



國立中山大學資訊工程學系

碩士論文

Department of Computer Science and Engineering

National Sun Yat-sen University

Master Thesis

基於隨機運算之數位訊號處理電路設計

Design of some DSP circuits based on stochastic computation

研究生：李晉維

Jin-Wei Li

指導教授：張雲南 博士

Dr. Yun-Nan Chang

中華民國 一百零三年 八月

August 2014



國立中山大學資訊工程學系

碩士論文

Department of Computer Science and Engineering

National Sun Yat-sen University

Master Thesis

基於隨機運算之數位訊號處理電路設計

Design of some DSP circuits based on stochastic computation

研究生：李晉維

Jin-Wei Li

指導教授：張雲南 博士

Dr. Yun-Nan Chang

中華民國 一百零三年 八月

August 2014

國立中山大學研究生學位論文審定書

本校資訊工程學系碩士班

研究生李晉維（學號：M013040082）所提論文

基於隨機運算之數位訊號處理電路設計

Design of some DSP circuits based on stochastic computation

於中華民國 103 年 8 月 27 日經本委員會審查並舉行口試，符合碩士學位論文標準。

學位考試委員簽章：

召集人 賴永康 賴永康

委員 張雲南 張雲南

委員 鄭獻榮 鄭獻榮

委員 陳春僥 陳春僥

委員 _____

委員 _____

指導教授(張雲南) 張雲南 (簽名)

摘要

隨機計算(Stochastic Computing, SC)由於其容錯率的特質，在近年來逐漸獲得關注。與傳統的二進制計算系統不同的是，隨機計算是以一串位元流來表示一個機率值。而其電路的設計可以使用低成本的邏輯閘組合來完成複雜的計算，藉以降低硬體的大小，節省成本。

然而，以往隨機計算的應用領域被限制在類神經網路、控制系統、影像處理還有低密度同位元檢查碼上。而本論文的目標是去探索更多可能的應用，故本論文嘗試實作了三種基於隨機計算的應用。第一種應用是繪製貝茲曲線電路的設計，由於貝茲曲線的計算公式可以使用 Bernstein polynomial 來表示，而 Bernstein polynomial 在隨機計算上可以使用簡單的邏輯閘組合來實現，所以貝茲曲線的繪製可以有效的利用隨機計算的特性來完成電路的設計。但是，我們的模擬結果顯示繪製的效果並不理想，儘管增加了位元流的長度，仍然無法接近理想上的曲線。而本論文也嘗試實作了基於線性內積模組的隨機離散餘弦轉換電路，然而使用在圖像處理上會有明顯的區塊化影響，造成圖像的失真。

最後，本論文提出基於隨機計算之去尾迴旋碼解碼器。其與傳統的二進制維特比(Viterbi)解碼器不同，不同的地方在於本論文是將去尾迴旋碼視為一種區塊碼，而其解碼器的設計是基於低密度同位元檢查碼的原理。由於變數節點(Variable Node)單元會因為同位元檢查矩陣的關係，導致硬體的電路過大。故本論文提出一個電路架構，藉由合併與篩選從檢查節點(Check Node)單元傳送給變數節點單元的資訊，藉以降低變數節點單元的輸入數，進而節省硬體的面積。而本論文是針對 LTE 之去尾迴旋碼標準而設計隨機解碼器，該標準之編碼長度為 40-bits 的標準所設計的解碼器。與傳統的維特比解碼器相比，本論文所提出的解碼器能節省超過 60% 的硬體面積。此外，本論文所提出的解碼器，解碼效能也比傳統的維特比解碼器要好。而我們的模擬結果也顯示，比較小的編碼長度，解碼效能還能比較長的編碼長度來的好。

關鍵詞：隨機計算、隨機解碼、低密度同位元檢查碼、去尾迴旋碼、貝茲曲線、
離散餘弦轉換



Abstract

Stochastic computing (SC) has recently gained attention due to its fault-tolerance property. Different from the ordinary binary computing, this unconventional approach represents numbers using the probability value of bit-streams. Very low-cost circuits that are highly resistant to manufacturing process variations and soft errors can thus be designed by stochastic logic. However, SC applications have been limited to the fields of neural networks, control systems, image processing, and low-density parity code (LDPC) applications. This thesis aims to explore more possible SC applications. In this thesis, three applications have been explored. The first application is the rendering of Bezier curve which can be expressed as a Bernstein polynomial and efficiently realized by SC. However, our simulation results show the quality of rendering result is not good, and can hardly approximate the ideal curve even with the increase of length of bit-stream used. Our thesis also considers the implementation of discrete cosine transform (DCT) based on the inner-product SC circuit. However, it will lead to obvious blocking effect if DCT is applied to image coding. Finally, this thesis proposed a novel implementation of SC decoder for tail-biting convolution code. Different from the ordinary Viterbi-based decoding approach, this thesis treated the tail-biting convolution code as a block code which can be further decoded based on the decoder architecture used for LDPC. The parity relationship of a convolution code may involve dozens of bits which may lead to very high implementation cost for variable node unit. This thesis proposed a simplified scheme by merging the check-to-variable messages in order to limit the number of variable-node input messages. Our implementation results show that for the tail-biting convolution code specified in LET (Long Term Evolution) standard with the block size of 40, more than 60% area can be saved for the proposed decoder

compared with the ordinary Viterbi decoder implementation. In addition, the bit-error-ratio (BER) of the proposed decoders is also better. Our simulation results also show that the BER achieved based on our decoder architecture will become better for smaller block sizes.

Keyword : stochastic computing, stochastic decoding, low-density parity-check (LDPC), tail-biting convolution, bezier curve, discrete cosine transform (DCT)

目錄

論文審定書.....	i
摘要.....	ii
Abstract.....	iv
目錄.....	vi
圖次.....	ix
表次.....	xii
第一章 概論.....	1
1.1 論文背景與動機.....	1
1.2 論文大綱.....	2
第二章 隨機計算背景與相關研究.....	3
2.1 隨機計算的數值表示方式.....	3
2.2 隨機計算的基本運算元件介紹.....	3
2.2.1 隨機乘法器.....	4
2.2.2 隨機加法器與隨機減法器.....	5
2.2.3 方程式合成.....	6
2.2.4 內積單元之隨機電路設計.....	7
2.2.5 互斥或閘與 J-K 正反器在隨機電路的應用.....	8
2.3 隨機亂數產生器設計原理.....	10
2.4 隨機計算的容錯率.....	12
2.5 隨機計算近年來相關應用.....	13
第三章 以隨機計算實現錯誤更正碼.....	16
3.1 低密度同位元檢查碼.....	16
3.1.1 Tanner Graph.....	16
3.1.2 和積演算法.....	18

3.2 使用隨機計算實現和積演算法	22
3.2.1 初始化	22
3.2.2 檢查節點運算.....	22
3.2.3 變數節點運算.....	23
3.2.4 字碼的硬判定(Hard Decision).....	25
3.3 變數節點在隨機計算下的栓鎖問題與解決方法	26
3.3.1 雜訊相依性調整(Noise-Dependent Scaling , NDS)	27
3.3.2 邊緣記憶體(Edge Memory , EM)	28
3.3.3 循跡預測記憶體(Tracking Forecast Memory , TFM).....	32
3.3.4 基於多數決機制的循跡預測記憶體(Majority- Based Tracking Forecast Memories , MTFM)	33
3.3.5 基於多數決機制的邊緣記憶體(Majority Edge Memory , MEM)..	37
3.4 去尾迴旋碼介紹	39
3.4.1 去尾迴旋碼編碼方法	40
3.4.2 產生去尾迴旋碼之生成矩陣.....	41
3.4.3 產生去尾迴旋碼之同位元檢查矩陣	42
第四章 基於隨機計算之影像處理應用實作.....	44
4.1 貝茲曲線.....	44
4.2 離散餘弦轉換(DCT).....	45
第五章 基於隨機計算之去尾迴旋碼解碼器實作.....	50
5.1 生成矩陣與同位元檢查矩陣產生	50
5.2 硬體架構.....	51
5.2.1 通道機率隨機亂數產生器(Channel Probability Stochastic Number Generator , CPSNG)	54
5.2.2 亂數產生引擎(Random Number Engine, RNE).....	54
5.2.3 檢查節點(Check Node , CN)	55

5.2.4 變數節點(Variable Node, VN)	56
5.2.5 字碼硬判定(Hard Decision , HD)	60
5.2.6 提早結束機制(Early Termination , ET).....	61
5.2.7 控制器(Controller).....	62
5.3 軟體驗證與 FPGA 驗證.....	62
5.3.1 軟體驗證	62
5.3.2 FPGA 驗證	63
第六章 實驗結果與效能比較.....	66
6.1 貝茲曲線實驗結果.....	66
6.2 離散餘弦轉換(DCT)實驗結果.....	67
6.3 基於隨機計算之去尾迴旋碼解碼器實驗結果與效能比較	68
6.3.1 解碼效能分析與硬體面積比較.....	68
6.3.2 去尾迴旋碼不同截斷長度之效能比較	71
第七章 結論與未來展望	73
7.1 結論.....	73
7.2 未來展望	73
參考文獻	74
附錄 A 固定取 18 條輸入的變數節點與檢查節點對應表格.....	77
附錄 B 40-bits 去尾迴旋碼解碼器之同位元檢查矩陣	81

圖次

圖 2-1	AND 閘的隨機無號數乘法運算	4
圖 2-2	XNOR 閘的隨機有號數乘法運算	4
圖 2-3	隨機加法器	5
圖 2-4	隨機減法器	6
圖 2-5	隨機 Bernstein 多項式電路架構	7
圖 2-6	傳統隨機內積模組	7
圖 2-7	總和加權內積模組	8
圖 2-8	以互斥或閘實現檢查節點運算	8
圖 2-9	J-K 複合電路	9
圖 2-10	J-K 正反器的複合電路狀態所構成的馬可夫鏈	9
圖 2-11	數值轉換電路	10
圖 2-12	數值轉換電路範例	11
圖 2-13	16 位元線性反饋移位暫存器	12
圖 2-14	二進制計算與隨機計算容錯率比較	13
圖 2-15	隨機與二進制邊緣偵測電路	15
圖 3-1	同位元檢查矩陣範例	17
圖 3-2	Tanner Graph 範例圖	17
圖 3-3	通訊系統資訊傳遞圖	18
圖 3-4	檢查節點電路傳輸路徑	20
圖 3-5	變數節點電路傳輸路徑(q_i, j)	21
圖 3-6	變數節點電路傳輸路徑(Q_i)	21
圖 3-7	檢查節點資訊傳送範例	23
圖 3-8	以互斥或閘實現檢查節點運算範例	23
圖 3-9	變數節點資訊傳送範例	24

圖 3-10	J-K 複合電路使用隨機計算實現範例	25
圖 3-11	J-K 複合電路的輸出狀態	27
圖 3-12	栓鎖問題範例	27
圖 3-13	兩輸入的 EM 變數節點電路	29
圖 3-14	三個輸入 EM 變數節點電路	30
圖 3-15	六個輸入 EM 變數節點電路	30
圖 3-16	區塊化兩輸入變數節點的隨機運算架構	30
圖 3-17	三輸入 EM 變數節點簡化圖	31
圖 3-18	六輸入 EM 變數節點簡化圖	31
圖 3-19	EM(TFM)在 Tanner Graph 的位置圖	31
圖 3-20	使用 TFM 的兩輸入變數節點	33
圖 3-21	MTFM 在 Tanner Graph 的位置圖	34
圖 3-22	6 級 MTFM 變數節點的輸入輸出方向圖	35
圖 3-23	六級 MTFM 變數節點電路圖	36
圖 3-24	MTFM 內部架構圖	36
圖 3-25	MEM 在 Tanner Graph 上的位置	37
圖 3-26	6 級 MEM 變數節點的輸出輸入圖	38
圖 3-27	6 級 MEM 變數節點電路圖	38
圖 3-28	MEM 內部架構圖	39
圖 3-29	(2,1,2)迴旋編碼器	40
圖 3-30	去尾迴旋碼初始狀態	40
圖 3-31	輸入 d1 進行編碼	41
圖 3-32	輸入 d2 進行編碼	42
圖 4-1	貝茲曲線的隨機電路(方程式合成)	44
圖 4-2	貝茲曲線的隨機電路(基本邏輯閘組合)	45
圖 4-3	JPEG 編碼步驟	46

圖 4-4	1-D DCT 子電路.....	47
圖 4-5	1-D DCT	48
圖 4-6	DCT 轉換流程圖	49
圖 4-7	FDCT 轉回圖片的步驟	49
圖 5-1	40-bit 迴旋編碼器初始狀態	50
圖 5-2	基於隨機計算之去尾迴旋碼解碼器硬體架構圖	52
圖 5-3	通道機率隨機亂數產生區塊(CPSNG BLOCK)內部架構圖...	52
圖 5-4	檢查節點區塊(CN BLOCK)內部架構圖	53
圖 5-5	變數節點區塊(VN BLOCK)內部架構圖.....	53
圖 5-6	硬判定區塊(HD BLOCK)內部架構圖	53
圖 5-7	通道機率隨機亂數產生器的方塊圖	54
圖 5-8	亂數產生引擎方塊圖	55
圖 5-9	多輸入檢查節點的隨機運算電路	56
圖 5-10	固定取 6 條資訊的變數節點運算.....	58
圖 5-11	取多條資訊的變數節點運算.....	58
圖 5-12	固定取 18 條資訊的變數節點運算	58
圖 5-13	6 級輸入 MEM 變數節點電路圖	60
圖 5-14	上下數飽和計數器	61
圖 5-15	提早結束機制範例.....	62
圖 5-16	軟體模擬通訊系統中的編解碼流程.....	63
圖 5-17	以 Slave 形式包裝的隨機解碼器方塊圖	64
圖 6-1	貝茲曲線模擬結果	66
圖 6-2	DCT 轉換模擬結果	67
圖 6-3	不同迭代次數對於解碼效能的影響	69
圖 6-4	Viterbi 與隨機 LDPC 解碼效能比較.....	71

表次

表 2-1	互斥或閘真值表	9
表 2-2	位元數與反饋方程式對應表格	12
表 2-3	邏輯閘在相關性的功能	14
表 3-1	和積演算法參數定義	19
表 5-1	附錄 B 範例	51
表 5-2	架構圖中訊號線的意義	51
表 5-3	檢查節點輸入個數分布	56
表 5-4	變數節點輸入數分布	57
表 5-5	固定取 18 條輸入的變數節點範例	60
表 6-1	傳統二進制 DCT 電路與隨機計算 DCT 電路硬體大小比較 ..	68
表 6-2	有提早結束機制的解碼器平均解碼週期.....	70
表 6-3	去尾迴旋解碼器與隨機 LDPC 解碼器硬體大小比較	70

第一章 概論

1.1 論文背景與動機

隨機計算[1](Stochastic computation, SC)是在 1960 年代提出的一種計算方式，有別於傳統的二進制表示法，隨機計算是採用位元流(bit-stream)的方式來表示一個機率的數值。舉例來說，假設有一串長度為 10 的位元流為 "1 0 0 0 1 1 0 0 0 0"，其中"1"的數目有 3 個，所以這串位元流的機率為 0.3。基於以上特性，隨機計算可以使用簡單的邏輯閘來實現加減法與乘法的運算，或是使用有限狀態機(Finite State Machine, FSM)來實現較複雜的運算，例如：指數運算、絕對值運算...等的應用。由此可知，隨機計算可以以較低的成本的元件來實現較複雜的運算。

隨機計算早年的使用一直侷限在神經網路[12]和控制系統的領域。但近年來，由於隨機計算的容錯能力與電路設計簡單....等的優點下，而被廣泛的應用在各個領域之中。例如影像處理[10],[14]、錯誤更正碼[2]-[9],[15]與數位信號處理[11],[17]的應用上。而本論文則是依據上述的各種研究中，嘗試找出更多的實際應用。例如文獻[11]所提出的加權內積模組，原本是使用在有限脈衝響應濾波器(Finite Impulse Response Filter, FIR filter)與無限脈衝響應濾波器(Infinite impulse Response Filter, IIR Filter)上。而本論文則嘗試使用加權內積模組實線離散餘弦轉換(Discrete Cosine Transform, DCT)電路。

另外，近年來的研究顯示，隨機計算使用在錯誤更正碼上能獲得不錯的效能，而較常被使用的錯誤更正碼為低密度同位元檢查碼(Low Density Parity Check, LDPC)。所以本論文則依據 LDPC 解碼器的設計原理，設計出了能解碼由去尾迴旋碼(Tail-biting Convolution Code)所編碼的資訊。

故本論文嘗試實作出三種應用電路，第一種為隨機貝茲曲線電路；第二種為隨機離散餘弦轉換電路；第三種為基於隨機計算之去尾迴旋碼解碼器。其詳細的設計方法將在以下章節說明。

1.2 論文大綱

本論文章節詳細劃分如下，第一章介紹論文背景與論文大綱；第二章介紹隨機計算的背景與基本隨機計算元件。內容包含隨機計算數值表示的方式、隨機計算的基本計算元件、隨機計算所需的亂數產生器設計、隨機計算的容錯性介紹與近年來隨機計算相關的應用；第三章介紹使用在隨機計算上的錯誤更正碼應用。內容包含使用隨機計算實現的低密度同位元檢查碼及其演算法介紹，與去尾迴旋碼的編碼方法說明；第四章為介紹基於隨機計算的貝茲曲線電路與離散餘弦轉換電路設計方法；第五章為隨機去尾迴旋碼解碼器電路設計方法；第六章為實驗結果與效能比較；第七章為結論與未來展望。

第二章 隨機計算背景與相關研究

本章節首先介紹隨機計算的數值表示方式，以及如何使用隨機計算來實現乘法器、加法器等的基本元件介紹。再來介紹使用隨機亂數產生器產生隨機位元流的方法，最後介紹隨機計算的容錯率，與相關性位元流[13],[14]的應用。

2.1 隨機計算的數值表示方式

由於隨機計算是以位元流(bit-stream)的方式來表示一個機率的數值，所以每串位元流可以表示的機率範圍為 0~1 之間。因此，若要表示負數，就必須將機率與數值的關係重新對應。

在無號數的表示方式中， x 為一串位元流(即 X)中 1 出現的機率，故其可表示範圍為 $[0,1]$ ，如式(1)。

$$x = P(X = 1), x \in [0,1] \quad (1)$$

而在有號數的表示方式中，可表示範圍為 $[-1,1]$ ，須由 $[0,1]$ 擴增至 $[-1,1]$ 如式(2)。

$$x = 2P(X = 1) - 1, x \in [-1,1] \quad (2)$$

2.2 隨機計算的基本運算元件介紹

本章節將詳細介紹使用在隨機計算系統下的基本元件。包含隨機乘法器、隨機加法器與隨機減法器、方程式合成、內積單元隨機之電路設計與錯誤更正碼之隨機電路設計。

2.2.1 隨機乘法器

隨機計算的乘法器是使用一個簡單的邏輯閘 AND 或是 XNOR 來實現無號數 (AND) 或是有號數 (XNOR) 的乘法。如圖 2-1 所示，假設有一個 AND 邏輯閘，有兩個輸入位元流 A、B，一個輸出位元流 Y。假設 $P_{(A=1)} = 4/8$ ， $P_{(B=1)} = 6/8$ ，則 $P_{(Y=1)} = P_{(A=1)} \times P_{(B=1)} = 3/8$ 。

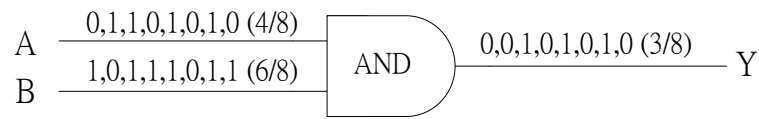


圖2-1 AND 閘的隨機無號數乘法運算

而上述乘法器，為無號數乘法器，若要實現有號數乘法器，就需要使用邏輯閘 XNOR 來實現。如圖 2-2 所示。其原理推導如式(3)所描述，其中 A、B 為輸入有號數隨機位元流，Y 為輸出有號數隨機位元流。

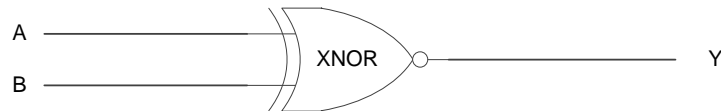


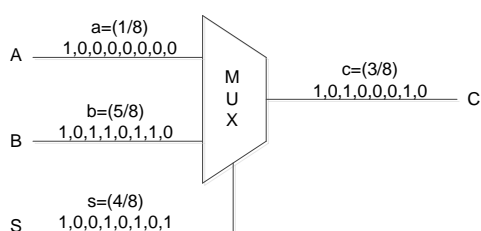
圖2-2 XNOR 閘的隨機有號數乘法運算

$$\begin{aligned}
 P_{(Y=1)} &= P_{(A=1 \& B=1)} + P_{(A=0 \& B=0)} = P_{(A=1)} \cdot P_{(B=1)} + P_{(A=0)} \cdot P_{(B=0)} \\
 &= P_{(A=1)} \cdot P_{(B=1)} + \overline{P_{(A=1)}} \cdot \overline{P_{(B=1)}}, \text{ where } \bar{P} = 1 - P \\
 &\because P_{(A=1)} = \frac{A+1}{2}, P_{(B=1)} = \frac{B+1}{2} \\
 \therefore P_{(Y=1)} &= \frac{[(A+1) \cdot (B+1) + (1-A) \cdot (1-B)]}{4} = \frac{(A \cdot B + 1)}{2} \\
 \therefore Y &= 2 \cdot P_{(Y=1)} - 1 = A \cdot B
 \end{aligned} \tag{3}$$

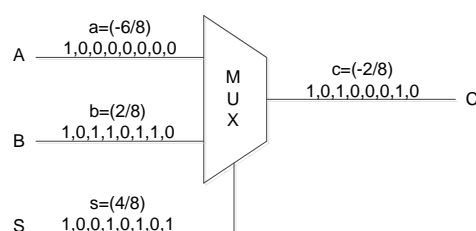
2.2.2 隨機加法器與隨機減法器

隨機運算的加法器是使用一個多工器(MUX)來實現有號數與無號數的加法。假設有一個多工器，有兩個輸入 A、B，一個輸出 C，一條選擇線 S，且 A、B、C、S 皆為位元流，而 $a = P_{(A=1)}$ 、 $b = P_{(B=1)}$ 、 $s = P_{(S=1)}$ 、 $c = P_{(C=1)}$ 。如圖 2-3(A) 為一無號數加法器，假設輸入 $a = 1/8$ ， $b = 5/8$ ， $s = 4/8$ ，則 c 使用式(4)的方法計算，由此可知 $c = 4/8 \times 1/8 + (1 - 4/8) \times 5/8 = 3/8$ 。

$$\begin{aligned} c = P_{(C=1)} &= P_{(S=1 \& A=1)} + P_{(S=0 \& B=1)} = P_{(S=1)} \cdot P_{(A=1)} + (1 - P_{(S=1)}) \cdot P_{(B=1)} \\ &= s \cdot a + (1 - s) \cdot b \end{aligned} \quad (4)$$



(A) 無號數加法器



(B) 有號數加法器

圖2-3 隨機加法器

圖 2-3(B)為一有號數加法器，其中 A、B 與 C 是使用有號數編碼，而 S 是使用無號數編碼。所以 $a = 2P_{(A=1)} - 1$ ， $b = 2P_{(B=1)} - 1$ ， $c = 2P_{(C=1)} - 1$ ， $s = P_{(S=1)}$ 基於以上編碼方式，c 仍然等於 $s \cdot a + (1 - s) \cdot b$ ，其證明如式(5)所示。

$$\begin{aligned} P_{(C=1)} &= P_{(S=1)} \cdot P_{(A=1)} + (1 - P_{(S=1)}) \cdot P_{(B=1)} \\ \frac{c+1}{2} &= s \cdot \frac{a+1}{2} + (1-s) \cdot \frac{b+1}{2} \\ c &= s \cdot a + (1-s) \cdot b \end{aligned} \quad (5)$$

再來是隨機減法器，其使用一組多工器加上一個反閘(NOT)，如圖 2-4 所示。因為數值相減會有負數的產生，而無號數無法表示負數，所以減法器只能使用在有號數的編碼系統上。其中 A，B，C 是使用有號數編碼，而 S 是使用無號數編碼，式(6)為其證明過程。

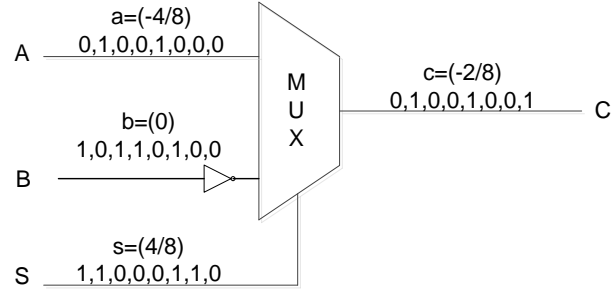


圖2-4 隨機減法器

$$P_{(C=1)} = P_{(S=1)} \cdot P_{(A=1)} + P_{(S=0)} \cdot P_{(B=0)}$$

$$\frac{c+1}{2} = s \cdot \frac{a+1}{2} + (1-s) \cdot \frac{1-b}{2} \quad (6)$$

$$c = s \cdot a - (1-s) \cdot b$$

2.2.3 方程式合成

本章節介紹如何使用隨機計算的原理實現 Bernstein 多項式[1]的電路，而 Bernstein 多項式如式(7)，其中 n 為階數，而 b_i 為多項式的係數。

$$B_n(t) = \sum_{i=0}^n b_{i,n} \cdot B_{i,n}(t) \quad (7)$$

式(7)中的 $B_{i,n}(x)$ 是 Bernstein 多項式的基底多項式，由式(8)得到。

$$B_{i,n}(t) = \binom{n}{i} t^i (1-t)^{n-i} \quad (8)$$

使用隨機計算原理所設計出來的 Bernstein 多項式，其設計電路如圖 2-5 所示。圖中，”+”區塊有 n 個輸入埠(X_1, \dots, X_n)，及 $\lceil \log_2(n+1) \rceil$ 個輸出埠，此區塊是計算 X_1, \dots, X_n 中 1 出現的數目，其目的是要模擬式(8)。而”MUX”區塊，有 n 個輸入埠(Z_1, \dots, Z_n)，而 Z_i 代表係數 $b_{i,n}$ ，且輸出結果為 Bernstein 多項式，如式(7)。

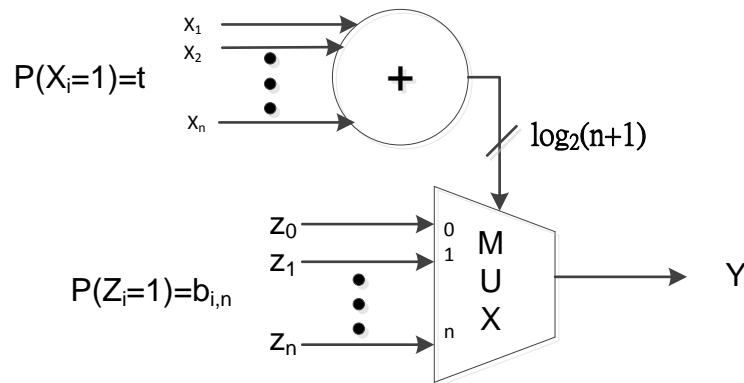


圖2-5 隨機 Bernstein 多項式電路架構

2.2.4 內積單元之隨機電路設計

內積單元是數位信號處理較常使用的元件。而內積單元的設計相當簡單，主要是由多個乘法器及加法器元件所組成。

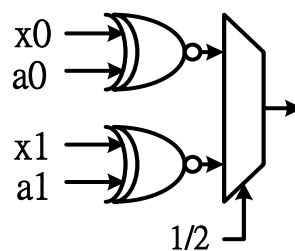


圖2-6 傳統隨機內積模組

由於隨機計算是以機率的形式來表示，故數值不能大於 1。因此，圖 2-6 的隨機加法器有一個係數 $1/2$ ，此因子是為了將兩個輸入相加後的和降到 1 以下。如圖 2-6 所示，內積後的結果為 $1/2(a_0x_0 + a_1x_1)$ 。

此外，由於內積模組會有一個 $1/2$ 的係數，在經過多個內積或是加減法運算之後，數值將會越來越小。因此為了增加 SC 內積單元的準確性，總和加權內積

模組因此被提出[11]，如圖 2-7 所示，多工器的控制訊號機率不再是 0.5，而是 $\frac{|a_0|}{|a_0|+|a_1|}$ ，其值是取決於係數的相對大小比。由此可知輸出方程式將會是 $\frac{|a_0|}{|a_0|+|a_1|} \text{sign}(a_0)x_0 + (1 - \frac{|a_0|}{|a_0|+|a_1|}) \text{sign}(a_1)x_1$ ，可寫成 $\frac{1}{|a_0|+|a_1|} (a_0x_0 + a_1x_1)$ 。此方法除了有更好的信號縮放比，還有另外的優點就是需要的隨機亂數產生器數量較少。

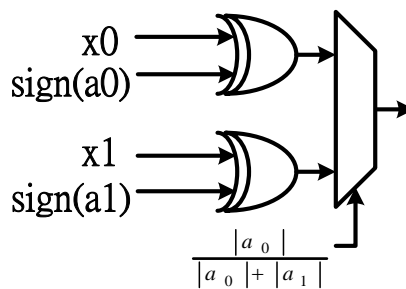


圖2-7 總和加權內積模組

2.2.5 互斥或閘與 J-K 正反器在隨機電路的應用

常用於錯誤更正碼上的原件為互斥或閘與 J-K 正反器，分別介紹如下。如圖 2-8 所示，互斥或閘可以執行 $P_C = P_A(1 - P_B) + P_B(1 - P_A)$ 的運算，其證明方法如表 2-1 互斥或閘的真值表可得知。只有在 A_i 、 B_i 等於 0、1 或是 1、0 的時候，輸出 C_i 會等於 1，由此可知 $P_C = P_A(1 - P_B) + P_B(1 - P_A)$ 。其中 A_i 、 B_i 與 C_i 為輸入與輸出的位元流， P_A 、 P_B 、 P_C 為 A_i 、 B_i 、 C_i 位元流中 "1" 的機率。

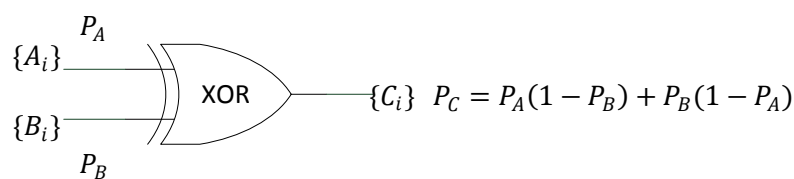


圖2-8 以互斥或閘實現檢查節點運算

P_A	P_B	P_C
0	0	0
0	1	1
1	0	1
1	1	1

表2-1 互斥或閘真值表

另一個基本元件為 J-K 複合電路，此電路是由 J-K 正反器(J-K Flip-Flop)、及閘(AND Gate)與反或閘(NOR Gate)所組成的複合電路。可以用來實現 $P_C = \frac{P_A P_B}{P_A P_B + (1 - P_A)(1 - P_B)}$ 的運算，如圖 2-9 所示。其中 A_i 、 B_i 與 C_i 為輸入與輸出的位元流， P_A 、 P_B 、 P_C 為 A_i 、 B_i 、 C_i 位元流中 "1" 的機率。證明方法可以使用 J-K 複合電路的馬可夫鏈(Markov Chain)，如圖 2-10 所示。輸出 C_i 為 "1" 的所有情況可由圖上動作得知，因此可得到式(9)結果。

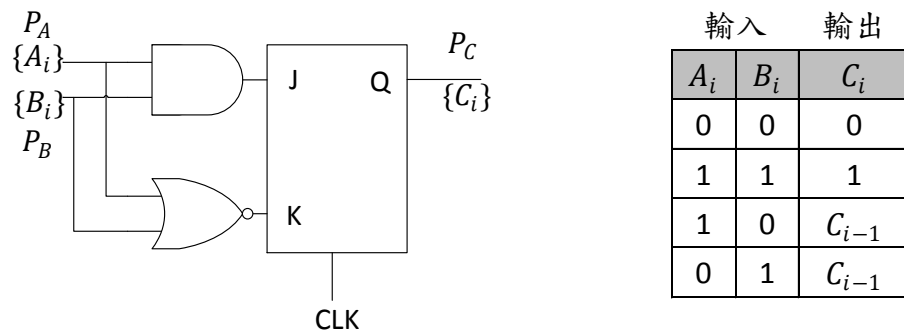


圖2-9 J-K 複合電路

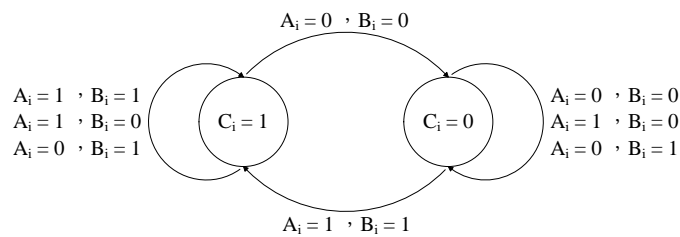


圖2-10 J-K 正反器的複合電路狀態所構成的馬可夫鏈

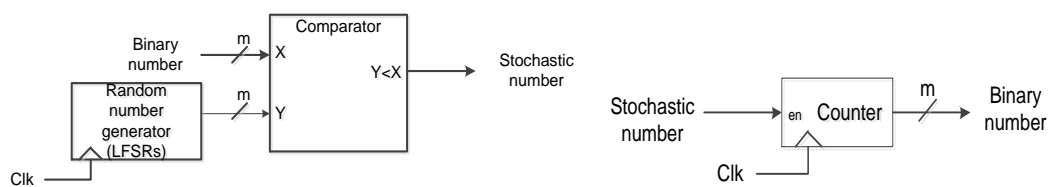
$$\begin{aligned}
P_C &= P_C[P_A P_B + P_A(1 - P_B) + (1 - P_A)P_B] + (1 - P_C)P_A P_B \\
&= P_C[P_A - 2P_A P_B + P_B] + P_A P_B \\
P_C &= \frac{P_A P_B}{1 - P_A + 2P_A P_B - P_B} = \frac{P_A P_B}{P_A P_B + (1 - P_A)(1 - P_B)}
\end{aligned} \tag{9}$$

2.3 隨機亂數產生器設計原理

為了要實現隨機計算，須將傳統的二進制所表示的數值，以隨機數呈現出來。所以我們需要一個隨機亂數產生器 (Stochastic Number Generator, SNG)，將二進制系統所表示的數轉換為隨機數。而 SNG 在隨機計算中，是一個很重要的基本元件。

如圖 2-11(A)所示，SNG 主要是由亂數產生器(Random number generator)及比較器(comparator)組成，其中 m 為隨機亂數產生器的精確度，能產生的亂數範圍為 $0 \sim 2^m - 1$ 。而產生亂數的方法是基於比較器比較的結果，如果比較器的輸入 $X > Y$ 輸出會等於 1，反之就等於 0。

而 SNG 產生出來的位元流在經過隨機計算之後，需要再轉回二進位數。其方法是使用一個計數器，用來計數位元流中 1 的個數，統計完的結果即為二進位數，如圖 2-11(B)所示。



(A) 二進位數轉隨機數(SNG)

(B) 隨機數轉二進位數

圖2-11 數值轉換電路

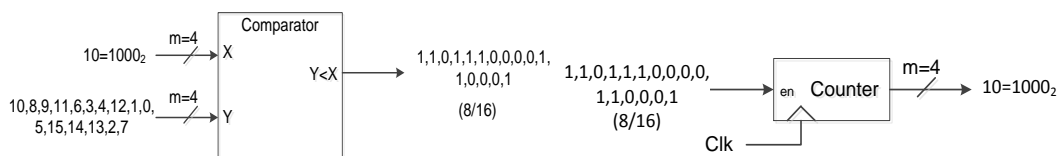


圖2-12 數值轉換電路範例

這裡我們舉例來說明，如圖 2-12，假設 $m=4$ ，若 Binary number = 1000_2 ，則 random number generator 產生的數會在 $[0,15]$ 區間之內，總共改變 16 次，且數值不會重複。再把 X 與 Y 相比會產生一串 16 位元的位元流，而這些數值的機率剛好為 $8/16$ 。若將產生出來的隨機數經過 counter，可以得到一個二進位數 8。

在圖 2-11(A)中還有一個重要的元件 Random number generator 要介紹。在這裡我們是使用線性反饋移位暫存器 (Linear Feedback Shift register, LFSR) 來實現亂數產生的功能。而 LFSR 會根據位元數目的不同，而有多種不同的電路形式。以下我們舉一個 16 位元的 LFSR 來當例子，如表 2-2，16 位元 LFSR 的反饋方程式為 $x^{16} + x^{14} + x^{13} + x^{11} + 1$ 。其電路如圖 2-13 所示，動作原理說明如下。我們先假設原本暫存器裡面的數值為 1,1,0,1,0,0,0,1,1,1,0,0,1,0,1,0(從左到右)，經過一個週期之後，暫存器右移，數值會依據反饋方程式做更新。所以新的數值變為 1,1,1,0,1,0,0,0,1,1,1,0,0,1,0,1(從左到右)，而 Random number generator 就是靠這樣的方式產生出隨機數的。

位元數	反饋方程式
2	$x^2 + x + 1$
3	$x^3 + x^2 + 1$
4	$x^4 + x^3 + 1$
5	$x^5 + x^3 + 1$
6	$x^6 + x^5 + 1$
7	$x^7 + x^6 + 1$

8	$x^8 + x^6 + x^5 + x^4 + 1$
9	$x^9 + x^5 + 1$
10	$x^{10} + x^7 + 1$
11	$x^{11} + x^9 + 1$
12	$x^{12} + x^{11} + x^{10} + x^4 + 1$
13	$x^{13} + x^{12} + x^{11} + x^8 + 1$
14	$x^{14} + x^{13} + x^{12} + x^2 + 1$
15	$x^{15} + x^{14} + 1$
16	$x^{16} + x^{14} + x^{13} + x^{11} + 1$

表2-2 位元數與反饋方程式對應表格

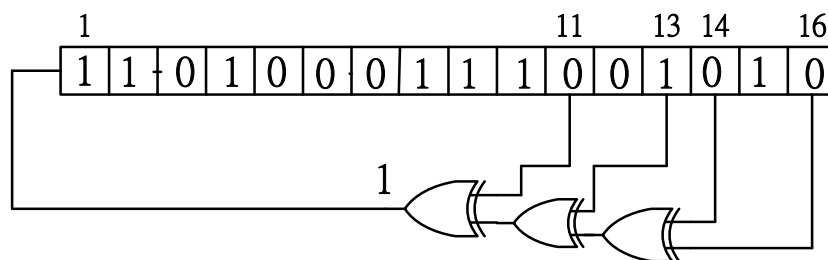
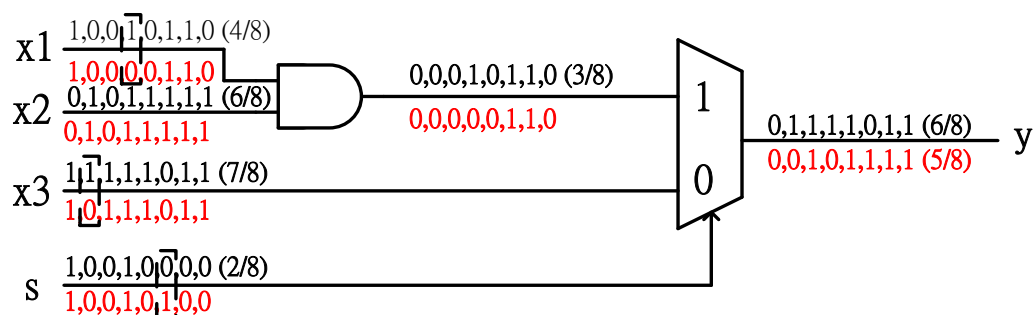


圖2-13 16 位元線性反饋移位暫存器

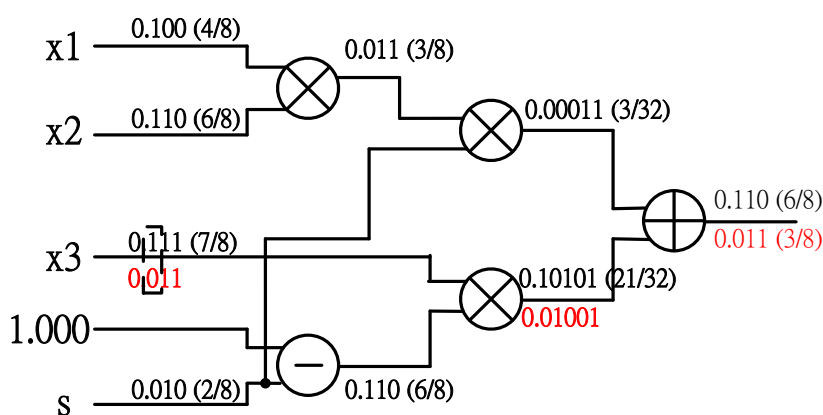
2.4 隨機計算的容錯率

隨機計算系統的準確度雖然不如傳統的二進制系統，但它與二進制計算系統比較起來還有一大優勢，那就是容錯率。假設在一個有雜訊的環境下，位元流受到雜訊的影響可能會改變(0 變 1，1 變 0)，使得接收到的位元流會有錯誤。所以在一般基於二進制表示法的資料運算時，如果是高位元受到雜訊影響，接收到錯誤的位元，這對計算的結果會產生很大的影響。反之，如果是隨機計算的話，因為每個位元的權重都是相同的，所以就算其中有一或兩個位元接收錯誤對於結果

的影響會較小，如圖 2-14 所示。



(A) 隨機計算 $y = x_1x_2s + x_3(1 - s)$



(B) 二進制計算 $y = x_1x_2s + x_3(1 - s)$

圖2-14 二進制計算與隨機計算容錯率比較

如圖 2-14，黑色數字是原本因該接收到的位元流，而紅色數字是因為雜訊影響造成接收到錯誤的位元。由圖 2-14(A)可知，就算有幾個位元接收錯誤，對於計算的結果影響並不會很大，但相對於圖 2-14 (B)所示，僅僅是一個位元的錯誤，對於結果來說就有很大的誤差。

2.5 隨機計算近年來相關應用

隨機計算在近年來的使用是越來越廣泛，應用的領域也越來越多。本小節主要是介紹隨機計算在近年來的相關應用，包括像相關性位元流的應用、影像處理、

數位信號處理與錯誤更正碼....等的應用。

➤ 相關性位元流的應用：

相對於前面 2.2 節所提到的基本元件，例如：乘法器、加法器...等應用，他們所輸入的位元流彼此都是沒有相關性的，也就是說給 SNG 起始的亂數種子是不同的。但本論文要進一步討論，如果位元流彼此是有相關性 [13],[14](種子相同)的話，會不會有完全不同的結果，其結果如表 2-3。OR 閘可以實現取最大值，AND 閘可以實現取最小值，XOR 閘可以實現相減取絕對值，而 MUX 則不變。

所以如果我們現在要實做一個邊緣偵測電路[14]，其公式如式(10)。只要使用一個多工器再加上兩個 XOR 閘就可以完成邊緣偵測電路的設計，如圖 2-15(A)所示，在與二進制的電路相比圖 2-15(B)，隨機計算相關性位元流的電路複雜度相對於二進制電路簡單不少。

$$Z_{i,j} = 0.5 \times (|X_{i,j} - X_{i+1,j+1}| + |X_{i,j+1} - X_{i+1,j}|) \quad (10)$$

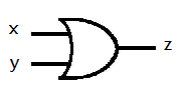

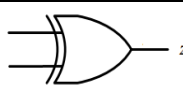
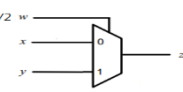
邏輯閘		功能
OR		$\text{Max}(P_x, P_y)$
AND		$\text{Min}(P_x, P_y)$
XOR		$P_Z = P_x - P_y $
MUX		$1/2(P_x + P_y)$

表2-3 邏輯閘在相關性的功能

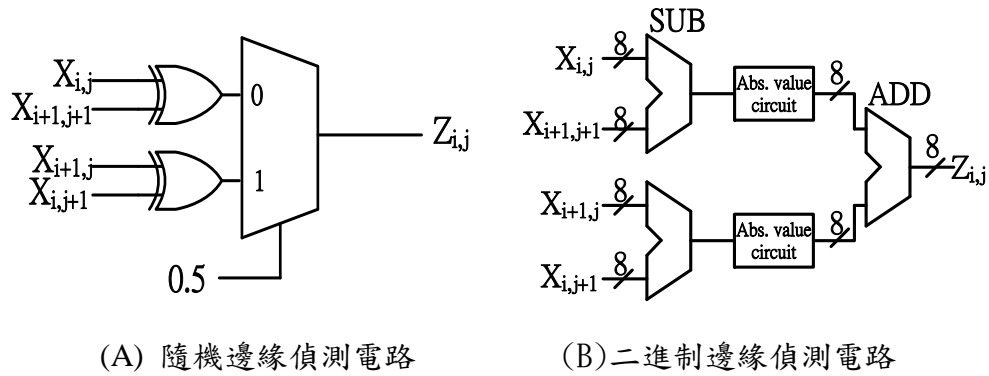


圖2-15 隨機與二進制邊緣偵測電路

➤ 影像處理的相關應用：

隨機計算使用在影像處理方面在近年來有越來越熱門的趨勢[10],[14]。

主要是應用隨機計算運算單元簡單的原理，來設計一些運算式較不複雜的影像處理功能。例如：邊緣偵測(Edge Detection)與雜訊減少(Noise Reduction).....等的功能。

➤ 數位信號處理的相關應用：

在信號處理方面，隨機計算主要是使用在有限脈衝響應濾波器(Finite Impulse Response Filter, FIR filter)與無限脈衝響應濾波器(Infinite impulse Response Filter, IIR Filter)[11],[17].....等的應用上。

➤ 錯誤更正碼的相關應用：

隨機計算使用在錯誤更正碼方面，可以說是這幾年來最主流的應用。除了本論文所使用的隨機LDPC解碼器外，還有隨機維特比(Viterbi)解碼器[18]的應用。

第三章 以隨機計算實現錯誤更正碼

傳統上資訊在傳輸的過程中會因為受到雜訊干擾而有錯誤的發生，進而造成接收端無法正確解碼資料。因此，就有了錯誤更正碼的概念誕生，錯誤更正碼的本意是在原始資料上再加上一些冗餘的資訊，讓接收端在接收到有錯誤的資料時，可以依據這些冗餘的資訊來修正錯誤的資訊。

由於錯誤更正碼的編解碼方法很多，本論文則是依據低密度同位元檢查碼的原理來設計隨機解碼器，詳細介紹如下。

3.1 低密度同位元檢查碼

LDPC 是由具稀疏矩陣性質的同位元檢查矩陣(Parity Check Matrix, H)所構成，而 LDPC 名稱中低密度所代表的是同位元檢查矩陣中"1"的密度。而這個特性的好處在於打散資料的相關性，所以當發生連續的錯誤時可以增加修正回正確數值的可能性。

LDPC 的解碼演算法眾多，包含有和積演算法(Sum Product Algorithm, SPA)、最小和演算法(Min-Sum Algorithm, MSA)、正規化最小和演算法(Normalized Min-Sum Algorithm, NMSA)與補償式最小和演算法(Offset Min-Sum Algorithm, OMSA)等的演算法。而本論文中採用的解碼演算法主要是基於和積演算法(SPA)，此演算法是所有解碼演算法中錯誤修正效果最好的演算法，但是相對其他演算法，計算複雜度也會相對較高。

以下小節將介紹以圖形化表示 H 矩陣的方法 Tanner Graph，與 LDPC 的解碼演算法和積演算法(SPA)。

3.1.1 Tanner Graph

Tanner Graph 是一種以圖形化的方式來表示同位元檢查矩陣資料的連線狀

況。其中同位元檢查矩陣每列運算單元稱為檢查節點(Check node)，而每行的運算單元稱為變數節點(Variable node)。同位元檢查矩陣中每個 1 所代表的是檢查節點與變數節點的連線，而這些連線也代表著資料的傳輸路徑。

這裡以圖 3-1 的同位元檢查矩陣來舉例繪製 Tanner Graph，圖中總共有 5 列，代表有 5 個檢查檢點，這裡以 C1~C5 來表示，而以 V1~V10 來表示共有 10 個變數節點。如圖 3-2，為圖 3-1 的範例所對應到的 Tanner Graph。圖 3-1，V1 行在 C1 與 C4 的位置有 1，表示 V1 與 C1 還有 C4 的節點是相連的，如圖 3-2 的虛線部分。而其他的節點也是依此方法連線，連線完成後就可以建構出圖 3-1 矩陣所對應的 Tanner graph。

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10
C1	1	1	1	0	1	1	0	0	0	0
C2	0	1	0	1	0	0	1	0	0	0
C3	0	1	1	0	1	0	0	1	0	0
C4	1	1	1	1	1	0	0	0	1	0
C5	0	1	0	0	1	0	0	0	0	1

圖3-1 同位元檢查矩陣範例

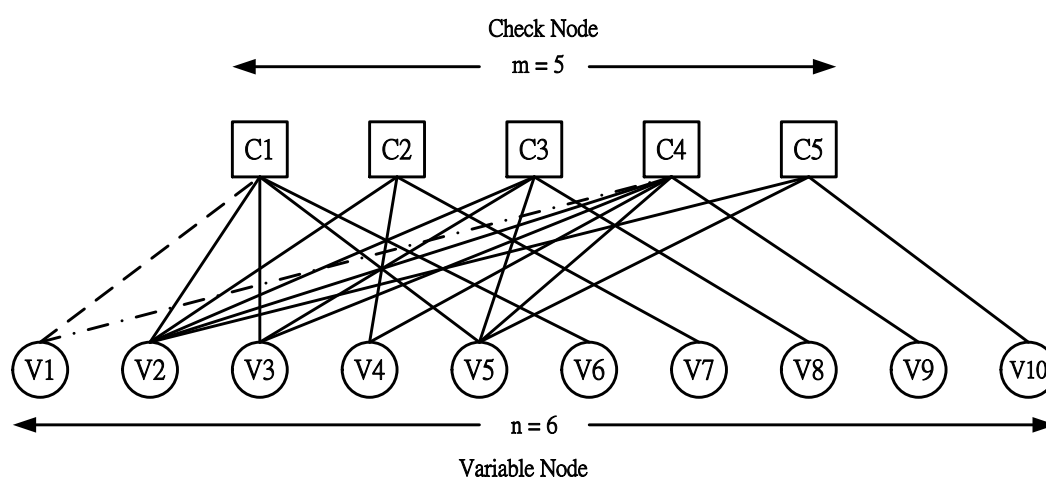


圖3-2 Tanner Graph 範例圖

3.1.2 和積演算法

和積演算法(Sum-of-Product Algorithm, SPA)為訊息傳遞演算法(Message Passing Algorithm, MPA)的一種。是利用事後機率(A Posteriori Probability, APP)在檢查節點與變數節點間多次迭代傳遞來修正錯誤的資料，以達到錯誤修正的效果。

在通訊系統中，編碼完後的訊號字碼(Codeword)，我們以 $v = [v_0 \ v_1 \ \dots \ v_n]$ 來代表，其中 v_i 為0或1。經過二進制相位偏移調變(Binary Phase Shift Keying, BPSK)轉換為類比訊號 $x = [x_0 \ x_1 \ \dots \ x_n]$ ，其中當 $v_i = 1, x_i = +1$ ，反之 $v_i = 0, x_i = -1$ 。訊號會再經過外加白色高斯雜訊通道(Additive White Gaussian Noise, AWGN)轉換為 $y = [y_0 \ y_1 \ \dots \ y_n]$ ，而 $y_i = x_i + n_i$ ， n_i 為外加白色雜訊， i 為 $0 \sim n$ 。

如圖 3-3，為整個信號從傳送端到接收端的傳遞圖。依序為訊號字碼輸入至編碼器，編碼器輸出加入冗餘資訊的字碼，在經過 BPSK 將字碼轉為代表 +1 或 -1 的類比電壓訊號，然後經過模擬的通道雜訊 AWGN，模擬實際傳送時的失真。而接收端，接收到經過通道雜訊後的類比訊號，經由公式轉為機率數值(Voltage to Probability)，接著才進行 SPA 的修正運算。

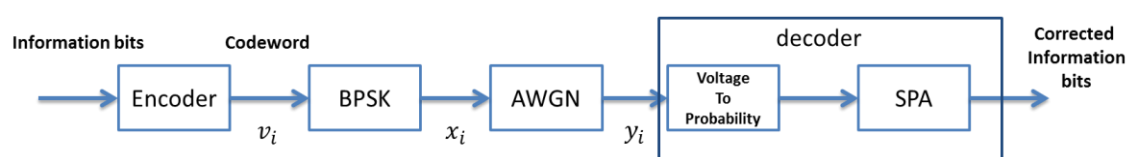


圖3-3 通訊系統資訊傳遞圖

訊息在 Tanner Graph 中傳遞分為兩個步驟，一個為檢查節點的運算，產生的資訊定義為 $r_{j,i}$ 。另一個為變數節點運算，產生的資訊定義為 $q_{i,j}$ 、 Q_i 。其中 i 是代表第幾個變數節點的索引位置， j 是代表第幾個檢查節點的索引位址。

在介紹和積演算法前，先定義一些參數，如表 3-1。

參數名稱	代表意義
$S_{-}V_j$	連接到檢查節點 C_j 的變數節點集合
$S_{-}V_j \setminus i$	除了第 i 個變數節點之外，連接到檢查節點的變數節點集合
$S_{-}C_i$	連接到變數節點 V_i 的檢查節點集合
$S_{-}C_i \setminus j$	除了第 j 個檢查節點之外，連接到變數節點 V_i 的檢查節點集合
$M_v(\sim i)$	來自變數節點 V_i 以外的所有資訊
$M_c(\sim j)$	來自檢查節點 C_j 以外的所有資訊
S_i	滿足變數節點 V_i 的檢查方程式
$q_{i,j}(b)$	$\Pr(V_i = b S_i, y_i, M_c(\sim j))$ ，其中 b 為 0 或 1
$r_{i,j}(b)$	$\Pr(V_i = b M_v(\sim i))$ ，其中 b 為 0 或 1

表3-1 和積演算法參數定義

和積演算法分為五個步驟，分別為初始化、檢查節點運算、變數節點運算(Q、q)與字碼判定。

步驟 1:初始化

初始化主要是將接收到的類比電壓資訊(y_i)，轉為機率值(Pch_i)，並且將此機率值指定給 $q_{i,j}(1)$ 與 $q_{i,j}(0)$ 當作初始值，如式(11)。

$$Pch_i = P(v_i = 1 | y_i) = \frac{1}{(1 + e^{\frac{-2*y_i}{\frac{N_0}{2}}})}$$

$$q_{i,j}(1) = Pch_i$$

$$q_{i,j}(0) = 1 - Pch_i$$
(11)

步驟 2:檢查節點運算

接收變數節點 $q_{i,j}$ 來的資訊，經由式(12)計算後可得到 $r_{j,i}$ ，其傳輸路徑如圖 3-4 所示，這裡以三個輸入的檢查節點為例，接收從 V_1 傳送來的數值 $q_{1,1}$ ，與 V_3 傳

送來的數值 $q_{3,1}$ ，並執行式(12)運算，計算出 $r_{1,2}$ 數值回傳 V_2 。如欲計算傳回 V_1 與 V_3 也是依此方法計算。如此就可以得到所有的 $r_{j,i}$ 。

$$r_{j,i}(0) = \frac{1}{2} + \frac{1}{2} \prod_{i' \in S_{V_j \setminus i}} (1 - 2q_{i',j}(1))$$

$$r_{j,i}(1) = 1 - r_{j,i}(0) \quad (12)$$

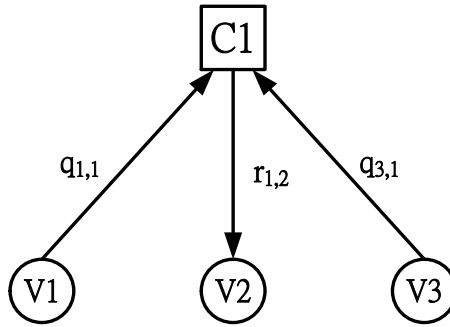


圖3-4 檢查節點電路傳輸路徑

步驟 3: 變數節點運算($q_{i,j}$)

接收檢查節點 $r_{j,i}$ 與通道 Pch_i 來的資訊，經由式(13)計算後可得到 $q_{i,j}$ ，其中 $K_{i,j}$ 為一調整係數，主要是用來正規化 $q_{i,j}(1)$ 和 $q_{i,j}(0)$ 。其傳輸路徑如圖 3-5 所示，這裡以三個輸入的變數節點為例，接收從 C_1 傳送來的數值 $r_{1,1}$ 、 C_2 傳送來的數值 $r_{2,1}$ 與通道傳送來的數值 Pch_1 ，並執行式(13)運算，計算出 $q_{1,3}$ 數值回傳 C_3 。如欲計算傳回 C_1 與 C_2 也是依此方法計算。如此就可得到所有的 $q_{i,j}$ 。

$$q_{i,j}(1) = K_{i,j}(Pch_i) \prod_{j' \in SC_{i/j}} r_{j',i}(1)$$

$$q_{i,j}(0) = K_{i,j}(1 - Pch_i) \prod_{j' \in SC_{i/j}} r_{j',i}(0) \quad (13)$$

$$q_{i,j}(1) + q_{i,j}(0) = 1$$

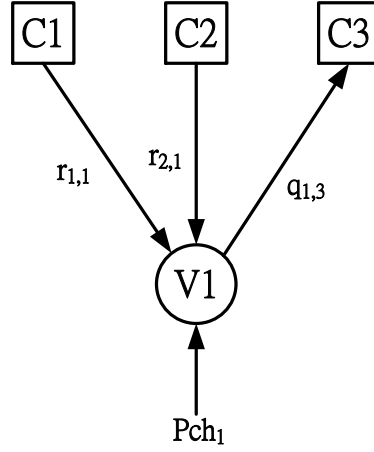


圖3-5 變數節點電路傳輸路徑($q_{i,j}$)

步驟 4: 變數節點運算(Q_i)

接收所有來自檢查節點 $r_{j,i}$ 與通道 Pch_i 傳來的數值，經由式(14)計算後可得到 Q_i ，其中 $K_{i,j}$ 為一調整係數，主要是用來正規化 $Q_i(1)$ 和 $Q_i(0)$ 。其傳輸路徑如圖3-6所示，這裡以三個輸入的變數節點為例，接收從 C_1 傳送來的數值 $r_{1,1}$ 、 C_2 傳送來的數值 $r_{2,1}$ 、 C_3 傳送來的數值 $r_{3,1}$ 與通道傳送來的數值 Pch_1 ，並執行式(14)運算，計算出 Q_i 數值以執行步驟5字碼的判定。

$$\begin{aligned}
 Q_i(1) &= K_i(Pch_i) \prod_{j \in SC_i} r_{j,i}(1) \\
 Q_i(0) &= K_i(1 - Pch_i) \prod_{j \in SC_i} r_{j,i}(0) \\
 Q_i(1) + Q_i(0) &= 1
 \end{aligned} \tag{14}$$

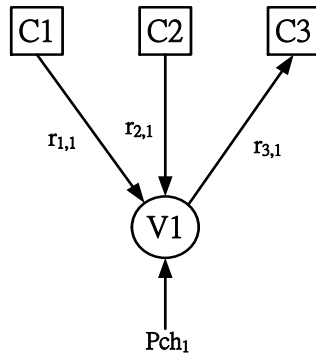


圖3-6 變數節點電路傳輸路徑(Q_i)

步驟 5:字碼判定

由步驟 4 所得的 $Q_i(1)$ ，來判斷第 i 個字碼的訊息如式(15)所示，當 $Q_i(1) \geq 0.5$ ，輸出為 1，反之為 0。

$$v_i = \begin{cases} 1, & \text{if } Q_i(1) \geq 0.5 \\ 0, & \text{otherwise} \end{cases} \quad (15)$$

執行完步驟 1~5，若解碼出來的字碼 v_i ，不能使式(16)成立，或是未達到最大設定的迭代次數，則回到步驟 2 繼續執行解碼運算，直至解碼成功。

$$v \cdot H^T = 0 \quad (16)$$

3.2 使用隨機計算實現和積演算法

在 3.1 節中，已介紹如何由 Tanner Graph 與和積演算法來實現 LDPC 解碼運算，在本節中將介紹如何使用隨機運算的方法來實現隨機 SPA 演算法的步驟。

3.2.1 初始化

當接收到通道傳來的類比電壓資訊，可以經由 3.1.2 節式(11)轉換為機率值，而此機率值可以由隨機亂數產生器(SNG)產生隨機亂數，產生出來的位元流的第一個位元就是 $q_{i,j}$ 的初始值。

3.2.2 檢查節點運算

在隨機運算中檢查節點的運算可以由互斥或閘(XOR Gate)來實現。如 2.2.5 節所述。至於為什麼可以使用互斥或閘來實現式(12)的運算式，這裡由三個輸入的檢查節點來說明。如圖 3-7(A)，要計算 $r_{1,1}$ 則需要變數節點來的資訊 $q_{2,1}$ 與 $q_{3,1}$ ，如式(17)。推導出來的結果與互斥或閘的運算式相同，所以這邊可以使用互斥或

開來當檢查節點的運算元件。

$$\begin{aligned}
 r_{1,1}(0) &= \frac{1}{2} + \frac{1}{2}[(1 - 2q_{2,1}(1))(1 - 2q_{3,1}(1))] \\
 r_{1,1}(1) &= 1 - r_{1,1}(0) \\
 &= q_{2,1}(1)(1 - q_{3,1}(1)) + q_{3,1}(1)(1 - q_{2,1}(1))
 \end{aligned} \tag{17}$$

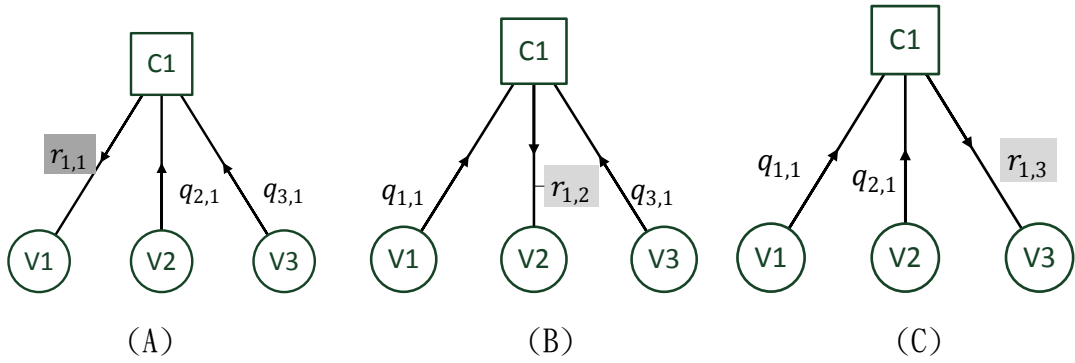


圖3-7 檢查節點資訊傳送範例

由此可知圖 3-7(B)與(C)也可依此方法推導出來，如式(18)，而隨機電路的實現方法則是使用互斥或閘如圖 3-8 所示。

$$\begin{aligned}
 r_{1,2} &= q_{1,1} \cdot (1 - q_{3,1}) + q_{3,1} \cdot (1 - q_{1,1}) \\
 r_{1,3} &= q_{1,1} \cdot (1 - q_{2,1}) + q_{2,1} \cdot (1 - q_{1,1})
 \end{aligned} \tag{18}$$

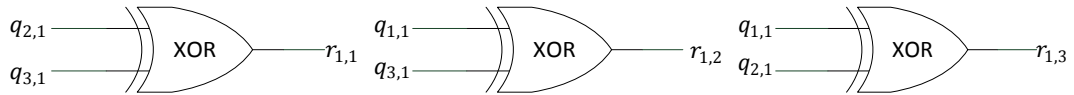


圖3-8 以互斥或閘實現檢查節點運算範例

3.2.3 變數節點運算

變數節點的運算可以由 2.2.5 節提到的 J-K 複合電路來實現。至於為什麼 J-K 複合電路可以實現變數節點的運算，我們以一個兩輸入的變數節點來舉例說明。

變數節點不外乎就是求 $q_{i,j}$ 與 Q_i ，所以如圖 3-9(A)，假設要求取 $q_{1,2}$ ，則需要 C_1 節點傳送來的訊息 $r_{1,1}$ 與通道傳來的訊息 Pch_1 ，再經由式(13)推導出式(19)。其運算出來的結果與隨機運算的 J-K 複合電路相等。所以這裡可以使用 J-K 複合電路來實現變數節點的運算。

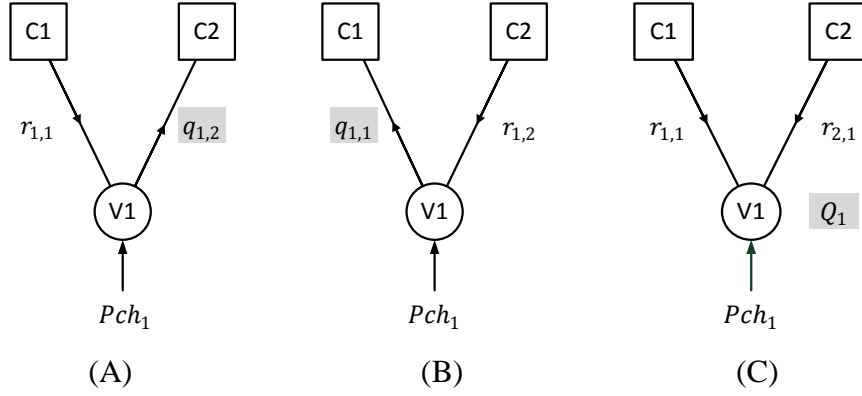


圖3-9 變數節點資訊傳送範例

$$\begin{aligned}
 q_{1,2}(1) &= K_{1,2} Pch_1 r_{1,1}(1) \\
 q_{1,2}(0) &= K_{1,2} (1 - Pch_1) (1 - r_{1,1}(1)) \\
 \therefore q_{1,2}(1) + q_{1,2}(0) &= 1 \\
 \therefore q_{1,2} &= \frac{Pch_1 \cdot r_{1,1}}{Pch_1 \cdot r_{1,1} + (1 - Pch_1) \cdot (1 - r_{1,1})}
 \end{aligned} \tag{19}$$

由此可知圖 3-9(B)也可依此方法推導出來。至於圖 3-9(C)為計算 Q 的資訊，這裡可以由式(14)推導出來，結果如式(20)所示。其隨機電路如圖 3-10 所示。

$$\begin{aligned}
 q_{1,1} &= \frac{Pch_1 \cdot r_{2,1}}{Pch_1 \cdot r_{2,1} + (1 - Pch_1) \cdot (1 - r_{2,1})} \\
 Q_1 &= \frac{Pch_1 \cdot r_{1,1} \cdot r_{2,1}}{Pch_1 \cdot r_{0,1} \cdot r_{1,1} + (1 - Pch_1) \cdot (1 - r_{0,1}) \cdot (1 - r_{1,1})}
 \end{aligned} \tag{20}$$

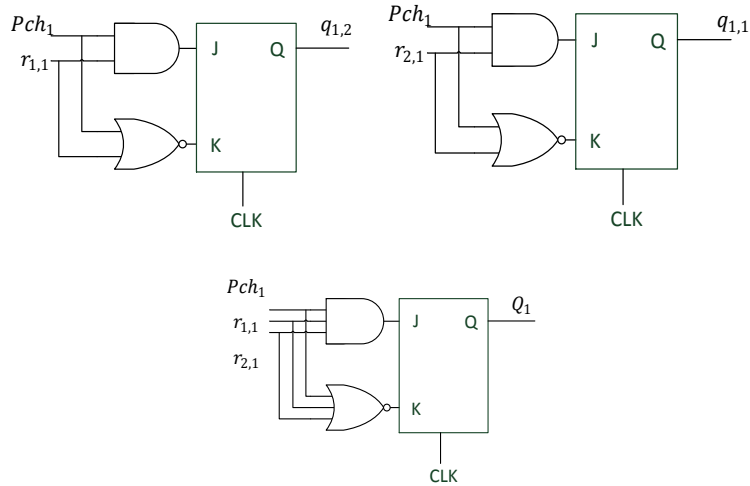


圖3-10 J-K 複合電路使用隨機計算實現範例

3.2.4 字碼的硬判定(Hard Decision)

字碼的硬判定是根據 3.2.3 節中的 $Q_i(t)$ 來判定解碼的結果，而每一個解碼週期變數節點都會傳一個位元的計算結果給硬判定模組，而硬判定模組會再根據所接收到的位元，依式(21)統計它的數值。一般是使用計數器來統計，但若解碼週期較長，計數器所需的記憶體大小就會相對的比較大。因此為了節省記憶體的用量，我們就需要去限制記憶體的大小，又由於迭代就多次後，變數節點傳送給字碼硬判定模組的 $Q_i(t)$ 就會越準確，所以我們只需要取得解碼完成前的幾個解碼週期數據，即可準確的依式(22)判斷要輸出的字碼結果。所以可以使用一個上下數飽和計數器來實現字碼硬判定模組的計算，如此即可記錄下解碼完成前的數據，做有效的輸出判斷，而不用使用很大的計數器，來統計全部的資訊。

$$\begin{cases} \text{Counter}_{HD}(t) = \text{Counter}_{HD}(t-1) + 1, \text{ 當 } Q(t) = 1 \\ \text{Counter}_{HD}(t) = \text{Counter}_{HD}(t-1) - 1, \text{ 當 } Q(t) = 0 \end{cases} \quad (21)$$

$$\begin{aligned} V_{\text{Decoded}}(t) &= 1, \text{ 當 } \text{Counter}_{HD}(t) \geq 0 \\ V_{\text{Decoded}}(t) &= 0, \text{ 當 } \text{Counter}_{HD}(t) < 0 \end{aligned} \quad (22)$$

3.3 變數節點在隨機計算下的栓鎖問題與解決方法

栓鎖問題(Latching Problem)主要是在隨機運算的變數節點上會發生的問題。由於在隨機運算中變數節點是使用 J-K 複合電路來實現，如圖 3-11 所示。當輸入 A_i 與 B_i 都相同 ($A_i = 0 \& B_i = 0 \mid A_i = 1 \& B_i = 1$)，則輸出 C_i 會等於輸入的數值 ($C_i = 0 \mid C_i = 1$)，此狀態稱為非持續狀態(Non-Hold State)。若是輸入 A_i 與 B_i 不同 ($A_i = 0 \& B_i = 1 \mid A_i = 1 \& B_i = 0$) 則輸出會等於之前一筆的輸出狀態 C_{i-1} ($C_i = C_{i-1}$)，此狀態稱為持續狀態(Hold-State)。所以如果輸入位元流有很多持續狀態，就會造成輸出結果有很大的誤差。以下以一個例子來說明，如圖 3-12 所示，假設輸入 $P_A = \frac{1}{10}$ 、 $P_B = \frac{9}{10}$ ，理論上依照式(19)， $P_C = \frac{5}{10}$ 。但在圖上所顯示出來的運算結果卻是 $P_C = \frac{1}{10}$ ，由此可知 J-K 複合電路的持續狀態會對結果造成很大的影響。進一步觀察會發現 P_A 與 P_B 的機率值都非常接近 0 或是 1，因此轉換為隨機位元流會有非常極端狀況。如 P_A 位元流中 0 的位元非常多，而 1 非常少， P_B 則是 1 非常多而 0 非常少。所以要改善這樣的狀況，就要使用雜訊相依性調整 (Noise-Dependent Scaling ,NDS)，來調整隨機數的機率值，讓它不要太靠近 0 或是 1，詳細的方法將在 3.3.1 節介紹。

由此可知，栓鎖問題對於解碼的效能有顯著的影響。因此在接下來的介紹主要都是針對改善栓鎖問題所提出來的的方法，在近期的文獻中，主要的改善方法有邊緣記憶體 (Edge Memory , EM)、循跡預測記憶體 (Tracking Forecast Memory ,TFM)、基於多數決機制的循跡預測記憶體 (Majority-Based Tracking Forecast Memories, MTFM) 與基於多數決機制的邊緣記憶體 (Majority Edge Memory , MEM)，以上方法將在接下來的小節中詳細介紹。

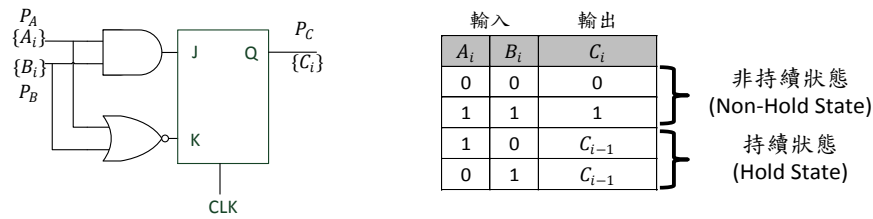


圖3-11 J-K 複合電路的輸出狀態

			隨機位元流									
			1	2	3	4	5	6	7	8	9	10
input	P_A	$\frac{1}{10}$	1	0	0	0	0	0	0	0	0	0
	P_B	$\frac{9}{10}$	1	0	1	1	1	1	1	1	1	1
output	P_C	$\frac{1}{10}$	1	0	0	0	0	0	0	0	0	0

圖3-12 栓鎖問題範例

3.3.1 雜訊相依性調整(Noise-Dependent Scaling , NDS)

栓鎖問題一般發生在訊號雜訊比(Signal-To-Noise Ratio , SNR)較高的時候，因為訊號雜訊比越高代表通道中訊號受到雜訊的影響越少，接收到的數據就越準確，因此通道機率就會趨近於 0 或 1。由圖 3-12 可知，如果這樣的機率轉換為隨機位元流，那位元流中的大部分位元都會是 0 或 1，那就會讓運算狀態一直保持在持續狀態，導致栓鎖問題發生。所以雜訊相依性調整(Noise-Dependent Scaling , NDS)機制就是為了解決栓鎖問題所提出來的解決方法。

在雜訊相依性調整的機制下，將通道接收的資訊 L_i 藉由一個調整參數來做縮小的調整，這樣的調整可以減少隨機解碼中栓鎖問題發生的頻率。這裡的 L_i 指的是通道資訊以 LLR(Log-likelihood ratio)的方式呈現，如式(23)所示。其中 L_i 若為負數，則代表通道資訊為 0 的機率較大，反之，若 L_i 為正數，則代表通道資訊為 1

的機率比較大。

$$L_i = \log \left(\frac{\Pr(v_i=1|y_i)}{\Pr(v_i=0|y_i)} \right) \quad (23)$$

至於 NDS 的調整方式如式(24)所示，其中 α 為一個常數， Y 為收到訊號的最大值， y_i 為第 i 個收到字碼類比訊號， N_0 為單邊雜訊功率譜密度， L'_i 代表 NDS 調整後的資訊 L_i 。

$$L'_i = \left(\frac{\alpha N_0}{Y} \right) L_i = \left(\frac{4\alpha}{Y} \right) y_i, \quad 0 < \alpha < Y, \quad 0 < N_0 < 1 \quad (24)$$

在隨機運算中，數值都是以機率的方式來進行運算，所以這裡需要將 L_i 轉換為通道機率(Pch_i)，如式(25)所示，才能進行隨機 LDPC 的解碼動作。

$$Pch_i = \Pr(v_i = 1|y_i) = 1 - \Pr(v_i = 0|y_i) = \frac{e^{L_i}}{e^{L_i} + 1} \quad (25)$$

簡單來講，NDS 是將過於接近 1 或是 0 的機率數值往 0.5 的方向調整，藉以降低持續狀態出現的機率，增加解碼的效能。

3.3.2 邊緣記憶體(Edge Memory , EM)

由於 J-K 複合電路會有栓鎖問題的產生，所以 EM 就是為了要改善此問題所提出的方法之一。如圖 3-13 為一兩輸入的 EM 變數節點電路。其中 $S(t)$ 代表運算狀態，當輸入 $A(t)$ 與 $B(t)$ 相同時，即是非持續狀態(Non-Hold State)，反之若是不同，則為持續狀態(Hold State)； $r(t)$ 代表再生資訊，只有在非持續狀態下，這個再生資訊才有意義，且會被記錄到 EM 中，同時當作輸出資訊； $r'(t)$ 代表最近的

再生資訊，當發生持續狀態時，會從 EM 中取得非持續狀態下所記錄的資訊當作輸出。

圖 3-13 中的 EM 是一個 M 位元的移位暫存器(M-bit Shift-Register)，其動作是使用先進先出(First In First Out ,FIFO)的策略，藉以將較舊的資訊，經過數次的迭代之後而排除。

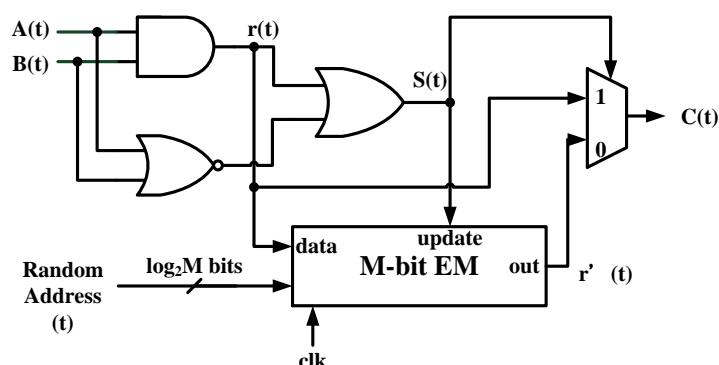


圖3-13 兩輸入的 EM 變數節點電路

而其動作我們分為非持續狀態，與持續狀態兩種來做說明。當發生非持續狀態時， $S(t)$ 等於 1，再生資訊 $r(t)$ 會更新 EM，並且當作此次計算之輸出資訊；而若發生持續狀態時， $S(t)$ 等於 0，再生資訊 $r(t)$ 不會更新 EM，輸出資訊是由 EM 根據 Random Address(t)選出的數值當作輸出。

當然變數節點也不單只有兩個輸入，如圖 3-14 與圖 3-15 為三個與六個輸入的 EM 變數節點電路。而圖 3-14 與圖 3-15 中，內嵌記憶體(Internal Memory , IM)為一容量較 EM 小的記憶體，其構造與 EM 相同，都是為了讓電路進入非持續狀態的機率增加所設計的。為了方便之後的章節說明，所以簡化電路的表示方法，如圖 3-16，我們將電路以區塊化的方式表達。所以圖 3-14 與圖 3-15 可以簡化成圖 3-17 與圖 3-18 所示。

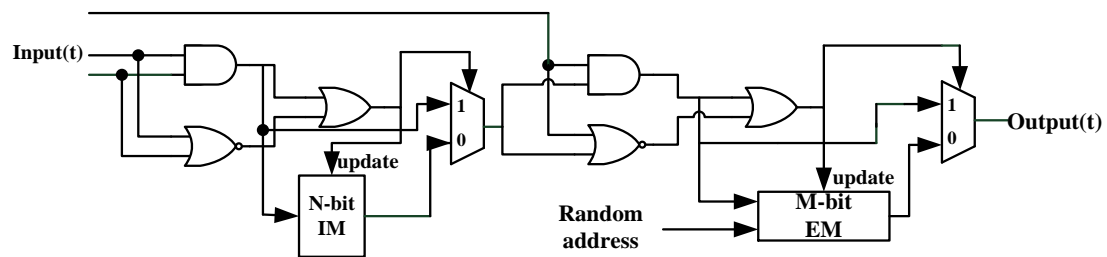


圖3-14 三個輸入 EM 變數節點電路

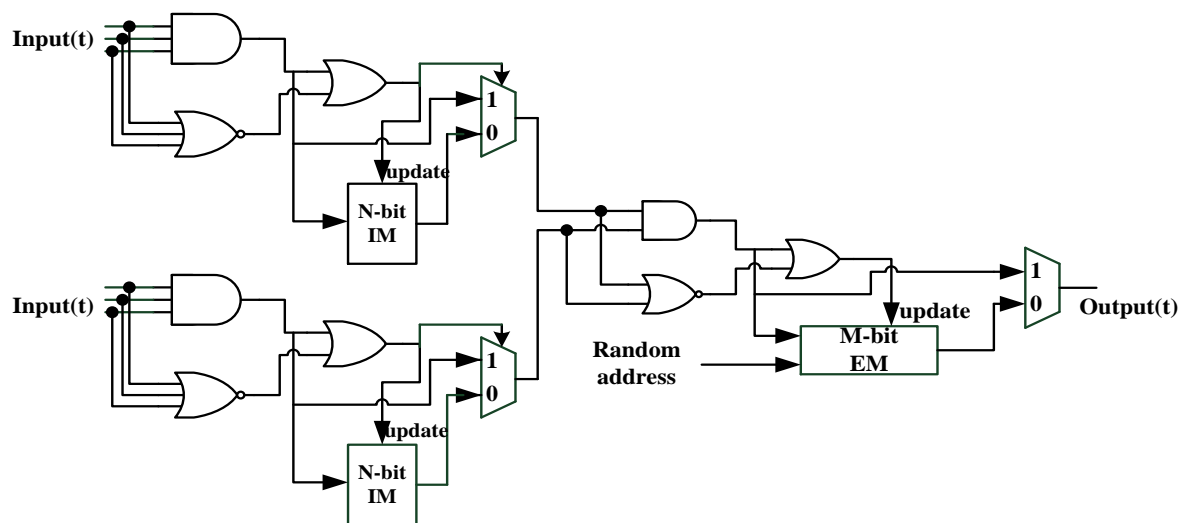


圖3-15 六個輸入 EM 變數節點電路

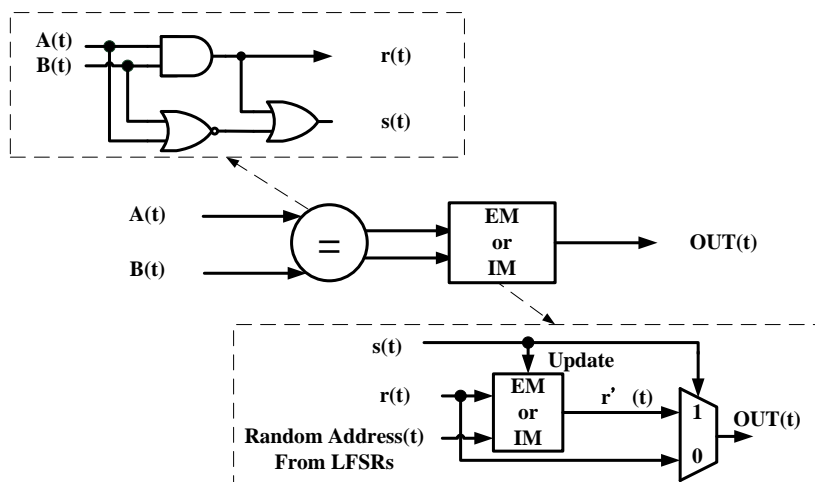


圖3-16 區塊化兩輸入變數節點的隨機運算架構

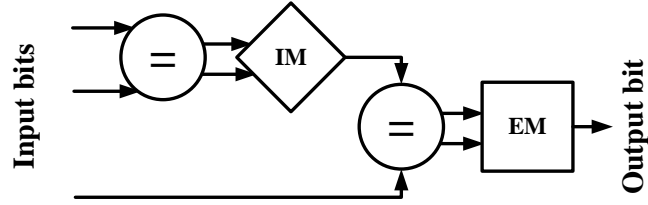


圖3-17 三輸入 EM 變數節點簡化圖

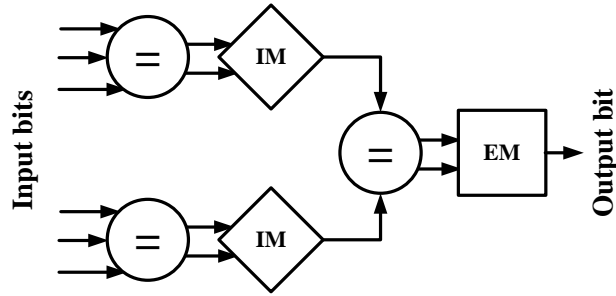


圖3-18 六輸入 EM 變數節點簡化圖

由多個輸入的變數節點觀察得知，當變數節點的輸入越多時，EM 就要使用越多個。如圖 3-19 所示，EM 的多寡與輸入的資訊數有關。因此，為了減少硬體的大小，才會有循跡預測記憶體(Tracking Forecast Memory, TFM)[8]、基於多數決機制的循跡預測記憶體(Majority-Based Tracking Forecast Memories, MTFM)[5]與基於多數決機制的邊緣記憶體(Majority Edge Memory, MEM)[15]的出現。

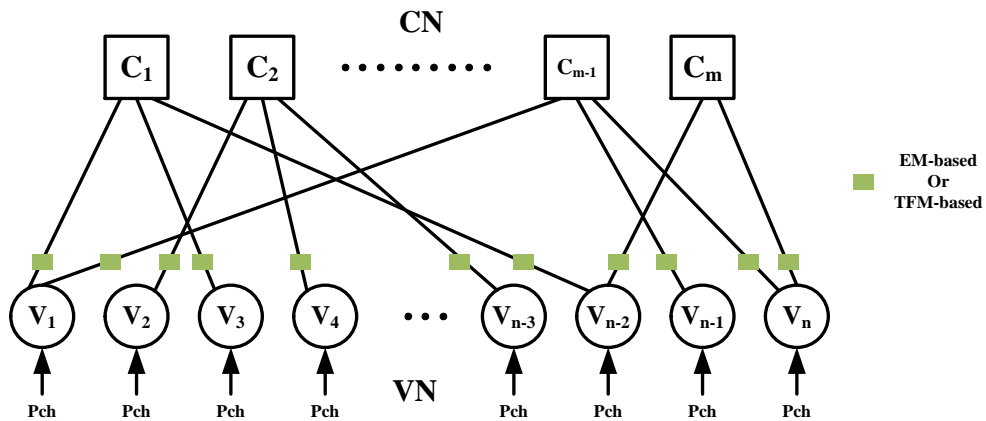


圖3-19 EM(TFM)在 Tanner Graph 的位置圖

3.3.3 循跡預測記憶體(Tracking Forecast Memory , TFM)

TFM 也是改善栓鎖問題的方法之一，由於 EM 會使用非常大的記憶體，所以 TFM 就是為了減少記憶體的使用量而發展出來的設計。EM 與 TFM 最大的不同，在於更新記憶體的方式。我們以例子來說明，假設 EM 的記憶體為 32-bits，如果這 32-bits 的記憶體中有 19 個 bits 為 1，那這個記憶體代表的機率是 19/32。而 TFM 與 EM 不同的地方在於 TFM 是使用二進制的方法來管理記憶體。所以我們假設 TFM 的記憶體大小為 7-bits，其記憶體的數值為”0011111”，所代表的數值為 31/127。由此可知 TFM 可以使用比 EM 更小的記憶體使用量來表示一個機率數值。

而 TFM 的計算方式是基於連續鬆弛法(Successive Relaxation ,SR)，這個方法常用於求線性方程式的解，以迭代的方式得到正確解，如式(26)。其中 $P(t)$ 為當下尚未更新的記憶體資訊，其值是介於 0~1 之間的機率值； $P(t+1)$ 為更新後的記憶體資訊，其值是介於 0~1 之間的機率值； $r(t)$ 為變數節點在非持續狀態所產生的再生資訊，其數值為 0 或 1； $\beta(t)$ 為鬆弛係數，其數值介於 0~1 之間的機率值，通常設定為 2 的負冪次方。

$$P(t + 1) = (1 - \beta(t)) \cdot P(t) + \beta(t) \cdot r(t) \quad (26)$$

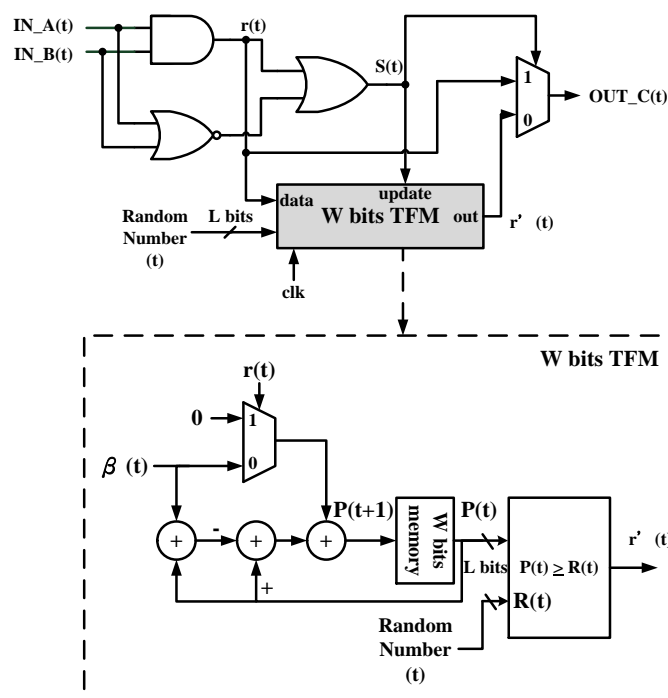


圖3-20 使用 TFM 的兩輸入變數節點

而 TFM 變數電路的設計如圖 3-20 所示，觀察圖中可以發現，TFM 電路與 EM 電路最大的不同在於記憶體電路的改變。其中 L 代表通道機率的精確度， W 代表機率更新計算的精確度。圖中取代 EM 的 W bits TFM 這個區塊主要是在執行式(26)的運算。其動作說明如下，這裡分為兩個部分，第一部分為非持續狀態的動作，第二部分為持續狀態的動作。當發生非持續狀態時， $S(t)$ 等於 1， W bits TFM 會根據輸入的再生資訊 $r(t)$ 執行式(26)的計算，並且更新 TFM 的記憶體，而輸出則為再生資訊 $r(t)$ 。而若當持續狀態發生時， $S(t)$ 等於 0，輸出會根據 Random Number(t) 與 TFM 的記憶體比較，決定輸出為 0 或 1。

3.3.4 基於多數決機制的循跡預測記憶體(Majority- Based Tracking Forecast Memories , MTFM)

由 3.3.2 與 3.3.3 節可知 TFM 使用記憶體的數量會比 EM 小很多，但是如果

使用的同位元檢查矩陣 H 很大的話，縱使使用 TFM，硬體的需求量也會非常的大，因此 MTFM 就是為了降低硬體的使用量所發展出來的設計。

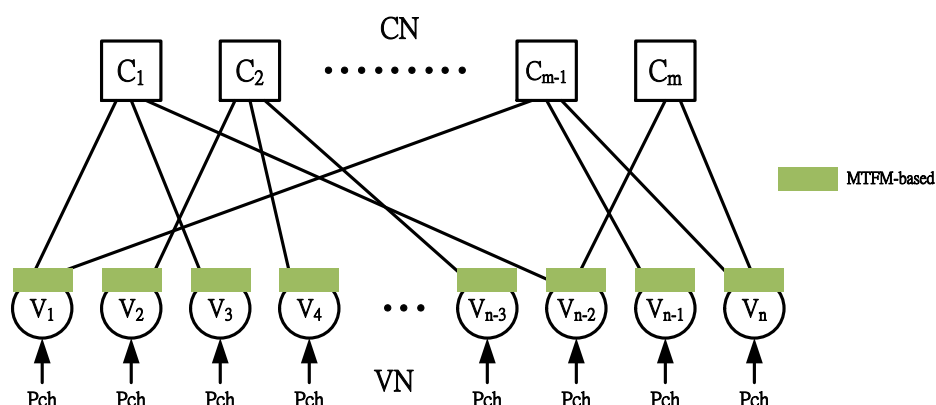


圖3-21 MTFM 在 Tanner Graph 的位置圖

這裡我們以例子來說明:假如使用 EM 或是 TFM，一個變數節點假設有 6 個輸入，需要使用 6 塊 EM 或是 TFM 記憶體。但是如果是 MTFM，硬體使用量將只有原本的 1/6。簡單來講，MTFM 就是一個變數節點只會有一塊記憶體，如圖 3-21 所示。

如圖 3-22，為一 6 級 MTFM 變數節點的 Tanner Graph，其所對應到的 MTFM 電路如圖 3-23 所示。觀察六級的 MTFM 電路可知，每個邊緣點的再生資訊 $r(t)$ 與運算狀態 $s(t)$ 會被分開計算，最終將所有計算的結果傳入 MTFM 中做最後的判定，來決定輸出的資訊。而其更新 MTFM 的機制與如何決定輸出的資訊，詳細說明如下。

根據圖 3-24 所示為 MTFM 內部架構圖，當輸入運算狀態 $s_{1,1}(t) \sim s_{1,6}(t)$ 若都為 1，TFM 會更新資訊。而更新的資訊是根據輸入再生資訊 $r_{1,1}(t) \sim r_{1,6}(t)$ 的多數來決定。也就是說，如果 $r_{1,1}(t) \sim r_{1,6}(t)$ 裡面 1 較多的話，就以 1 來當 TFM 更新的輸入 $r_1(t)$ 。而輸出 $q_{1,1}(t) \sim q_{1,6}(t)$ 會分別輸出 $r_{1,1}(t) \sim r_{1,6}(t)$ 的資訊。

反之，若 $s_{1,1}(t) \sim s_{1,6}(t)$ 有一組不為 1，則 TFM 不會更新， $r_1'(t)$ 會根據 Random Number(t) 選擇輸出資訊。而輸出 $q_{1,1} \sim q_{1,6}$ 則會依自己邊緣點的運算狀態 $s_{1,1}(t) \sim s_{1,6}(t)$ 來決定輸出的資訊。假設 $s_{1,1}(t)$ 等於 1，輸出 $q_{1,1}(t)$ 等於 $r_{1,1}(t)$ 。若

$s_{1,1}(t)$ 等於 0，輸出 $q_{1,1}(t)$ 等於 $r_{1,1}'(t)$ ，依此類推。至於 $Q_1(t)$ 的輸出是根據輸入的 $R_{1,1}(t) \sim R_{1,6}(t)$ 與 $Pch_1(t)$ 的多數資訊來判定，也就是說，若 1 的資訊較多，輸出 $Q_1(t)$ 就是 1，反之則為 0。

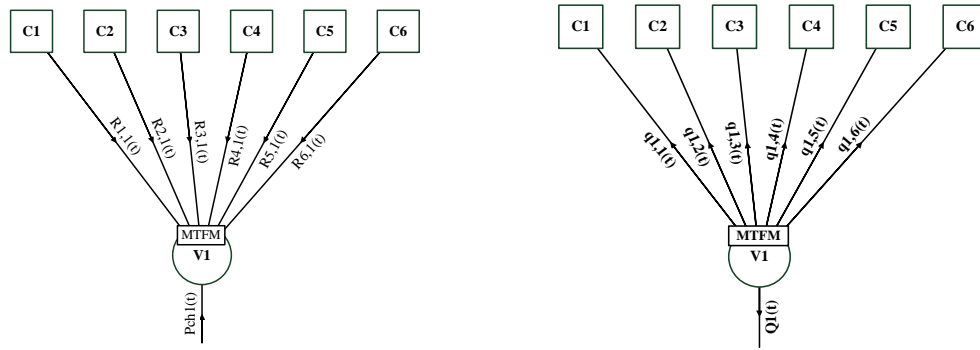


圖3-22 6 級 MTFM 變數節點的輸入輸出方向圖

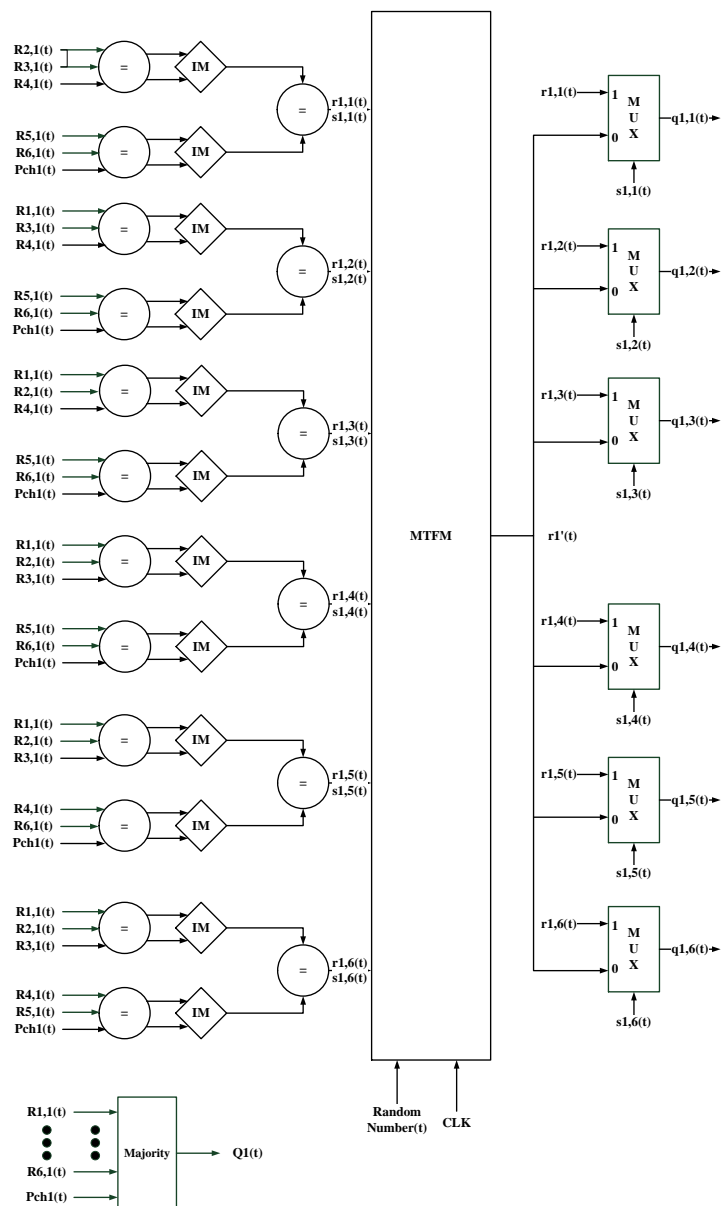


圖3-23 六級 MTFM 變數節點電路圖

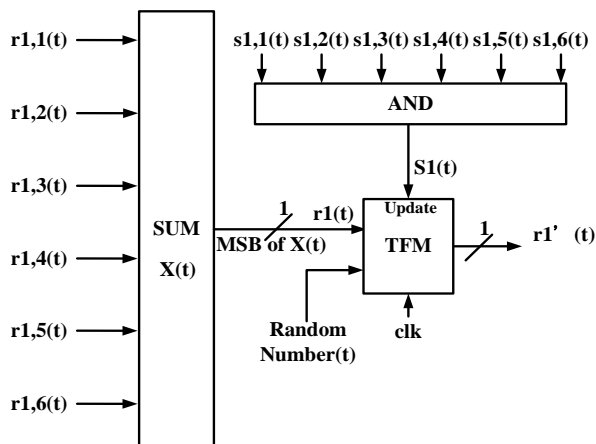


圖3-24 MTFM 內部架構圖

3.3.5 基於多數決機制的邊緣記憶體(Majority Edge

Memory , MEM)

由之前的章節介紹可知，EM 會有記憶體使用量過多的缺點，因此為了解決記憶體用量上的瓶頸，而有了 TFM 與 MTFM 的設計提出。而[15]則依據 EM 的記憶體管理概念，與 MTFM 的記憶體更新機制，設計出了基於多數決機制的邊緣記憶體(Majority Edge Memory , MEM)。

MEM 主要改善了 EM 電路使用過多記憶體的缺點，而解碼的效能上也比使用 TFM 與 MTFM 的設計來的佳。所以本論文的隨機去尾迴旋碼解碼器是使用此架構做為設計之依據。

如圖 3-25 所示為 MEM 在 Tanner Graph 上的位置，與 MTFM 相同，都是在一個變數節點上使用一塊記憶體空間。

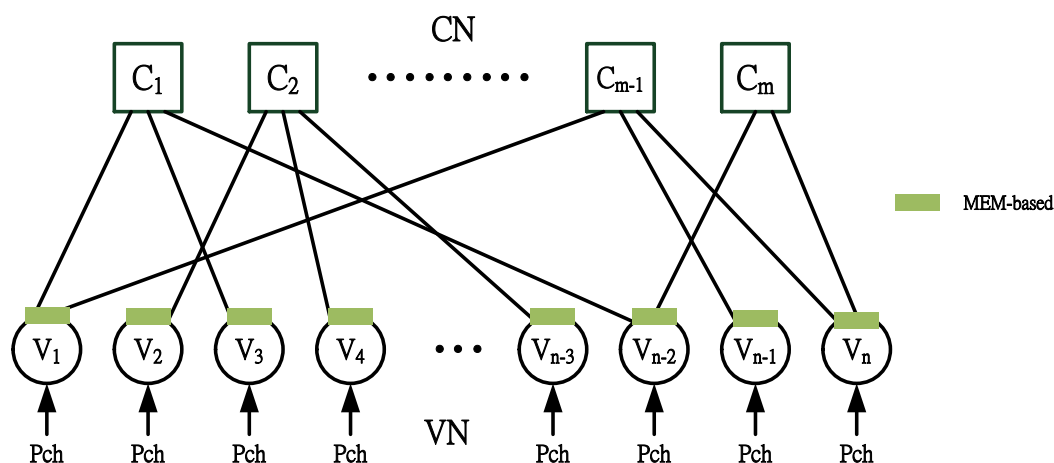


圖3-25 MEM 在 Tanner Graph 上的位置

如圖 3-26 為一 6 級 MEM 的 Tanner Graph，其所對應到的 MEM 變數節點電路圖如圖 3-27 所示。MEM 變數節點的架構大致上與 MTFM 變數節點相同，最大的差別就在於將 TFM 的儲存模式改為 EM。而更新 MEM 與輸出 q 的方式也與 MTFM 相同，唯一不同的就是 Q 的計算方式，詳細說明如下。

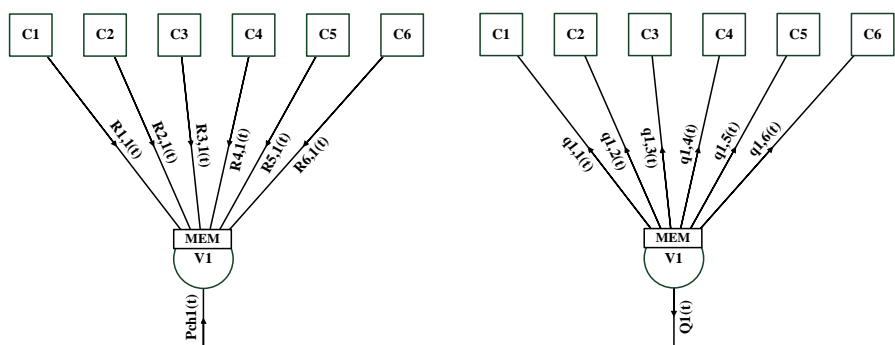


圖3-26 6 級 MEM 變數節點的輸出輸入圖

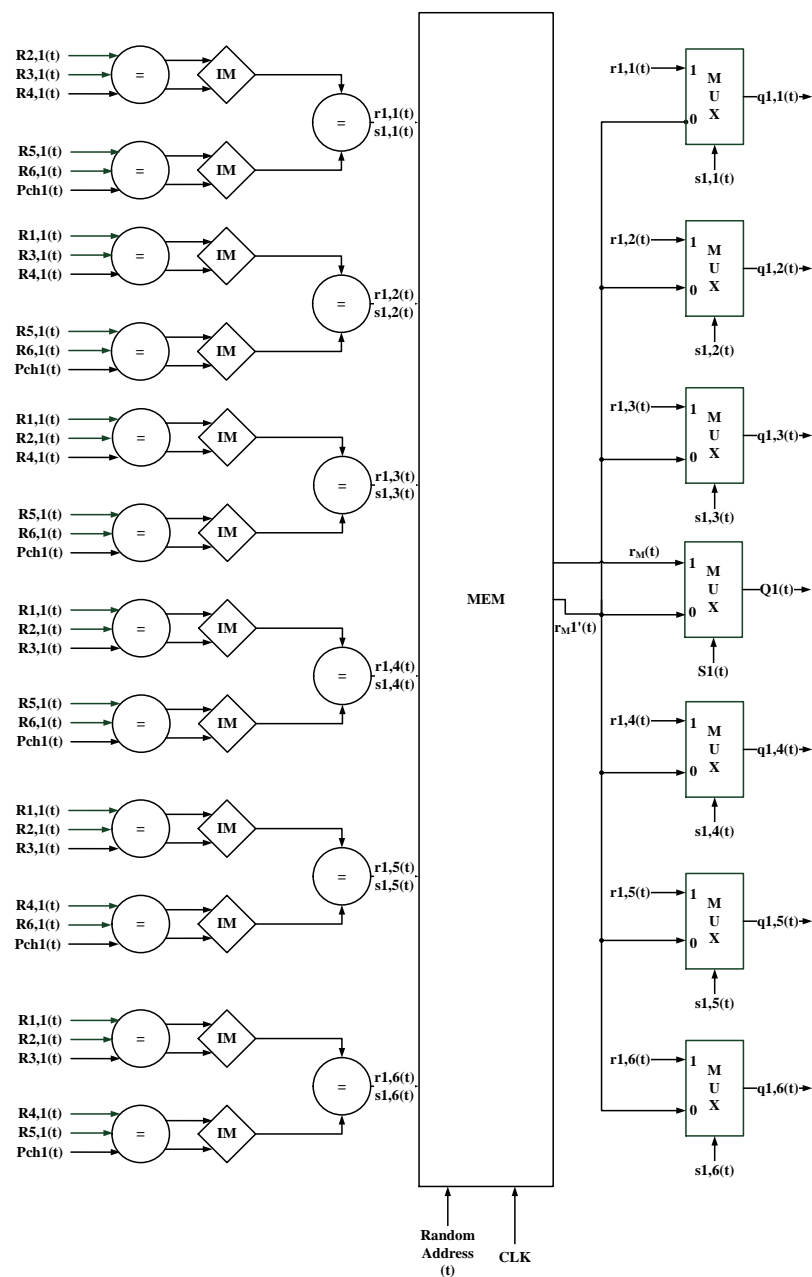


圖3-27 6 級 MEM 變數節點電路圖

根據圖 3-28 的 MEM 內部架構圖，若運算狀態 $s_{1,1}(t) \sim s_{1,6}(t)$ 都為 1，EM 會依據 $r_M 1(t)$ 更新資訊，而 $r_M 1(t)$ 的數值則是根據 $r_{1,1}(t) \sim r_{1,6}(t)$ 的多數來決定。至於輸出 $q_{1,1}(t) \sim q_{1,6}(t)$ 會分別等於 $r_{1,1}(t) \sim r_{1,6}(t)$ ，而輸出 $Q_1(t)$ 會等於 $r_M 1(t)$ 。

反之，若 $s_{1,1}(t) \sim s_{1,6}(t)$ 有一個不為 1，則 EM 不會更新， $r_M 1'(t)$ 會根據 Random Address(t) 選擇資訊。而輸出 $q_{1,1}(t) \sim q_{1,6}(t)$ 會分別根據 $s_{1,1}(t) \sim s_{1,6}(t)$ 的狀態選擇輸出結果，至於 $Q_1(t)$ 會等於 $r_M 1'(t)$ 。

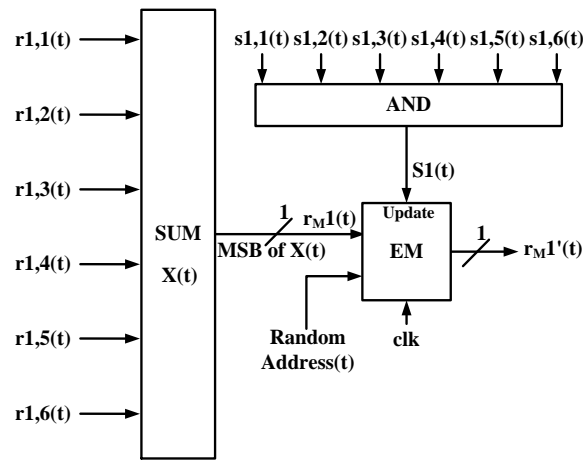


圖3-28 MEM 內部架構圖

3.4 去尾迴旋碼介紹

如前面章節所介紹，LDPC 解碼器一般是用來解碼 LDPC 編碼器所編碼的資訊。但本論文提出另一種應用，那就是使用 LDPC 解碼器解碼由去尾迴旋編碼器所編碼的資訊。由於去尾迴旋碼編碼時，輸入與輸出之間會有同位元的關係，所以可以藉由此關係，產生輸入與輸出所對應的生成矩陣 (Generator Matrix, G Matrix)。之後可以藉由簡單的數學運算將生成矩陣轉換為同位元檢查矩陣 H。有了同位元檢查矩陣，才能使用 LDPC 進行解碼運算。

接下來將介紹去尾迴旋碼的編碼方法，與生成矩陣、同位元檢查矩陣的產生方法。

3.4.1 去尾迴旋碼編碼方法

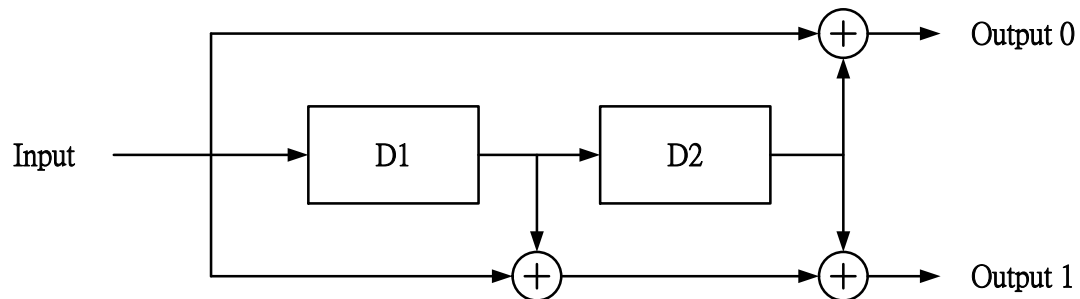


圖3-29 (2,1,2)迴旋編碼器

圖 3-29 為一(2,1,2)迴旋編碼器，其中(2,1,2)我們以(n,k,m)來代表，n 表示編碼後的輸出位元數，k 表示輸入的位元數，m 表示迴旋編碼器所使用的位移暫存器個數。以(2,1,2)迴旋編碼器來說，代表 1 bit 的位元輸入，在經過 2 個位移暫存器後，產生 2 bits 的輸出。

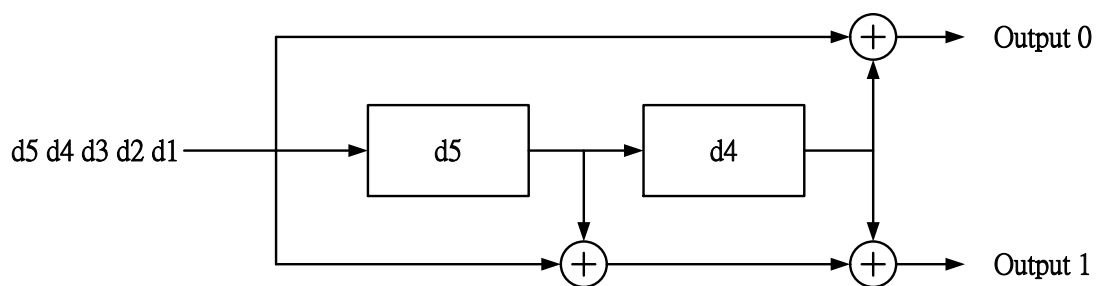


圖3-30 去尾迴旋碼初始狀態

去尾迴旋碼的編碼，會先將要編碼的位元流的最後 m 個位元先放到暫存器中再進行編碼。如圖 3-30，假設要編碼的位元流長度有 5 bits(d1~d5)，在編碼前會先將位元流的最後 2 個位元先存入暫存器，接著才開始編碼。這樣的編碼方法不但不會造成碼率的下降，解碼時錯誤修正的能力也不會受到影響。

3.4.2 產生去尾迴旋碼之生成矩陣

經由 3.4.1 節的編碼方法，我們可以把它轉為生成矩陣(Generator Matrix, G)。有了生成矩陣，就可以經由數學運算，得出同位元檢查矩陣(Parity Check Matrix, H)。以下將舉例說明如何產生生成矩陣。

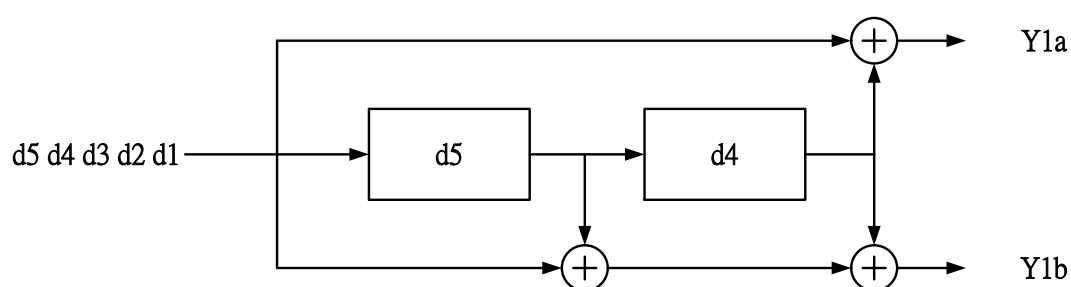


圖3-31 輸入 d1 進行編碼

如圖 3-31，當輸入 d1 位元進行編碼時，輸出 Y1a 與 Y1b 分別為 $d1 \otimes d4$ 與 $d1 \otimes d4 \otimes d5$ 。這裡我們可以用一個矩陣的方式來表示輸入與輸出的關係，這個矩陣就是生成矩陣。如式(27)所示，Y1a 為 $d1 \otimes d4$ ，所以在第一行 d1 與 d4 的位置標註 1，其他位置標註 0。Y1b 為 $d1 \otimes d4 \otimes d5$ ，所以在第二行的 d1、d4 與 d5 的位置標註 1，其他標註 0。

$$G = \begin{matrix} d1 \\ d2 \\ d3 \\ d4 \\ d5 \end{matrix} \begin{bmatrix} 1 & 1 & & & \\ 0 & 0 & & & \\ 0 & 0 & \dots & & \\ 1 & 1 & & & \\ 0 & 1 & & & \end{bmatrix} \quad (27)$$

接著對 d2 編碼，如圖 3-32，Y2a 為 $d2 \otimes d5$ ，Y2b 為 $d1 \otimes d2 \otimes d5$ ，對應到生成矩陣如式(28)第三行與第四行的部分。由此可知 d3、d4 與 d5 也採用相同的方法依此類推，因此當 d1~d5 都編碼完後，會得到一個 5×10 的生成矩陣如式(29)。在 3.4.3 節將詳細介紹如何利用生成矩陣，得出同位元檢查矩陣。

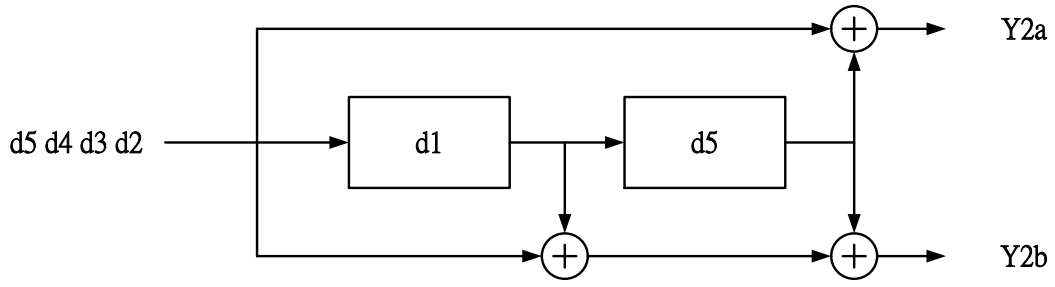


圖3-32 輸入 d2 進行編碼

$$G = \begin{matrix} d1 \\ d2 \\ d3 \\ d4 \\ d5 \end{matrix} \begin{bmatrix} 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \dots\dots \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 \end{bmatrix} \quad (28)$$

$$G = \begin{bmatrix} 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix} \quad (29)$$

3.4.3 產生去尾迴旋碼之同位元檢查矩陣

由 3.4.2 節所舉的例子，得到了生成矩陣。但是如果要用 LDPC 來解碼，就必須要有同位元檢查矩陣 H。以下介紹詳細轉換過程：

首先將 3.4.2 節的 G 矩陣利用高斯消去法(Gaussian Elimination)將其轉換為 [I|P]矩陣的形式，其中 I 為單位矩陣，P 為扣除掉單位矩陣後剩下的矩陣如式(30)所示。

$$G = [I|P] = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 & 1 \end{bmatrix} \quad (30)$$

接下來產生 H 矩陣，將 G 矩陣的 P 部分做轉置再與單位矩陣 I 結合，其形

式為 $[P^T|I]$ ，為一 5×10 大小矩陣，如式(31)所示。

$$H = [P^T|I] = \begin{bmatrix} 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \quad (31)$$

第四章 基於隨機計算之影像處理應用實作

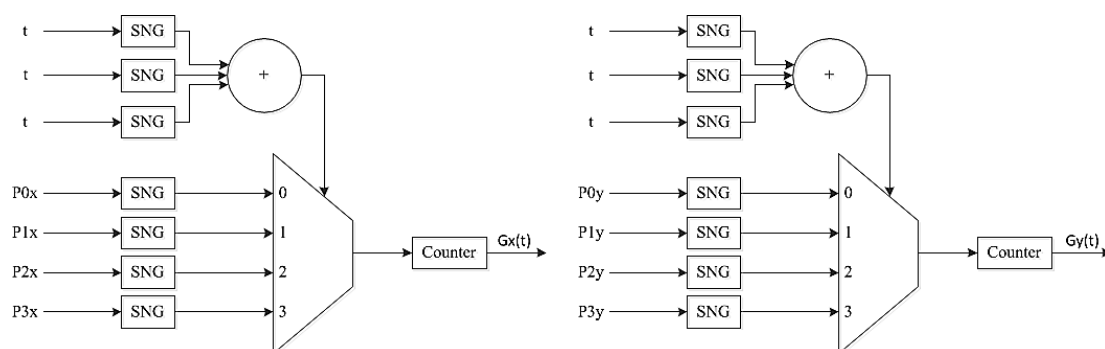
本章節將實作貝茲曲線(Bezier Curve)與離散餘弦轉換(Discrete Cosine Transform, DCT)的隨機計算應用。

4.1 貝茲曲線

在這裡我們是實作三次方貝茲曲線，其公式如式(32)與式(33)所示，其中 P_{0x} 代表輸入第一組座標值的 x 座標， P_{0y} 代表輸入第一組座標值的 y 座標，其餘的三個輸入依此類推。透過觀察公式可以發現，貝茲曲線可以使用我們 2.2.3 所介紹的 Bernstein 方程式合成的方法來設計隨機貝茲曲線的電路，圖 4-1(A)與(B)分別為 x 軸與 y 軸的電路。

$$Gx(t) = P_{0x}(1-t)^3 + 3P_{1x}t(1-t)^2 + 3P_{2x}t^2(1-t) + P_{3x}t^3 \quad (32)$$

$$Gy(t) = P_{0y}(1-t)^3 + 3P_{1y}t(1-t)^2 + 3P_{2y}t^2(1-t) + P_{3y}t^3 \quad (33)$$



(A) 貝茲曲線 x 軸的隨機電路

(B) 貝茲曲線 y 軸的隨機電路

圖4-1 貝茲曲線的隨機電路(方程式合成)

當然實作貝茲曲線還有另外一種方法，就是直接將式(32)與式(33)直接使用

乘法器，加法器或是反向器等...的基本元件組合起來，如圖 4-2 的電路，乘法的部份我們使用 AND 邏輯閘，而 NOT 閘是 $1-t$ 的計算，最後再使用一個加法器來做加總的動作。

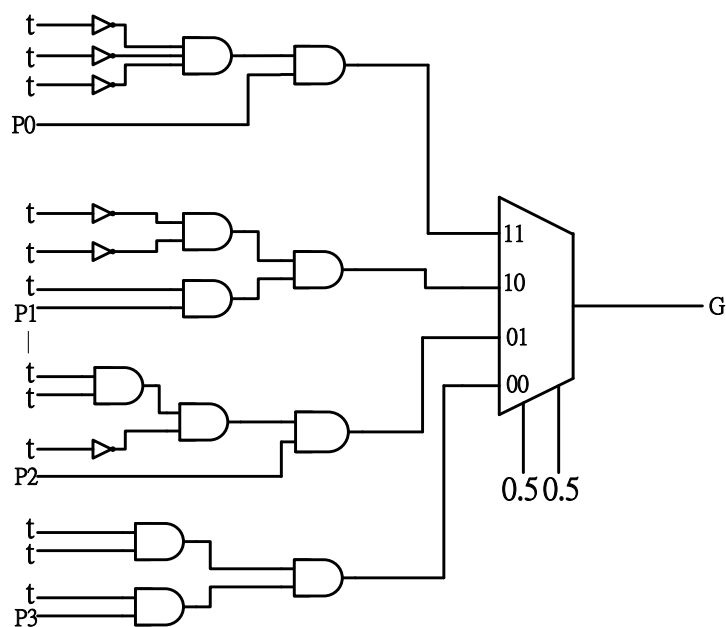


圖4-2 貝茲曲線的隨機電路(基本邏輯閘組合)

4.2 離散餘弦轉換(DCT)

離散餘弦轉換是 JPEG 圖片壓縮的一個重要處理過程，但是在這裡我們只實作 JPEG 壓縮裡 DCT 轉換的部分，並觀察隨機 DCT 轉換後圖片的誤差值。

首先，傳統 JPEG 編碼的步驟如圖 4-3 所示，第一步先將我們輸入的圖檔，轉換成很多 8×8 的小區塊，第二步再對每一塊 8×8 的小區塊做 2 次 DCT 轉換(2-D DCT)，第三步再做一個量化(Quantization)步驟，第四步是對量化完的結果做一個壓縮，稱為熵編碼技術(entropy coding)，完成以上四個步驟，就是一張 JPEG 編碼的圖檔了。

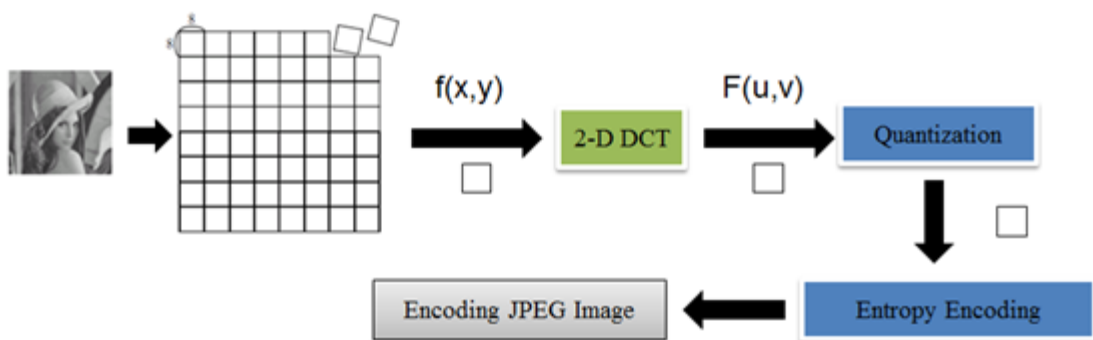


圖4-3 JPEG 編碼步驟

如果要對 8*8 的區塊做 DCT 轉換，我們需要一個 8 點 DCT 的轉換矩陣，式(34)為一 8 點 DCT 的計算方法，其中 $e_{(k)}$ 如式(35)所示。而式(36)為矩陣形式的 8 點 DCT， $x_{(0)} \sim x_{(7)}$ 為圖片輸入的機率值， $X_{(0)} \sim X_{(7)}$ 為圖片經過計算完後輸出的機率值，其中 $c_i = \cos i\pi/16$ 。

$$X_{(k)} = e_{(k)} \sum_{n=0}^7 x_{(n)} \cos\left[\frac{(2n+1)\pi k}{16}\right], k = 0, 1 \dots, 7 \quad (34)$$

$$e_{(k)} = \begin{cases} \frac{1}{\sqrt{2}}, & \text{if } k = 0 \\ 1, & \text{otherwise} \end{cases} \quad (35)$$

$$\begin{bmatrix} X(0) \\ X(1) \\ X(2) \\ X(3) \\ X(4) \\ X(5) \\ X(6) \\ X(7) \end{bmatrix} = \begin{bmatrix} c_4 & c_4 & c_4 & c_4 & c_4 & c_4 & c_4 & c_4 \\ c_1 & c3 & c5 & c7 & -c7 & -c5 & -c3 & -c1 \\ c_2 & c6 & -c6 & -c2 & -c2 & -c6 & c6 & c2 \\ c_3 & -c7 & -c1 & -c5 & c5 & c1 & c7 & -c3 \\ c_4 & -c4 & -c4 & c4 & c4 & -c4 & -c4 & c4 \\ c_5 & -c1 & c7 & c3 & c3 & -c7 & c1 & -c5 \\ c_6 & -c2 & c2 & -c6 & -c6 & c2 & -c2 & c6 \\ c_7 & -c5 & c3 & -c1 & c1 & -c3 & c5 & -c7 \end{bmatrix} \begin{bmatrix} x(0) \\ x(1) \\ x(2) \\ x(3) \\ x(4) \\ x(5) \\ x(6) \\ x(7) \end{bmatrix} \quad (36)$$

觀察式(36)，可以知道 DCT 就是由很多的乘加器所組成的原件，所以這裡

我們就用 2.2.4 節所提到的內積模組來進行電路的設計，如圖 4-4 為式(36)中藍色框框乘以 $x(0) \sim x(7)$ ，並且會得到計算結果 $X(0)$ 。由此可知會有 8 個圖 4-4 的模組，組成一個 1-D DCT，如圖 4-5。圖 4-5 中 ROW1~ROW8 代表的式(36)中 8×8 的矩陣，ROW1 代表第一列的係數，ROW2 代表第二列，以此類推。

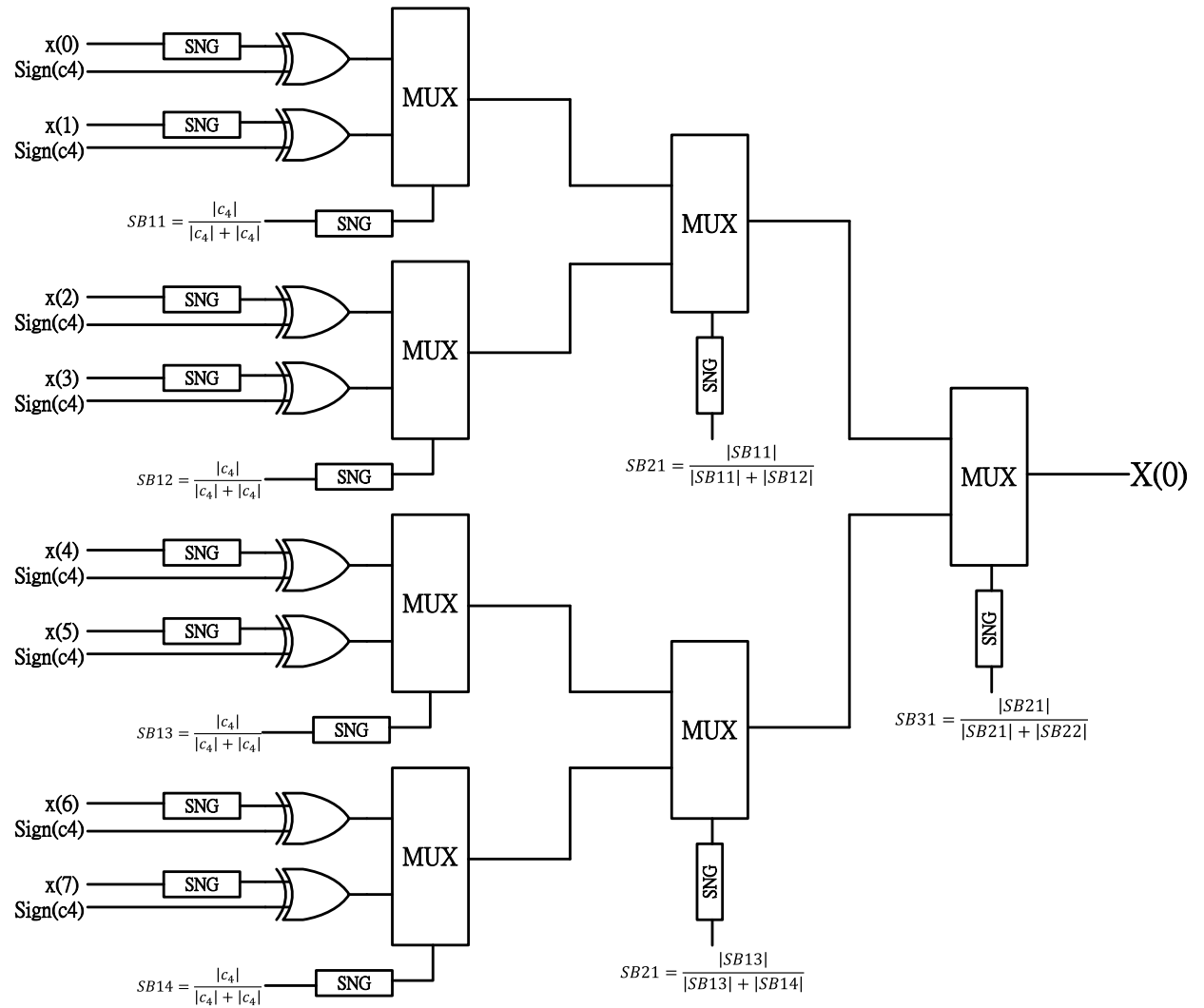


圖4-4 1-D DCT 子電路

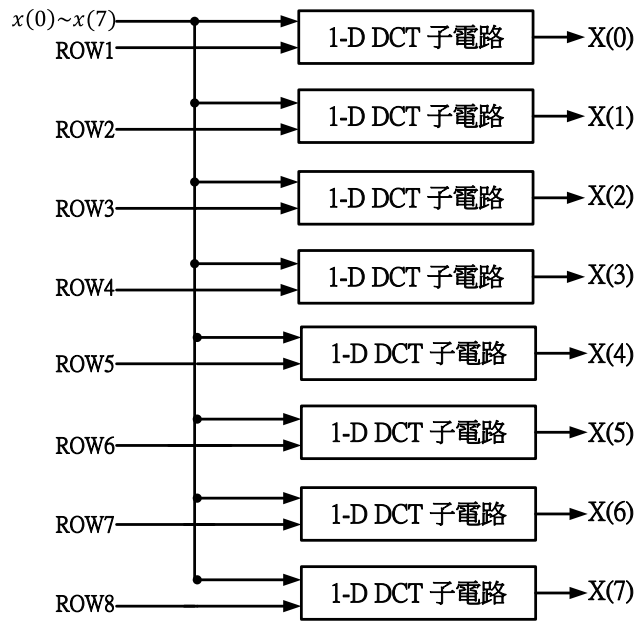


圖 4-5 1-D DCT

再來我們將介紹實作的過程，如圖 4-6 為實作步驟，首先將 8*8 Sub-graph 的矩陣數值先減掉 128，再除以一個縮放的係數(這裡為 128)將數值轉為機率的形式(-1~1 之間)。因為位元流無法表達負數，所以需要將機率全轉為 0~1 之間，這個動作我們稱做 N2P 轉換(也就是將輸入數值+1/2)。經過前面運算，這時候的 8*8 矩陣數值都為 0~1 的機率值，下一步就要將這些機率值轉為位元流的形式(這裡我們使用 1024 位元)。接下來就是做 2-D DCT 正轉換(2-D FDCT)，轉換完後我們須將這些位元流再轉回二進制數值。由於我們主要是觀察 DCT 在使用隨機算後所造成的誤差，所以接下來我們只需將產生的圖檔再經由 DCT 逆轉換(IDCT)就可以觀察到有誤差的圖了，如圖 4-7 所示。

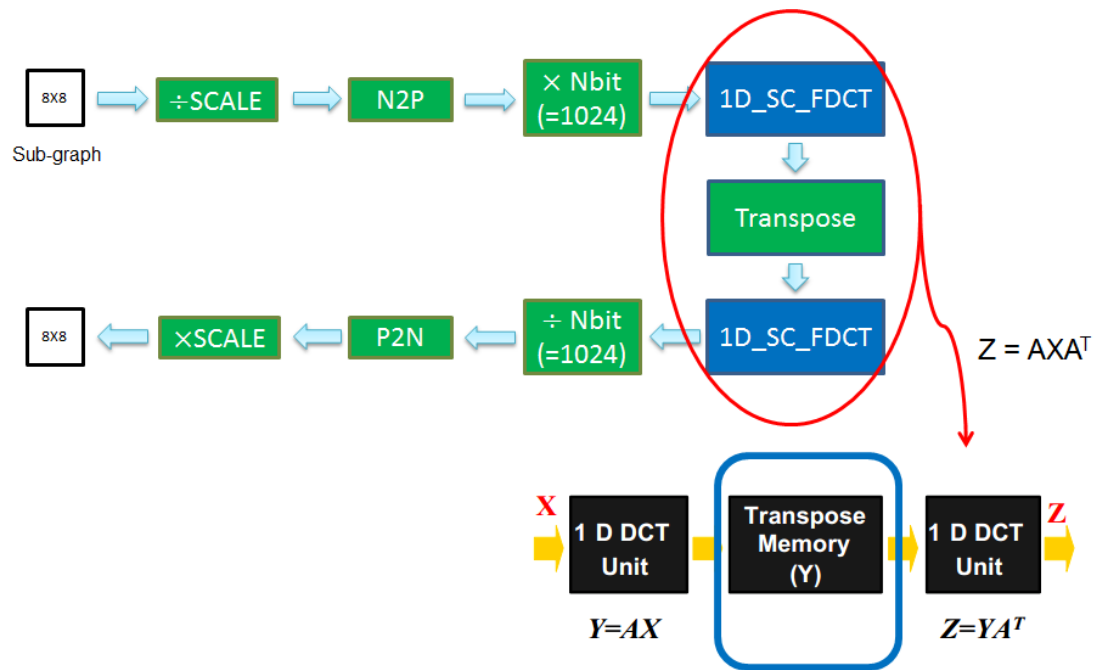


圖4-6 DCT 轉換流程圖

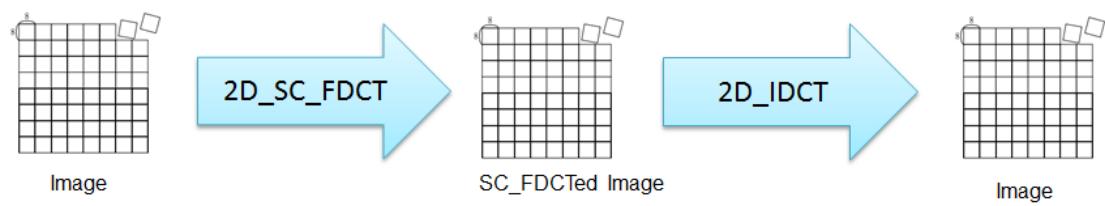


圖4-7 FDCT 轉回圖片的步驟

第五章 基於隨機計算之去尾迴旋碼解碼器實作

本論文的隨機去尾迴旋碼解碼器是根據長期演進技術(Long Term Evolution , LTE)標準下的去尾迴旋碼(Tail-biting Convolution Code) 編碼器所設計的電路。去尾迴旋碼的截斷長度(1 個碼框(Frame)的大小)形式有很多種，一般常用的為 1 個碼框 40-bits 的編碼長度。所以本論文是以此為設計之依據。

5.1 生成矩陣與同位元檢查矩陣產生

本論文是根據 LTE 標準下的(3,1,6)去尾迴旋碼編碼器架構，如圖 5-1 所示，為 40-bits 迴旋碼編碼器初始狀態。輸入的截斷長度為 40-bits，在經過迴旋碼編碼器後的資訊為 120-bits。所以我們可以根據 3.4.2 節的方法產生 1 個 40×120 的生成矩陣(G matrix)，再經由高斯消除法將生成矩陣化簡為[I|P]形式的矩陣，如 3.4.3 節的介紹。最後會得到一個 80×120 的同位元檢查矩陣(H matrix)，如附錄 B 所示，這裡我們以表 5-1 來說明附錄 B 的表示方法。表 5-1 中，左邊欄位代表第幾個檢查節點而右邊欄位代表第幾個檢查節點與第幾個變數節點有連線(Tanner Graph)。如表 5-1 所示，以第 1 個檢查節點來說，代表他與第 2、3、4、6....等有連線的關係。

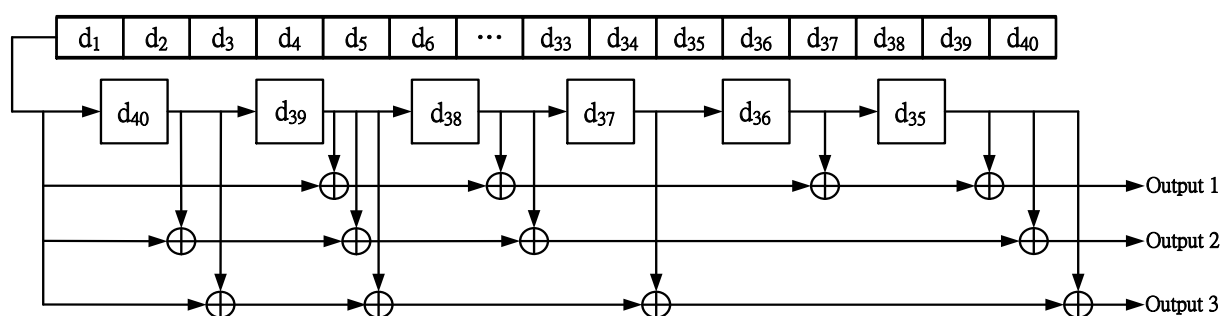


圖5-1 40-bit 迴旋編碼器初始狀態

檢查節點	檢查節點連線到變數節點的索引值													
1	2	3	4	6	7	9	10	11	14	16	17	19	20	41
2	2	3	8	10	11	14	15	19	20	42				

表5-1 附錄 B 範例

5.2 硬體架構

如圖 5-2 為基於隨機計算之去尾迴旋解碼器的硬體架構圖，大致上分為通道機率隨機亂數產生區塊 (CPSNG BLOCK)、亂數產生模組(RNE)、檢查節點區塊 (CN BLOCK)、變數節點區塊(VN BLOCK)、硬判定區塊(HD BLOCK)、提早結束模組(ET)與控制電路模組(Controller)。其中通道機率隨機亂數產生區塊、檢查節點區塊、變數節點區塊與硬判定區塊會再根據同位元檢查矩陣的大小而分為更小的模組。如圖 5-3、圖 5-4、圖 5-5 與圖 5-6 所示。

至於同位元檢查矩陣 H 的大小，由於本論文是使用 40-bits 的 LTE 去尾迴旋碼編碼器編碼。如 5.1 節所介紹的轉換方法，求得 H 矩陣為一 80×120 大小的檢查矩陣。而圖 5-2 中 M 、 N 代表的是檢查節點與變數節點的個數，所以 M 與 N 分別等於 80 與 120。而 L 代表通道機率的精確度，本論文中設定為 7。

如表 5-2 為解釋圖中訊號線傳送的訊息。

訊號線	傳送訊息
V_Pch	字碼的通道機率
SC_Pch	V_Pch 的隨機位元流
RAND_NUM_Pch	欲產生 V_Pch 所需要的隨機亂數
RAND_ADDR_EM	EM 所需的亂數位址
RAND_ADDR_IM	IM 所需的亂數位址
q、Q	經過變數節點運算所得到的資訊
r	經過檢查節點運算所得到的資訊
FINISH	解碼結束資訊
START	開始解碼的訊號
INITIAL	初始化記憶體體的訊號
STOP	解碼停止資訊

表5-2 架構圖中訊號線的意義

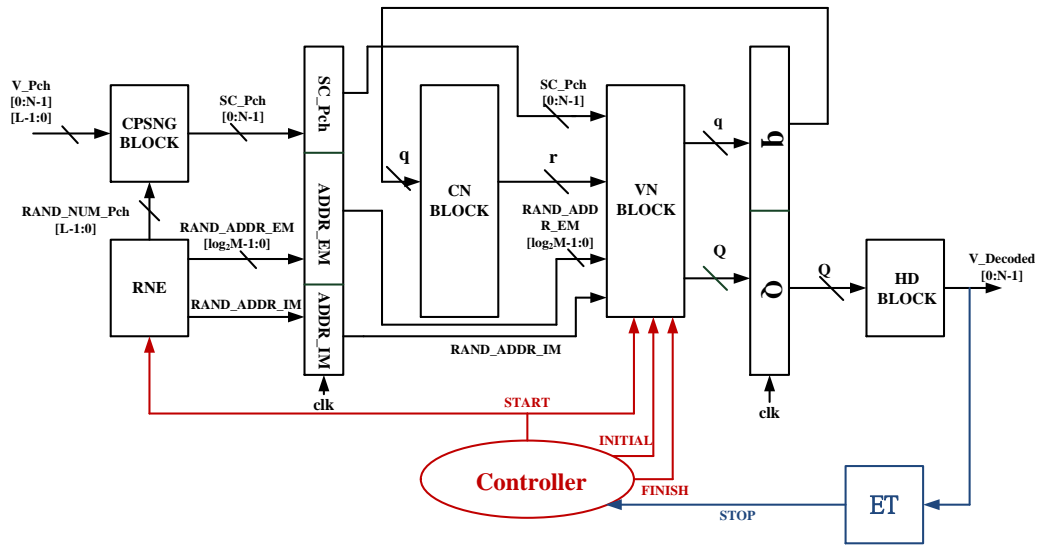


圖5-2 基於隨機計算之去尾迴旋碼解碼器硬體架構圖

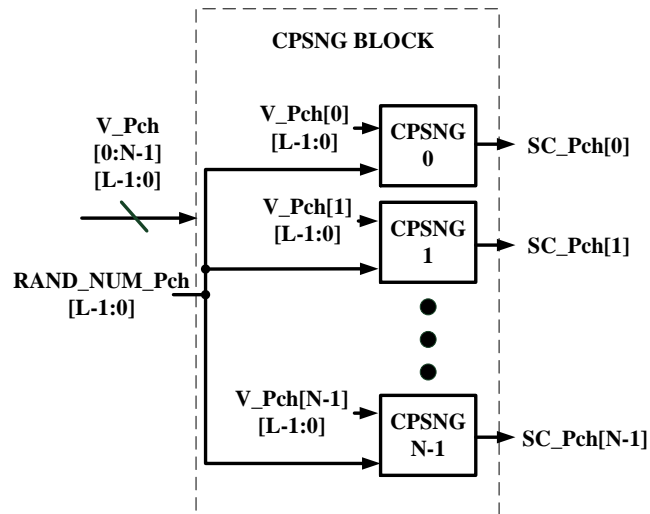


圖5-3 通道機率隨機亂數產生區塊(CPSNG BLOCK)內部架構圖

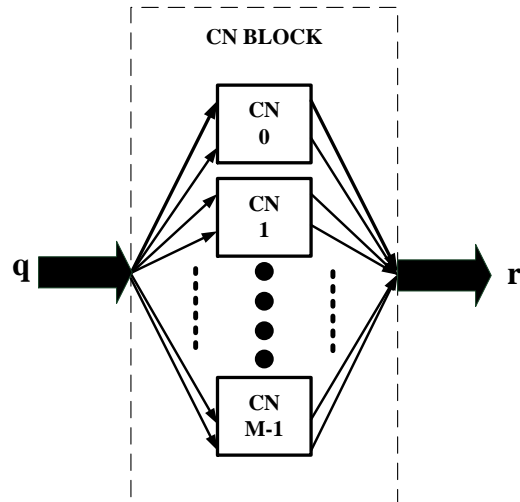


圖5-4 檢查節點區塊(CN BLOCK)內部架構圖

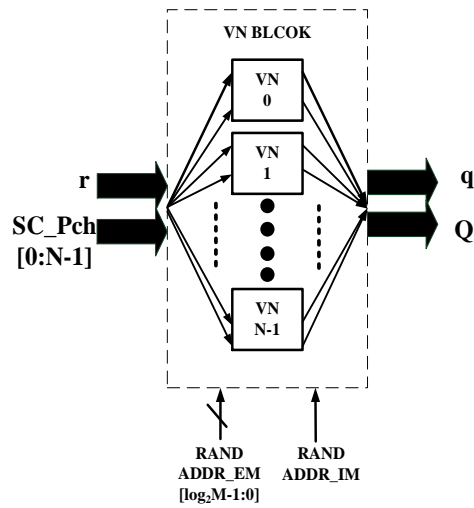


圖5-5 變數節點區塊(VN BLOCK)內部架構圖

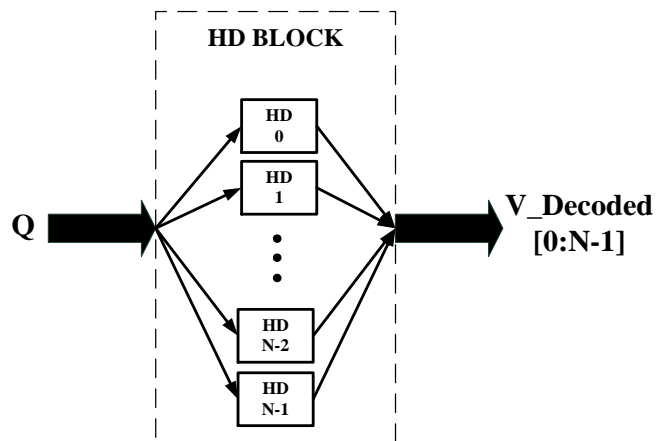


圖5-6 硬判定區塊(HD BLOCK)內部架構圖

以下將詳細介紹各個區塊或模組的設計與詳細的電路架構，分為通道機率隨機亂數產生器(CPSNG)、亂數產生引擎(RNE)、檢查節點(CN)、變數節點(VN)、字碼硬判定(HD)、提早結束機制(ET)與控制器(Controller)。

5.2.1 通道機率隨機亂數產生器(Channel Probability

Stochastic Number Generator , CPSNG)

如圖 5-7 所示，為一通道機率隨機亂數產生器的方塊圖。由於隨機運算是使用位元流的方式來計算數值，所以須先將接收到的通道機率值，藉由 CPSNG 產生位元流。其動作是將接收到的通道機率值先乘以 2^L ，其中 L 代表機率的精確度。再與接收到的隨機亂數做比較，若通道的數值大於隨機亂數的數值，輸出隨機數為 1，反之為 0。簡單來講 CPSNG 就是在做解碼的運算前，先將機率值轉為位元流的一個模組。

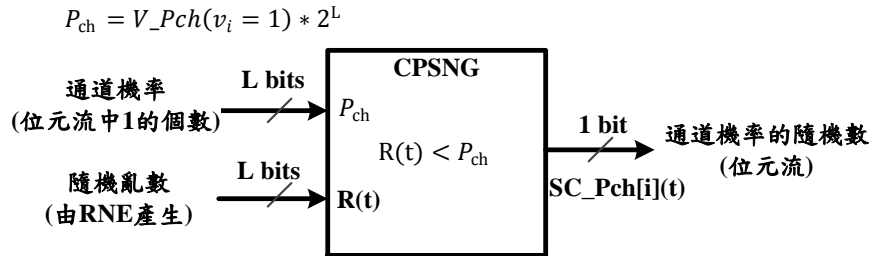


圖5-7 通道機率隨機亂數產生器的方塊圖

5.2.2 亂數產生引擎(Random Number Engine, RNE)

隨機亂數產生引擎(Random Number Engine , RNE)，主要產生三種亂數，分別為給 CPSNG 所需的亂數與給 EM、IM 所需的亂數位址。在本論文中，CPSNG 所設定的精確度 L 為 7-bits，所以如圖 5-8 中的 LFSRs_7 bits 就是給 CPSNG 的隨機亂數值(0~127)。而 EM 與 IM 的大小分別設定 32-bits 與 2-bits，所以 EM 的

LFSR 設定為 5-bits(0~31)。而 IM 因為只有 2-bits，所以選擇線只會有 0 與 1 兩種數值，因此這裡我們使用一個 zero_one_gen 的電路來產生所需的位址。而圖中的 SEED_7bits 與 SEED_5bits 是 LFSRs_7 bits 與 LFSRs_5 bits 的亂數種子。

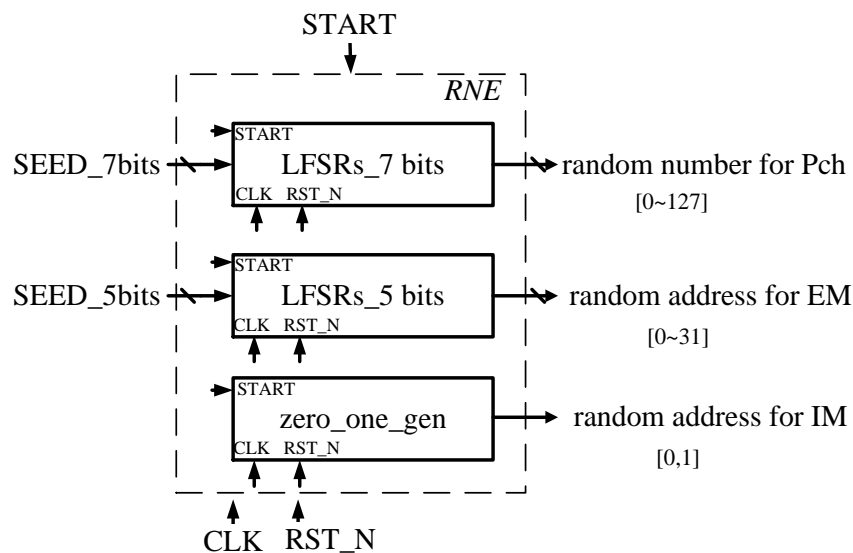


圖5-8 亂數產生引擎方塊圖

5.2.3 檢查節點(Check Node , CN)

由於本論文是使用 LTE 標準下的去尾迴旋碼編碼器所設計的隨機去尾迴旋碼解碼電路。所以我們觀察 40-bits 編碼長度所產生的同位元檢查矩陣 H，可以發現檢查節點的輸入個數分布甚廣，如表 5-3 所示。

檢查節點輸入數	檢查節點個數(共 80 個)
6	2
8	8
10	6
12	8
14	12
16	4
18	9

20	9
22	10
24	7
26	5

表5-3 檢查節點輸入個數分布

檢查節點電路的設計非常簡單，不論檢查節點的輸入個數為多少，只要依據圖 5-9 的設計方法設計就可以了。

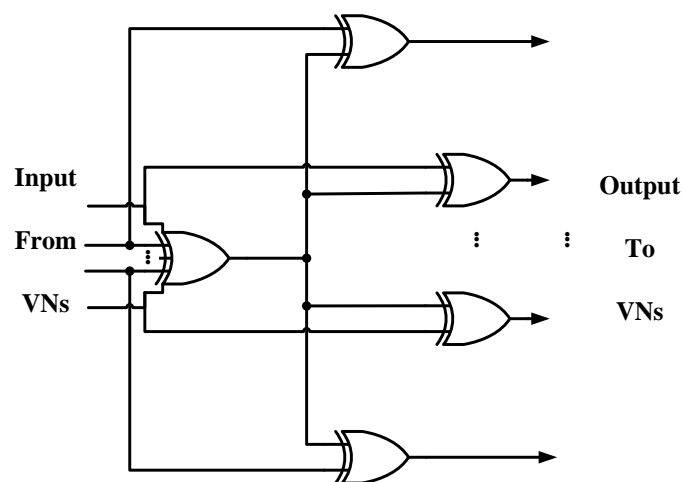


圖5-9 多輸入檢查節點的隨機運算電路

5.2.4 變數節點(Variable Node, VN)

由同位元檢查矩陣 H 可知，變數節點的輸入數會有超過 1 個以上的輸入，如表 5-4 所示。其中有 40 組變數節點的輸入會在 1 個以上，甚至多達 48 個。因此若實作一個 48 個輸入的變數節點，硬體將會變得非常大，而且也不一定會準確。

所以本論文提出另一種實作方法，那就是只取變數節點的部分資訊。如圖 5-10 所示，一開始我們在每個變數節點固定只取 6 條輸入，但由於取得資訊過少，解碼效能並不好。因此我們增加取得每個變數節點的輸入資訊如圖 5-11 所示，

其中 Majority 模組是由加法器構成，主要目的是統計輸入的資訊 0 或 1 誰為多數，並且把多數的資訊傳送給變數節點。

至於變數節點的輸入數要如何選擇，這裡我們嘗試了兩種選取模式，第一種是在每個變數節點固定取 18 條輸入，但是由表 5-4 中可以發現，有些變數節點的輸入數少於 18 條，而這些少於 18 條輸入的變數節點，我們的處理方式是重覆選取相同的輸入線，也就是說，如果變數節點的輸入數只有 16 條，那這 16 條中，就會有 2 條線會被重複利用到。第二種是根據變數節點的輸入數多寡彈性選擇部分輸入數，其中輸入數再 14~27 條的，我們固定選取其中 18 條，不足的部分一樣是重覆選取，而輸入數再 32~48 我們固定選取其中 30 條。而取得的部分輸入資訊再將它平均分配給 6 個 Majority 模組，而 Majority 模組會根據輸入的多數決當作輸出，傳送給變數節點。

經過實際模擬之後，本論文發現每個變數節點固定取 18 條的解碼效能會最好，因此變數節點的硬體設計就是依此為設計的依據如圖 5-12 所示。

變數節點輸入數	變數節點個數	變數節點輸入數	變數節點個數
1	80	34	1
14	2	35	5
15	1	36	1
16	2	37	2
18	3	39	1
19	1	41	2
21	3	43	2
26	2	44	5
27	4	47	1
32	1	48	1

表5-4 變數節點輸入數分布

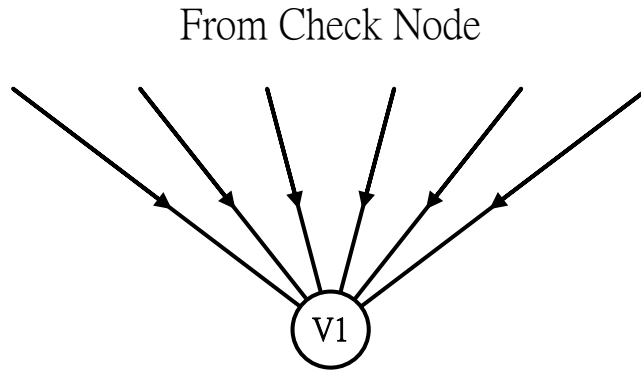


圖5-10 固定取 6 條資訊的變數節點運算

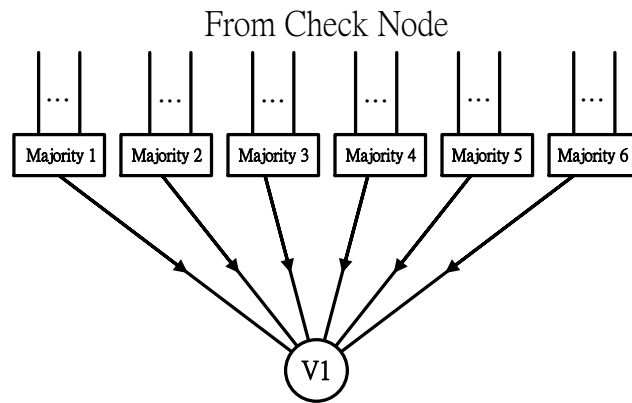


圖5-11 取多條資訊的變數節點運算

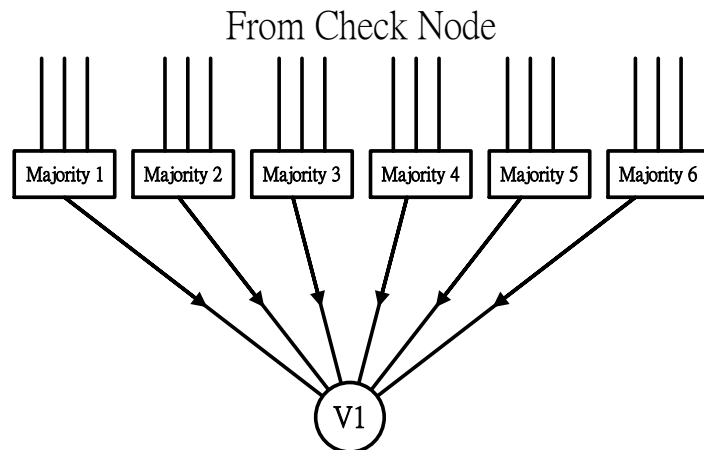


圖5-12 固定取 18 條資訊的變數節點運算

至於變數節點連接到檢查的 18 條輸入要如何選擇，我們就以本論文所設計的 40-bits 去尾迴旋碼解碼器來說明。由同位元檢查矩陣可知，有 40 組變數節點的輸入數會在 1 個以上。由於每組變數節點要固定取 18 個輸入，所以這 40 組變

數節點共需取 720 條從檢查節點來的資訊。而我們將這 720 條資訊平均分配給 80 個檢查節點，所以每個檢查節點會分配到 9 條需要傳給變數節點的連線。這樣的分配方法可以讓每個檢查節點傳送給變數節點的權重相同，而不會有過度偏重於某幾個檢查節點的問題。

而變數節點與檢查節點的詳細連線表格如附錄 A 所示，這裡我們舉其中幾個變數節點來說明表格的意義，如表 5-5 所示，青色框框的 3、6、39 代表第 1 個變數節點的 Majority1 模組的三條輸入分別與第 3、6、39 個檢查節點連線。而橘色框框代表第 2 變數節點的 Majority 3 模組的三條輸入與第 8、24、42 個檢查節點連線，其餘的依此類推。待連線完成之後，可以觀察到每個檢查節點的連線數都為 9 條，而每個變數節點連線數都為 18 條。而表 5-5 中有部分變數節點會重覆連到相同的檢查節點，這是因為這些變數節點原始的輸入數少於 18 條，所以不足的部分，本論文是使用重覆的方式將它補齊。

Variable Node	1	2
Majority 1	3	2
	6	3
	39	30
Majority 2	14	3
	15	6
	42	39
Majority 3	24	8
	43	24
	45	42
Majority 4	51	24
	57	30
	69	45
Majority 5	60	39
	68	39
	75	61
Majority 6	72	42

	77	67
	77	80

表5-5 固定取 18 條輸入的變數節點範例

如圖 5-13 為 6 級的 MEM 變數節點電路圖，其中 IM 與 EM 在本論文中分別設定為 2-bits 與 32-bits。

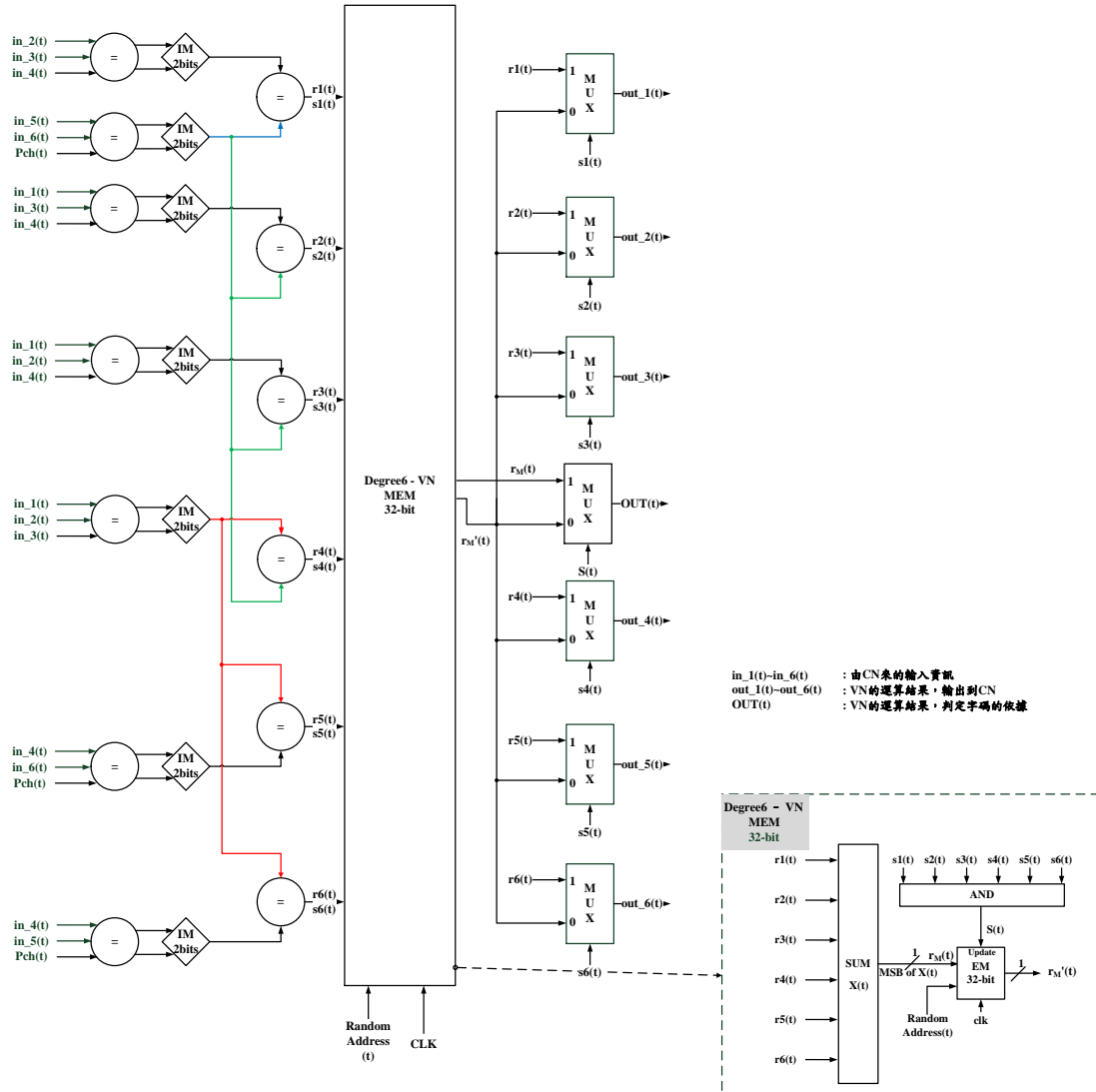


圖5-13 6 級輸入 MEM 變數節點電路圖

如圖 5-13，綠色與紅色線代表的是共用 IM 的機制，由於 IM 模組接收到的資訊重複性很高，所以這裡我們就將重複的部分共用，藉以減少硬體的大小。

5.2.5 字碼硬判定(Hard Decision , HD)

由於每個時脈週期隨機解碼器都會有一個位元的輸出 Q 傳送至字碼硬判定

模組。而此模組主要的功能就是將接收到的字碼記錄起來，並且根據記錄下來的資訊，決定輸出的結果是要 0 或是 1。

實際上，字碼硬判定模組的硬體是由上下數飽和計數器所構成，如圖 5-14 所示。如果 Q 的輸入是 1，計數器就加 1，反之就減 1。而最後輸出字碼的判定則是依據飽和計數器的符號位元來判定，如式(37)所示。

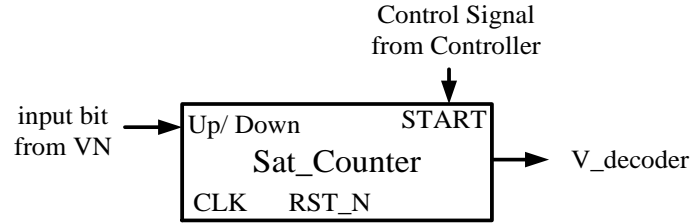


圖5-14 上下數飽和計數器

$$v_i = \begin{cases} 1, & \text{當 Counter_Value (t) } \geq 0 \\ 0, & \text{當 Counter_Value (t) } < 0 \end{cases} \quad (37)$$

5.2.6 提早結束機制(Early Termination , ET)

由於解碼器一開始接收到從通道來的訊息有可能受到雜訊的干擾較小，所以訊息本身並沒有錯誤的資訊，或是錯誤的資訊很少。因此，只需經過幾次的解碼週期，錯誤的資訊就會被修正回來。而提早結束機制(Early Termination , ET)就是當錯誤資訊修正完成時，會提早中斷解碼的動作，避免浪費多餘的時間與電量的一種機制。

本論文提早結束機制是根據 $v \cdot H^T$ 如果等於 0 就停止解碼動作，而 v 是字碼硬判定的輸出結果、 H 是同位元檢查矩陣。電路上的實作方法是以 XOR 閘來實現，以下舉一個例子來說明電路的設計方法，如圖 5-15 所示，將 v 矩陣乘上 H^T ，電路上是以互斥或閘來實作，輸出為 $R1$ 、 $R2$ 、 $R3$ 與 $R4$ 。而 NOR 閘主要是判斷每列的計算結果是否都等於 0，如果都等於 0 則 STOP 訊號拉起，傳送給控制器

(Controller)，硬體則提早結束此次的計算。

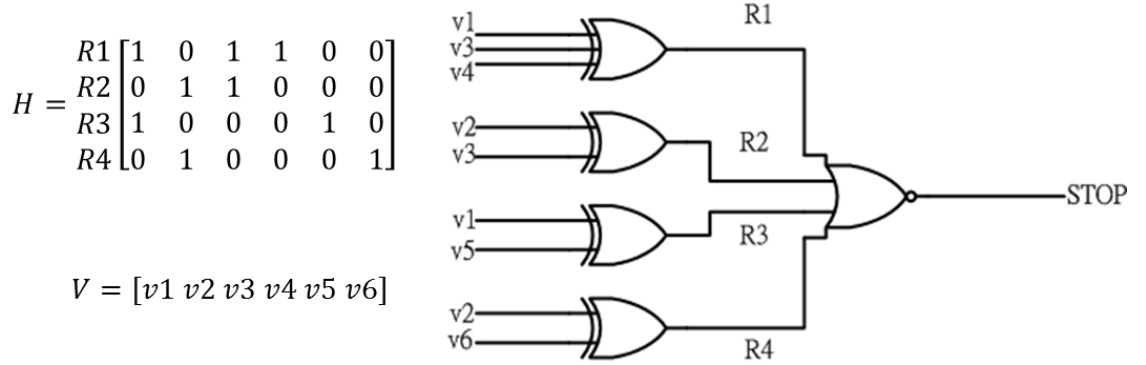


圖5-15 提早結束機制範例

5.2.7 控制器(Controller)

如圖 5-2 的架構圖中，控制器(Controller)有三個輸出訊號，分別為 START、INITIAL 與 FINISH 訊號線以及一個輸入訊號 STOP。其中 START 訊號線代表的是開始的訊號，而 FINISH 訊號線則是如果解碼到達最大的迭代次數或是 STOP 訊號拉起時，則會拉起停止硬體動作。

而 INITIAL 訊號則是初始化訊號，主要的目的是要加速解碼收斂的速度，而初始化的方法是將通道機率的部分位元流先存入 MEM 中。

5.3 軟體驗證與 FPGA 驗證

本章節介紹軟體模擬的實驗步驟，與硬體平台的設置方法。

5.3.1 軟體驗證

如圖 5-16 所示為整個使用軟體編解碼的流程圖，本論文是使用 MATLAB 來當作軟體模擬的平台。而其詳細的編解碼流程，與模擬通道雜訊的方法如下，首先產生出原始的碼字(未編碼前的長度，本論文為 40-bits)，經由去尾迴旋編碼器編碼，產生有加冗餘資訊的碼字(編碼後為 120-bits)。再來使用軟體模擬通道的環境(BPSK、AWGN)產生出有雜訊干擾的類比電壓資訊。最後由解碼器接收類

比電壓資訊，且將其轉換為機率表示的數值，經由隨機 LDPC 解碼器解碼得到正確的碼字。

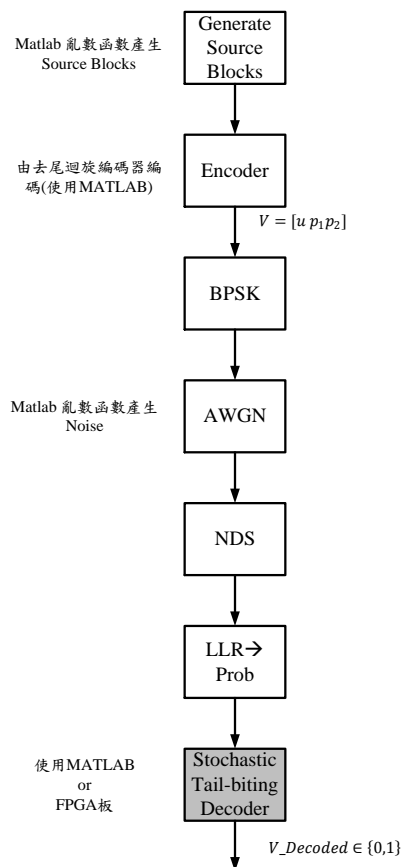


圖5-16 軟體模擬通訊系統中的編解碼流程

5.3.2 FPGA 驗證

為了要加速數據模擬的速度，與確定所寫的硬體是否可以成功的製作成晶片。在這裡我們需要進行 FPGA 板的驗證。在本論文中，我們將隨機去尾迴旋碼解碼的部分製作成硬體，以加速解碼的速度。而本論文使用的 FPGA 驗證平台是由虹晶科技所設計的 FPGA 開發板，型號為 MDK-330，內有一顆 ARM9 與 ARM11 的 CPU，而其晶片型號為 Virtex-5 XCVLX330FF1760。由於只有隨機 LDPC 解碼的部分是使用硬體來實現，因此編碼、加入通道雜訊與最後錯誤率的計算還是使用 MATLAB 來實作，其流程圖如圖 5-16 所示。

而要讓 PC 與 FPGA 板之間能進行溝通則須仰賴 AMBA 協定(Advanced

Microcontroller Bus Architecture)，而本論文所使用的 AMBA 協定為 AHB(Advanced High-Performance Bus)。所以在進行 FPGA 板驗證前，我們需要
先將解碼器的硬體外層包裝成 AHB 形式，而這層包裝稱之為 Wrapper。而 Wrapper
又分為 Master 與 Slave 兩種形式，本論文是使用 Slave 的包裝形式。

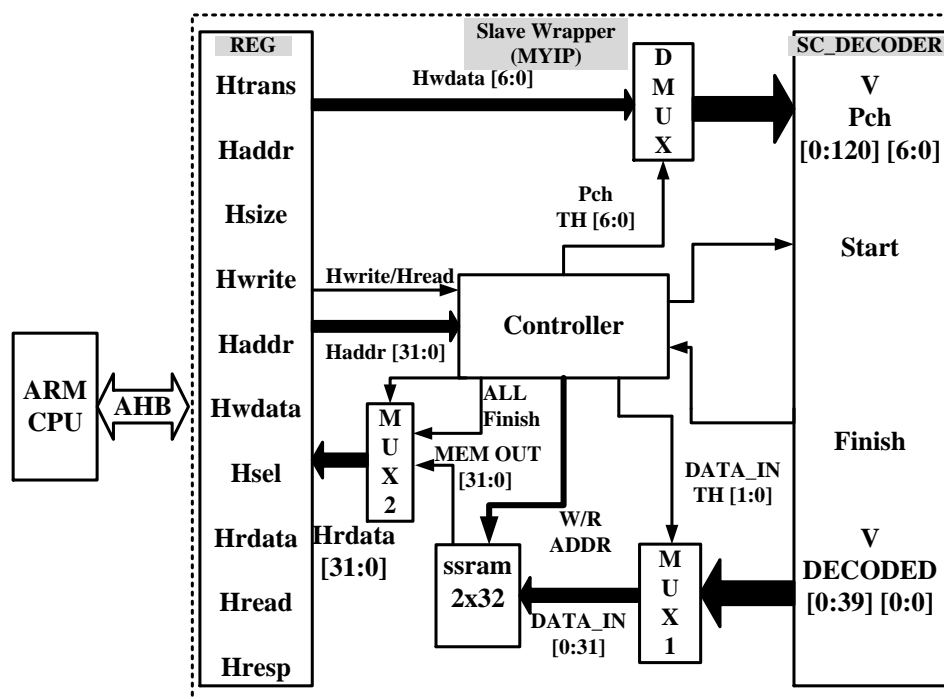


圖5-17 以 Slave 形式包裝的隨機解碼器方塊圖

圖 5-17 為使用 AHB 協定包裝的硬體，而每個區塊的功能介紹如下：

- DMUX：將通道機率輸入到對應的解碼器輸入接腳。
- MUX1：將解碼完的結果寫入記憶體對應的位置中。
- MUX2：將解碼完的字碼或是解碼完成的訊號送至 CPU。
- ssram_2x32：儲存解碼完的資訊(40bits)。
- Controller：傳送控制訊號至各個模組與接收硬體訊號和動作命令，詳細介紹如下：
 - ◆ Hwrite / Hread：寫入狀態或讀出狀態。
 - ◆ Haddr：位址資訊。
 - ◆ Start：開始解碼運算的訊號。

- ◆ Finish：解碼完成的運算訊號。
- ◆ Pch_TH：第幾個字碼的選擇資訊。
- ◆ DATA_IN_TH：選擇哪組字碼區塊（32 個字碼）當輸出。
- ◆ ALL_Finish：解碼動作是否完成且以將解碼結果寫入記憶體。
- ◆ W/R：讀寫記憶體的訊號。

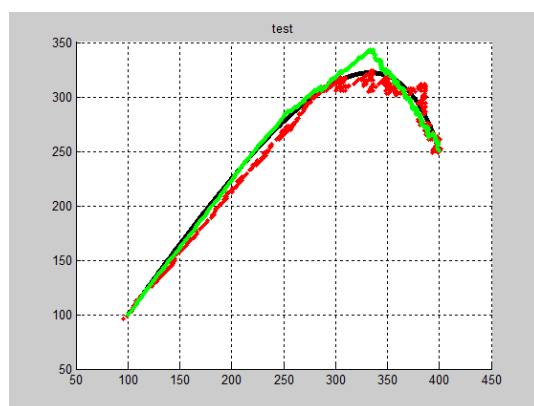
而圖 5-17 的動作流程說明如下，首先 Controller 發送 Start 訊號給解碼器電路，而解碼電路根據 DMUX 的選擇線選擇相對應的輸入，而輸入的資訊為 120 位元。緊接著，解碼器進行解碼動作，等到解碼完成後，解碼器發送 Finish 訊號給 Controller，而 Controller 再根據 MUX1 選擇線選擇解碼完的碼字結果 40bits，將其儲存在 ssram_2x32 的記憶體中，儲存完成後 Controller 再發送 ALL_Finish 給 ARM CPU 端，告知解碼完成，資料已可讀取，並可進行下一筆資料的運算。

第六章 實驗結果與效能比較

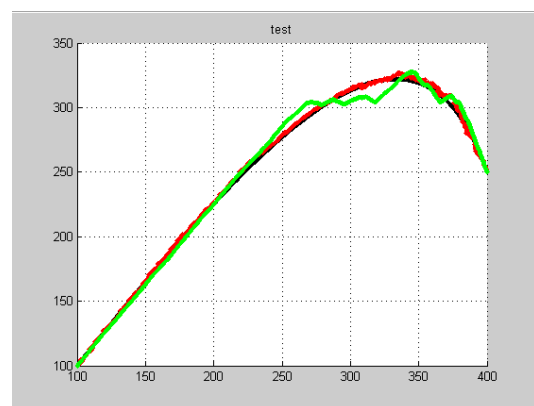
本章節主要是展示貝茲曲線與離散餘弦轉換的實驗結果，與去尾迴旋碼解碼器的效能比較。

6.1 貝茲曲線實驗結果

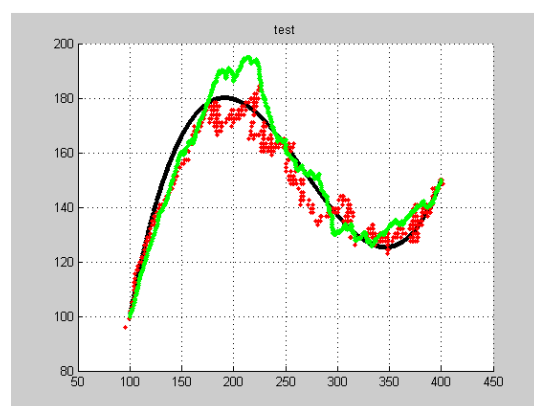
貝茲曲線的實作如 4.1 節所述，共有兩種實作的方法。第一種是使用方程式合成的方法，第二種是使用邏輯閘所組成的電路。其結果如圖 6-1 所示，其中黑色線是式(32)與式(33)計算出來的結果，紅色線是邏輯閘組合實現的結果，綠色線是使用方程式合成的結果。



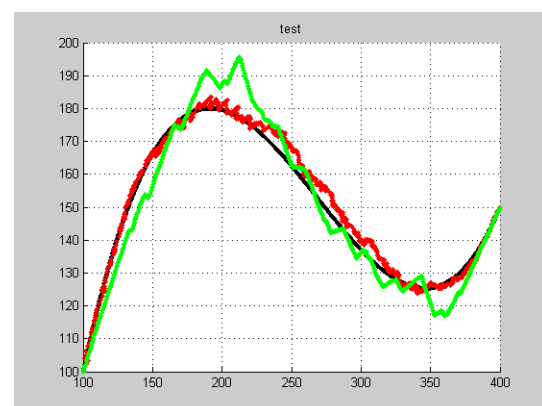
(A) LFSR 長度=12 位元(數據 1)



(B) LFSR 長度=15 位元(數據 1)



(C) LFSR 長度=12 位元(數據 2)



(D) LFSR 長度= 15 位元(數據 2)

圖6-1 貝茲曲線模擬結果

由圖 6-1(B)與(D)中可以發現，使用方程式合成的方法並不會因為位元流的長度變長而較準確。反而是邏輯閘組合的方法，在較長的位元流計算中能取得優勢。而圖 6-1(C)與(D)因為給定的數據是屬於較彎曲的形式，不像圖 6-1(A)與(B)那樣的平滑，所以在進行隨機計算的時候會造成較大的誤差。

6.2 離散餘弦轉換(DCT)實驗結果

如 4.2 節所介紹，DCT 的實作部分主要是測試數據在經過隨機計算後，會造成多少的誤差。所以在實作的部分，我們是將圖檔經由 2D-FDCT 的轉換後，其餘的部分是由軟體來實作。

如圖 6-2 為 DCT 模擬結果，由(B)圖可以發現，使用隨機計算後的結果與理想上的圖相比，會有些微的誤差。而硬體的大小如表 6-1 所示，可以觀察到使用隨機計算實現的硬體大小明顯小於使用二進制電路的硬體大小，大約節省了 1/3 的空間。



(A)理想上 DCT 運算結果



(B)隨機 DCT 運算結果

圖6-2 DCT 轉換模擬結果

	傳統二進制 DCT 電路	隨機計算 DCT 電路
CMOS Technology	Synopsys 90nm	Synopsys 90nm
Clock Frequency	50 MHz	100 MHz
Total Cell Area (um ²)	87,490um ²	25,073um ²

表6-1 傳統二進制 DCT 電路與隨機計算 DCT 電路硬體大小比較

6.3 基於隨機計算之去尾迴旋碼解碼器實驗結果與效能比較

由 6.1 與 6.2 節的實驗結果發現，隨機計算使用在圖學上都會有較明顯可見的誤差，所以近年來隨機計算的應用較常使用在錯誤更正碼的領域上。以下將分析本論文實作之隨機去尾迴旋碼解碼器效能與面積比較，另外本論文還分析去尾迴旋碼不同截斷長度使用在隨機去尾迴旋碼解碼器上的效能。

6.3.1 解碼效能分析與硬體面積比較

本論文的解碼器設計是基於 LTE 標準下的去尾迴旋碼編碼方法所設計的隨機去尾迴旋碼解碼電路。而我們所使用的去尾迴旋碼編碼長度是 1 個 Frame 為 40bits，所以產生出來的同位元檢查矩陣大小為 80x120，故本論文的隨機解碼電路就是依此矩陣所設計的。

如圖 6-3 為使用隨機去尾迴旋碼解碼器解碼的效能比較圖，圖中紅色線為使用 Viterbi 解碼器的結果，也是我們主要比較的目標，而黑色線是使用和積演算法的結果，其變數節點的計算資訊是所有檢查節點傳入的資訊，與隨機去尾迴旋碼解碼器只取部分的輸入資訊來運算有很大的不同，因此其解碼結果代表隨機去尾迴旋碼解碼器能解碼的最理想效能。綠色線、藍色線與粉紅色線為使用隨機去尾迴旋碼解碼器解碼且迭帶次數((或稱解碼週期)Decoding Cycle , DC)為 128、256 與 512 的解碼效能。而淺藍色線是有加入提早結束機制且迭帶次數最多為 512

的解碼效能。由圖 6-3 中我們可以發現到，在迭帶次數為 128 的時候，還有部分 E_b/N_0 的解碼效能比 Viterbi 差，但在迭帶次數等於 256 之後的效能都比 Viterbi 要好。而有加入提早結束機制的解碼器，可以在不影響解碼效能的情況下節省掉不少的解碼週期，如表 6-2 所示。而使用和積演算法的效能，則優於隨機去尾迴旋碼解碼器，其原因在於和積演算法是使用全部的資訊做運算，所以計算的準確度較高，進而增進解碼的效能。

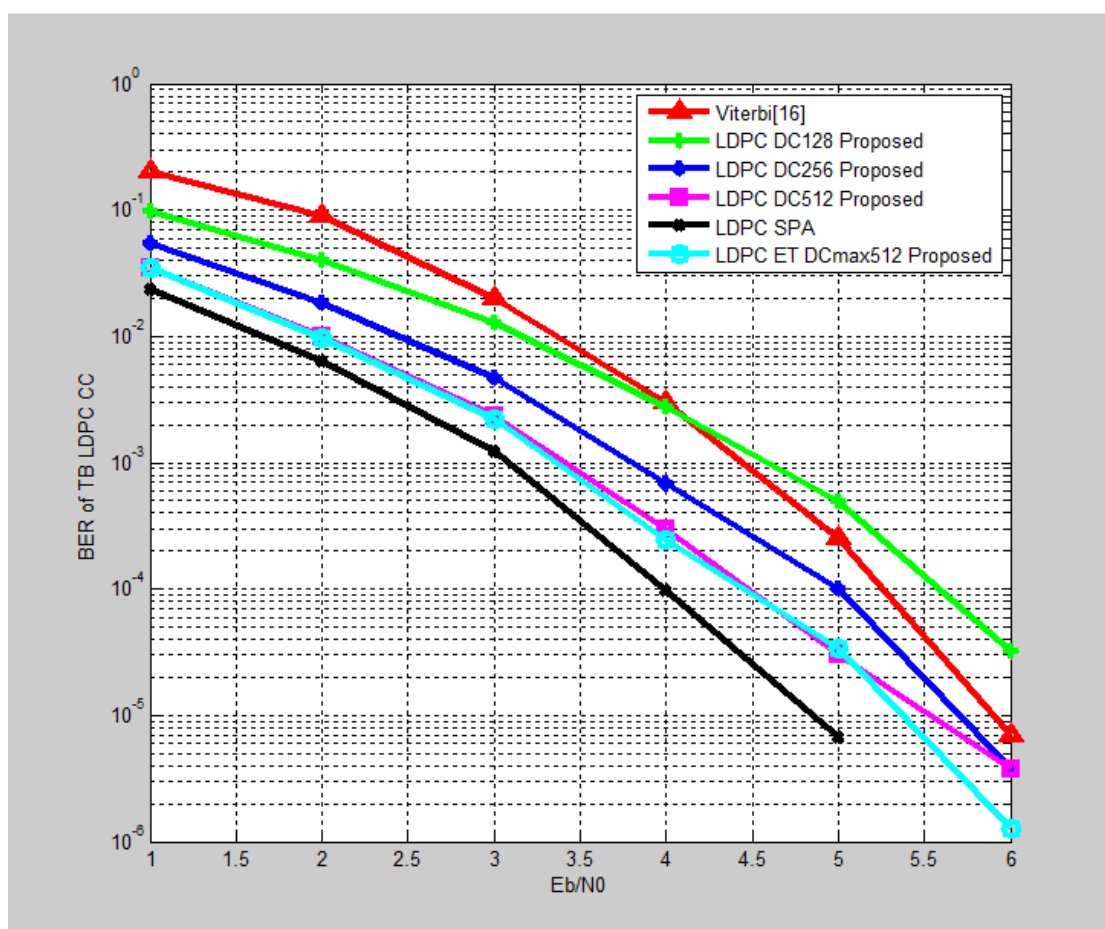


圖6-3 不同迭代次數對於解碼效能的影響

(Viterbi 數據是參考[16]，而 LDPC 的數據是使用 FPGA 硬體驗證，並且每個 E_b/N_0 測試 20000 組數據)

	$E_b/N_0=1$	$E_b/N_0=2$	$E_b/N_0=3$	$E_b/N_0=4$	$E_b/N_0=5$	$E_b/N_0=6$
平均解碼週期(DC)	180	110	71	51	42	38

表6-2 有提早結束機制的解碼器平均解碼週期

接下來為硬體面積的比較，本論文的隨機去尾迴旋碼解碼器是跟傳統去尾迴旋解碼器與預先追朔去尾迴旋解碼器[16]做比較。由表 6-3 可以發現，本論文所提出的隨機去尾迴旋碼解碼器硬體面積明顯小於傳統去尾迴旋碼解碼器與預先追朔去尾迴旋碼解碼器，分別解省了 68%與 48%的硬體空間，而其時脈週期 (Clock Frequency)可以達到 455MHz。

	傳統去尾迴旋 解碼器[19]	預先追朔去尾迴旋 解碼器[16]	隨機計算去尾 迴旋碼解碼器
CMOS Technology	TSMC 90nm	TSMC 90nm	TSMC 90nm
Clock Frequency	100 MHz	100 MHz	455 MHz
Total Cell Area (μm^2)	432,163 μm^2	267,926 μm^2	137,754 μm^2

表6-3 去尾迴旋解碼器與隨機 LDPC 解碼器硬體大小比較

6.3.2 去尾迴旋碼不同截斷長度之效能比較

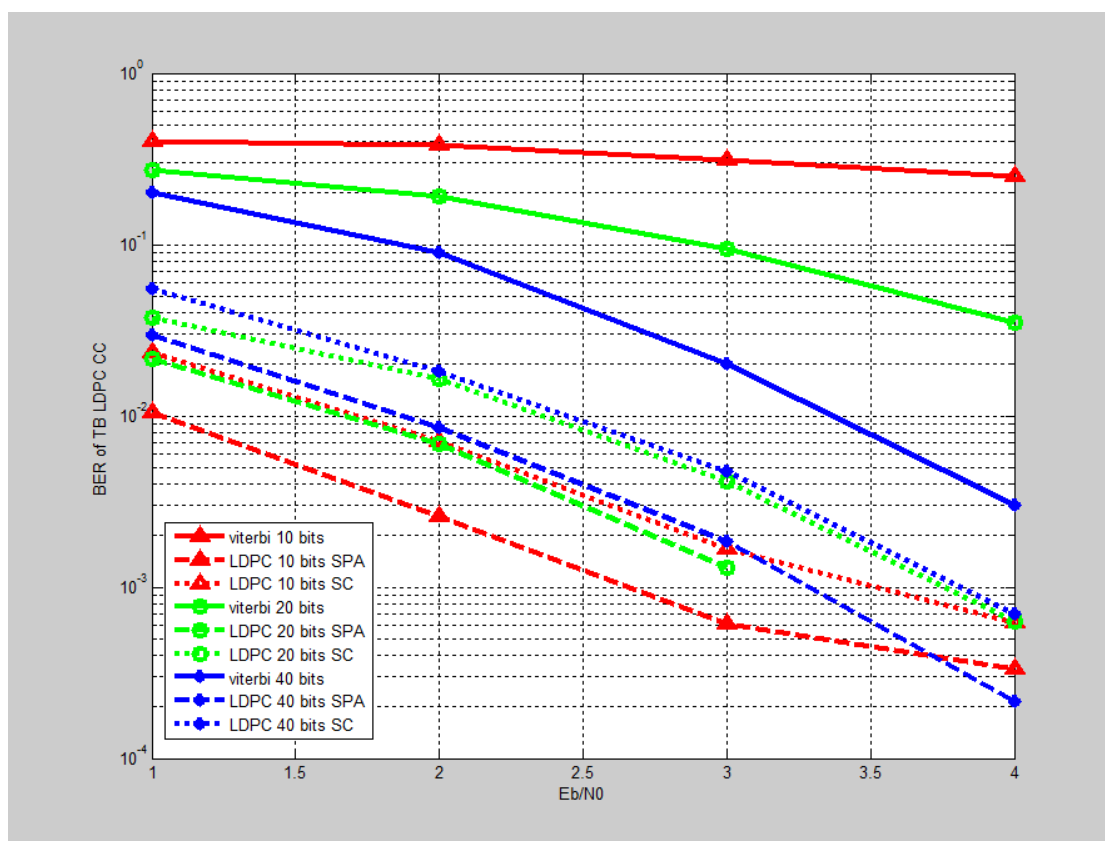


圖6-4 Viterbi 與隨機 LDPC 解碼效能比較

(Viterbi 數據是參考[16]，而 LDPC 截斷長度 10 位元與 20 位元是使用 MATLAB 驗證，截斷長度 40 位元是使用 FPGA 驗證。而迭代次數皆設為 256)

圖 6-4 為 Viterbi 與隨機去尾迴旋碼解碼效能比較，其中點線部分是代表隨機去尾迴旋碼解碼效能，虛線代表 LDPC SPA 解碼效能，實線部分代表是 Viterbi 解碼效能。而紅色線代表是截斷長度為 10 位元的解碼結果、綠色線代表的是截斷長度為 20 位元的解碼結果、而藍色線代表的是截斷長度為 40 位元的解碼結果。

由圖 6-4 中可以發現，隨機去尾迴旋碼的解碼效能皆優於 Viterbi 解碼器。而使用隨機計算所設計的電路，在截斷長度為 10 位元時，隨機去尾迴旋碼的解碼效能還優於截斷長度為 20 位元與 40 位元解碼結果。一般來說截斷長度越長，解

碼效果應該會較好，但在這裡結果卻是相反。所以我們是否解碼器的設計可以使用 10 位元的截斷長度來取代 40 位元的截斷長度，這就是未來要討論的問題了。

第七章 結論與未來展望

7.1 結論

本論文嘗試實作了三種由隨機計算實現的應用，第一種應用是使用在貝茲曲線的電路上，並且應用了兩種實作的方法，分別為方程式合成與邏輯閘組合的電路。第二種應用是 DCT 轉換的實作，藉由數個內積單元電路組成的 DCT 電路，可以以較小的硬體空間，實作出 DCT 轉換的功能。第三種是錯誤更正碼的應用，我們設計出的隨機去尾迴旋碼解碼器，可以解碼由去尾迴旋編碼器所編碼的資訊。並且在硬體大小優於 Viterbi 解碼器的情況下，解碼效能還優於 Viterbi 解碼器。

7.2 未來展望

✧ 增加隨機計算使用在影像處理的精確度

由於使用隨機計算會有一定的誤差，所以在計算貝茲曲線等需要高精確度的計算中，隨機計算就比較沒有優勢。所以該如何改進精確度的問題，是接下來相當重要的一個研究方向。

✧ 增加隨機計算使用在去尾迴旋碼解碼上的效能

雖然使用 LDPC 解碼器的解碼結果已經比 Viterbi 解碼器還要好了，但是離 LDPC，SPA 演算法的理論數值還是有很大的差距。所以，該如何增進解碼效能，也是接下來的一個研究課題。

✧ 找出更多可以使用隨機運算的應用

隨機計算是一種新的計算方法，它能以較少的硬體成本完成一些複雜的計算。雖然說隨機計算的精確度仍有待加強，但是在較不需要非常精確數值的系統上，隨機計算仍然有一定的優勢。例如像錯誤更正碼等的應用。

參考文獻

- [1] A. Alaghi and J. P. Hayes, “Survey of Stochastic Computing,” *ACM Transactions on Embedded Computing Systems*, vol. 12, no. 92, pp. 92:1-92:19, May. 2013.
- [2] C. Winstead, V.C. Gaudet, A. Rapley, and C. Schlegel, “Stochastic Iterative Decoders,” in *IEEE International Symposium on Information Theory. 2005. ISIT 2005.*, pp.1116-1120, Sept. 2005.
- [3] S.S. Tehrani, W.J. Gross, and S. Mannor, “Stochastic Decoding of LDPC Codes,” in *IEEE Communications Letters, 2006*, pp. 716-718, Oct. 2006.
- [4] F.-P. Leduc, S. Hemati, W.J. Gross, and S. Mannor, “A Relaxed Half-Stochastic Iterative Decoder for LDPC Codes,” in *IEEE Global Telecommunications Conference, 2009, GLOBECOM. 2009*, pp. 1-6, Dec. 2009.
- [5] S.S. Tehrani, A. Naderi, G.-A. Kamendje, G.-A., S.Hemati, and S. Mannor, W.J. Gross, “Majority-Based Tracking Forecast Memories for Stochastic LDPC Decoding,” *IEEE Transactions on Signal Processing, 2010, TSP 2010*, vol. 58,no. 9, pp.4883-4896, Sept. 2010.
- [6] A. Naderi, S. Mannor, M. Sawan and W.J. Gross, “Delayed Stochastic Decoding of LDPC Codes,” *IEEE Transactions on Signal Processing, 2011, TSP. 2011*, vol. 59, no. 11 , pp. 5617-5626, Nov. 2011.
- [7] M. Maamoun, R. Bradai, A. Naderi, R. Beguenance and M. Sawan, “Controlled Start-Up Stochastic Decoding of LDPC Codes,” in *IEEE 11th International New Circuits and Systems Conference, 2013, NEWCAS. 2013*, pp. 1-4, June. 2013.
- [8] S.S. Tehrani, A. Naderi, G.-A. Kamendje, S. Mannor, and W.J. Gross, “Tracking Forecast Memories in Stochastic Decoders,” in *IEEE International Conference on Acoustics, Speech and Signal Processing, 2009, ICSSAP 2009*, pp. 561-564 ,

April. 2009.

- [9] S.S. Tehrani, S. Mannor, and W.J. Gross, "Fully Parallel Stochastic LDPC Decoders," *IEEE Transactions on Signal Processing*, 2008, TSP 2008, vol. 56, no. 11, pp. 5692-5703, Nov. 2008.
- [10] P. Li, D.J.Lilja, W. Qian, K.Bazargan, and M.D.Riedel, "Computation on Stochastic Bit Streams Digital Image Processing Case Studies," *IEEE Transactions on Very Large Scale Integration(VLSI) Systems*, vol.22, no .3, pp.449-462, March. 2014.
- [11] Y.-N. Chang, and K.K. Parchi, "Architectures for Digital Filters Using Stochastic Computing," in *2013 IEEE International Conference on Acoustics, Speech and Signal Processing(ICASSP)*, pp.2697-2701, May. 2013.
- [12] B. D. Brown, and H. C. Card, "Stochastic Neural Computation I: Computational Elements," *IEEE Transactions on Computers*, vol. 50, no. 9, pp. 891–905, September. 2001.
- [13] A. Alaghi, and J.P. Hayes, "Exploiting Correlation in Stochastic Circuit Design," in *2013 IEEE 31st International Conference on Computer Design(ICCD)*, pp.39-46, Oct. 2013.
- [14] A. Alaghi , C. Li , and J. P. Hayes, "Stochastic Circuits for Real-Time Image-Processing Applications," in *2013 50th ACM/EDAC/IEEE Design Automation Conference(DAC)*, pp.1-6, May 29 2013-June 7 2013.
- [15] 王暉翔, "基於 IEEE 802.16e 標準的隨機運算低密度同位元檢查碼解碼器設計", *國立中山大學資訊工程學系碩士論文*, 2014.
- [16] 馮皓, "具高效儲存及低功率架構之去尾迴旋解碼器", *國立中山大學資訊工程學系碩士論文*, 2014.
- [17] K.K.Parhi , and Y.Liu, "Architectures for IIR Digital Filters Using Stochastic Computing", in *2014 IEEE International Symposium on Circuits and*

Systems(ISCAS), pp.373-376, June 2014.

- [18] T.-H.Chen, and J. P. Hayes, "Design of Stochastic Viterbi Decoders for Convolutional Codes" , in *2013 IEEE 16th International Symposium on Design and Diagnostics of Electronic Circuits & Systems(DDECS)* , pp.66-71, April 2013.
- [19] Y. Gang ,A. T. Erdogan, and T.Arslan , "An Efficient Pre-Traceback Architecture for The Viterbi Decoder Targeting Wireless Communication Applications" , *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol.53, No.9, PP.1918-1927, Sept 2006.

附錄 A 固定取 18 條輸入的變數節點與檢查節點對應表格

Variable Node	1	2	3	4	5	6	7	8	9	10
Majority 1	3	30	2	9	3	1	7	2	1	15
	6	2	9	11	7	49	1	24	6	6
	39	3	2	9	12	12	15	3	3	6
Majority 2	14	39	5	13	5	4	16	24	3	21
	15	3	12	21	10	51	4	33	12	15
	42	6	5	21	15	21	30	6	9	9
Majority 3	24	42	18	19	11	6	20	48	4	24
	43	8	13	24	27	57	18	34	15	60
	45	24	18	24	27	33	36	18	12	15
Majority 4	57	45	33	27	12	7	22	57	6	30
	69	24	51	45	29	66	20	35	21	65
	51	30	33	27	39	45	42	33	18	21
Majority 5	68	61	51	37	15	8	36	60	51	51
	75	39	58	48	30	78	28	38	42	68
	60	39	51	36	48	51	45	48	30	24
Majority 6	72	67	80	48	21	10	52	79	54	54
	77	80	60	55	36	79	42	41	51	70
	77	42	57	48	60	57	54	60	54	27

11	12	13	14	15	16	17	18	19	20	21	22	23
6	18	9	9	14	17	8	32	17	1	4	7	10
9	3	3	2	5	5	1	4	1	4	7	10	13
1	4	7	1	2	1	4	13	10	19	16	19	22
15	33	12	12	18	40	11	34	25	2	5	8	11
18	27	12	9	17	16	8	11	2	5	8	11	14
2	5	8	11	4	7	10	22	14	20	23	25	29
33	39	27	36	21	49	20	44	62	10	35	41	26
30	33	14	18	23	27	36	14	7	10	13	16	19
5	7	11	13	13	8	19	23	16	28	32	34	38
36	42	48	57	27	53	23	46	64	13	38	50	63
33	42	19	21	25	30	39	54	54	17	20	23	28
10	8	17	14	16	20	22	25	26	31	34	35	40
39	45	54	66	30	76	26	56	65	29	47	58	69
73	48	26	45	32	36	45	57	57	37	22	26	46
17	11	36	17	20	23	28	26	28	32	71	38	41
42	60	60	73	45	78	29	75	70	77	55	74	71
76	67	60	63	54	39	48	71	74	47	31	44	53
25	14	57	26	29	54	31	31	29	35	74	43	44

24	25	26	27	28	29	30	31	32	33	34	35	36
13	16	19	22	25	28	31	34	37	40	43	46	49
16	19	22	25	28	31	34	37	40	43	46	49	52
32	34	37	40	37	46	49	52	37	40	43	46	58
14	17	20	23	26	29	32	35	38	41	44	47	50
17	20	23	26	29	32	35	38	41	44	47	50	53
44	35	46	50	43	47	59	61	38	41	59	47	64
16	19	22	31	34	44	40	37	41	56	47	69	52
22	32	31	34	41	37	43	46	53	58	52	62	61
50	44	47	64	53	56	62	62	41	49	61	49	67
31	25	23	38	59	62	58	43	46	68	55	73	71
25	52	35	49	44	40	47	71	56	67	65	66	63
69	50	53	70	73	58	66	63	53	52	68	55	70
43	28	28	69	66	71	65	50	49	77	64	78	73
29	59	38	56	64	55	50	72	64	78	69	76	68
72	65	55	72	79	68	71	67	73	69	76	63	78
59	32	35	75	70	72	74	53	75	79	67	80	76
40	62	50	61	72	59	75	74	70	80	73	79	77
74	69	56	76	80	75	78	70	74	73	77	77	80

37	38	39	40
52	55	58	61
55	58	61	62
52	55	59	61
53	56	59	64
56	59	64	65
56	58	63	62
61	67	62	66
68	63	66	67
66	65	67	64
63	72	63	68
74	70	71	69
68	66	71	65
77	79	65	70
76	73	72	75
72	75	75	79
78	80	76	74
80	79	77	78
79	78	76	80

附錄 B 40-bits 去尾迴旋碼解碼器之同位元檢查矩陣

檢查節點	檢查節點連線到變數節點的索引值													
1	2	3	4	6	7	9	10	11	14	16	17	19	20	41
2	2	3	8	10	11	14	15	19	20	42				
3	1	2	5	8	9	12	13	43						
4	1	2	4	5	6	7	8	9	11	12	15	17	18	20
	21	44												
5	1	3	5	9	10	11	12	15	16	20	21	45		
6	1	2	6	8	9	10	11	46						
7	2	3	5	6	7	10	12	13	16	18	19	21	22	47
8	1	2	6	10	11	12	13	16	17	21	22	48		
9	1	3	4	8	9	10	11	13	14	49				
10	5	6	8	9	10	11	13	14	17	19	20	22	23	50
11	4	5	9	11	12	13	14	17	18	22	23	51		
12	2	3	5	6	9	13	14	52						
13	2	3	4	6	11	12	14	15	18	20	21	23	24	53
14	1	2	4	7	10	12	13	14	15	18	19	23	24	54
15	1	3	4	5	6	7	9	10	11	55				
16	5	6	7	9	12	13	15	16	19	21	22	24	25	56
17	4	5	7	10	11	13	14	15	16	19	20	24	25	57
18	2	3	7	8	9	11	12	14	15	58				
19	1	2	4	9	13	14	16	17	20	22	23	25	26	59
20	7	8	10	11	12	14	15	16	17	20	21	25	26	60
21	1	2	4	5	6	9	10	14	15	61				
22	1	2	5	7	8	10	14	15	17	18	21	23	24	26
	27	62												
23	1	3	6	9	11	12	13	15	16	17	18	21	22	26
	27	63												
24	1	2	4	8	10	64								
25	1	3	4	5	6	8	9	11	15	16	18	19	22	24
	25	27	28	65										
26	1	2	6	8	9	10	12	13	14	16	17	18	19	22
	23	27	28	66										
27	2	3	4	5	8	9	10	12	13	15	16	67		
28	2	3	7	12	16	17	19	20	23	25	26	28	29	68

29	2	3	5	6	8	11	13	14	15	17	18	19	20	23
	24	28	29	69										
30	1	2	5	7	9	10	11	15	16	70				
31	5	6	10	13	17	18	20	21	24	26	27	29	30	71
32	1	3	5	8	10	12	14	15	16	18	19	20	21	24
	25	29	30	72										
33	2	3	5	6	8	11	12	73						
34	8	9	11	14	18	19	21	22	25	27	28	30	31	74
35	1	3	4	8	9	10	11	13	15	16	17	19	20	21
	22	25	26	30	31	75								
36	2	3	4	5	7	9	11	13	14	16	17	76		
37	2	3	4	6	8	9	12	15	19	20	22	23	26	28
	29	31	32	77										
38	2	3	7	8	9	11	12	14	16	17	18	20	21	22
	23	26	27	31	32	78								
39	1	2	5	11	12	16	17	79						
40	2	3	4	5	7	8	13	16	20	21	23	24	27	29
	30	32	33	80										
41	2	3	4	5	8	9	10	12	13	15	17	18	19	21
	22	23	24	27	28	32	33	81						
42	1	2	6	7	9	11	12	82						
43	1	3	5	7	8	9	14	17	21	22	24	25	28	30
	31	33	34	83										
44	2	3	4	5	7	9	11	13	14	16	18	19	20	22
	23	24	25	28	29	33	34	84						
45	1	2	4	5	6	7	12	14	15	17	18	85		
46	2	3	9	10	15	18	22	23	25	26	29	31	32	34
	35	86												
47	1	2	4	5	6	7	12	14	15	17	19	20	21	23
	24	25	26	29	30	34	35	87						
48	4	5	8	12	13	17	18	88						
49	1	2	4	5	6	8	10	11	16	19	23	24	26	27
	30	32	33	35	36	89								
50	4	5	7	8	9	10	13	15	16	18	20	21	22	24
	25	26	27	30	31	35	36	90						
51	1	3	6	9	10	91								
52	1	3	4	5	6	7	8	10	11	12	17	20	24	25
	27	28	31	33	34	36	37	92						

53	2	3	4	6	7	9	10	11	14	16	17	19	21	22
	23	25	26	27	28	31	32	36	37	93				
54	4	5	7	8	9	10	13	15	16	18	19	94		
55	1	3	4	7	8	11	12	13	18	21	25	26	28	29
	32	34	35	37	38	95								
56	1	2	4	5	6	7	8	9	11	12	15	17	18	20
	22	23	24	26	27	28	29	32	33	37	38	96		
57	1	3	6	7	8	9	10	13	14	18	19	97		
58	1	3	4	7	9	12	13	14	19	22	26	27	29	30
	33	35	36	38	39	98								
59	2	3	5	6	7	10	12	13	16	18	19	21	23	24
	25	27	28	29	30	33	34	38	39	99				
60	1	3	5	8	10	12	13	100						
61	1	2	6	7	8	13	14	15	20	23	27	28	30	31
	34	36	37	39	40	101								
62	5	6	8	9	10	11	13	14	17	19	20	22	24	25
	26	28	29	30	31	34	35	39	40	102				
63	1	2	3	4	5	6	10	12	13	14	16	17	18	19
	22	23	27	29	31	34	35	36	37	38	39	103		
64	2	10	12	13	15	17	18	19	21	22	23	24	27	28
	32	34	36	39	40	104								
65	1	5	10	11	13	15	16	17	19	20	21	22	25	26
	30	32	34	37	38	39	40	105						
66	4	5	6	7	8	9	11	13	14	15	17	18	19	20
	23	24	28	30	32	35	36	37	38	39	40	106		
67	1	2	3	4	6	10	11	12	17	20	24	25	27	28
	31	33	34	36	38	39	40	107						
68	1	2	3	5	6	8	10	11	13	19	20	21	26	29
	33	34	36	37	40	108								
69	1	5	7	13	15	17	20	21	22	23	24	25	27	33
	34	35	40	109										
70	1	2	3	7	9	10	11	14	16	17	19	21	22	23
	25	26	27	28	31	32	36	38	40	110				
71	2	6	8	11	13	16	17	18	19	20	21	23	29	30
	31	36	39	111										
72	1	2	3	5	6	8	12	13	17	19	21	24	25	26
	27	28	29	31	37	38	39	112						
73	3	4	8	10	11	13	14	15	16	19	20	24	26	28

	31 32 33 34 35 36 38 113
74	1 3 5 6 10 12 14 17 18 19 20 21 22 24 30 31 32 37 40 114
75	1 3 4 5 8 10 13 14 18 20 22 25 26 27 28 29 30 32 38 39 40 115
76	1 3 7 9 10 11 12 14 15 16 17 20 21 25 27 29 32 33 34 35 36 37 39 116
77	1 2 3 5 8 9 10 11 12 14 15 16 17 20 21 25 27 29 32 33 34 35 36 37 39 117
78	2 5 6 7 8 9 11 12 13 15 16 17 18 21 22 26 28 30 33 34 35 36 37 38 40 118
79	1 2 6 8 11 12 13 15 16 17 18 21 22 26 28 30 33 34 35 36 37 38 40 119
80	2 3 5 9 11 12 13 15 16 17 18 21 22 26 28 30 33 34 35 36 37 38 40 120