1 byte(即 1 個符號元)，此時 $\text{feedback}=g32+\text{din\_r}$，並且同時將訊號直接輸出 $\text{enc\_dout}=\text{din\_r}$ 。

反饋移位電路運算分為兩階段：

(1) 反饋值輸入變數-常數乘法器

$$g0 = \text{feedback} \cdot \alpha^{18}$$
$$g1 = \text{feedback} \cdot \alpha^{251}$$
$$\vdots$$
$$g31 = \text{feedback} \cdot \alpha^{11}$$
$$g32 = g31\_r$$

(2) 暫存器移位

$$g0\_r = g0$$
$$g1\_r = g0\_r + g1$$
$$\vdots$$
$$g31\_r = g30\_r + g31$$

(三) 當 $k \le cnt < n$ ：經過 k(訊息符號元序列長度)個時刻後，訊息符號元輸入完畢，此時已產生奇偶查核符號元並儲存於 $g0\_r$, $g1\_r$, …, $g31\_r$ 暫存器中。因此，反饋值歸零 $\text{feedback} = 8'b0$ ，並繼續將每個暫存器中的資料依序移位至暫存器 $g31\_r$ 後輸出 $\text{enc\_dout} = g32$ 。反饋移位電路運算則為：

(1) 反饋輸入為零

$$g0 = g1 = \cdots = g31 = 0$$
$$g32 = g31\_r$$

(2) 暫存器移位

$$g0\_r = 0$$
$$g1\_r = g0\_r$$
$$\vdots$$
$$g31\_r = g30\_r$$

圖 4.5 為此電路運算流程圖。

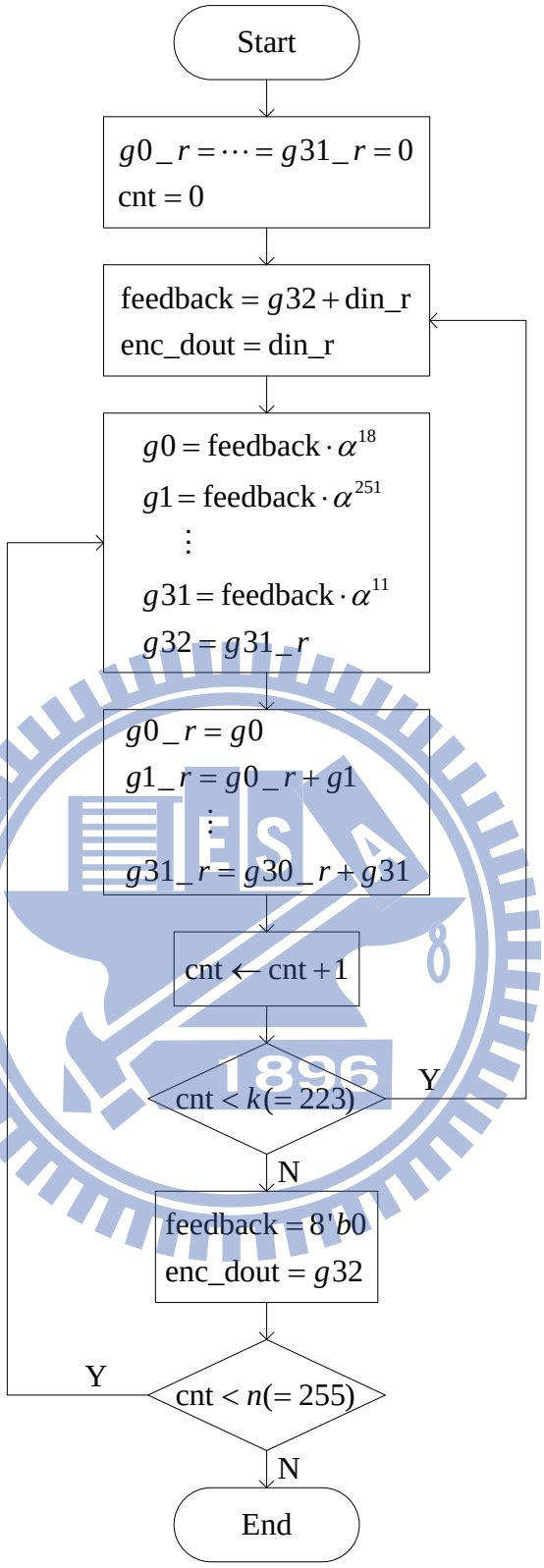

圖 4.5　　編碼電路流程圖

4.3　RS解碼電路分析

RS 解碼於硬體電路設計中，主要將其分為四個區塊：徵狀值計算、Berlekamp-Massey、

Chien's search 以及 Forney。由於 RS 硬體解碼速度主要受限於徵狀值計算電路與 Chien's search 電路，它們皆需要 n 個循環時間才能完成，而原先 Berlekamp-Massey 電路架構只需 $4t \sim 6t$ 個循環時間，但其因包含多項式運算需要 $2t \sim 3t$ 個有限場乘法器，故在此運用[11]所提出的序列解碼架構，其增加了循環時間數，但不影響整體解碼速度，並其可將乘法器個數減少至 3 個，降低了電路的複雜度，其原理於 4.3.2 節中將介紹。

4.3.1 徵狀值運算電路

為了提高運算速度，徵狀值在電路中實現的計算方式採用具有遞迴關係的(52)式，其對於每個單一徵狀值皆僅需使用一個變數-常數乘法器即可，而變數-常數乘法器則是依據 $g(X)$ 的根來決定；另一方面，由於接收訊號多項式冪次為 $n-1$，因此，每個單一徵狀值電路皆必須遞迴執行 n 次才能完成整個多項式的求值運算，但所有 $2t$ 個徵狀值可同時並行處理，故可將其整合設計為一個平行運算架構的電路，而此電路共包含 $2t$ 個變數-常數乘法器：$\alpha, \alpha^2, \cdots, \alpha^{2t}$。在此，我們實現之 $(255, 223, 16)$ RS碼共有 $32(=2t)$ 個徵狀值產生，其電路如圖 4.6所示。

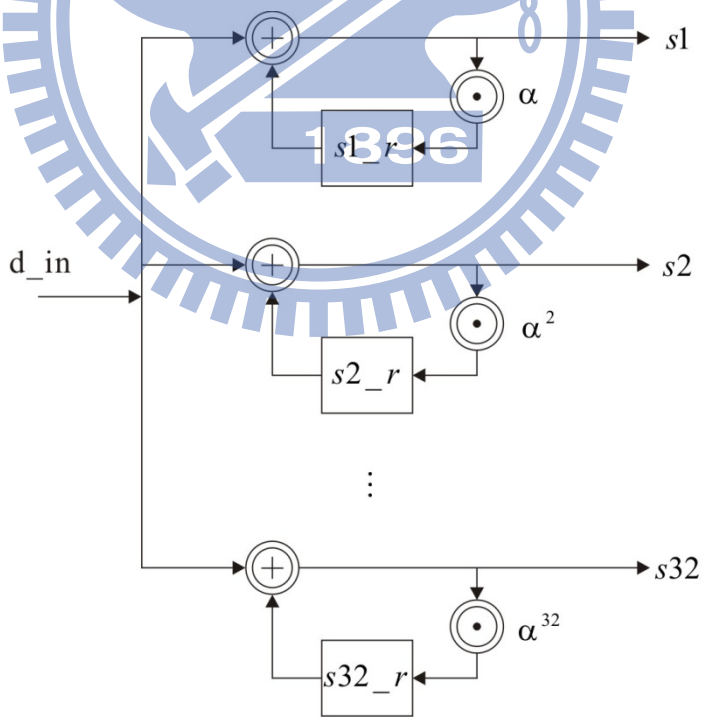

圖 4.6　　32 個徵狀值平行運算電路

其中，$s1\_r, s2\_r, \cdots, s32\_r$ 為儲存每次遞迴運算值的暫存器，而我們接收到的訊號由 d_in 輸入，於每個時刻輸入一個符號元(1 byte)，依序將 $r_{n-1}, r_{n-2}, \cdots, r_0$ 全部輸入完

畢，因此，此電路需要 $n(=255)$ 個循環時間才能完成計算。

4.3.2 **Berlekamp-Massey電路**

如同 3.4 節所介紹，我們實現 Berlekamp-Massey 演算法的流程如圖 4.7 所示。然而，由於我們實現之 RS 碼有 32 個徵狀值，因此，我們在此的 Berlekamp-Massey 演算法整個流程需疊代 32 次。參數定義為：

r 為循環次數。

L 為 $L(X)$ 的冪次。

$delta\_r$ 表示差異值。

$B(X)$ 為輔助多項式。

$L(X)$ 為錯誤位置多項式。

$T(X)$ 為暫存修正後之 $L(X)$ 的多項式。

其中，與 3.4 節描述的步驟上僅有些微差異，即循環次數 r 在流程中累加的時間不同，因而使得數學表示上有些微不同。

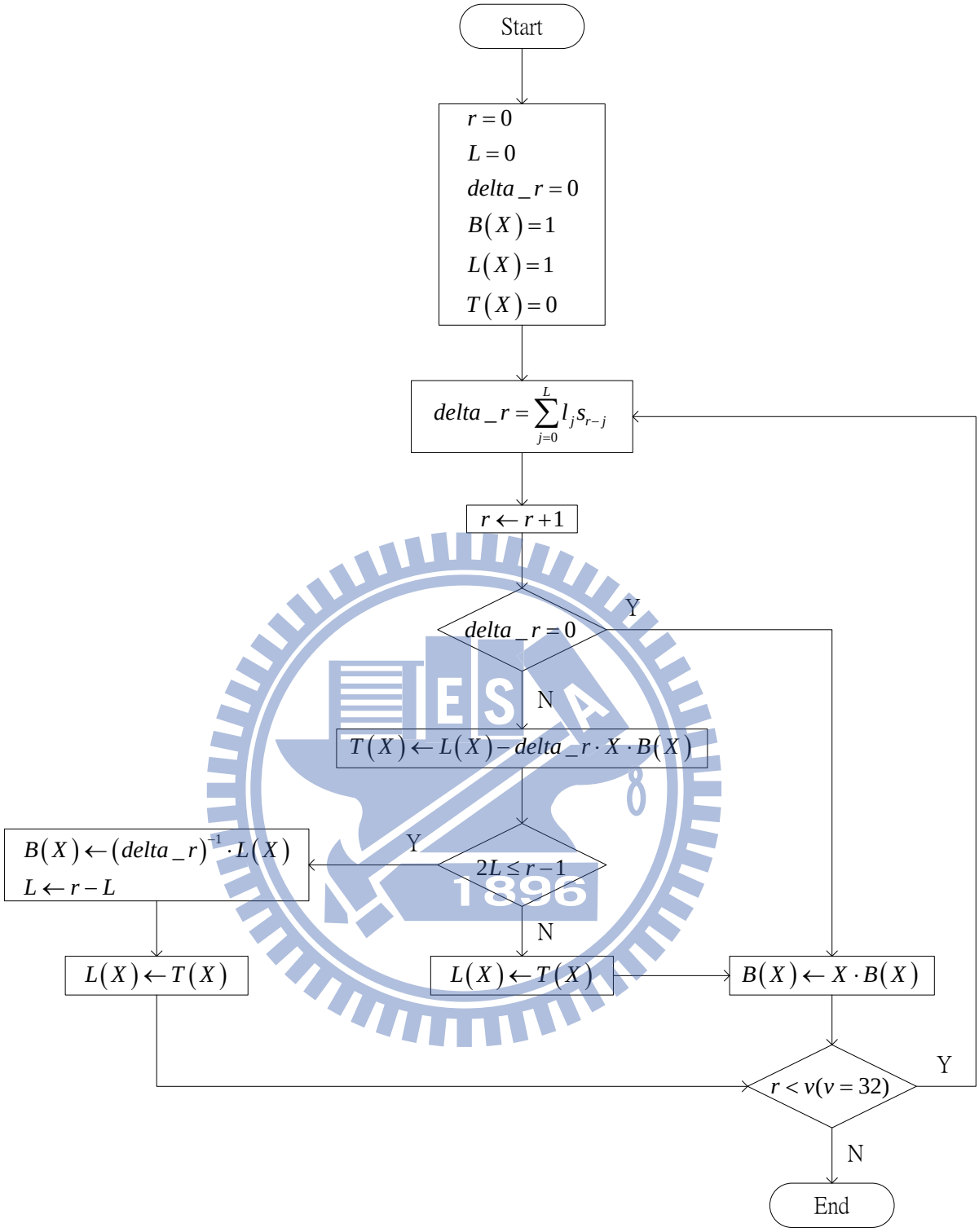

圖 4.7　　Berlekamp-Massey 電路運算流程圖

在此，Berlekamp-Massey 演算法採用序列解碼的方式實現，序列解碼的原理是將多項式分解成係數的運算，演算法中主要反覆疊代的兩個式子如下：

計算差異值 $d^{(r)}$

$$d^{(r)} = l_0^{(r)} s_r + l_1^{(r)} s_{r-1} + \cdots + l_{L_r}^{(r)} s_{r-L_r} \tag{78}$$

修正錯誤位置多項式

$$L^{(r)}(X) = L^{(r-1)}(X) - d^{(r-1)} XB^{(r-1)}(X) \tag{79}$$

其中 $0 \le r \le 2t-1$，L_r 表示 $L^{(r)}(X)$ 的冪次。將(78)與(79)數學式分解為以下的係數型態：

$$d_j^{(r)} = \begin{cases} l_0^{(r)} s_r, & \text{for } j = 0 \\ d_{j-1}^{(r)} + l_j^{(r)} s_{r-j}, & \text{for } 1 \le j \le L_r \end{cases} \tag{80}$$

$$l_j^{(r)} = \begin{cases} l_0^{(r-1)}, & \text{for } j = 0 \\ l_j^{(r-1)} - d^{(r-1)} b_{j-1}^{(r-1)}, & \text{for } 1 \le j \le L_r \end{cases} \tag{81}$$

其中，$d_j^{(r)}$ 為計算 $d^{(r)}$ 第 j 次的部分結果；而 $l_j^{(r)}$ 與 $b_j^{(r)}$ 分別為 $L^{(r)}(X)$ 和 $B^{(r)}(X)$ 的係數。換句話說，即是將第 r 次的疊代再分解成 $L_r + 1$ 個循環 $(L_r \le t)$ 來執行。由此得知，於每個循環中，計算 $d_j^{(r)}$ 只需要一個變數-變數乘法器，計算 $l_j^{(r)}$ 至多只需要兩個變數-變數乘法器(其中一個包含在修正輔助多項式時，$L^{(r-1)}(X)$ 的係數與乘法反元素之乘法運算)，因此，它取代了原先整個多項式同時運算的方法，大量減少乘法器的個數，因而簡化電路。此電路架構使用了 3 個有限場乘法器與 1 個乘法反元素，如圖 4.8。

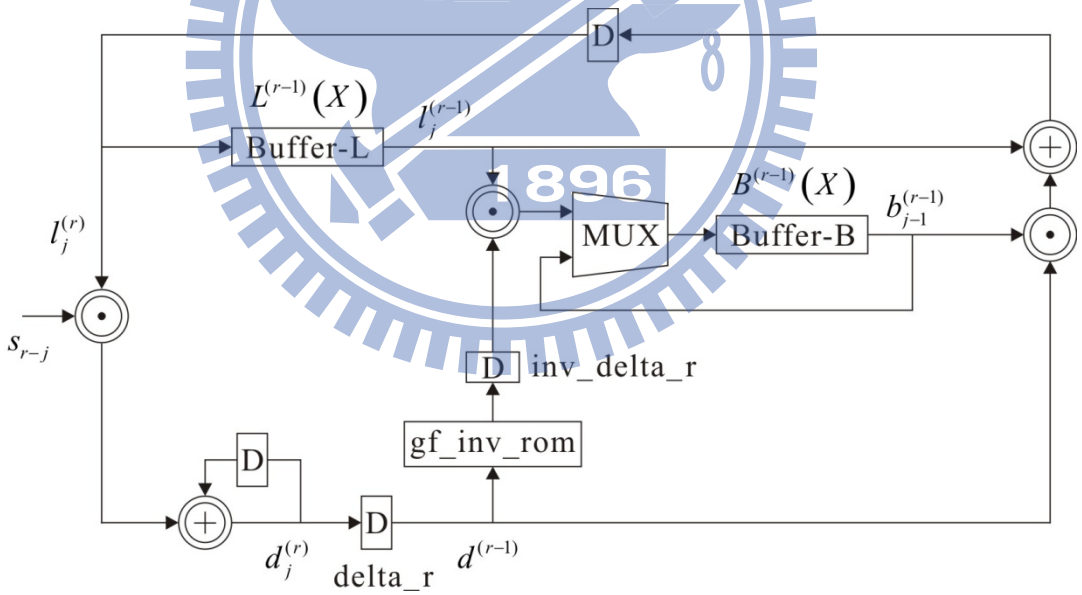

圖 4.8　　Berlekamp-Massey 序列解碼電路

接下來將進一步分析 Berlekamp-Massey 每一個步驟的詳細電路運算。首先，我們提供以下六個長度為 364 位元的暫存器，用來暫存多項式的係數以完成序列解碼的運算過程，而它們可儲存冪次為 32 的多項式($\because 8 \times 33 = 264$)。雖然提供的暫存器最多皆可儲存至冪次為 32 之多項式，但我們實際運算所求得的多項式冪次最高只有 16。

B_x_Q：儲存 $B(X)$ 的係數 b_j。

L_x_Q：儲存$L(X)$的係數l_j。

T_x_Q：儲存$T(X)$的係數，主要用來暫存修正之後$L(X)$的係數。

R_0_Q、R_1_Q、R_2_Q：當要執行序列運算時，可透過這些暫存器將係數以位移的方式來完成。

在此，我們有另外建立一個模組，其主要功能是用來控制這些多項式暫存器的執行動作。

(一) 初始化

暫存器 B_x_Q 與 L_x_Q 皆初始化為 1。其餘為 0。

(二) 計算 $delta\_r = \sum_{j=0}^{L} l_j s_{r-j}$

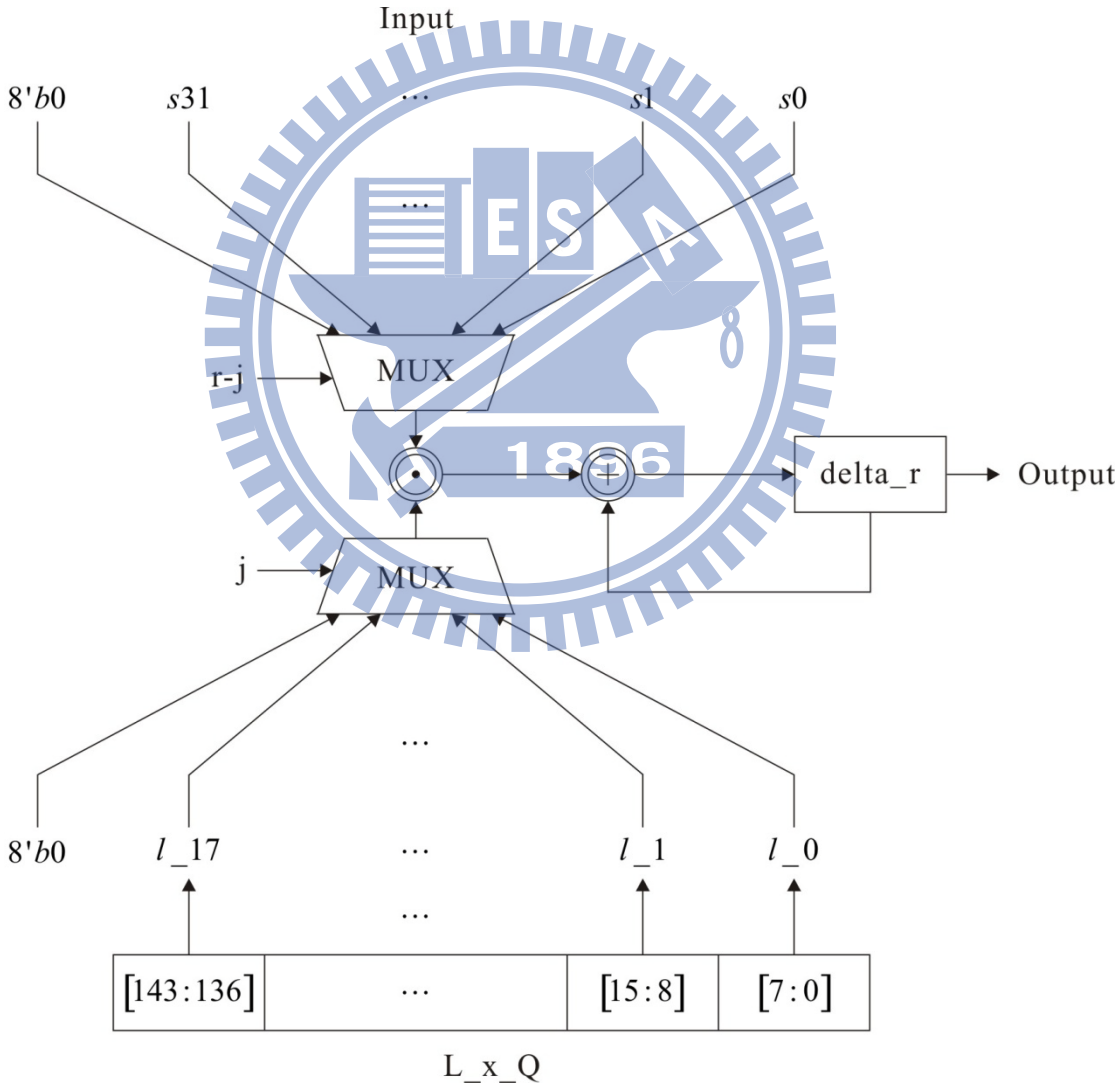

圖 4.9　　Berlekamp-Massey 差異值(delta_r)運算電路

如圖 4.9，首先，我們從 L_x_Q 暫存器取出每次疊代後所求得的$L(X)$係數，接著

藉由兩個多工器，分別選擇適當的徵狀值 s_{r-j} 與 $L(X)$ 的係數 l_j 輸出以相乘，再利用一個反饋電路執行累加動作，反覆 L+1 個時刻即可。

(三) 判別 $delta\_r$ 是否為零

(1) $delta\_r \neq 0$

(i) 如圖 4.10，二選一多工器選擇 B_x_Q 暫存器為輸出至 R_0_din 暫存器，並將其 load 至 R_0_Q 暫存器中，準備進行下一步運算。

(ii) 同時亦將 L_x_Q 暫存器中的值 load 至 R_1_Q 暫存器做準備。

(iii) 跳至(四)。

(2) $delta\_r = 0$

(i) 將 B_x_Q 暫存器向左位移一次(最低位 B_x_Q$[7:0]$補零)，得到 $X \cdot B(X)$。

(ii) 回到(二)。

(四) 計算 $T(X) \leftarrow L(X) - delta\_r \cdot X \cdot B(X)$。

分為三個步驟執行：

(1) $X \cdot B(X)$

(i) 如圖 4.10，此時，R_0_Q 暫存器中已是 $B(X)$ 的係數。

(ii) 接著再將 R_0_Q 暫存器內的值向左位移一次即可。

(2) $delta\_r \cdot (X \cdot B(X))$

(i) 如圖 4.10，繼續上一個步驟，此時 R_0_Q 暫存器中所儲存的值為 $X \cdot B(X)$ 的係數。

(ii) R_0_Q 暫存器透過不斷向左位移將 $X \cdot B(X)$ 每個冪次項係數位移至最高位 R_0_Q$[255:248]$與 $delta\_r$ 做相乘，每次位移時，最低位 R_0_Q$[7:0]$補零，並將 $delta\_r \cdot$ R_0_Q$[255:248]$結果輸入至 R_2_Q$[7:0]$。R_2_Q 暫存器則是將結果由最低位 R_2_Q$[7:0]$輸入，藉由向左位移的方式依序將多項式的每個冪次項係數存至暫存器中。故此步驟 R_0_Q 暫存器與 R_2_Q 暫存器於每個時刻皆需向左位移一次，重複 32 個時刻(∵暫存器以每 byte 為一單位之長度為 33)。

(iii) 經過 32 個時刻後，R_2_Q 暫存器中即為 $delta\_r \cdot X \cdot B(X)$ 的係數，此時，

二選一多工器選擇 R_2_Q 暫存器為輸出至 R_0_din 暫存器，並將其 load 至 R_0_Q 暫存器中，結束 $delta\_r \cdot X \cdot B(X)$ 之運算。

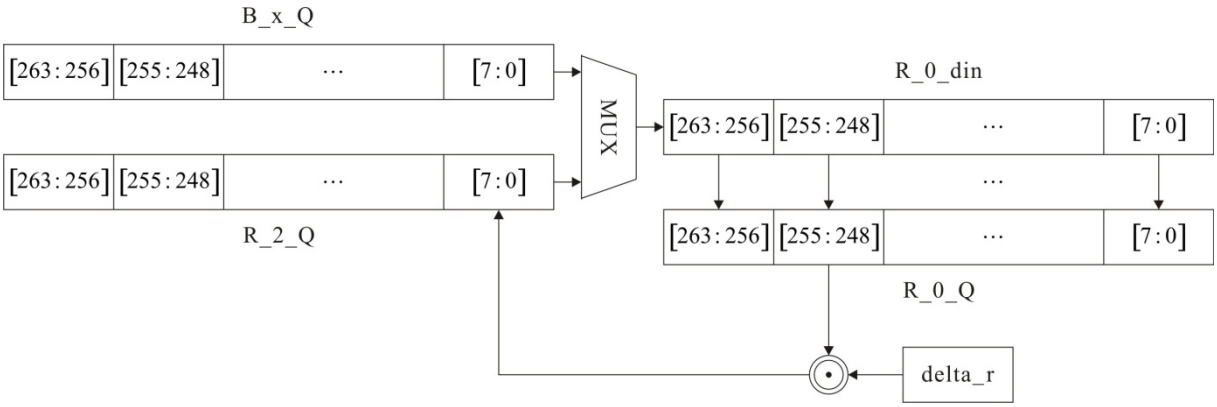

圖 4.10　Berlekamp-Massey 計算 $delta\_r \cdot X \cdot B(X)$ 之電路

(3) $T(X) \leftarrow L(X) - (delta\_r \cdot X \cdot B(X))$

(i) 如圖 4.11，先前已先將 L_x_Q 暫存器中的值($L(X)$ 的係數)load 至 R_1_Q 暫存器。然而，繼續上一個步驟，此時 R_0_Q 暫存器中所儲存的值為 $delta\_r \cdot X \cdot B(X)$ 的係數。

(ii) R_0_Q 暫存器與 R_1_Q 暫存器同時皆向左位移，將每個冪次項係數位移至最高位 R_0_Q$[255:248]$與 R_1_Q$[255:248]$，再彼此做相加，並將結果輸入至 R_2_Q$[7:0]$。R_2_Q 暫存器同樣藉由向左位移的方式將運算完的結果存至暫存器中。故每個時刻 R_0_Q 暫存器、R_1_Q 暫存器與 R_2_Q 暫存器皆需向左位移一次，其中 R_0_Q$[7:0]$與 R_1_Q$[7:0]$皆補零，重複 32 個時刻。

(iii) 經過 32 個時刻後，R_2_Q 暫存器中即儲存 $L(X) - (delta\_r \cdot X \cdot B(X))$ 的係數，此時，將其 load 至 T_x_Q 暫存器中，完成 $T(X) \leftarrow L(X) - (delta\_r \cdot X \cdot B(X))$ 的運算。

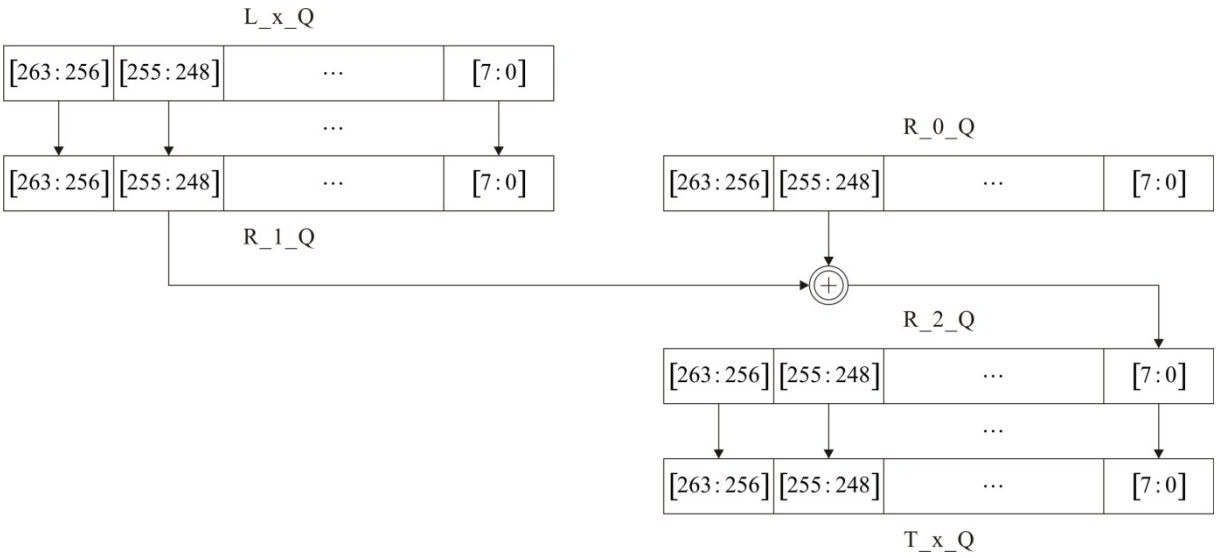

圖 4.11　Berlekamp-Massey 計算 $T(X) \leftarrow L(X) - (delta\_r \cdot X \cdot B(X))$ 之電路

(五) 判別 $2L$ 是否小於或等於 $r-1$

(1) $2L \le r-1$

(i) 如圖 4.12，將 L_x_Q 暫存器中 $L(X)$ 的係數同樣再 load 一次至 R_1_Q 暫存器。

(ii) 跳至(六)。

(2) $2L > r-1$

(i) 此時 T_x_Q 暫存器中的值為多項式 $T(X) \leftarrow L(X) - (delta\_r \cdot X \cdot B(X))$ 的係數，故可直接將其 load 回至 L_x_Q 暫存器更新 $L(X)$。

(ii) 將 B_x_Q 暫存器向左位移一次(最低位 B_x_Q$[7:0]$ 補零)，得到 $X \cdot B(X)$。

(iii) 回到(二)。

(六) $B(X) \leftarrow (delta\_r)^{-1} \cdot L(X)$

(i) 如圖 4.12，此時，R_1_Q 暫存器中已是 $L(X)$ 的係數。

(ii) R_1_Q 暫存器一樣反覆向左位移的動作，每次位移時，最低位 R_1_Q$[7:0]$ 補零，其將 $L(X)$ 每個冪次項係數位移至最高位 R_1_Q$[255:248]$ 與 inv_delta_r 做相乘，並將 $inv\_delta\_r \cdot$ R_1_Q$[255:248]$ 結果輸入至 R_2_Q$[7:0]$。然而，再次以同樣向左位移的方式將運算完的結果存至 R_2_Q 暫存器中。故此步驟 R_1_Q 暫存器與 R_2_Q 暫存器於每個時刻皆需向左位移一次，重複 32 個時

刻。

(iii) 經過 32 個時刻後，R_2_Q 暫存器中即儲存多項式$(delta_r)^{-1}\cdot L(X)$的係數，最後，再將其 load 至 B_x_Q 暫存器中，完成更新$B(X)\leftarrow(delta_r)^{-1}\cdot L(X)$的動作。

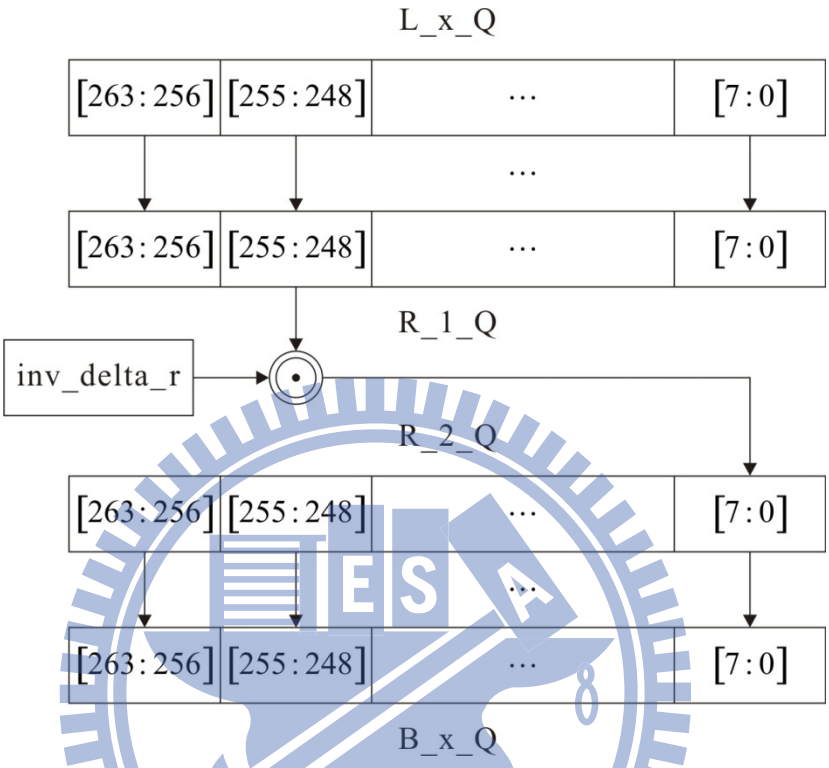

圖 4.12 Berlekamp-Massey 計算$B(X)\leftarrow(delta_r)^{-1}\cdot L(X)$之電路

(iv) 完成(i)~(iii)後，要再將暫存於 T_x_Q 中新的$L(X)$係數(此時為多項式$T(X)\leftarrow L(X)-(delta_r\cdot X\cdot B(X))$的係數)load 回至 L_x_Q 暫存器以完成$L(X)$之更新。

(v) 回到(二)。

4.3.3 Chien search電路

根據 3.5 節的數學推導得知，將元素代入錯誤位置多項式求根時，代入α^{l}與代入α^{l+1}它們兩者之間多項式的每一項皆符合(64)式之關係，因此參照圖 3.3，符合我們實現的 RS 碼其搜尋錯誤位置多項式的根之電路圖為圖 4.13，共有 17 的暫存器r_i與 16 個變數-常數乘法器α, α^2, …, α^{16}。當電路啟動時，先將錯誤位置多項式$L(X)$的係數l_i輸入至

暫存器中，接著於每個循環不斷累乘暫存器中的值即可，直到循環完$2^m(=256)$(其為伽羅瓦場包含之所有元素個數)次後停止。

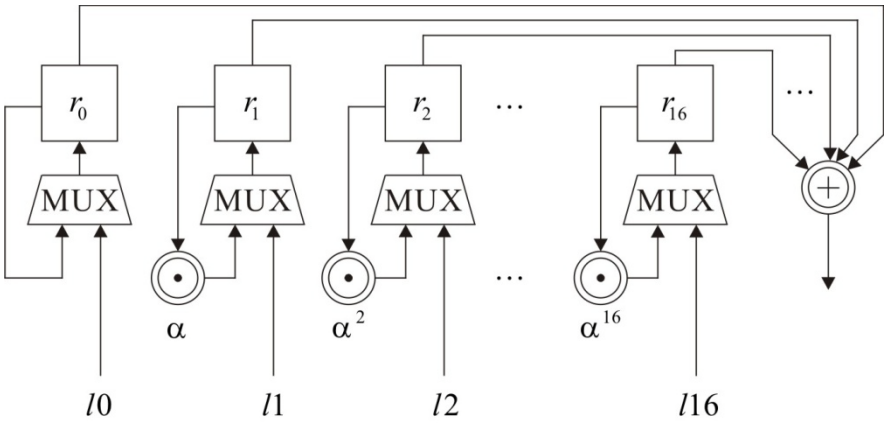

圖 4.13 Chien search 電路

而圖 4.14 為其電路流程圖，其中參數為：r_i 為暫存器，j 為找出之第 j 個根，Num_Roots 為找出之根的個數，n 為循環次數，root_j 為儲存第 j 個根的 power。

在此另外討論，若同以徵狀值運算電路的方式實現此演算法，則需使用 2^m-1 個變數-常數乘法器分別來進行每個元素代入 $\sigma(X)$ 之運算，相較之下，上述的電路僅需 t 個變數-常數乘法器，由於 $t \ll 2^m-1$，故乘法器個數相對減少非常多，若在節省電路設計成本及降低複雜度的考量下，很明顯地此為更好的設計方式。

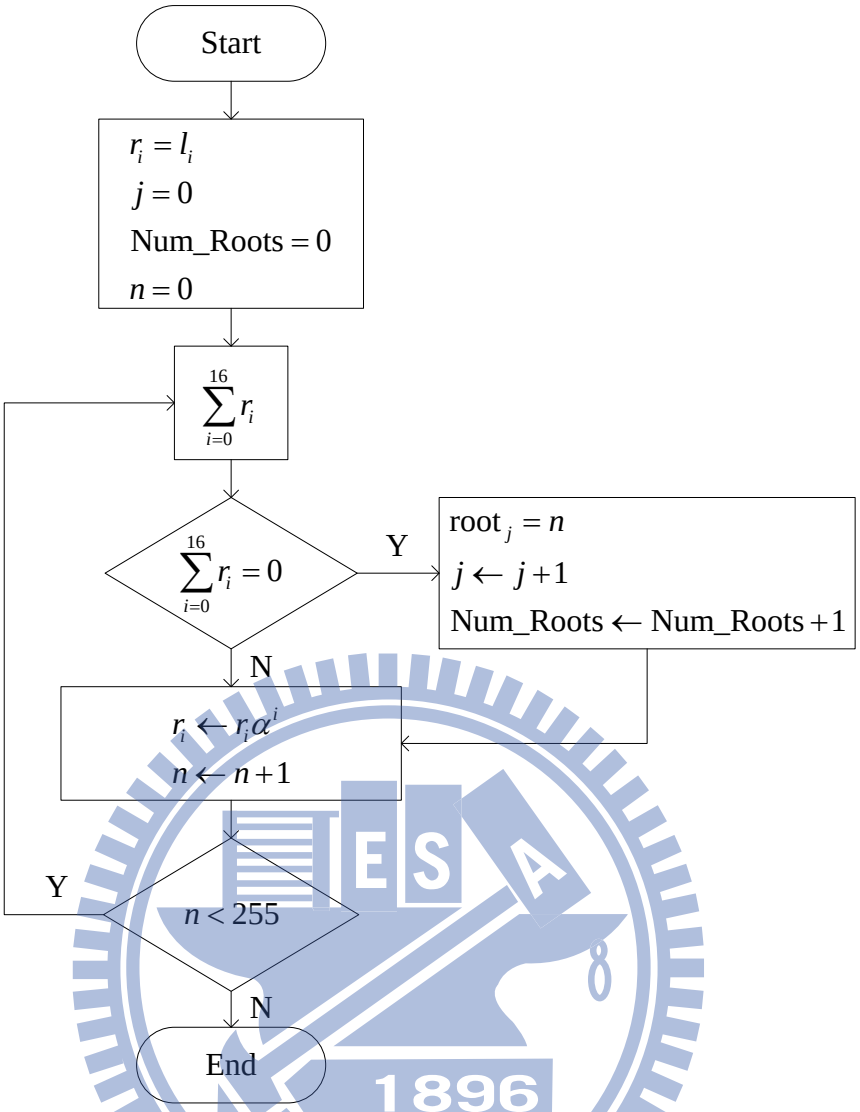

圖 4.14　　Chien search 電路運算流程圖

(一) 初始化，並將 l_i load 至暫存器 r_i。

(二) 藉由圖 4.13 之電路計算 $\sum_{i=0}^{16} r_i$。於第 $n=0$ 個循環時，即計算 $L(1)=\sum_{i=0}^{16} r_i$。

(三) 判別 $\sum_{i=0}^{16} r_i$ 是否為零，若為零，則將此元素之 power 儲存於 root_j 中。在此，因為於第 n 個循環時，電路即計算元素 α^n 是否為 $L(X)$ 的根，故 n 亦為此元素之 power。

(四) 累乘暫存器中的值 $r_i \cdot \alpha^i$，目的是要計算將下一個元素 α^{n+1} 代入 $L(X)$ 之結果，故回到(二)直到 $n=255$。

因此，最後我們可找出錯誤位置 root_j，其中 $j \le t$，以及錯誤個數 Num_Roots。

4.3.4 Forney電路

圖 4.15 為 Forney 演算法的電路流程圖，大致分為兩階段，第一階段要先找到關鍵方程式$\Omega(X)$，第二階段再將 Chien search 找到的v個根R_j一一代入多項式求得相對應之v個錯誤值，其同時分別先將R_j代入$\Omega(X)$與$\sigma'(X)$後再相除即可。

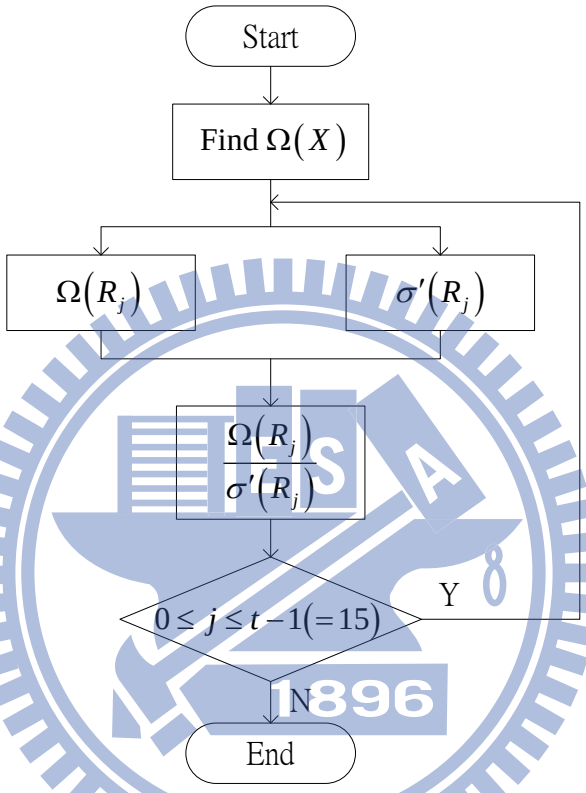

圖 4.15　　Forney 電路運算流程圖

首先，找出$\Omega(X)$。從(67)式與(68)式得知$\Omega(X)$的每一項係數為

$$\Omega^{(i)} = l_0 s_i + l_1 s_{i-1} + \cdots + l_i s_0 = \sum_{j=0}^{i} l_j s_{i-j} \tag{82}$$

分解後如下

$$\Omega_j^{(i)} = \begin{cases} l_0 s_i, & \text{for } j = 0 \\ \Omega_{j-1}^{(i)} + l_j s_{i-j}, & \text{for } 1 \le j \le i \end{cases} \tag{83}$$

$\Omega_j^{(i)}$為$\Omega^{(i)}$的第 j 次計算結果，故$\Omega(X)$每一項係數需$i+1(=15+1)$個循環時間才能計算完成。同理，我們可利用 Berlekamp-Massey 電路找出$\Omega(X)$的所有係數，如圖 4.16，其與求差異值的電路相同，此時，由於$L(X)$已滿足廣義牛頓恆等式的所有徵狀值，故無差異值。

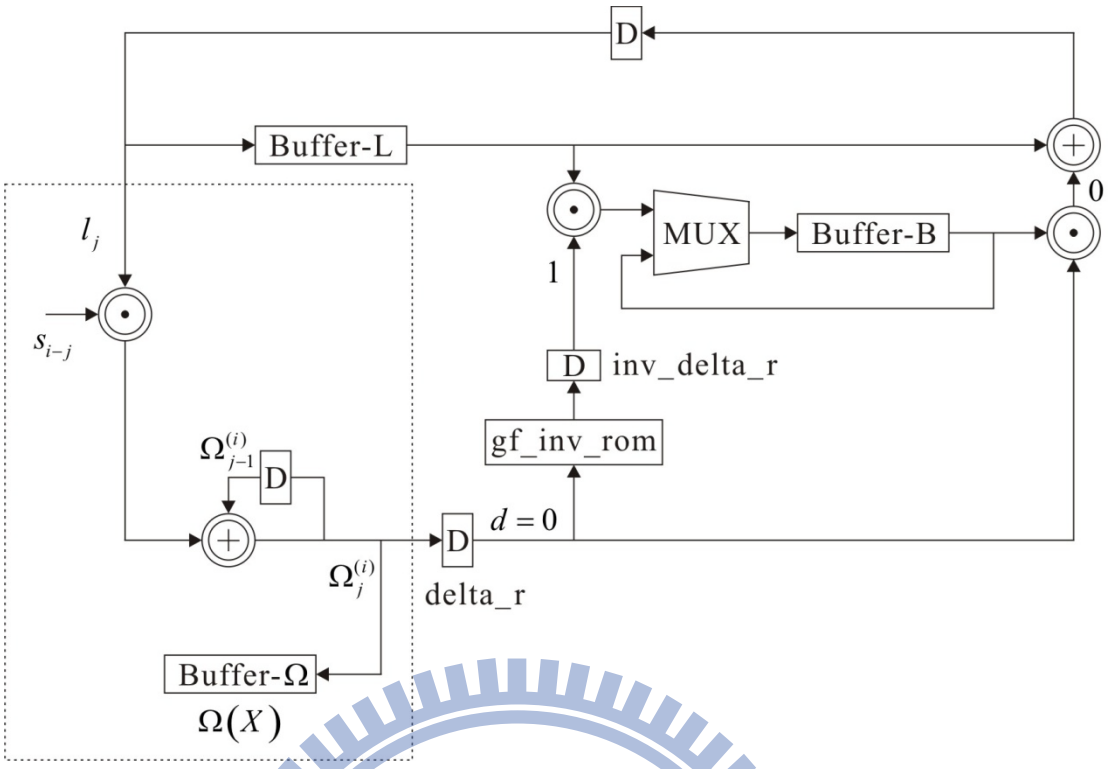

圖 4.16　　尋找關鍵方程式$\Omega(X)$電路

接下來是多項式的代值運算，我們一樣將(72)、(73)式拆解成$v(=16)$個循環來完成，數學分解式如下：

$$R_j^{(i)} = \begin{cases} 1, & \text{for } i = 0 \\ R_j^{(i-1)} \cdot R_j, & \text{for } 1 \le i \le v-1 \end{cases} \tag{84}$$

$$\Omega^{(i)}\left(R_j\right) = \begin{cases} \Omega_0, & \text{for } i = 0 \\ \Omega^{(i-1)}\left(R_j\right) + \Omega_i \cdot R_j^{(i)}, & \text{for } 1 \le i \le v-1 \end{cases} \tag{85}$$

$$\sigma'^{(i)}\left(R_j\right) = \begin{cases} l_0, & \text{for } i = 0 \\ \sigma'^{(i-1)}\left(R_j\right) + 0 \cdot R_j^{(i)}, & \text{for } 1 \le i \le v-1, i \in odd \\ \sigma'^{(i-1)}\left(R_j\right) + l_{i+1} \cdot R_j^{(i)}, & \text{for } 1 \le i \le v-1, i \in even \end{cases} \tag{86}$$

其中，$R_j^{(i)}$為於第i個循環時R_j的i次方(即R_j累乘i次)，$\Omega^{(i)}\left(R_j\right)$為$\Omega\left(R_j\right)$於第$i$個循環計算結果，$\sigma'^{(i)}\left(R_j\right)$為$\sigma'\left(R_j\right)$於第$i$個循環計算結果，其運算電路如圖 4.17 所示。然而，由於錯誤個數v未知，故我們計算都以最大值$t=16$來假設。

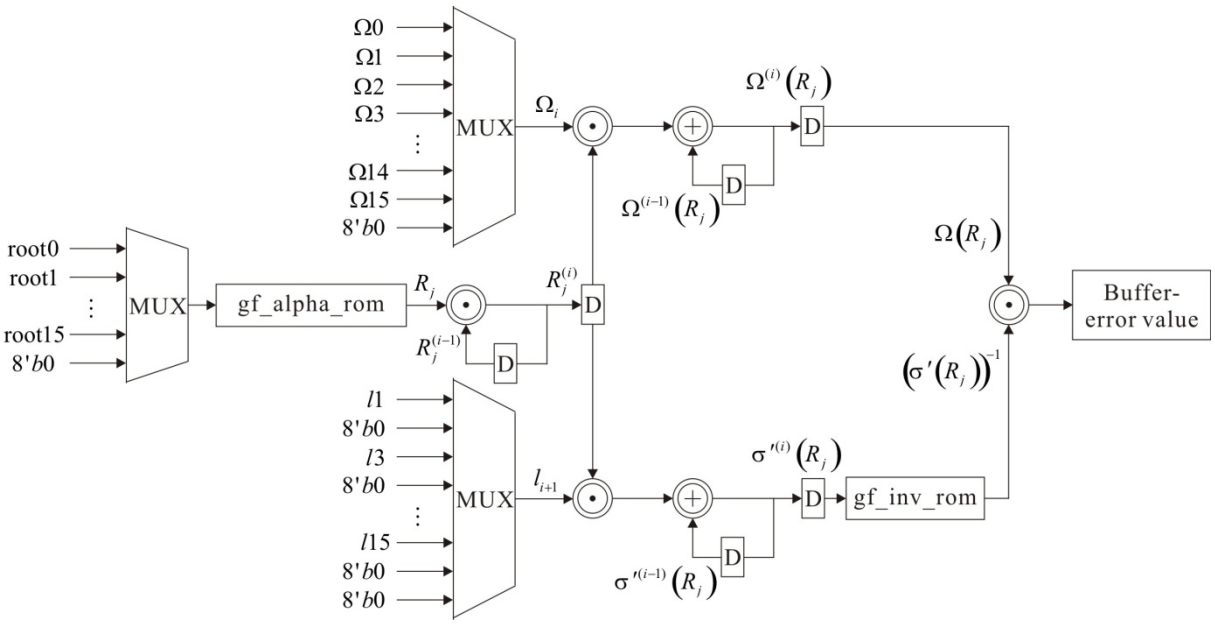

圖 4.17　　錯誤值運算電路

(一) 將 Chien search 找出之根 root_j 輸入電路，並藉由多工器來選擇要輸入的根，但由於 root_j 為元素的 power，因此我們需要透過查表才能得到其相對應元素的值 R_j，$R_j = \text{gf\_alpha\_rom}\left[\text{root}_j\right]$。

(二) 另外兩個多功器同時選擇 $\Omega(X)$ 和 $\sigma'(X)$ 的第 i 項係數輸入電路與 $R_j^{(i)}$ 相乘，其中 $\sigma'(X)$ 的奇數項係數為零，而偶數項係數為 l_{i+1}。

(三) 經過 t 個循環得到 $\Omega(R_j)$ 和 $\sigma'(R_j)$ 後，$\sigma'(R_j)$ 需藉由查表找到其相對應之反元素 $\left(\sigma'(R_j)\right)^{-1}$，$\left(\sigma'(R_j)\right)^{-1} = \text{gf\_inv\_rom}\left[\sigma'(R_j)\right]$，最後將它們彼此相乘即可找到錯誤位置 root_j 其相對應之錯誤值。

因此，此整個電路亦要循環 t 次，才能計算出 t 個錯誤值。

第5章　實驗模擬與結果分析

在前面的章節已介紹 RS 碼的演算流程及分析其運算電路架構，接下來我們將利用模擬軟體 ModelSim，來針對第四章中的 (255, 223, 16) RS 碼編解碼器之每個模組電路進行各自的模擬驗證。首先我們先使用 C 語言撰寫編解碼演算法並模擬，接著再透過 ModelSim 模擬所設計的 Verilog 硬體描述語言，比較模擬結果以完成 function 驗證的工作。

5.1　編解碼器之模擬驗證

我們測試的訊息訊號序列為 0, 1, …, 222，首先將其經過編碼器加以編碼，再於接收端將編碼後的碼字 207, 208, …, 222 訊號處手動加入錯誤(皆改成錯誤訊號 100) ，此即為我們測試的接收訊號，接著再依序模擬解碼端每個演算法模組，最後驗證解碼結果能將錯誤訊號更正回正確的訊號。

5.1.1　編碼器

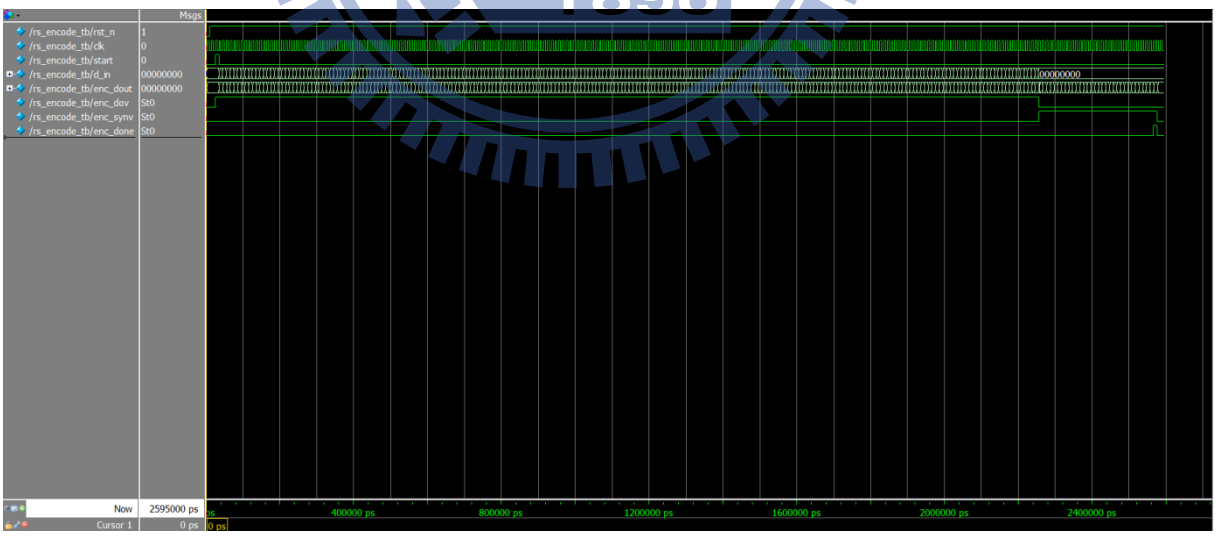

圖 5.1　　編碼器時序波形圖

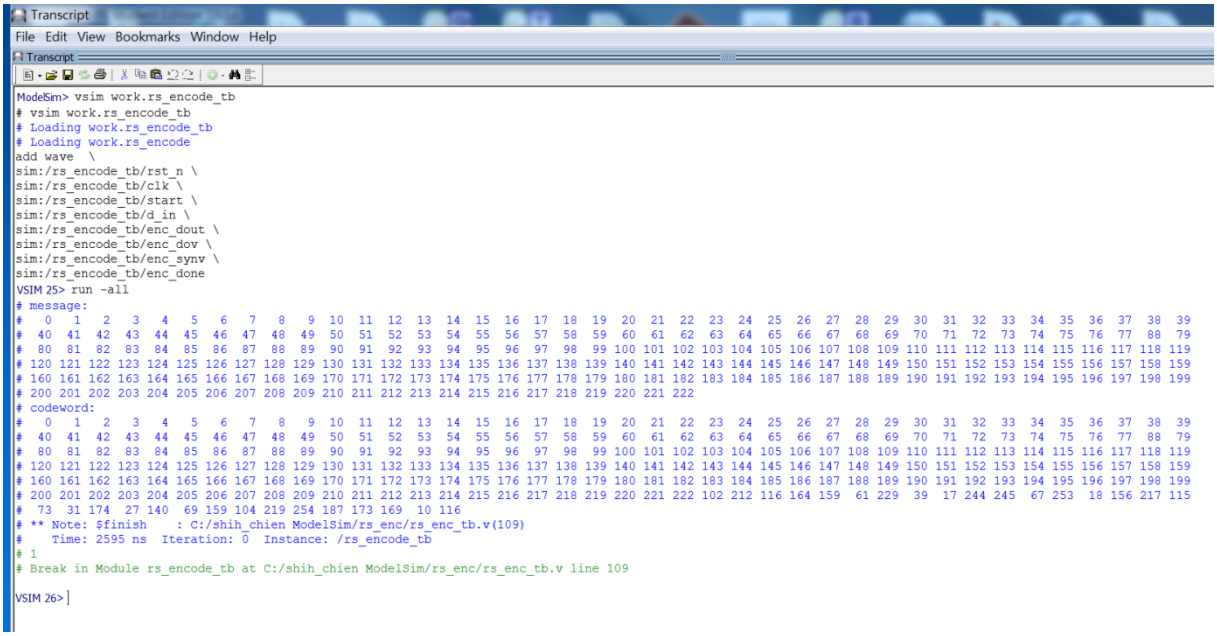

圖 5.2　　編碼器模擬結果

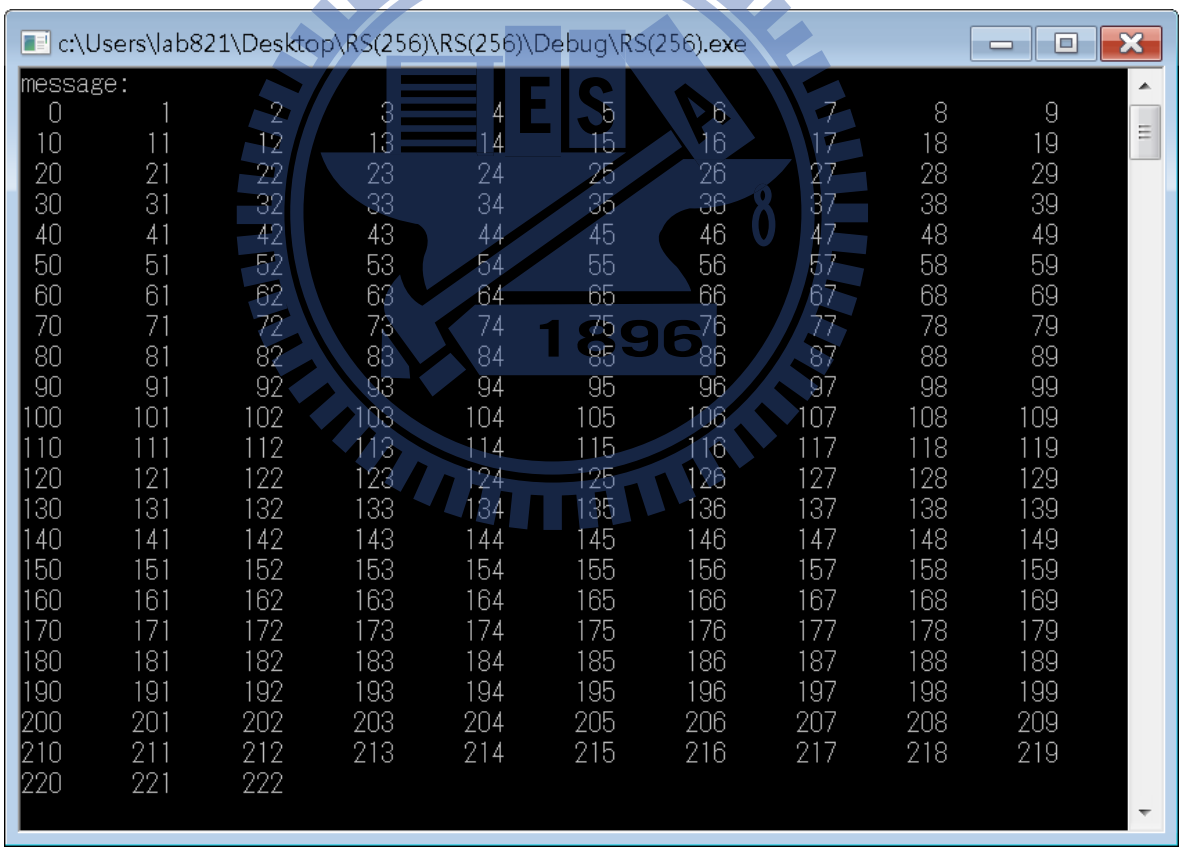

圖 5.3　　C 語言-編碼前之訊息訊號

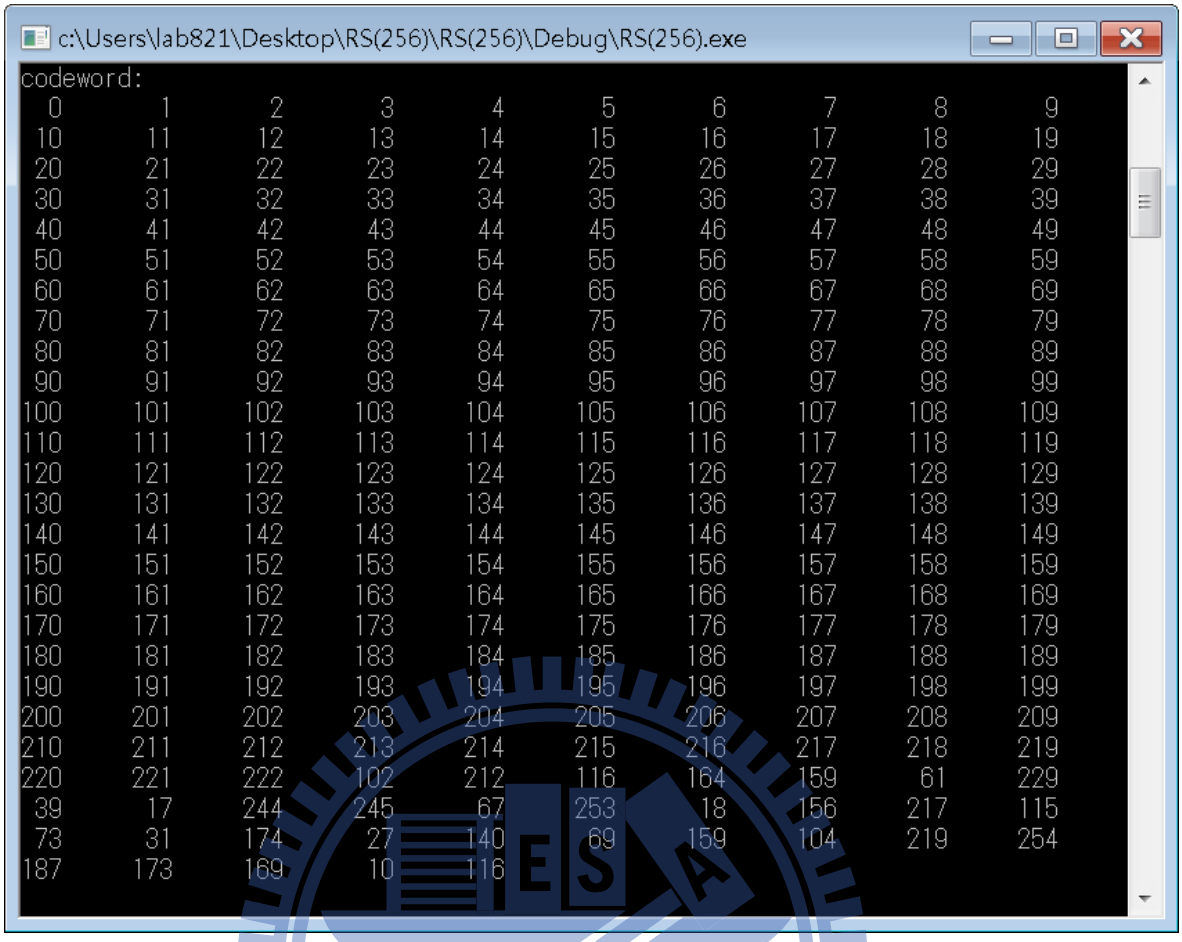

圖 5.4　　C 語言-編碼後之碼字

5.1.2　徵狀值運算器

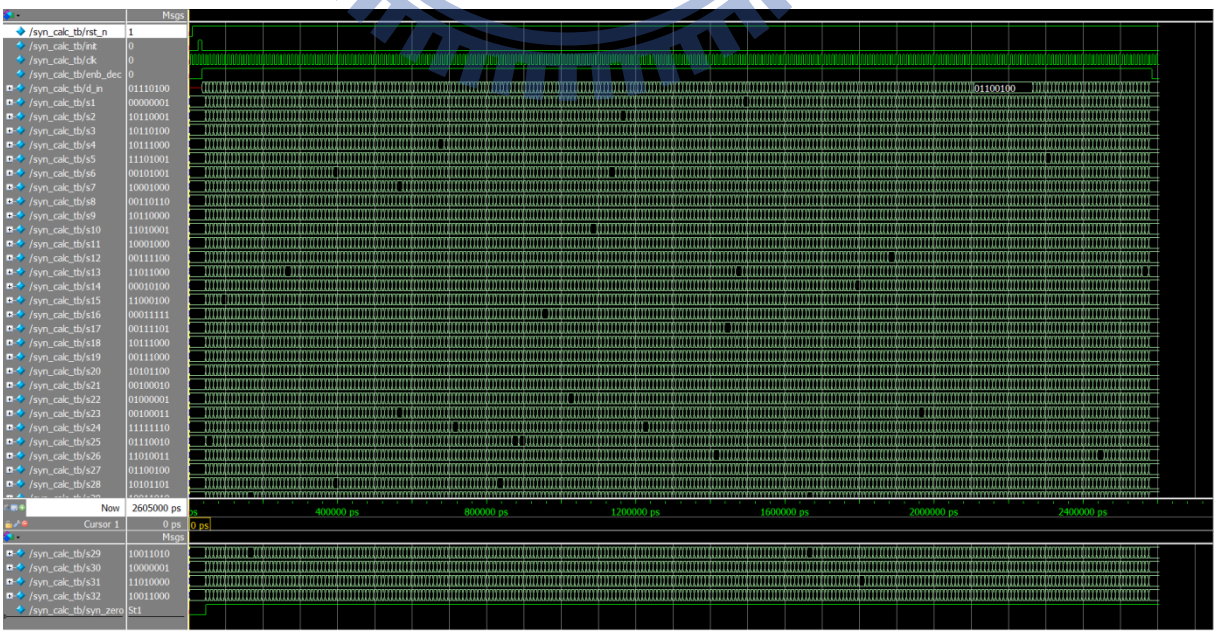

圖 5.5　　徵狀值運算器時序波形圖

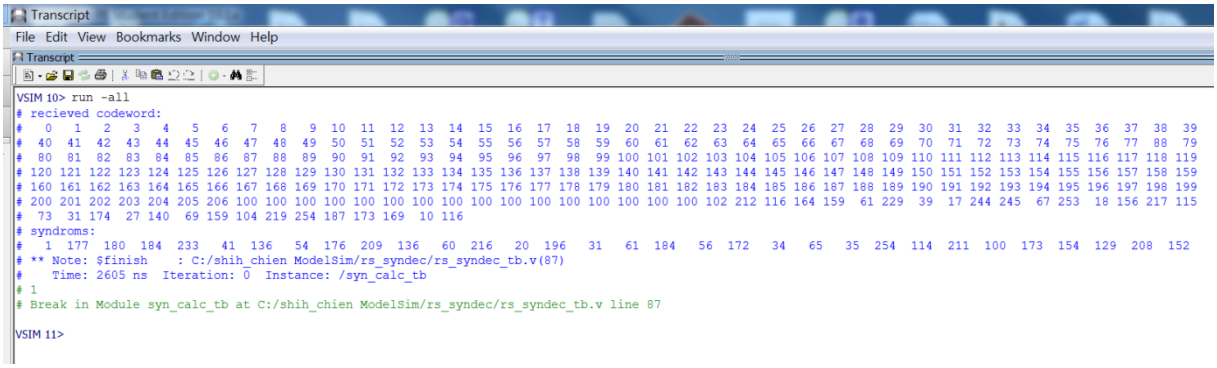

圖 5.6　　徵狀值運算器模擬結果

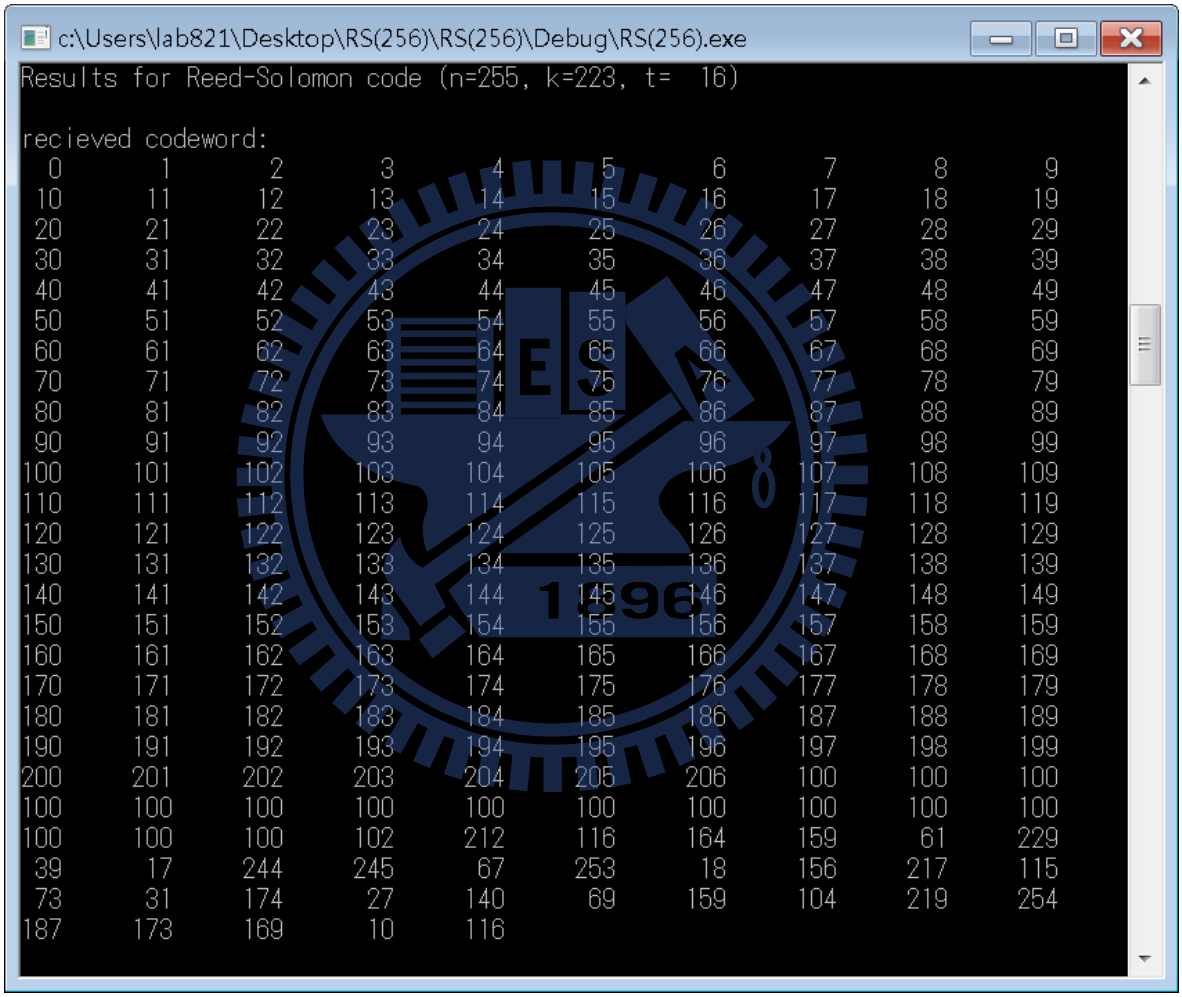

圖 5.7　　C 語言-接收訊號

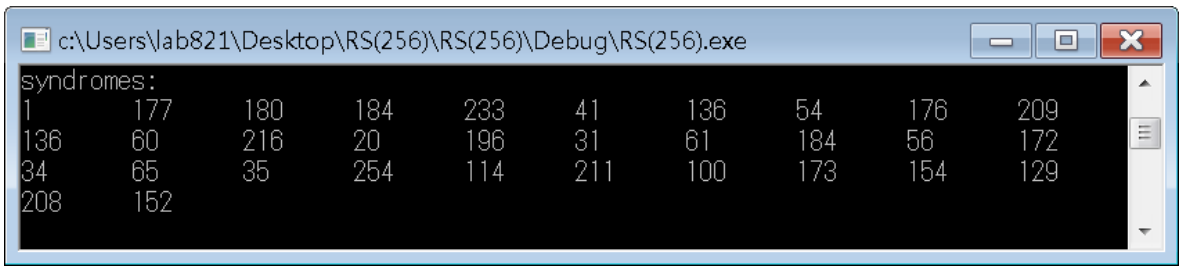

圖 5.8　　C 語言-徵狀值

5.1.3 Berlekamp-Massey錯誤位置多項式產生器

圖 5.9　　Berlekamp-Massey 錯誤位置多項式產生器時序波形圖

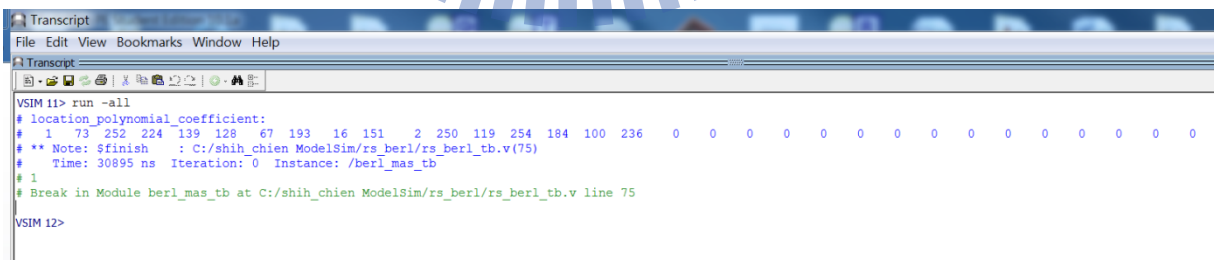

圖 5.10　　Berlekamp-Massey 錯誤位置多項式產生器模擬結果

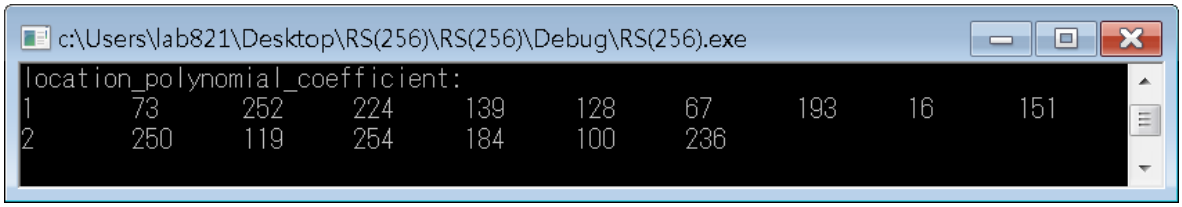

圖 5.11　　C 語言-錯誤位置多項式之係數

5.1.4 Chien search錯誤位置運算器

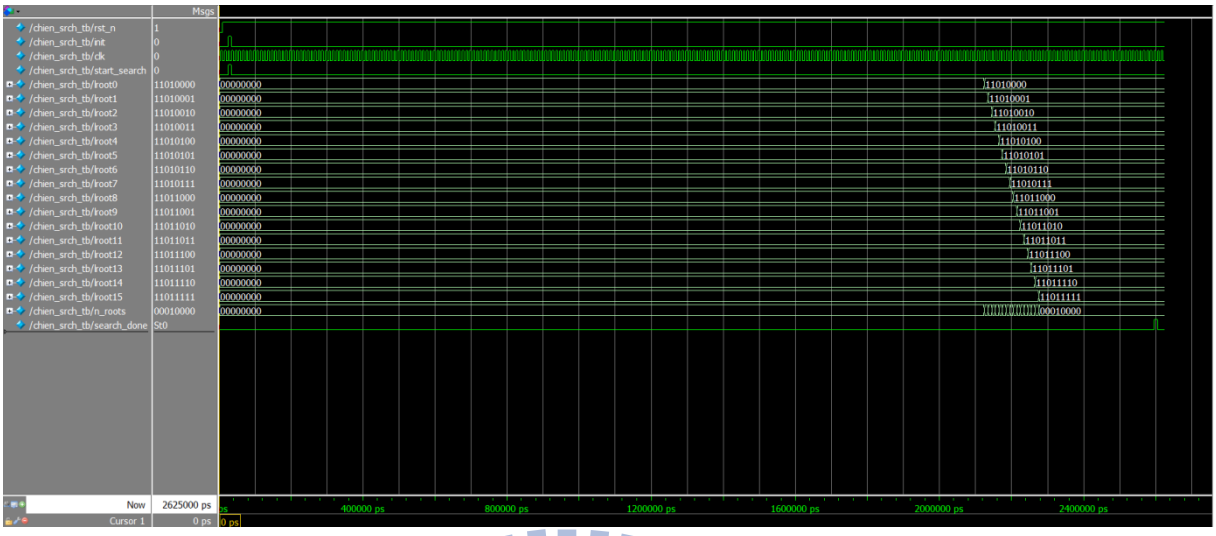

圖 5.12 Chien search 錯誤位置運算器時序波形圖

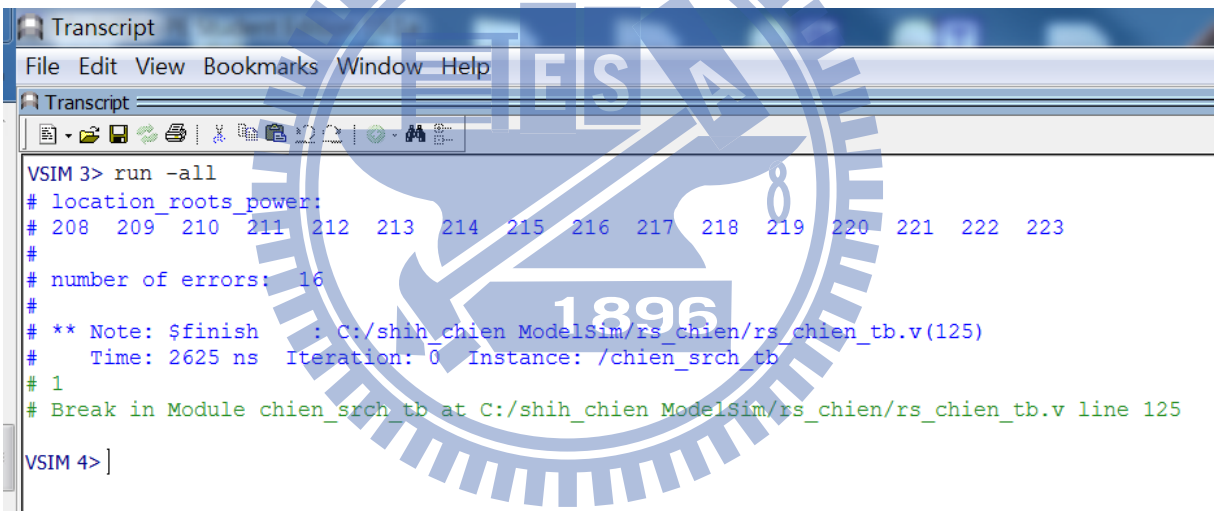

圖 5.13 Chien search 錯誤位置運算器模擬結果

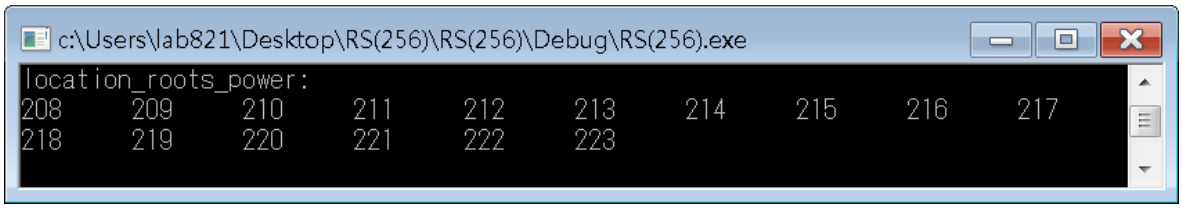

圖 5.14 C 語言-錯誤位置多項式之根的 power

5.1.5 Forney錯誤值運算器

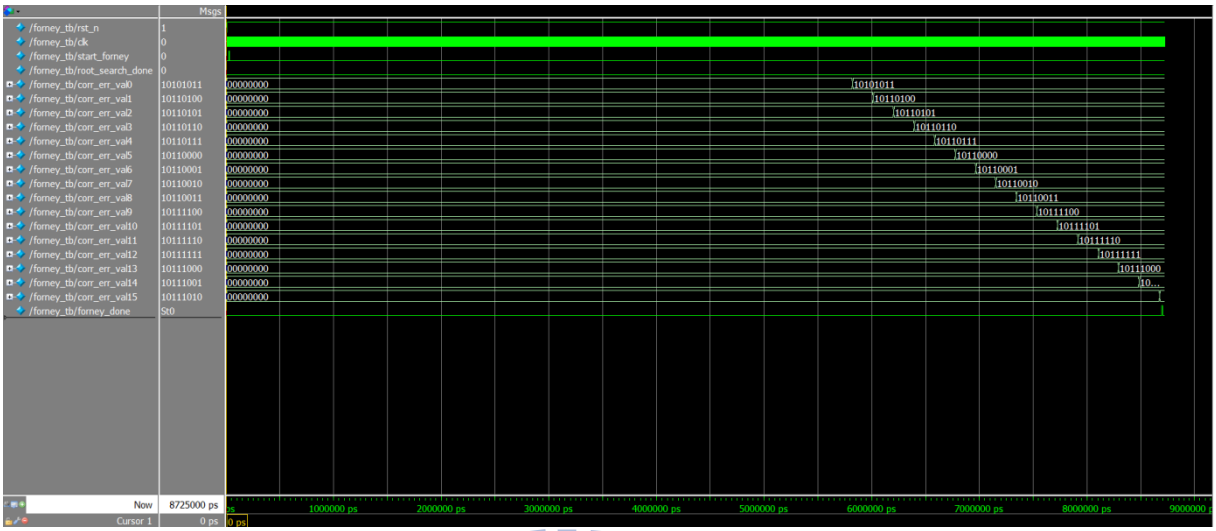

圖 5.15 Forney 錯誤值運算器時序波形圖

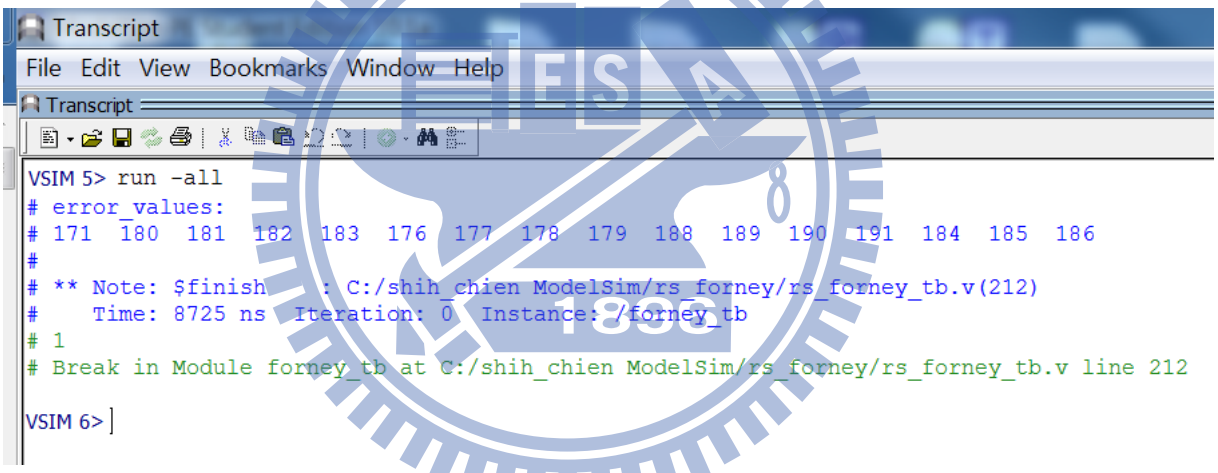

圖 5.16 Forney 錯誤值運算器模擬結果

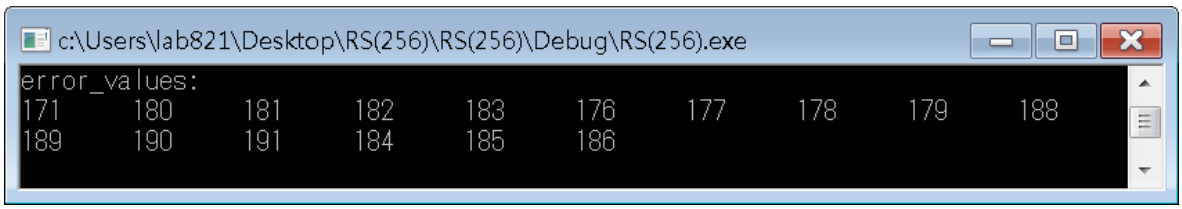

圖 5.17 C 語言-相對應於錯誤位置之錯誤值

5.2 結果分析

於進行模擬中，因為我們計算徵狀值時，訊號是由第一個 byte(=0)開始輸入電路運算，所以第一個 byte 是多項式的最高階，故在此解出的根其 power 就是錯誤位置，不需要再另外透過式子 $2^m - 1 - l$ 之運算。經由 ModelSim 模擬後，我們得到的錯誤位置及相對應錯誤值如表 5.1 所示。

錯誤位置	正確之原始訊號	錯誤值	錯誤值對應之二進位
208	207	171	10101011
209	208	180	10110100
210	209	181	10110101
211	210	182	10110110
212	211	183	10110111
213	212	176	10110000
214	213	177	10110001
215	214	178	10110010
216	215	179	10110011
217	216	188	10111100
218	217	189	10111101
219	218	190	10111110
220	219	191	10111111
221	220	184	10111000
222	221	185	10111001
223	222	186	10111010

表 5.1 模擬解碼結果

我們利用找到的錯誤位置與錯誤值進行錯誤更正的動作，而此動作即是將錯誤訊號加上找到的錯誤值(此加法為二進制 XOR 運算)。在此，錯誤訊號 100 對應之二進位為 01100100，我們以錯誤位置 208 為例，其錯誤值 171 對應之二進位為 10101011，更正結果如(87)式。依此類推，即可將所有錯誤訊號更正回來。

$$(01100100) \oplus (10101011) = 11001111 (= 207)_{10} \tag{87}$$

最後結果與 C 語言的結果一致，如圖 5.18 所示。

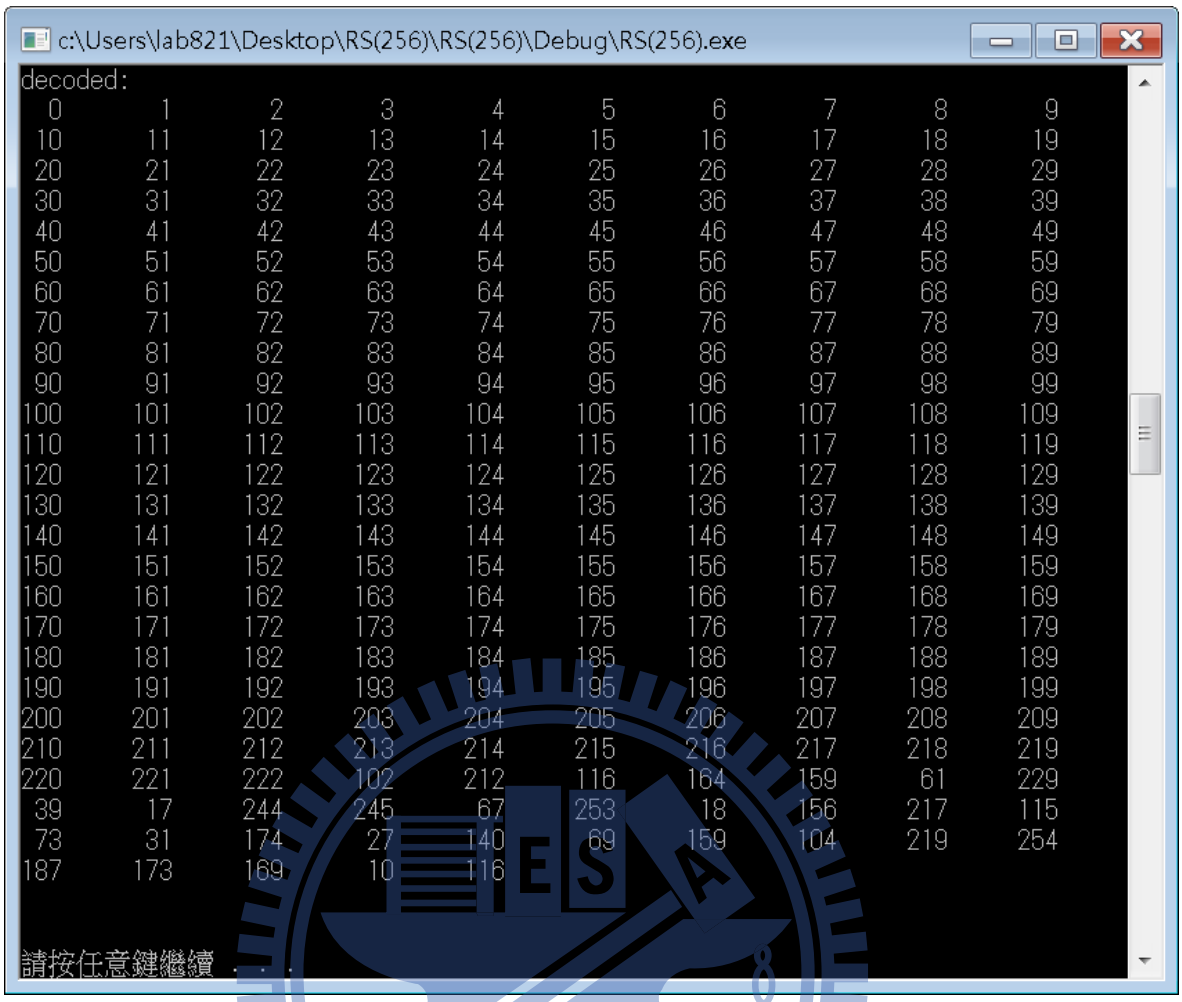

圖 5.18　C 語言-解碼後之訊號

在此統計每個模組進行模擬運算所需的時間如表 5.2，以 clock 為單位。

編碼器	徵狀值運算器	Berlekamp-Massey	Chien search	Forney
255	255	3086	258	868

表 5.2　模擬時間

由於多項式運算採序列解碼方式，故其中以包含許多多項式運算的 Berlekamp-Massey 以及 Forney 這兩個模組所需的時間花費較長。

第6章　結論與未來展望

本論文主要針對 RS 碼編解碼器之硬體電路運算進行分析，以 Verilog 硬體描述語言設計 (255, 223, 16) RS 碼的運算電路架構，並在 ModelSim 軟體下進行模擬。由於 RS 碼是基於有限場元素來運作，因此有限場運算器設計是電路架構的基本要素，當中以乘法器較為複雜，必須先透過數學推導得知其規則後才能進行下一步之設計，與一般通用乘法器不同，尤其是變數-常數乘法器更需依照每個不同的常數去設計，但此方式仍是系統評估後較有效率的選擇。另外我們反元素運算使用查表方式以避免有限場反向器的設計，雖然佔用了 256 bytes 記憶體資源，但卻可使電路複雜度大幅降低，相較之下，對於電路效益是提升的。解碼程序中，Berlekamp-Massey 演算法與 Forney 演算法皆包含多項式運算，若同時運算整個多項式，實現電路之複雜度與成本皆需付出較大的代價，然而，我們採用將多項式分解為序列解碼的方式，於每個時刻皆僅運算一個係數符號元 (1 byte)，如此一來，可達到大量簡化電路的效果；除此之外，徵狀值運算器與 Chien search 電路雖然皆屬於多項式求值運算，但兩者卻以不同的系統化規則來實現，其目的仍是為了減少變數-常數乘法器個數的使用，總歸而言，整體解碼仍以 Berlekamp-Massey 演算法電路最為複雜。由此可知，RS 碼的編解碼器設計首要工作必須先考慮參數，無論是本質多項式的不同抑或是生成多項式的改變皆會影響電路架構，因此我們需在了解系統所採用的 RS 碼規格後再進行硬體設計。本研究已對 RS 碼傳統的硬式決策解碼做詳細的運算分析與實作探討，瞭解基礎的運作以利未來的改善工作。

如同緒論所述，針對 RS 碼之解碼演算法後續已被學界廣泛分析與改良，但其中軟式決策解碼可能會面臨解碼器高複雜度的困難，故如何同時滿足低複雜度與低錯誤率的需求，截至目前仍為學者研究之議題。因此，我們未來目標為加入可更正抹除(erasure)訊號功能並朝向軟式決策解碼器的方向進行研究，以實作為考量，期許能夠實際應用在產業層面。

參考文獻

[1] I. S. Reed and G. Solomon, “Polynomial Codes over Certain Finite Fields,” *J. Soc. Ind. Appl. Math.*, vol. 8, pp. 300-304, June 1960.

[2] W. W. Peterson, “Encoding and Error-Correction Procedures for the Bose-Chaudhuri Codes,” *IRE Trans. Inform. Thoery*, vol. IT-6, pp. 459-470, Sept. 1960.

[3] D. Gorenstein and N. Zierler, “A Class of Cyclic Linear Error-Correcting Codes in p^m Symbols,” *J. Soc. Ind. Appl. Math.*, vol. 9, pp. 207-214, June 1961.

[4] R. Chien, “Cyclic Decoding Procedure for the Bose-Chaudhuri-Hocquenghem Codes,” *IEEE Trans. Inform. Theory*, vol. IT-10, no. 4, pp. 357-363, Oct. 1964.

[5] G. D. Forney, Jr., “On Decoding BCH Codes,” *IEEE Trans. Inform. Theory*, vol. IT-11, no. 4, pp. 549-557, Oct. 1965.

[6] E. R. Berlekamp, *Algebraic Coding Theory*, New York: McGraw-Hill, 1968.

[7] J. L. Massey, “Shift-Register Synthesis and BCH Decoding,” *IEEE Trans. Inform. Theory*, vol. IT-15, pp. 122-127, Jan. 1969.

[8] Y. Sugiyama, M. Kasahara, S. Hirasawa, and T. Namekawa, “A Method for Solving Key Equation for Decoding Goppa Codes,” *Inform. Control*, vol. 27, pp. 87-99, Jan. 1975.

[9] X. Youzhi, “Implementation of Berlekamp-Massey Algorithm without Inversion,” *IEE Proc. I Commun. Speech Vis.*, vol. 138, pp. 138-140, June 1991.

[10] I. S. Reed, M. T. Shih, and T. K. Truong, “VLSI Design of Inverse-Free Berlekamp-Massey Algorithm,” *Proc. Inst. Elect. Eng. pt. E*, vol. 138, pp. 295-298, Sept. 1991.

[11] H. C. Chang and C. B. Shung, “New Serial Architecture for the Berlekamp-Massey Algorithm,” *IEEE Trans. Commun.*, vol. 47, no. 4, pp. 481-483, Apr. 1999.

[12] R. Koetter and A. Vardy, “Algebraic Soft-Decision Decoding of Reed-Solomon Codes,” *IEEE Trans. Inform. Theory*, vol. 49, no. 11, pp. 2809-2825, Nov. 2003.

[13] J. Jiang and K. R. Narayanan, “Iterative Soft-Input Soft-Output Decoding of Reed-Solomon Codes by Adapting the Parity-Check Matrix,” *IEEE Trans. Inform. Theory*, vol. 52, no. 8, pp. 3746-3756, Aug. 2006.

[14] S. Lin and D. J. Costello, Jr., *Error Control Coding: Fundamentals and Applications*, 2nd ed. NJ: Pearson Education, 2004.

[15] J. G. Proakis and M. Salehi, *Digital Communications*, 5th ed. New York: McGraw- Hill, 2008.

[16] SourceForge.net, “Reed-Solomon Core Compiler - Project Web Hosting - Open Source Software,” [Online]. Available: http://rstk.sourceforge.net/

[17] J. Bhasker, *Verilog HDL Synthesis: A Practical Primer*, Star Galaxy Publishing, 1998.