國立交通大學

電信工程學系碩士班
碩士論文

使用 FPGA 實現應用於網路安全之可延展的
字樣比對架構

Implementation of a Scalable Pattern Matching Architecture for Network Security Applications using FPGA

研究生：李世弘

指導教授：李程輝　教授

中華民國九十七年六月

使用 FPGA 實現應用於網路安全之可延展的字樣比對架構

Implementation of a Scalable Pattern Matching Architecture for Network Security Applications using FPGA

研究生：李世弘　　Student：Shih-Hung Lee

指導教授：李程輝 教授　　Advisor：Prof. Tsern-Huei Lee

國 立 交 通 大 學

電 信 工 程 學 系 碩 士 班

碩 士 論 文

A Thesis

Submitted to Department of Communication Engineering

College of Electrical and Computer Engineering

National Chiao Tung University

in Partial Fulfillment of the Requirements

For the Degree of

Master of Science

in

Communication Engineering

June 2008

Hsinchu, Taiwan, Republic of China

中華民國九十七年六月

使用 FPGA 實現應用於網路安全之可延展的字樣比對架構

學生：李世弘　　指導教授：李程輝 教授

國 立 交 通 大 學
電 信 工 程 學 系 碩 士 班

摘要

因為字樣比對的準確性，使其技術近年來被廣泛地運用到一些網際網路的應用，例如入侵偵測，防毒。現今網路入侵偵測系統在偵測網路封包的有效負載(payload)時，是檢查其有效負載是否與所設定的網路安全規範一致。這個過程，往往被稱之深度封包檢測(deep packet inspection)，偵測有效負載中之任意起始位置是否涉及到預先定義的字樣或關鍵字。字樣比對是一項需要高度計算密集的工作，因此其潛在的瓶頸問題是，無法快速處理。因為傳統的實現於軟體的字樣比對無法跟上日益增加的網路速度，因此實現於硬體的解決方式被相繼的提出。本論文將實現 NTL 實驗室所提出一個新穎的字樣比對架構，並將其一個稱之為預先過濾器的前置處理器實現於 FPGA。我們將展現出它如何有效率的比對成千上萬的字樣。最後，我們將字樣比對實現於 Xilinx Virtex II Pro ML310 FPGA，並得到一些詳細的數據和結果。

Implementation of a Scalable Pattern Matching Architecture for Network Security Applications using FPGA

student：Shih-Hong Lee Advisor：Prof. Tsern-Huei Lee

Department of Communication Engineering
National Chiao Tung University

Abstract

Because of its accuracy, pattern matching technique has recently been applied to Internet security applications such as intrusion detection, anti-virus. Modern Network Intrusion Detection Systems (NIDS) detect the network packet payload to check if it conforms to the security policies of the given network. This step, often called that deep packet inspection, involves detection of predefined patterns or keywords starting at an arbitrary location in the payload. Pattern matching is a computationally intensive work. It has a potential bottleneck without high-speed processing. Since the conventional software-implemented pattern matching have not kept pace with the increasing network speeds, hardware-implemented solutions have been introduced. In this paper we will realize the NTL laboratory to propose a novel scalable pattern matching architecture, and the pre-processor, call that pre-filter, which implemented to FPGA. We show how Pre-filters can be used effectively to perform pattern matching for thousands of strings. Finally, we give the details of our implementation of pattern matching technique on Xilinx Virtex II Pro ML310 FPGA.

誌 謝

誠摯的感謝指導教授 李 程輝 博士，在我研究所就讀期間悉心地教導與指點，得以一窺網路安全領域的深奧，使我在這兩年中獲益匪淺。雖然我不是一個能適時舉一反三的學生，但謝謝您總是包容著我的不細心，期待著我下次的完整解答。雖然我不是一個善於表達的學生，但謝謝您總是仔細地聽完我想要對您的陳述。您對學問的嚴謹態度更是學生作學問的一個優良典範。

感謝 NTL 實驗室，博士班-景融 學長、郁文 學長、迺倫 學姊；已畢碩士班-嘉旂 學長、建成 學長、柏庚 學長、登煌 學長；同儕-耀誼、勁文、凱文、明智、明鑫、錫堯；學弟-俊德、佑信、松晏、家豪、鈞傑，在我課業、生活以及研究上不吝嗇地給我最大的關懷與指教，使我一路茁壯，讓我碩士生涯過得很充實和愉快。謝謝每一個曾經伴我成長的戰友們，我以你們為傲。

感謝 CIC 這兩年來無怨無悔地充當我的衣食父母，也謝謝蔡博、雅慧、淑娟、維蓓、宗盈，...等各位大哥哥、大姐姐們的提攜與照顧。

最後，感謝我的父親-李 炳森 先生、母親-李張 彩霞 女士、陪伴我五年多的女友-吳雅婷 小妹妹、摯友-林文翔以及各位親朋好友，謝謝你們默默地支持我，鼓勵我。我由衷地感謝每一個曾經為我付出的人，我沒有辜負您們的期盼，我的成就與驕傲全因您們而得，我將一切的榮耀都奉獻給您們，謝謝。

謹將此論文獻給所有愛我及我愛的人

西元 2008 年 6 月 於新竹交大

目　錄

表目錄

圖目錄

第一章
簡 介

1.1 研究背景

網際網路的快速成長，和出現一些嶄新的應用，如 P2P 檔案共享、視訊點播，以及電子商務的興起，大大地提升了使用者在網路上的網路流量。網路的速度和頻寬也迅速地增加，以滿足使用者的需求。正因如此，網路上的一些惡意攻擊，如拒絕服務（Denial of Service, DoS）、電子郵件病毒(E-mail virus)以及網際網路蠕蟲(Internet worm)，可以更快和更具破壞性的進行攻擊。例如，Code Red 蠕蟲和 SQL Slammer 蠕蟲在數分鐘到數小時間就造成世界上數十億美金的損失。

字樣比對(Pattern matching)是電腦科學中的一個基本問題，其研究內容在資訊檢索、資料壓縮、搜尋引擎、入侵偵測、內容過濾以及基因排序等都佔據了重要的一環。在字樣比對演算法中常被提及的有 **KMP**, *D.E. Knuth, J.H. Morris and V.R. Pratt [1]*、**Boyer-Moore (BM)**, *R.S. Boyer and J.S. Moore[2]*、 **Wu-Manber**, *S. Wu and U. Manber [3]*、**Aho-Corasick (AC)**, *A.V. Aho and M.J. Corasick [4]*。而當我們在探討字樣比對演算法時，經常將它們區分成，

以複雜度的觀點：

可分為『線性(linear)』/『子線性(sub-linear)』字樣比對演算法。

AC 字樣比對演算法是一種典型且常見的線性時間演算法，它將字樣(pattern)建構成有限狀態機(Finite state automaton)，然後在輸入端一個接一個(one by one)輸入字元(character)，再根據現在狀態(current state)與輸入的字元(input character)，將狀態轉移至下一個狀態(next state)。AC 演算法的時間複雜度為 O(n)，因此它在最壞情況(worst case)的表現

尤佳。

BM 字樣比對演算法則是典型的子線性時間的演算法，在很多情況(時間)下，它每次移動時，都跳躍(移動)一段長距離的位置，使得它就時間複雜度而言優過於線性時間複雜度，我們稱之為子線性時間複雜度。此演算法的設計在一般情況(average case)下的表現突出，時間複雜度為 O(n+m)，其中 n 為輸入欲比對之字串(string)的長度，m 為字樣長度；但在最壞情況下的表現就差強人意，時間複雜度為 O(n*m)。

以可比對字樣多寡的觀點：

可分為『單一(single)』/『多(multiple)』字樣比對演算法。

上述，BM 演算法為單一字樣比對演算法，意指當輸入字串餵進時，僅單一字樣與其進行比對工作。

AC 則為多字樣比對演算法，意指當輸入字串餵進時，由多字樣所建構成的有限狀態機會與其進行比對工作，因此，我們可能會在輸入字串掃描完畢後，同時偵測到多個字樣。正如所述，AC 演算法同時具備『線性』及『多』的好處，因此經常被運用作為字樣比對中的搜尋引擎。在第二章中，我們會再詳細的解說 AC 的動作/操作原理。

實現於軟體的字樣比對也正因網際網路的快速成長，無法跟上日益增加的網路速度，使其無法有效與即時地替我們防堵網路上的惡意攻擊。由於無法快速的處理網路上的流量，這個先天性的缺憾，使得人們開始將注意力轉移至以硬體化的方式實現字樣比對的工作。相關的論文有如雨後般的春筍，絡繹不絕。

1.2 研究動機

幾乎當前市場上的網路入侵檢測系統(Network Intrusion Detection System, NIDS)都是依賴一種資料收集(data collection)的機制，此機制屬於被動協定分析(passive protocol analysis)，這種模式在本質上是有先天性的缺陷。被動協定分析的意思是說，入侵偵測系統在網路上是處於被動地觀看網路流量，並詳細地檢查流量中是否有可疑字樣的蹤跡。因此，我們可以大致上歸類出下列幾個主要的問題：

(1) 入侵偵測系統是否有足夠的訊息去偵測網路上流量？

(2) 入侵偵測系統對於可疑字樣的操作模式是屬於 fail open，意指當入侵偵測系統失敗時，網路仍是會通的，不會因為入侵偵測系統的失敗，而造成網路中斷的情形，但是此時網路的安全性是有疑慮的。這種操作模式與防火牆相異，防火牆對於可疑字樣的操作模式是屬於 fail close，意指當防火牆失敗時，網路是不通的，以保護使用端的安全。

(3) 基於軟體的入侵偵測系統是否能夠在網際網路的快速發展以及網路速度與頻寬日益增加下，還能有效地與即時地偵測出可疑的活動？

本論文的研究重心將致力於解決上述第三個問題。很明顯地，基於軟體的入侵偵測系統在處理速度上遇到了瓶頸，這激發我們去為這問題找到一個新的答案。在 NTL 實驗室，李 程輝 教授和黃 迺倫 學姐的通力合作下，開發出一個新穎的可延展之字樣比對架構，此新穎的字樣比對架構不僅在比對的速度上有著優越的表現，且所需儲存的空間也非常的節省，若以總和效能來進行比較，我們的可延展之字樣比對架構是現今的字樣比對領域中的領頭者。在有著優秀的字樣比對演算法下，我們更進一步地將它實現於硬體之上，以嚴謹和謹慎的觀點去評估其實際實現於 FPGA 的可延展之字樣比對架構能有著如何的表現。

第二章
相 關 工 作

2.1 Aho-Corasick 演算法

在 1975 年 6 月發表於 Communication of the ACM 的"Efficient String Matching: An Aid to Bibliographic Search"為貝爾實驗室的 Alfred V. Aho 和 Margaret J. Corasick 兩位先生共同提出。此篇論文提出了一個有效地利用有限狀態機(Finite State Machine)建構出快速且簡單的字樣比對。由於此篇論文為一古典字典比對搜尋演算法，有效率地執行比對的工作且更具備了同時多字樣比對的功能，因此常被人們應用於相關地方，如字樣比對，基因比對，入侵偵測系統等，常簡稱為 AC 演算法。

AC 是一種基於有限狀態機的演算法，在進行比對前，都會先對字樣集合內的所有字樣先行預先處理，建構出樹狀的有限狀態機；然後，僅須對輸入字串搜描一次，即可找到與字樣集合內匹配的字樣。AC 演算法預先處理部份包含了三個函數，轉移函數(Goto function)，失效函數(failure function)與輸出函數(output function)。

轉移函數，表示在現在狀態(cuurent state)下讀入一個待測的輸入字串之字元後，將轉移到下個狀態(next state)，這步驟將在建構出有限狀態機；如圖-2.1～圖-2.4 所示，在字樣集合內有{he, she, his, hers}等 4 個字樣，我們將依次建構出屬於此字樣集合的樹狀的有限狀態機。首先，設置一個初始狀態(initial state)為狀態 0，依字樣集合內的字樣排列順序建構出。當我們在將一字樣要建立於圖中時，皆從狀態 0 開始，若在現在狀態下，讀入一字樣之字元，下一個狀態不存在，則我們須編比已有狀態編號大 1 的一個新狀態，並用一條有向線從此現有狀態指向

此新狀態且在有向線上標註此字樣之字元；若下一個狀態存在，則我們繼續走下去，直至此字樣建構於圖上。

首先，依字樣集合所示，我們須先建構字樣 he。一開始時，整個有限狀態機僅存在一個狀態 0，所以狀態 0 必無一條轉移函數經由字元 h 指向另一個狀態，因此我們編入一新狀態為狀態 1，並在由狀態 0 至狀態 1 的有向線上標註 h，函數式為 $g(0,h)=1$。在狀態 1 下，此字樣下一個字元 e 讀入待編，我們可知狀態 1 並無接任何的下一個狀態，因此我們編入一新狀態為狀態 2，並在由狀態 1 至狀態 2 的有向線上標註 e，函數式為 $g(1,e)=2$。由於目前所編入的字樣開頭僅為 h，故其餘字元的轉移函數皆回至狀態 0，如圖-2.1 所示。

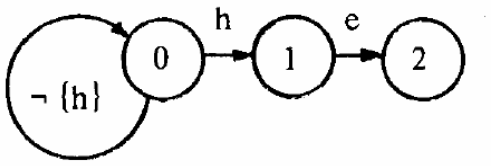

圖-2.1：由{he, she, his, hers}字樣集合所建構之有限狀態機-1

字樣 he 建構完畢，依次待建構字樣為 she。狀態 0 無一條轉移函數經由字元 s 指向另一個狀態，因此我們編入一新狀態比目前現有狀態編號最大者加 1，為狀態 3，並在由狀態 0 至狀態 3 的有向線上標註 s，函數式為 $g(0,s)=3$。在狀態 3 下，此字樣下一個字元 h 讀入待編，我們可知狀態 3 並無接任何的下一個狀態，因此我們編入一新狀態比目前現有狀態編號最大者加 1，為狀態 4，並在由狀態 3 至狀態 4 的有向線上標註 h，函數式為 $g(3,h)=4$。在狀態 4 下，此字樣下一個字元 h 讀入待編，我們可知狀態 4 並無接任何的下一個狀態，因此我們編入一新狀態比目前現有狀態編號最大者加 1，為狀態 5，並在由狀態 4 至狀態 5 的有向線上標註 e，函數式為 $g(4,e)=5$。由於目前所編入的字樣開頭僅為 h 與 s，故其餘字元的轉移函數皆回至狀態 0，如圖-2.2 所示。

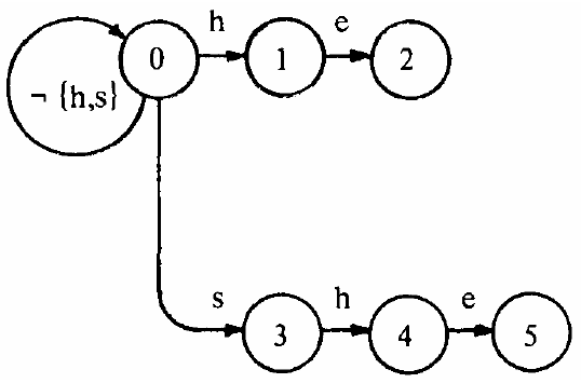

圖-2.2：由{he, she, his, hers}字樣集合所建構之有限狀態機-2

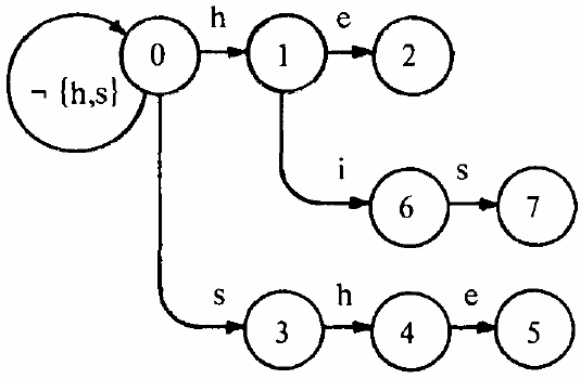

圖-2.3：由{he, she, his, hers}字樣集合所建構之有限狀態機-3

依此類推，若目前待建構字樣為字樣集合的最後字樣 hers。狀態 0 已存在一條轉移函數經由字元 h 指向狀態 1，因此我們不需編入一新狀態，下一個狀態將停留在狀態 1，函數式為 $g(0,h)=1$。在狀態 1 下，此字樣下一個字元 e 讀入待編，我們可知狀態 1 已存在一條轉移函數經由字元 e 指向狀態 2，因此我們不需編入一新狀態，我們下一個狀態將停留在狀態 2，函數式為 $g(1,e)=2$。在狀態 2 下，此字樣下一個字元 r 讀入待編，我們可知狀態 2 並無接任何的下一個狀態，因此我們編入一新狀態比目前現有狀態編號最大者加 1，為狀態 8，並在由狀態 2 至狀態 8 的有向線上標註 r，函數式為 $g(2,r)=8$。在狀態 8 下，此字樣下一個字元 s 讀入待編，我們可知狀態 8 並無接任何的下一個狀態，因此我們編入一新狀態比目前現有狀態編號最大者加 1，為狀態 9，並在由狀態 8 至狀態 9 的有向線上標註 s，函數式為 $g(8,s)=9$。由於目前所編入的字樣開頭僅為 h 與 s，故其餘字元的轉移函數皆回至狀態 0，如圖-2.4 所示。

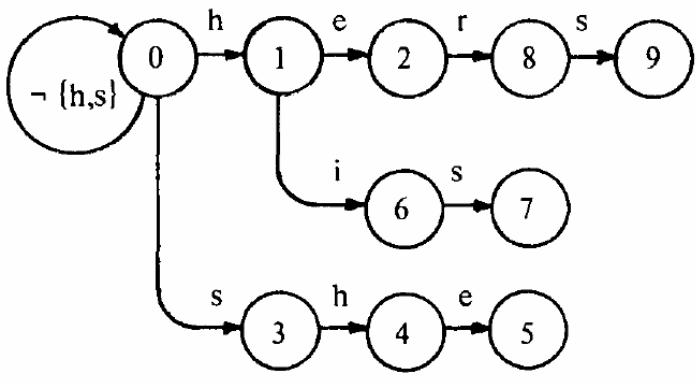

圖-2.4：由{he, she, his, hers}字樣集合所建構之有限狀態機-4

失效函數，用來指示某個『現在狀態』下，當讀入一輸入字串之字元後無法至下一個狀態時(意指這個現在狀態為此路徑(分支)的最後狀態)，我們須將轉移到的狀態則由此失效函數決定之。圖-2.5 為由此字樣集合所加入之失效函數後的有限狀態機。

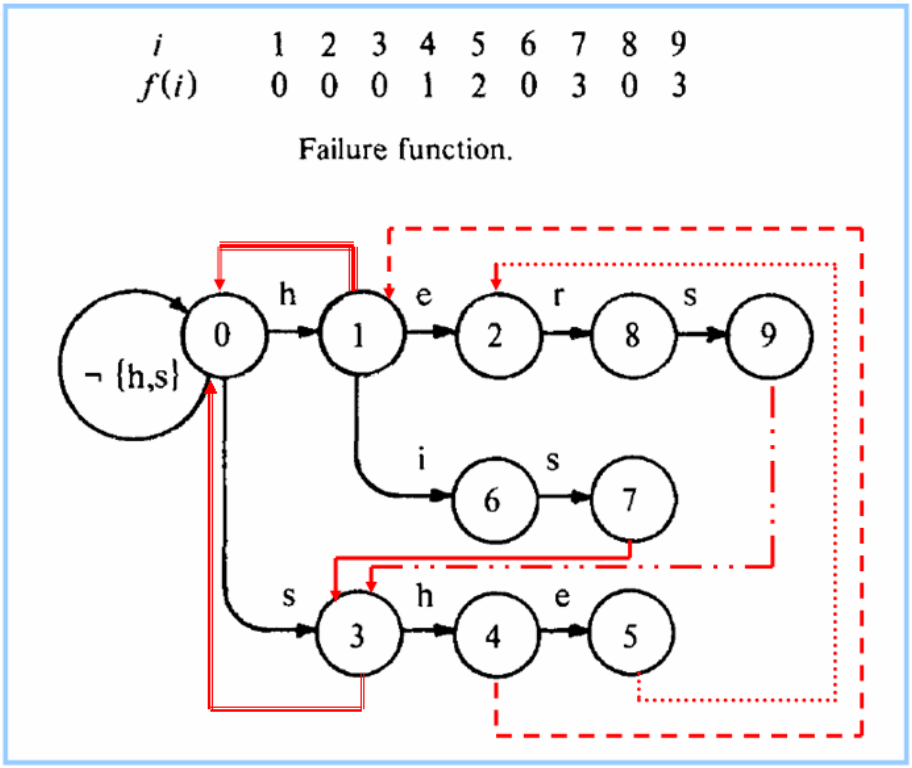

圖-2.5：{he, she, his, hers}此字樣集合所加入失效函數的有限狀態機

失效函數 f 為逐層建構而得。當第一層，也就是狀態 1 與狀態 3 發生失效，依演算法之定義，則下一個狀態將回至初始狀態。其餘則遵循著 $f(s)=g(f(s'),a)$，其中狀態 s' 為狀態 s 的父狀態，如下所示：

- 當狀態 2 發生失效，則 $f(2)=g(f(1),e)=g(0,e)=0$
- 當狀態 4 發生失效，則 $f(4)=g(f(4),h)=g(0,h)=1$
- 當狀態 5 發生失效，則 $f(5)=g(f(4),e)=g(1,e)=2$
- 當狀態 6 發生失效，則 $f(6)=g(f(1),i)=g(0,i)=0$
- 當狀態 7 發生失效，則 $f(7)=g(f(6),s)=g(0,s)=3$
- 當狀態 8 發生失效，則 $f(8)=g(f(2),r)=g(0,r)=0$
- 當狀態 2 發生失效，則 $f(9)=g(f(8),s)=g(0,s)=3$

輸出函數，用來指示在比對過程中，輸出所匹配的字樣之最後狀態所停留之處。圖-2.6 為此字樣集合的輸出狀態。

i	$output(i)$
2	{he}
5	{she, he}
7	{his}
9	{hers}

Output function.

圖-2.6：{he, she, his, hers}此字樣集合的輸出狀態

2.2 Wu-Manber 演算法

在 1994 年 5 月發表的"A fast algorithm for multi-pattern searching"為 Sun Wu 和 Udi Manber 兩位先生共同提出。此演算法採用的跳躍不可能匹配之字元與雜湊陣列(Hash array)方式加速比對字樣的進行。

Wu-Manber 亦須對字樣集合內的所有字樣進行預先處理之動作，分別建構

出 SHIFT 表(SHIFT table)，HASH 表(HASH table)，以及 PREFIX 表(PREFIX table)等三個表。SHIFT 表用於指示當在掃描輸入字串時，根據所讀入之輸入字元串決定可以跳躍的字元數(距離)，如果所對應之值為 0，則表示可能產生匹配的情況，這時我們須交由 HASH 表與 PREFIX 表更進一步的驗證判斷，以決定匹配了哪些可能的字樣，再則驗證哪一個或哪些字樣完全匹配。

假設字樣集合中的最短字樣的長度為 m，則我們將取字樣集合內的每一個字樣前 m 個字元(每一個字樣等長)進行多字樣比對，這樣的動作是為了增進比對之速度與簡單性(不必去在意各字樣的長度，因為已將每一個字樣取等長)。

令 $X = x_1 x_2 ... x_B$ 為輸入字元串中的待比對之區塊字串。區塊字串 X 將透過雜湊函數映射得到一雜湊值，並將此雜湊值作為 SHIFT 表的一個索引值(index)，由 SHIFT 表得到的值將代表此區塊字串可跳躍(移動)的位元組數。

$$SHIFT[i] = \begin{cases} m - B + 1 & \text{X does not appear as a substring in any pattern of P} \\ m - q & \text{X appears in some patterns} \end{cases}$$

如上述方程式所示，我們經 X 得到一雜湊值為 i 所可移動的式子將分為兩大類。第一類，X 不屬於字樣集合內的一個字樣；第二類，X 可能出現於多個字樣之中。

如圖-2.7~圖-2.8 所示，我們將舉一個例子說明 SHIFT[i]第一類的移動情況。現在區塊字串 X 為『agc』，並且在字樣集合內存在了兩個字樣，分別為 p_1 與 p_2。從圖-2.7 所示，我們可以看到區塊字串 X 並不屬於 p_1 與 p_2，因此可移動距離為 $m-b+1=7-3+1=5$，如圖-2.8 所示，下次的區塊字串 X 將為『rbc』。

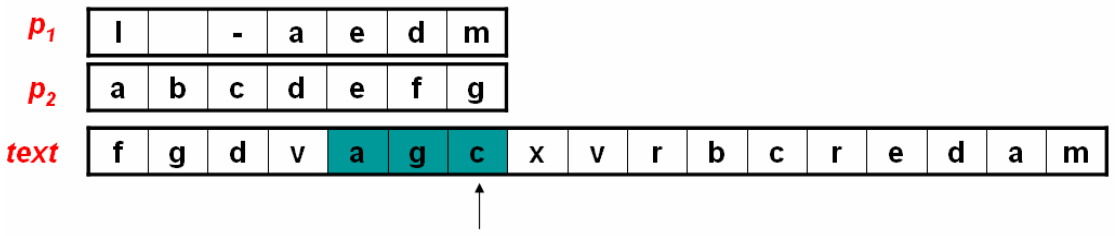

圖-2.7：SHIFT[i]第一類的移動說明-1

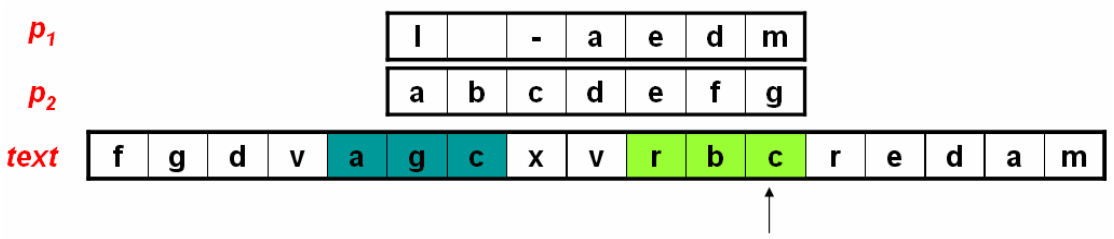

圖-2.8：SHIFT[*i*]第一類的移動說明-2

如圖-2.9~圖-2.12 所示，我們將舉一個例子說明 SHIFT[*i*]第二類的移動情況。現在區塊字串 X 為『agc』，並且在字樣集合內存在了兩個字樣，分別為 p_1 與 p_2。從圖-2.9~圖-2.10 所示，我們可以看到區塊字串 X 同時存在於 p_1 與 p_2，因此可移動距離為 $m-q$，其中 q 為區塊字串 X 結束於 p_1 與 p_2 的最大的位置。如圖-2.11 所示，區塊字串 X 的最後一個字元結束的位置為 p_1 的第 3 的字元，而區塊字串 X 的最後一個字元結束的位置為 p_2 的第 5 的字元。因此，我們的 q 值將選擇 5，可移動距離將為 $m-q=7-5=2$，如圖-2.12 所示，下次的區塊字串 X 將為『cxv』。

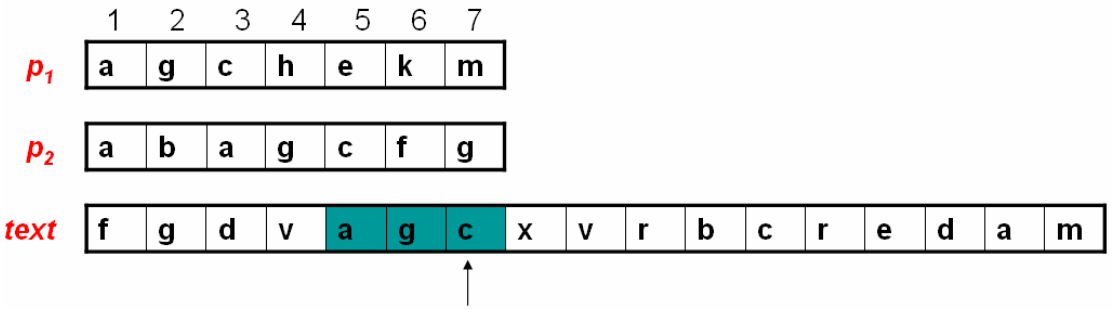

圖-2.9：SHIFT[*i*]第二類的移動說明-1

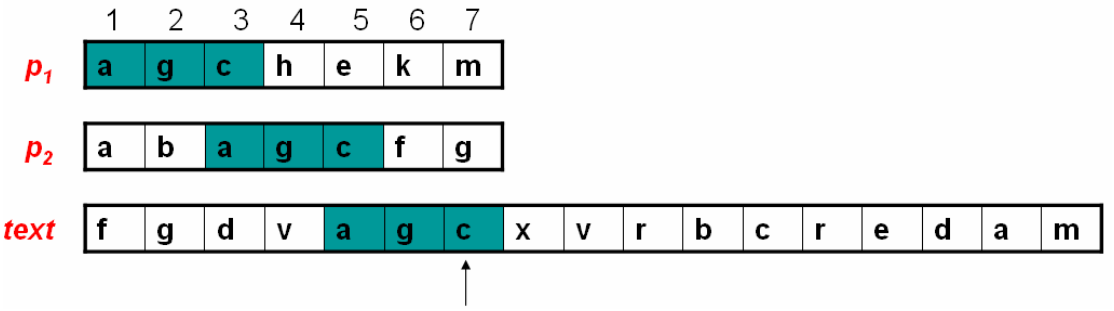

圖-2.10：SHIFT[*i*]第二類的移動說明-2

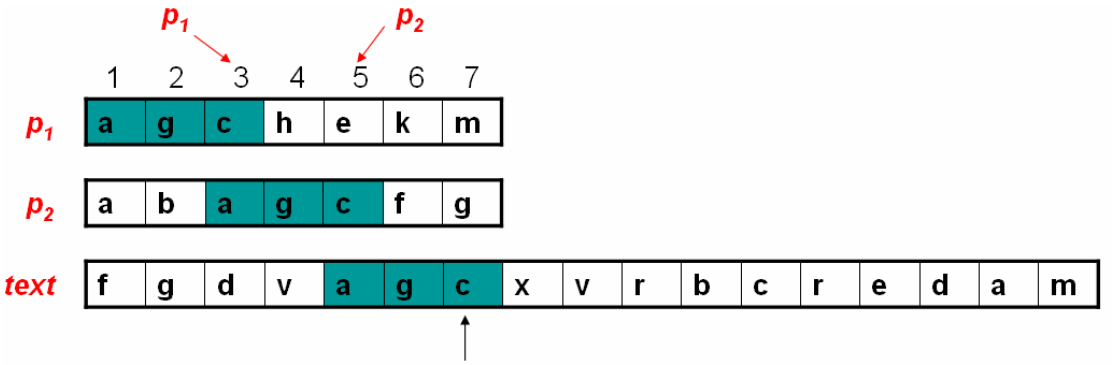

圖-2.11：SHIFT[*i*]第二類的移動說明-3

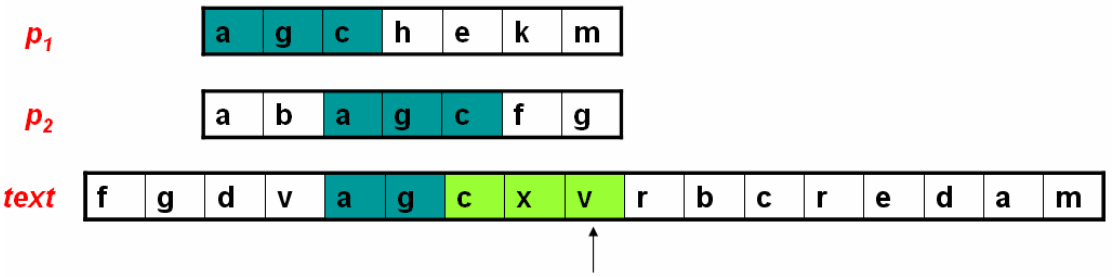

圖-2.12：SHIFT[*i*]第二類的移動說明-4

假設現在正比對的區塊字串 X 的雜湊值 h，如果 SHIFT[*h*]=0，則表示可能產生的匹配，因此我們須進一步地利用此雜湊值 h 去查詢 HASH 表，HASH[*h*]=p，其中 p 將作為字樣鏈表(pattern list)與 PREFIX 表的一個指示值。字樣鏈表儲存的是後 B 個字元的雜湊值與區塊字串 X 的雜湊值 h 一樣的字樣；PREFIX 表則儲存的是這些字樣的前 B 個字元的雜湊值，這樣的作法將有利於減少比對的次數，因為僅當這些字樣的前 B 個字元的雜湊值和後 B 個字元的雜湊值與輸入字串的前 B 個字元的雜湊值和後 B 個字元的雜湊值一樣時我們才需進行一個一個位元組的詳細比對。

第三章

應用於網路安全中可延展的字樣比對架構之預先過濾器

3.1 字樣比對架構的介紹

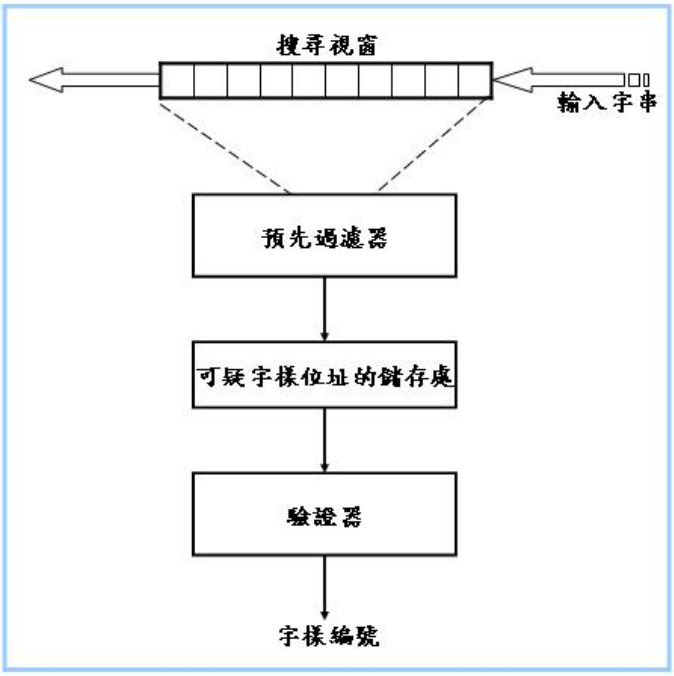

圖-3.1：我們所提出的字樣比對架構

圖-3.1 為我們所提出的字樣比對架構。此系統將輸入字串先透過預先過濾器的處理，將可疑字樣的位址記錄下，然後驗證器會根據紀錄在可疑字樣位址儲存處中的每一筆資料一一進行比對，驗證由此可疑字樣的起始位址是否真的存在一個屬於字樣集合(Patterns set)中的成員。一個透過預先過濾器處理的字樣比對架構大幅縮減系統對於輸入字串的處理時間，使其在網際網路的快速成長下還能保證系統安全地受到保護。我們將此字樣比對架構主要劃分成兩個部份，預先過濾

器以及驗證器，分別由我以及實驗室的另一位同學－古凱文所負責。因此，接下來我將針對我所負責的部份，預先過濾器，進行詳細的敘述與說明。

3.2 預先過濾器架構及相關定義

3.2.1. 搜尋視窗(Search Window)

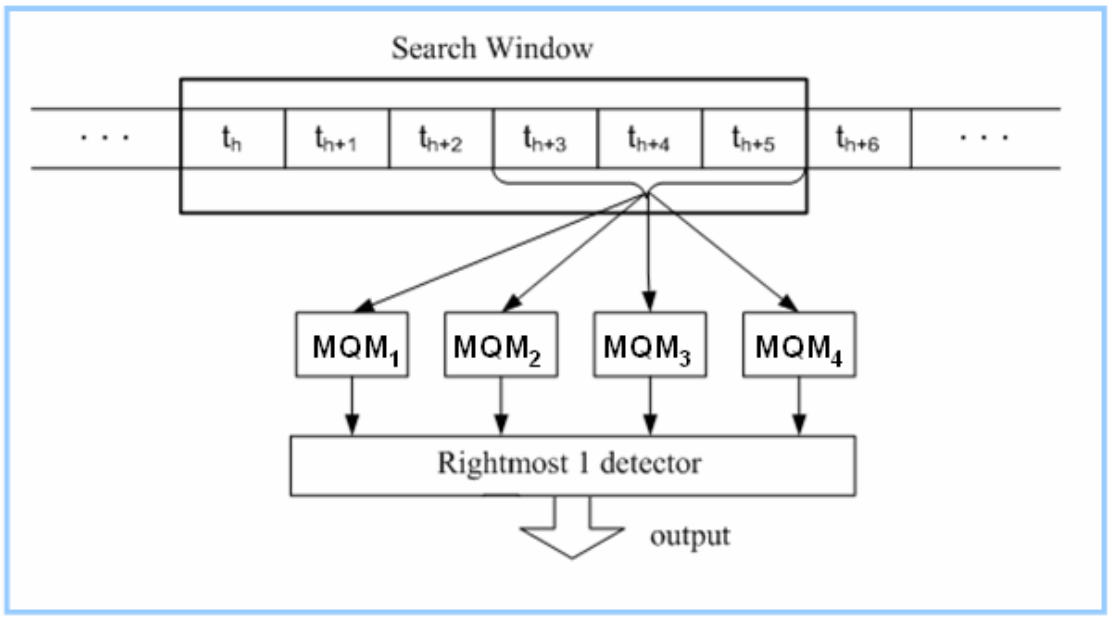

圖-3.2：m=6, k=3 的預先過濾器

圖-3.2 為一個 m=6, k=3 的預先過濾器，變數 m 表示我們所設定的搜尋視窗(Search window)的長度，我們稱 m 為視窗長度(Window length)，其單位為位元組(byte)；變數 k 表示我們每一次所去觀測搜尋視窗的後幾個位元組，我們稱 k 為區塊大小(Block size)。視窗長度以及區塊大小的選定將會影響我們接下來要討論的成員詢問模塊的個數。

3.2.2. 成員詢問模塊(Membership Query Module)

我們假設有一字樣集合儲存 N 個字樣，為了加速系統的處理速度，我們僅取字樣集合中每一個字樣的前 m 個位元組與輸入字串進行粗步的比對，假如比對匹配，則我們須將搜尋視窗的起始位址儲存至可疑字樣位址儲存處，驗證器則會針對其儲存於可疑字樣位址儲存處的每一筆資料做進一步詳細地判斷，驗證由此可疑字樣位址為起始是否真的存在一個屬於字樣集合中的成員。m，也是我們先前所提及須設定的視窗長度。以圖-3.2 為例，我們僅取字樣集合中每一個字樣的前 6 個位元組並且我們每次僅觀測搜尋視窗的後 3 個位元組。接下來我們將開始介紹如何將每一個字樣的前 m 個位元組分成 k 個位元組為一個群組(Group)，分別儲存至不同的成員詢問模塊(Membership Query Module, MQM)。首先，假設我們存在一個字樣為『*b9c0012e8a272e3226dd022e882743e2f2c3*』(此字樣取之於 ClamAV，以 16 進制表示之)，依圖-3.2，所設定之視窗長度為 6，因此我們將取下此字樣的前 6 個位元組，也就是『*b9c0012e8a27*』，並將分成 3 個位元組為一個群組分別儲存至不同的成員詢問模塊，我們分類的方式為

- 第一個子字樣(Sub-pattern)為第一個位元組至第三個位元組，『*b9c001*』，儲存至第一個成員詢問模塊(MQM1)。
- 第二個子字樣為第二個位元組至第四個位元組，『*c0012e*』，儲存至第二個成員詢問模塊(MQM2)。
- 第三個子字樣為第三個位元組至第五個位元組，『*012e8a*』，儲存至第三個成員詢問模塊(MQM3)。
- 第四個子字樣為第四個位元組至第六個位元組，『*2e8a27*』，儲存至第四個成員詢問模塊(MQM4)。

依上述的簡例，我們可以得知，當 m=6, k=3 時，我們的分類方式共需四個成員詢問模塊。現在我們將利用一些數學式子的推論來得知，若已知 m 和 k，則需多少個成員詢問模塊。假設字樣集合中有一字樣為 P^i ，為字樣集合中的第 i 個字樣，則有一子字樣 $p_1^i p_2^i ....p_k^i$ (P^i 的第一個位元組至第 k 個位元組)將儲存至第一個成員詢問模塊，一子字樣 $p_2^i p_3^i ....p_{k+1}^i$ 將儲存至第二個成員詢問模塊，依此類

推，則最後一個成員詢問模塊儲存的子字樣為 $p^i_{m-k+1}p^i_{m-k+2}....p^i_m$ 。**由此可知，若已知 m 和 k，則需 m-k+1 個成員詢問模塊**。接著，我們將說明成員詢問模塊在預先過濾器中的角色與功能。

每一個成員詢問模塊在預先過濾器中的角色都是擔任一個雜湊陣列(Hash array)，預設值為每陣列中的儲存空間都設定為 0。就先前所述，分配至每一個成員詢問模塊中的每一個子字樣並不是真實地儲存到每一個成員詢問模塊，而是每一個成員詢問模塊中的每一個子字樣都會經由同樣的雜湊函數(Hash function)，得到其雜湊值(Hash value)，並以此雜湊值為位址(Address)，將雜湊陣列中由此位址所指向的儲存空間設定為 1。因此，當成員詢問模塊所回饋值為 1，表示在搜尋視窗所觀測的區塊大小***可能***存在於該成員詢問模塊之中。利用雜湊的儲存方式雖擁有可大幅減低我們所儲存的空間和準確不遺漏(True negative)的特性，但付出的代價就是可能發生誤報(False positive)。例 3.1 說明如何將子字樣經由雜湊函數所得到的雜湊值，儲存至雜湊陣列之中的一個簡要例子。

例 3.1：

假設我們欲將一個位元組(8 個位元)的輸入值，經有一雜湊函數，轉換成為 3 個位元的雜湊值。則

- 輸入值值域 A = {0,1,..,255}，雜湊值值域 B={0,1, ..,7}。
- 3 個位元的雜湊值，告訴我們此雜湊陣列的大小為 2^3。
- 根據此題意，我們可定義出雜湊函數為一個隨機產生的 8*3 矩陣，D=[d_{8*3}]。
- 將雜湊陣列 HA 的儲存空間初始為 0。

$$D = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 1 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}$$

HA

0	0	0	0	0	0	0	0

- 假設現有其輸入值:　$D0_{16} \rightarrow 11010000_2$ 與 $82_{16} \rightarrow 10000010_2$

$$h(11010000) = (111 \bullet d_1) \oplus (111 \bullet d_2) \oplus (000 \bullet d_3) \oplus (111 \bullet d_4) \oplus (000 \bullet d_5) \oplus (000 \bullet d_6) \oplus (000 \bullet d_7) \oplus (000 \bullet d_8)$$
$$= d_1 \oplus d_2 \oplus d_4 = 110 \oplus 001 \oplus 101$$
$$= 010 \rightarrow 2(decimal)$$
$$h(10000010) = d_1 \oplus d_7 = 111 \rightarrow 7(decimal)$$

- 將 $D0_{16}$ 與 82_{16} 所得之雜湊值作為 HA 的位址，所指向的儲存空間設定為 1

HA	0	0	1	0	0	0	0	1

由上述簡例中我們可知，輸入值 8 個位元將由 3 個位元的雜湊值代替，減少了我們須儲存的空間，但不可避免的是，當輸入值為 $0A_{16}$ 時，經同一雜湊函數得到的雜湊值為 2，這將與 $D0_{16}$ 所產生的雜湊值一樣，都指向雜湊陣列 HA 的第二個儲存空間，因此發生了誤報。

3.2.3. 最右位元偵測器(Rightmost Bit Detector)

最右位元偵測器為偵測 $MQM_1MQM_2 \ldots MQM_{m-k+1}$ 因輸入值$(t_{h+3}t_{h+4}t_{h+5})$ 的輸入所得的輸出值組合中最右邊的位元為 1 的位置，此功能為判斷我們下一次可以從輸入字串中再餵進幾個位元組個數。我們先以圖-3.2 為一說明簡例(例 3.2)，說明每一輸出值組合可移動之位元組個數，在了解其意義後，我們再推廣為廣式。

例 3.2：如圖-3.2 所示，$MQM_1MQM_2MQM_3MQM_4$ 因輸入值$(t_{h+3}t_{h+4}t_{h+5})$ 的輸入所得的所有可能的輸出值組合為

0000, 0001, 0010, 0011, 0100, 0101, 0110, 0111, 1000, 1001, 1010, 1011, 1100, 1101, 1110, 1111，共 16 種可能的組合。

說明 1：

$MQM_1MQM_2MQM_3MQM_4$ 因區塊大小$(t_{h+3}t_{h+4}t_{h+5})$ 的輸入所得的輸出為 0010，表示此輸入值極有可能為 MQM_3 中的一個子字樣，由於 MQM_3

儲存的是字樣的第三個位元組至第五個位元組，因此如圖-3.3 所示

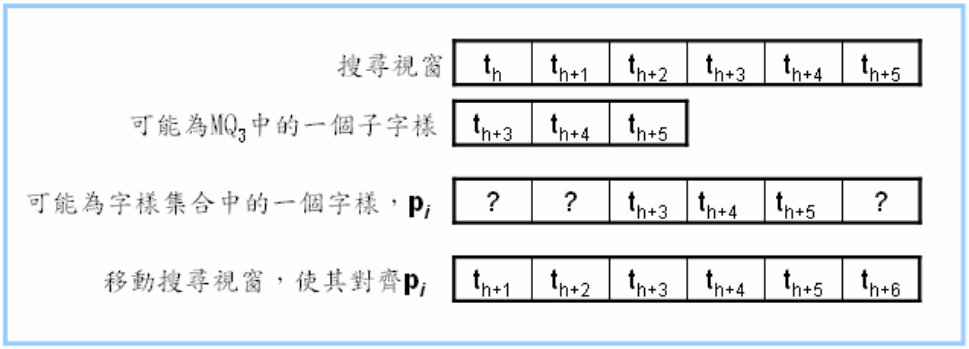

圖-3.3：最右位元偵測器說明 1

當其輸出值為 0010 時，搜尋視窗為了避免遺漏對 p_i 字樣的偵測，所以僅可移動一個位元組，使其下次搜尋視窗將停留在 $t_{h+1}t_{h+2}t_{h+3}t_{h+4}t_{h+5}t_{h+6}$ 的位置上，並將區塊大小，$t_{h+4}t_{h+5}t_{h+6}$，經由雜湊函數所得的雜湊值輸入至成員詢問模塊，去得知下次可移動的位元組個數。

說明 2：

$MQM_1MQM_2MQM_3MQM_4$ 因區塊大小($t_{h+3}t_{h+4}t_{h+5}$) 的輸入所得的輸出為 1100，表示此輸入值極有可能為 MQM_1 與 MQM_2 中的一個子字樣，由於 MQM_1 儲存的是字樣的第一個位元組至第三個位元組，MQM_2 儲存的是字樣的第二個位元組至第四個位元組，因此如圖-3.4 所示

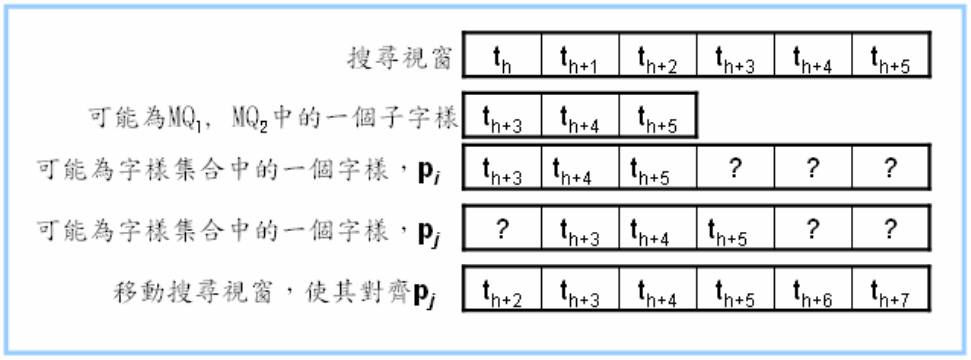

圖-3.4：最右位元偵測器說明 2

當其輸出值為 1100 時，搜尋視窗為了避免遺漏對 p_j 字樣的偵測，所以僅可移動二個位元組，使其下次搜尋視窗將停留在 $t_{h+2}t_{h+3}t_{h+4}t_{h+5}t_{h+6}t_{h+7}$ 的位置上，並將區塊大小，$t_{h+5}t_{h+6}t_{h+7}$，經由雜湊函數所得的雜湊值輸入至成員詢問模塊，去得知下次可移動的位元組個數。

說明 3：

$MQM_1MQM_2MQM_3MQM_4$ 因區塊大小($t_{h+3}t_{h+4}t_{h+5}$) 的輸入所得的輸出為 1001，表示此輸入值極有可能為 MQM_1 與 MQM_4 中的一個子字樣，由於 MQM_1 儲存的是字樣的第一個位元組至第三個位元組，MQM_4 儲存的是字樣的第四個位元組至第六個位元組，因此如圖-3.5 所示

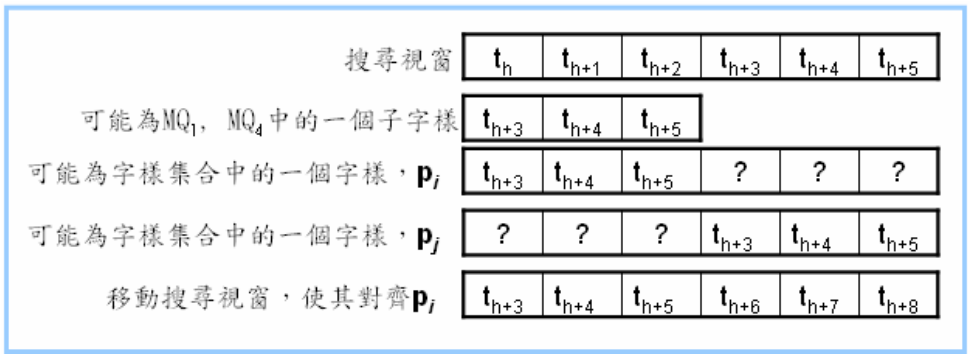

圖-3.5：最右位元偵測器說明 3

當其輸出值為 1001 時，由於 MQM_4 的輸出值為 1，這不但告訴我們搜尋視窗所觀測的區塊大小存在於 MQM_4，而且整個搜尋視窗極有可能為字樣集合中的一員。因此，我們須將此時搜尋視窗的起始位址儲存於可疑字樣位址儲存處，也就是輸入字串中 t_h 所代表的位址，等待由驗證器去驗證由此可疑字樣位址為起始是否真的存在一個屬於字樣集合中的成員。故，**當 MQM_4 的輸出值為 1 時，我們不但須去判斷下次可移動的位元組個數，還須將其此時搜尋視窗的起始位址儲存於可疑字樣位址儲存處**。這也就是意味著搜尋視窗為了避免遺漏對 p_j 字樣的偵測，將其此時搜尋視窗的起始位址儲存於可疑字樣位址儲存處，並且為了避免遺漏對 p_i 字樣的偵測，所以僅可移動三個位元組，使其下次搜尋視窗將停留在 $t_{h+3}t_{h+4}t_{h+5}t_{h+6}t_{h+7}t_{h+8}$ 的位置上，並將區塊大小，$t_{h+6}t_{h+7}t_{h+8}$，經由雜湊函數所得的雜湊值輸入至成員詢問模塊，去得知下次可移動的位元組個數。

說明 4：

$MQM_1MQM_2MQM_3MQM_4$ 因區塊大小($t_{h+3}t_{h+4}t_{h+5}$) 的輸入所得的輸出為 0000，表示此輸入值不存在於 MQM_1～MQM_4 中的任一子字樣，因此如圖-3.6 所示

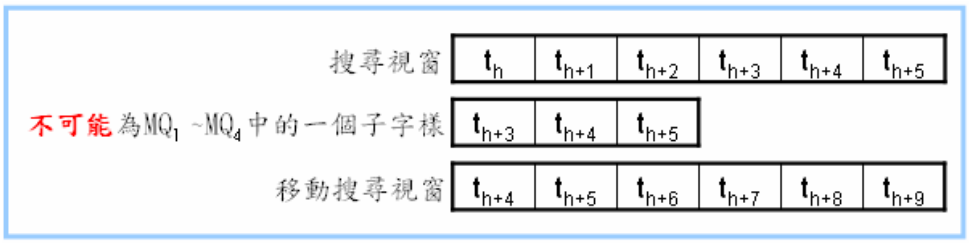

圖-3.6：最右位元偵測器說明 4

當其輸出值為 0000 時，搜尋視窗將可移動最大的距離，四個位元組，使其下次搜尋視窗將停留在 $t_{h+4}t_{h+5}t_{h+6}t_{h+7}t_{h+8}t_{h+9}$ 的位置上，並將區塊大小，$t_{h+7}t_{h+8}t_{h+9}$，經由雜湊函數所得的雜湊值輸入至成員詢問模塊，去得知下次可移動的位元組個數。

備註：

最大移動距離為四個位元組，而不是六個位元組。因為我們僅確定 $t_{h+3}t_{h+4}t_{h+5}$ 不可能會存在於 MQM_1～MQM_4 中，意味著 $t_ht_{h+1}t_{h+2}t_{h+3}t_{h+4}t_{h+5}$ 不可能會存在於字樣集合之中，但若我們將其移動六個位元組，下次搜尋視窗內容為 $t_{h+6}t_{h+7}t_{h+8}t_{h+9}t_{h+10}t_{h+11}$，這表示我們認定 $t_{h+4}t_{h+5}t_{h+6}$ 的子字樣是不可能會存在於 MQM_1～MQM_4 中，則我們將錯過對 $t_{h+4}t_{h+5}t_{h+6}t_{h+7}t_{h+8}t_{h+9}$ 的偵測，這會使我們可能遭遇到攻擊。因此，依此例，最大可移動的距離為四個位元組。

說明 5：

$MQM_1MQM_2MQM_3MQM_4$ 因區塊大小($t_{h+3}t_{h+4}t_{h+5}$) 的輸入所得的輸出為 1111，表示此輸入值極有可能為 MQM_1～MQM_4 中的一個子字樣，由於 MQM_1 儲存的是字樣的第一個位元組至第三個位元組，MQM_2 儲存的是字樣的第二個位元組至第四個位元組，MQM_3 儲存的是字樣的第三個位元組至第五個位元組，MQM_4 儲存的是字樣的第四個位元組至第六個位元組，因此如圖-3.7 所示

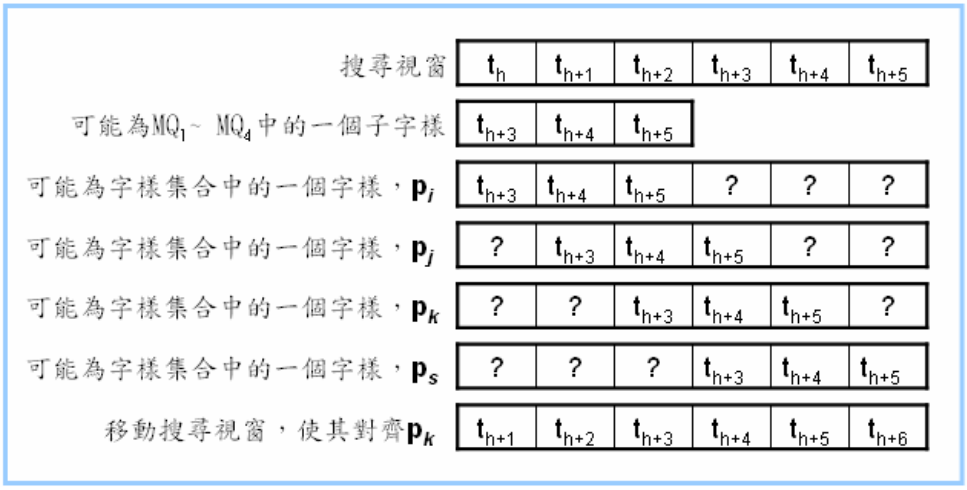

圖-3.7：最右位元偵測器說明 5

當其輸出值為 1111 時，依前面說明 1～說明 4 所述，搜尋視窗為了避免遺漏對 p_s 字樣的偵測，將其此時搜尋視窗的起始位址儲存於可疑字樣位址儲存處，並且為了避免遺漏對 p_k 字樣的偵測，所以僅可移動一個位元組，使其下次搜尋視窗將停留在 $t_{h+1}t_{h+2}t_{h+3}t_{h+4}t_{h+5}t_{h+6}$ 的位置上，並將區塊大小，$t_{h+4}t_{h+5}t_{h+6}$，經由雜湊函數所得的雜湊值輸入至成員詢問模塊，去得知下次可移動的位元組個數。

由例 3.2 的舉列說明中，我們可歸納出，當 MQM_1～MQM_4 的輸出值中，MQ_1M～MQM_3 將決定搜尋視窗可移動之距離且由最右邊位元為 1 的做決定；MQM_4 的輸出值 1 或 0 將決定我們是否須將搜尋視窗的起始位址儲存於可疑字樣位址儲存處。MQM_4 的輸出值為 1 表示須儲存之，反之，不須儲存。這也就是為什麼我們將『決定可移動的位元組個數』的功能命名為**最右位元偵測器**。最後，若推廣成有成員詢問模塊 $MQM_1MQM_2…MQM_{m-k+1}$，我們將在下述的通式中歸納出由 $MQM_1MQM_2…MQM_{m-k+1}$ 因輸入值($t_{h+3}t_{h+4}t_{h+5}$) 的輸入所得的輸出值組合中，如何決定可移動的位元組個數。

- MQM_{m-k+1} 為 0 時
  - 『決定可移動的位元組個數』＝ m-k+1-*id*，其中 *id* 指的是最右邊位元為 1 是發生在第幾個成員詢問模塊。
- MQM_{m-k+1} 為 1 時
  - 『決定可移動的位元組個數』＝ m-k+1-*id*，其中 *id* 指的是最右邊

位元為 1 是發生在第幾個成員詢問模塊。

- 須將現搜尋視窗的起始位址儲存於可疑字樣位址儲存處。

3.2.4. 主位元組列(Master Bitmap)

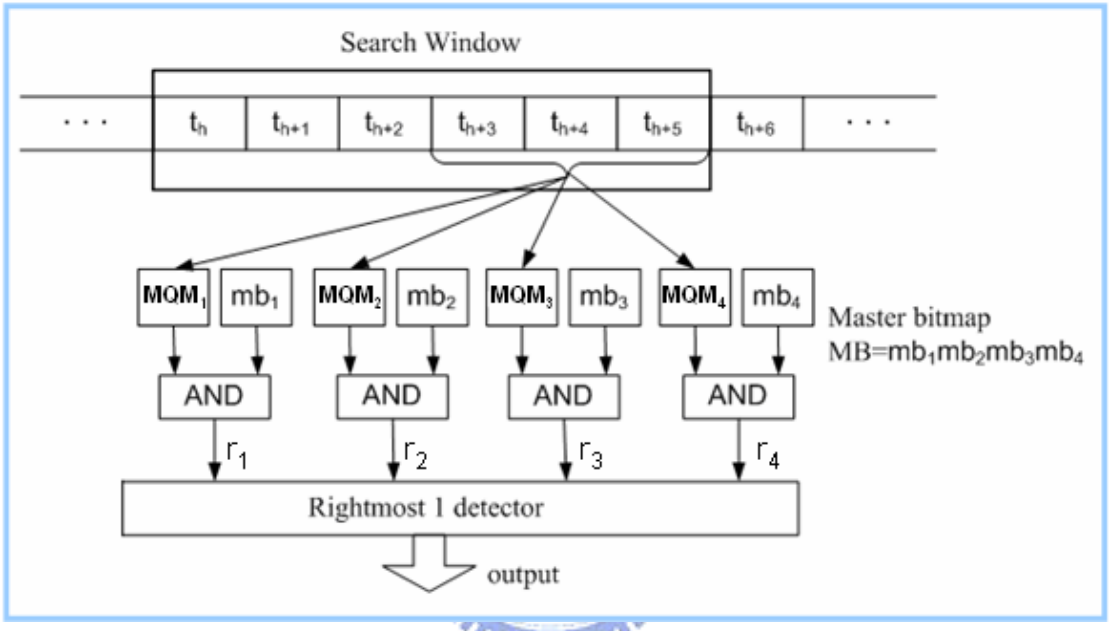

圖-3.8：具有主位元組列的預先過濾器

圖-3.8 為一具有主位元組列的預先過濾器。主位元組列，mb_1、mb_2、mb_3 以及 mb_4 的儲存規則如下之歸納：

- 主位元組列的初始值

  $MB = mb_1 mb_2 mb_3 mb_4 = 1111$

- 情況 1：r_4 的值為 1

  將搜尋視窗的起始位址儲存於可疑字樣位址儲存處，並根據 $r_1r_2r_3$ 的值來決定可向前移動之位元組個數。

- 情況 2：$r_1r_2r_3$ 的值為 000

  表示可向前移動之位元組個數為 4 個。因此，MB **向右位移** 4 個位元組並將其不足之處補上一個 1，即 $MB_{current} = mb_1 mb_2 mb_3 mb_4$，$MB_{next} = 1111$。

- 情況 3：$r_1r_2r_3$ 的值為 100

表示可向前移動之位元組個數為 3 個。因此，MB 向右位移 3 個位元組並將其不足之處補上一個 1，即 $MB_{current} = mb_1 mb_2 mb_3 mb_4$，$MB_{next} = 111 mb_1$。

- 情況 4：$r_1 r_2 r_3$ 的值為 $r_1 10$

  表示可向前移動之位元組個數為 2 個。因此，MB 向右位移 2 個位元組並將其不足之處補上一個 1，即 $MB_{current} = mb_1 mb_2 mb_3 mb_4$，$MB_{next} = 11 mb_1 mb_2$。

- 情況 5：$r_1 r_2 r_3$ 的值為 $r_1\ r_2 1$

  表示可向前移動之位元組個數為 1 個。因此，MB 向右位移 1 個位元組並將其不足之處補上一個 1，即 $MB_{current} = mb_1 mb_2 mb_3 mb_4$，$MB_{next} = 1 mb_1 mb_2 mb_3$。

主位元組列的功用為提升每一次可移動位元組的個數，使其整體吞吐量(Throughput)可以大幅提升。我們的實驗結果將可告訴我們，**有**主位元組列之預先過濾器比**沒有**主位元組列之預先過濾器的吞吐量最高可高於近 50%，我們將在其後加以說明。在這我們將舉例(例 3.3)說明主位元組列所帶來的優點。

例 3.3：現有一預先過濾器，其 m=6, k=3，假設第一次與第二次成員詢問模塊的輸出值分別為 1010 以及 0010。

- **沒有**主位元組列之預先過濾器

  搜尋視窗**每一次僅可向前移動一個位元組**，如圖-3.9 所示。

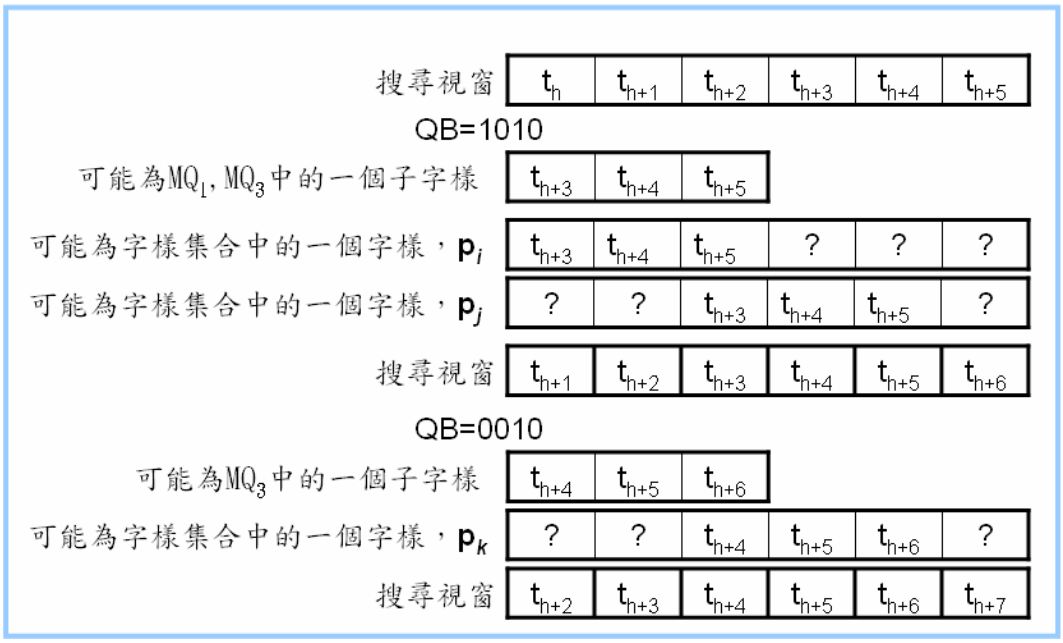

圖-3.9：不具有主位元組列之預先過濾器的搜尋視窗移動概觀

■ 有主位元組列之預先過濾器

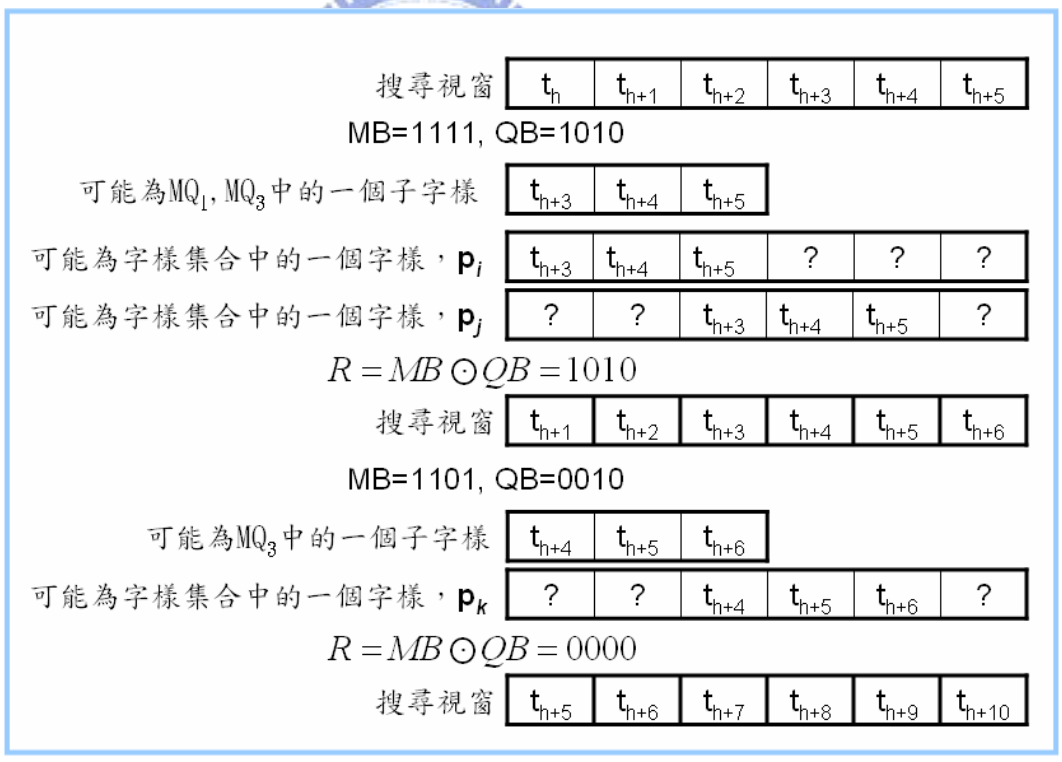

圖-3.10：具有主位元組列之預先過濾器的搜尋視窗移動概觀

➢ 第一次成員詢問模塊的輸出值，1010，將與主位元組列的初始值，1111，作『逐位和操作(Bitwise AND operation)』。所得之結果為

$R = r_1 r_2 r_3 r_4 = \{mqm_1 \square\ mb_1,\ mqm_2 \square\ mb_1,\ mqm_3 \square\ mb_1,\ mqm_4 \square\ mb_1\} = 1010$，根據主位元組列的規則表示

- ✓ 不須將起始位址儲存於可疑字樣位址儲存處。
- ✓ 可向前移動之位元組個數為 1 個。因此，MB 向右位移 1 個位元組並將其不足之處補上一個 1，即 $MB_{current} = 1111$，$MB_{next} = 1101$。

➢ 第二次成員詢問模塊的輸出值，0010，將與主位元組列，1101，作逐位和操作。所得之結果為 $R = \{0 \square\ 1,\ 0 \square\ 1,\ 1 \square\ 0,\ 0 \square\ 1\} = 0000$，根據主位元組列的規則表示

- ✓ 不須將起始位址儲存於可疑字樣位址儲存處。
- ✓ 可向前移動之位元組個數為 4 個。因此，MB 向右位移 4 個位元組並將其不足之處補上一個 1，即 $MB_{current} = 1101$，$MB_{next} = 1111$。

我們從例 3.3 明顯地得知，第二次的結果，**有**主位元組列之預先過濾器可向前移動的位元組個數是**沒有**主位元組列之預先過濾器的 4 倍，這說明了主位元組列功能的必要性與優越性。

第四章

基於 FPGA 的預先過濾器

4.1 字樣比對架構的系統雛型

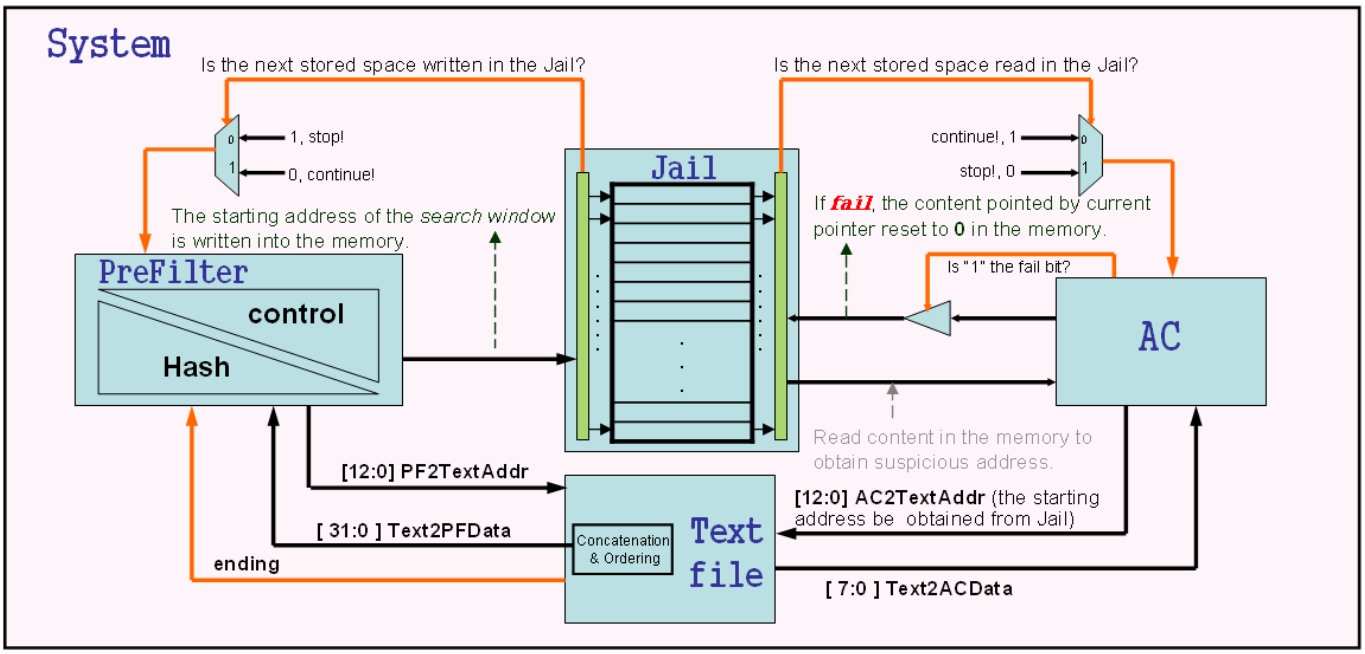

圖-4.1：我所提出實現於 FPGA 的字樣比對系統雛型

圖-4.1 為我所提出將要實現於 FPGA 的字樣比對系統雛型。圖中主要分成四大部分，分別為測試文件檔(Text File)、可疑位址儲存處(Jail)、預先過濾器(Prefilter)，以及驗證器(AC)。在這，我將利用子節 4.3、子節 4.4 及子節 4.5 分別針對前三部分的實作細節作一一深入的介紹與分析。

4.2 測試板-Xilinx Virtex II Pro ML310 介紹

在進入正題前我們要先針對我們所選用的 FPGA 板子作一大致上的說明，因為所選用作為測試板的內部元件將會對我們所設計的系統雛型有所牽動與影響。

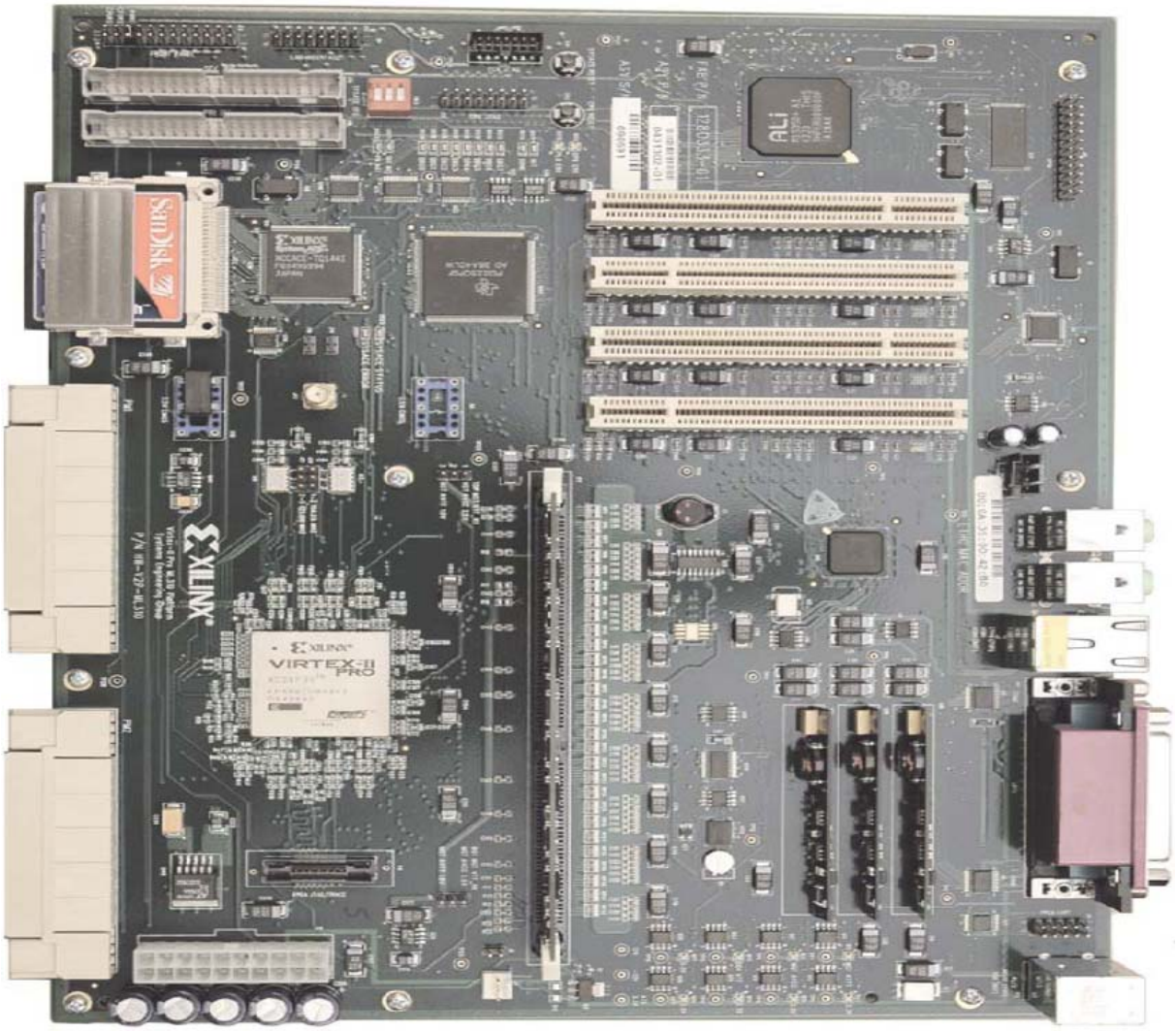

圖-4.2：Xilinx Virtex II Pro ML310 FPGA 測試板

我們所選用的測試板為 Xilinx 公司所出產的 Virtex II Pro ML310，如圖-4.2 所示。Virtex II Pro 系列的 FPGA 採用 0.13um、1.5V 的處理技術所製造而成，整合了嵌入式 PowerPC405 處理器和 3.125 Gbps RocketIO 串行收發器(Serial transceiver)。圖-4.3 為 ML310 FPGA 測試板的家族成員，而我們選用的是 XVC2VP30，在這我們須特別留意的是它的 BlockRAM 個數。XVC2VP30 共有 136 個 BlockRAM，每一個 BlockRAM 可以儲存 18,432(18K)個位元，故最高可

用的 BlockRAM 空間為 2448K 個位元。每一 BlockRAM 的儲存空間可分成資料記憶體(Data memory，16K 個位元)以及同位檢查記憶體(Parity memory，2K 個位元)兩個部份。所謂的同位檢查記憶體指的是，當每一個位址所取得的位元數稱為 OB(Obtained Bits)，則每一個位址內可加入的同位檢查位元(Parity bit)的個數為 $\left\lceil \frac{OB}{8} \right\rceil$。因此，同位檢查記憶體最大容量共 2K 個位元。此外，每一個位址所取得的資料寬度可配置為 1、2、4、9(8 個資料位元加 1 個同位檢查位元)、18(16 個資料位元加 2 個同位檢查位元)以及 36 個位元((32 個資料位元加 4 個同位檢查位元))，依使用者設定。

Device(1)	RocketIO Transceiver Blocks	PowerPC Processor Blocks	Logic Cells(2)	CLB (1 = 4 slices = max 128 bits)		18 X 18 Bit Multiplier Blocks	Block SelectRAM+		DCMs	Maximum User I/O Pads
				Slices	Max Distr RAM (Kb)		18 Kb Blocks	Max Block RAM (Kb)		
XC2VP2	4	0	3,168	1,408	44	12	12	216	4	204
XC2VP4	4	1	6,768	3,008	94	28	28	504	4	348
XC2VP7	8	1	11,088	4,928	154	44	44	792	4	396
XC2VP20	8	2	20,880	9,280	290	88	88	1,584	8	564
XC2VPX20	8(4)	1	22,032	9,792	306	88	88	1,584	8	552
XC2VP30	8	2	30,816	13,696	428	136	136	2,448	8	644
XC2VP40	0(3), 8, or 12	2	43,632	19,392	606	192	192	3,456	8	804
XC2VP50	0(3) or 16	2	53,136	23,616	738	232	232	4,176	8	852
XC2VP70	16 or 20	2	74,448	33,088	1,034	328	328	5,904	8	996
XC2VPX70	20(4)	2	74,448	33,088	1,034	308	308	5,544	8	992
XC2VP100	0(3) or 20	2	99,216	44,096	1,378	444	444	7,992	12	1,164

圖-4.3：ML310 FPGA 測試板的家族成員

每一個 BlockRAM 皆可宣告成單埠同步塊記憶體(Single-Port Synchronous Block RAM)或雙埠同步塊記憶體(Dual-Port Synchronous Block RAM)。其宣告方式如下：

- 單埠同步塊記憶體，宣告為 RAMB16_Sn，n 依使用者設定。

```
RAMB16_Sn user_instance_name (.DO (user_DO),
                              .DOP (user_DOP),
                              .ADDR (user_ADDR),
                                   .CLK (user_CLK),
                                   .DI (user_DI),
                                   .DIP (user_DIP),
                                   .ENB(user_EN),
                                   .SSR (user_SSR),
                                   .WE (user_WE));

defparam user_instance_name.INIT_00 = 256'h_hex_value;
defparam user_instance_name.INIT_01 = 256'h_hex_value;
defparam user_instance_name.INIT_02 = 256'h_hex_value;
```

defparam *user_instance_name*.INIT_03 = 256’h_*hex_value*
defparam *user_instance_name*.INIT_04 = 256’h_*hex_value*;
defparam *user_instance_name*.INIT_05 = 256’h_*hex_value*;
defparam *user_instance_name*.INIT_06 = 256’h_*hex_value*;
defparam *user_instance_name*.INIT_07 = 256’h_*hex_value*;
defparam *user_instance_name*.INIT_08 = 256’h_*hex_value*;
defparam *user_instance_name*.INIT_09 = 256’h_*hex_value*;
defparam *user_instance_name*.INIT_0A = 256’h_*hex_value*;
defparam *user_instance_name*.INIT_0B = 256’h_*hex_value*;
defparam *user_instance_name*.INIT_0C = 256’h_*hex_value*;
defparam *user_instance_name*.INIT_0D = 256’h_*hex_value*;
defparam *user_instance_name*.INIT_0E = 256’h_*hex_value*;
defparam *user_instance_name*.INIT_0F = 256’h_*hex_value*;
defparam *user_instance_name*.INIT_10 = 256’h_*hex_value*;
defparam *user_instance_name*.INIT_11 = 256’h_*hex_value*;
defparam *user_instance_name*.INIT_12 = 256’h_*hex_value*;
defparam *user_instance_name*.INIT_13 = 256’h_*hex_value*;
defparam *user_instance_name*.INIT_14 = 256’h_*hex_value*;
defparam *user_instance_name*.INIT_15 = 256’h_*hex_value*;
defparam *user_instance_name*.INIT_16 = 256’h_*hex_value*;
defparam *user_instance_name*.INIT_17 = 256’h_*hex_value*;
defparam *user_instance_name*.INIT_18 = 256’h_*hex_value*;
defparam *user_instance_name*.INIT_19 = 256’h_*hex_value*;
defparam *user_instance_name*.INIT_1A = 256’h_*hex_value*;
defparam *user_instance_name*.INIT_1B = 256’h_*hex_value*;
defparam *user_instance_name*.INIT_1C = 256’h_*hex_value*;
defparam *user_instance_name*.INIT_1D = 256’h_*hex_value*;
defparam *user_instance_name*.INIT_1E = 256’h_*hex_value*;
defparam *user_instance_name*.INIT_1F = 256’h_*hex_value*;
defparam *user_instance_name*.INIT_20 = 256’h_*hex_value*;
defparam *user_instance_name*.INIT_21 = 256’h_*hex_value*;
defparam *user_instance_name*.INIT_22 = 256’h_*hex_value*;
defparam *user_instance_name*.INIT_23 = 256’h_*hex_value*;
defparam *user_instance_name*.INIT_24 = 256’h_*hex_value*;
defparam *user_instance_name*.INIT_25 = 256’h_*hex_value*;
defparam *user_instance_name*.INIT_26 = 256’h_*hex_value*;
defparam *user_instance_name*.INIT_27 = 256’h_*hex_value*;
defparam *user_instance_name*.INIT_28 = 256’h_*hex_value*;
defparam *user_instance_name*.INIT_29 = 256’h_*hex_value*;
defparam *user_instance_name*.INIT_2A = 256’h_*hex_value*;
defparam *user_instance_name*.INIT_2B = 256’h_*hex_value*;
defparam *user_instance_name*.INIT_2C = 256’h_*hex_value*;
defparam *user_instance_name*.INIT_2D = 256’h_*hex_value*;
defparam *user_instance_name*.INIT_2E = 256’h_*hex_value*;
defparam *user_instance_name*.INIT_2F = 256’h_*hex_value*;
defparam *user_instance_name*.INIT_30 = 256’h_*hex_value*;
defparam *user_instance_name*.INIT_31 = 256’h_*hex_value*;
defparam *user_instance_name*.INIT_32 = 256’h_*hex_value*;
defparam *user_instance_name*.INIT_33 = 256’h_*hex_value*;
defparam *user_instance_name*.INIT_34 = 256’h_*hex_value*;

defparam *user_instance_name*.INIT_35 = 256'h_*hex_value*;
defparam *user_instance_name*.INIT_36 = 256'h_*hex_value*;
defparam *user_instance_name*.INIT_37 = 256'h_*hex_value*;
defparam *user_instance_name*.INIT_38 = 256'h_*hex_value*;
defparam *user_instance_name*.INIT_39 = 256'h_*hex_value*;
defparam *user_instance_name*.INIT_3A = 256'h_*hex_value*;
defparam *user_instance_name*.INIT_3B = 256'h_*hex_value*;
defparam *user_instance_name*.INIT_3C = 256'h_*hex_value*;
defparam *user_instance_name*.INIT_3D = 256'h_*hex_value*;
defparam *user_instance_name*.INIT_3E = 256'h_*hex_value*;
defparam *user_instance_name*.INIT_3F = 256'h_*hex_value*;
defparam *user_instance_name*.INIT_A = *bit_value*;
defparam *user_instance_name*.INIT_B = *bit_value*;
defparam *user_instance_name*.INITP_00 = 256'h_*hex_value*;
defparam *user_instance_name*.INITP_01 = 256'h_*hex_value*;
defparam *user_instance_name*.INITP_02 = 256'h_*hex_value*;
defparam *user_instance_name*.INITP_03 = 256'h_*hex_value*;
defparam *user_instance_name*.INITP_04 = 256'h_*hex_value*;
defparam *user_instance_name*.INITP_05 = 256'h_*hex_value*;
defparam *user_instance_name*.INITP_06 = 256'h_*hex_value*;
defparam *user_instance_name*.INITP_07 = 256'h_*hex_value*;
defparam *user_instance_name*.SRVAL_A = *bit_value*;
defparam *user_instance_name*.SRVAL_B = *bit_value*;
defparam *user_instance_name*.WRITE_MODE_A = *string_value*;
defparam *user_instance_name*.WRITE_MODE_B = *string_value*;

上述中，正體字為宣告式，固定不變；斜體字為使用者所設定之變數或初始值。

■ 雙埠同步塊記憶體，RAMB16_Sm_Sn，m 與 n 依使用者設定。

RAMB16_Sm_Sn *user_instance_name* (.DOA (*user_DOA*),
.DOB (*user_DOB*),
.DOPA (*user_DOPA*),
.DOPB (*user_DOPB*),
.ADDRA (*user_ADDRA*),
.ADDRB (*user_ADDRB*),
.CLKA (*user_CLKA*),
.CLKB (*user_CLKB*),
.DIA *(user_DIA*),
.DIB (*user_DIB*),
.DIPA (*user_DIPA*),
.DIPB (*user_DIB*),
.ENA (*user_ENA*),
.ENB (*user_ENB*),
.SSRA (*user_SSRA*),
.SSRB (*user_SSRB*),
.WEA (*user_WEA*),
.WEB (*user_WEB*));

defparam *user_instance_name*.INIT_00 = 256’h_*hex_value*;
defparam *user_instance_name*.INIT_01 = 256’h_*hex_value*;
defparam *user_instance_name*.INIT_02 = 256’h_*hex_value*;
defparam *user_instance_name*.INIT_03 = 256’h_*hex_value*
defparam *user_instance_name*.INIT_04 = 256’h_*hex_value*;
defparam *user_instance_name*.INIT_05 = 256’h_*hex_value*;
defparam *user_instance_name*.INIT_06 = 256’h_*hex_value*;
defparam *user_instance_name*.INIT_07 = 256’h_*hex_value*;
defparam *user_instance_name*.INIT_08 = 256’h_*hex_value*;
defparam *user_instance_name*.INIT_09 = 256’h_*hex_value*;
defparam *user_instance_name*.INIT_0A = 256’h_*hex_value*;
defparam *user_instance_name*.INIT_0B = 256’h_*hex_value*;
defparam *user_instance_name*.INIT_0C = 256’h_*hex_value*;
defparam *user_instance_name*.INIT_0D = 256’h_*hex_value*;
defparam *user_instance_name*.INIT_0E = 256’h_*hex_value*;
defparam *user_instance_name*.INIT_0F = 256’h_*hex_value*;
defparam *user_instance_name*.INIT_10 = 256’h_*hex_value*;
defparam *user_instance_name*.INIT_11 = 256’h_*hex_value*;
defparam *user_instance_name*.INIT_12 = 256’h_*hex_value*;
defparam *user_instance_name*.INIT_13 = 256’h_*hex_value*;
defparam *user_instance_name*.INIT_14 = 256’h_*hex_value*;
defparam *user_instance_name*.INIT_15 = 256’h_*hex_value*;
defparam *user_instance_name*.INIT_16 = 256’h_*hex_value*;
defparam *user_instance_name*.INIT_17 = 256’h_*hex_value*;
defparam *user_instance_name*.INIT_18 = 256’h_*hex_value*;
defparam *user_instance_name*.INIT_19 = 256’h_*hex_value*;
defparam *user_instance_name*.INIT_1A = 256’h_*hex_value*;
defparam *user_instance_name*.INIT_1B = 256’h_*hex_value*;
defparam *user_instance_name*.INIT_1C = 256’h_*hex_value*;
defparam *user_instance_name*.INIT_1D = 256’h_*hex_value*;
defparam *user_instance_name*.INIT_1E = 256’h_*hex_value*;
defparam *user_instance_name*.INIT_1F = 256’h_*hex_value*;
defparam *user_instance_name*.INIT_20 = 256’h_*hex_value*;
defparam *user_instance_name*.INIT_21 = 256’h_*hex_value*;
defparam *user_instance_name*.INIT_22 = 256’h_*hex_value*;
defparam *user_instance_name*.INIT_23 = 256’h_*hex_value*;
defparam *user_instance_name*.INIT_24 = 256’h_*hex_value*;
defparam *user_instance_name*.INIT_25 = 256’h_*hex_value*;
defparam *user_instance_name*.INIT_26 = 256’h_*hex_value*;
defparam *user_instance_name*.INIT_27 = 256’h_*hex_value*;
defparam *user_instance_name*.INIT_28 = 256’h_*hex_value*;
defparam *user_instance_name*.INIT_29 = 256’h_*hex_value*;
defparam *user_instance_name*.INIT_2A = 256’h_*hex_value*;
defparam *user_instance_name*.INIT_2B = 256’h_*hex_value*;
defparam *user_instance_name*.INIT_2C = 256’h_*hex_value*;
defparam *user_instance_name*.INIT_2D = 256’h_*hex_value*;
defparam *user_instance_name*.INIT_2E = 256’h_*hex_value*;
defparam *user_instance_name*.INIT_2F = 256’h_*hex_value*;
defparam *user_instance_name*.INIT_30 = 256’h_*hex_value*;

defparam *user_instance_name*.INIT_31 = 256'h_*hex_value*;
defparam *user_instance_name*.INIT_32 = 256'h_*hex_value*;
defparam *user_instance_name*.INIT_33 = 256'h_*hex_value*;
defparam *user_instance_name*.INIT_34 = 256'h_*hex_value*;
defparam *user_instance_name*.INIT_35 = 256'h_*hex_value*;
defparam *user_instance_name*.INIT_36 = 256'h_*hex_value*;
defparam *user_instance_name*.INIT_37 = 256'h_*hex_value*;
defparam *user_instance_name*.INIT_38 = 256'h_*hex_value*;
defparam *user_instance_name*.INIT_39 = 256'h_*hex_value*;
defparam *user_instance_name*.INIT_3A = 256'h_*hex_value*;
defparam *user_instance_name*.INIT_3B = 256'h_*hex_value*;
defparam *user_instance_name*.INIT_3C = 256'h_*hex_value*;
defparam *user_instance_name*.INIT_3D = 256'h_*hex_value*;
defparam *user_instance_name*.INIT_3E = 256'h_*hex_value*;
defparam *user_instance_name*.INIT_3F = 256'h_*hex_value*;
defparam *user_instance_name*.INIT_A = *bit_value*;
defparam *user_instance_name*.INIT_B = *bit_value*;
defparam *user_instance_name*.INITP_00 = 256'h_*hex_value*;
defparam *user_instance_name*.INITP_01 = 256'h_*hex_value*;
defparam *user_instance_name*.INITP_02 = 256'h_*hex_value*;
defparam *user_instance_name*.INITP_03 = 256'h_*hex_value*;
defparam *user_instance_name*.INITP_04 = 256'h_*hex_value*;
defparam *user_instance_name*.INITP_05 = 256'h_*hex_value*;
defparam *user_instance_name*.INITP_06 = 256'h_*hex_value*;
defparam *user_instance_name*.INITP_07 = 256'h_*hex_value*;
defparam *user_instance_name*.SRVAL_A = *bit_value*;
defparam *user_instance_name*.SRVAL_B = *bit_value*;
defparam *user_instance_name*.WRITE_MODE_A = *string_value*;
defparam *user_instance_name*.WRITE_MODE_B = *string_value*;

上述中，正體字為宣告式，固定不變；斜體字為使用者所設定之變數或初始值。

接下來我們將分節去討論如何搭配測試板的特性與功能來實現系統各部分之操作。

4.3 測試文字檔產生與儲存設計

我們用來測試的文字檔是先透過軟體(C++)隨機產生 0_{10}~255_{10}(00_{16}~FF_{16})的

值，並且以十六進制儲存之，總共產生 64K 個位元(4 個 BlockRAM，其後會說明為何隨機產生的文字檔大小是 4 個 BlockRAM)。在設計測試文字檔(圖-4.1 中的 Text File 區塊)儲存方式時，我們須將搜尋視窗的參數 k 列入考慮。

我們此次的研究是將預先過濾器的搜尋視窗長度(m)設定為 10(因為 ClamAV 所釋放出的公開病毒碼中，最短長度為 10 個位元組)，搜尋視窗所設定區塊大小(k)為 4，這告訴了我們，預先過濾器每一次操作都是觀看搜尋視窗後 4 個位元組。因此，**預先過濾器須有能力在一次的讀取時間內能同時餵入屬於現搜尋視窗中的後 4 個位元組**。而驗證器(由 AC 演算法為實作主軸)的運作模式為當執行時，它每一次都需要讀取 1 個位元組，根據此位元組與現在狀態去決定下次所停留的狀態。

總觀上述，我們所實現的測試文字檔必須滿足可同時讀取 4 個位元組與每一次讀取 1 個位元組的操作模式，因此我們選擇 BlockRAM 的宣告型態為 RAMB16_S9_S9，這個型態每次所讀取出的資料為 8 個位元(等於 1 個位元組)，如圖-4.4 所示；並且將測試文字檔分成 4 個群組(text_bram0、text_bram1、text_bram2，以及 text_bram3)，分別儲存於 4 個 BlockRAM 之中，使其預先過濾器操作時可同時讀取到 4 個位元組。從圖-4.5 的簡例中，我們可以更清楚地了解其設計的準則與儲存的原理。從圖中我們可以得知，有一測試文字檔為『abcdefghijk0123456#*(@!dsaok.....』，則

- BlockRAM 的宣告型態為 RAMB16_S9_S9，表示資料記憶體的每一個儲存空間為 8 個位元。由於一個 BlockRAM 的資料記憶體大小為 16K 個位元，因此表示了共有 2^{11} 個儲存空間，其中 11 這個數字即為此 BlockRAM 的位址匯流排寬度。
- 我們將測試文字檔分成 4 個群組，分別儲存於 text_bram0、text_bram1、text_bram2，以及 text_bram3 等 4 個 BlockRAM。
- 存入方式：
  - text_bram0：儲存測試文字檔的第 $4i$ 個位元組。
  - text_bram1：儲存測試文字檔的第 $4i+1$ 個位元組。
  - text_bram2：儲存測試文字檔的第 $4i+2$ 個位元組。
  - text_bram3：儲存測試文字檔的第 $4i+3$ 個位元組。

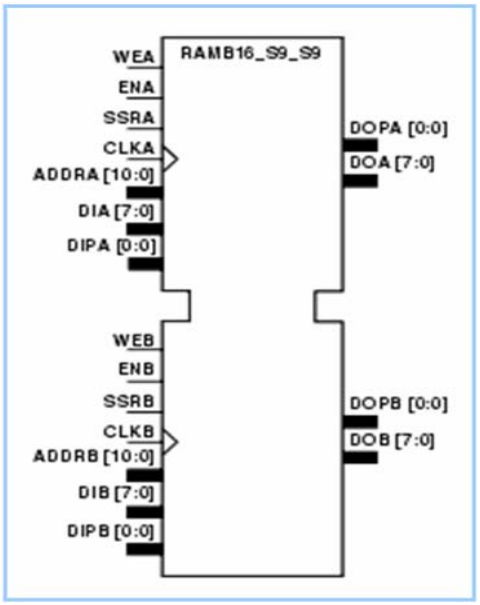

圖-4.4：text_bram0 ~ text_bram3 的宣告型態為 RAMB16_S9_S9

測試文字檔:abcdefghijk0123456#*(@!dsaok.....

存入之位址 / BlockRAM編號	0...0000	0...0001	0...0010	0...0011	0...0100	0...0101	0...0110
TextRAM0	a	e	i	1	5	(	s
TextRAM1	b	f	j	2	6	@	a
TextRAM2	c	g	k	3	#	!	o
TextRAM3	d	h	0	4	*	d	k

圖-4.5：測試文字檔設計簡例

我們將 Text File 區塊單獨寫成一個模組(module)，稱為 TextRAM 模組，其內部包含了測試文字檔 text_bram0、text_bram1、text_bram2，以及 text_bram3 與一些控制電路。在 TextRAM 模組中，對於預先過濾器而言，它須控制預先過濾器是否應停止運作。**若當我們測試文字檔內儲存之內容都已經偵測完畢或待偵測(剩餘)位元組個數低於下次須餵入的位元組個數時，TextRAM 模組須告知 PreFilter 模組應停止運作**，因為已無或不足之位元組個數可以再餵入 PreFilter 模組。而後我們都統稱儲存測試文字檔的 BlockRAM 為 text_bram 群。

4.4 可疑位址的存取設計

我們在第三章中已說明，當我們抓到一個可疑字樣時須將**搜尋視窗的起始位址**儲存至可疑字樣位址儲存處(圖 4-1 中 Jail 區塊)，待驗證器決策是否真為字樣集合中的一員。

上述 4.3 節中，我們所選定實現 TextRAM 模組的 text_bram 儲存型態為 RAMB16_S9_S9，共有 2^{11} 個儲存空間(意指此 BlockRAM 儲存型態的位址匯流排寬度為 11 個位元)。Jail 若須將搜尋視窗的起始位址存入，則其儲存空間須大於或等於 11+2 個位元(11 個位元的位址匯流排寬度加上 2 個位元的 text_bram 編號)。因此，我們選定一個適當可實現儲存可疑字樣位址的 BlockRAM 儲存型態為 RAMB16_S18_S18，此儲存型態表示資料記憶體的每一個儲存空間為 16 個位元，如圖-4.6 所示。

我們將 Jail 區塊中用來儲存可疑字樣位址的 BlockRAM 稱為 jail_bram。由於一個 jail_bram 的資料記憶體大小為 16K 個位元，因此表示了共有 2^{10} 個儲存空間，其中 10 這個數字即為此 jail_bram 的位址匯流排寬度。我們針對 RAMB16_S18_S18 的儲存空間之設計有如下的歸納：

- 儲存空間之位元數：16 位元
- 儲存空間之初始值(initial value):000_00_00000000000
- 可疑字樣的起始位址存入 jail_bram 形式：100_??_XXXXXXXXXXX
  - ??：表示此可疑字樣的起始位址是屬於測試文字檔中哪一個 BlockRAM (text_bram0、text_bram1、text_bram2、text_bram3)，佔用 2 個位元。
  - XXXXXXXXXXX：表示此編號??之 text_bram 的位址，佔用 11 個位元。
  - 當預先過濾器偵測到可疑字樣時，它會將其搜尋視窗的起始位址記錄下來，並將可疑字樣的起始位址存入於 jail_bram 之中。儲存之形式為 100_??_XXXXXXXXXXX。意指當儲存空間最高位元設定

為 1，驗證器須將處理之。

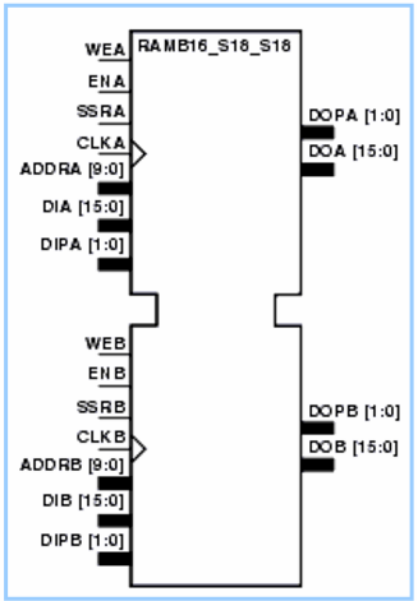

圖-4.6：jail_bram 的宣告型態為 RAMB16_S18_S18

我們將 Jail 區塊單獨寫成一個模組並命名成 JailRAM。在 JailRAM 模組中有一控制電路負責偵測 jail_bram 下一個儲存空間內的儲存值中最高位元是否為 1，若否，PreFilter 區塊將可再存入所找到的可疑字樣的起始位址至 jail_bram；若是，PreFilter 區塊則須暫停運作並保持著現有狀態直至 JailRAM 模組回報 jail_bram 下一個儲存空間內的儲存值為初始值狀態(000_00_00000000000)方可再行運作。jail_bram 回報它下一個儲存空間不為初始值狀態時，這表示驗證器(AC)尚未或正在對此儲存空間內所儲存的可疑字樣起始位址作一仔細之比對，如果這時 PreFilter 區塊沒有暫停運作並將所找到的可疑字樣的起始位址儲存至此儲存空間內，則即產生資料的複寫(overwrite)。因此，我們須等待驗證器對此儲存空間內所儲存的可疑字樣起始位址處理完畢後，將此儲存空間內的值重新儲存為初始值狀態，這時即表示此儲存空間內的儲存值中最高位元不為 1，PreFilter 區塊若有找到可疑的字樣，則可將可疑字樣的起始位址儲存至此儲存空間內。

4.5 預先過濾器設計

在詳細介紹完『測試文字檔的產生與設計』和『可疑位址的存取設計』後，我們將有足夠的資訊來完成預先過濾器之設計。如圖-4.1 所示，PreFilter 區塊即為我們此系統最大的賣點，預先過濾器。我們將單獨把 PreFilter 區塊寫成一個模組並命名為 Prefilter。

圖-4.7 為我們欲實現於 FPGA 之預先過濾器的雛型架構；如圖所示，在 Prefilter 模組中，我們將預先過濾器內的功能區分成兩部分，其一為『雜湊產生器(Hash generator)』，另一為『輸入控制器(Input controller)』，也就是 HashGen 模組和 InContr 模組，而在 HashGen 模組中又包含一個 HashRule 模組。我們將在下述次子節中分別去介紹。

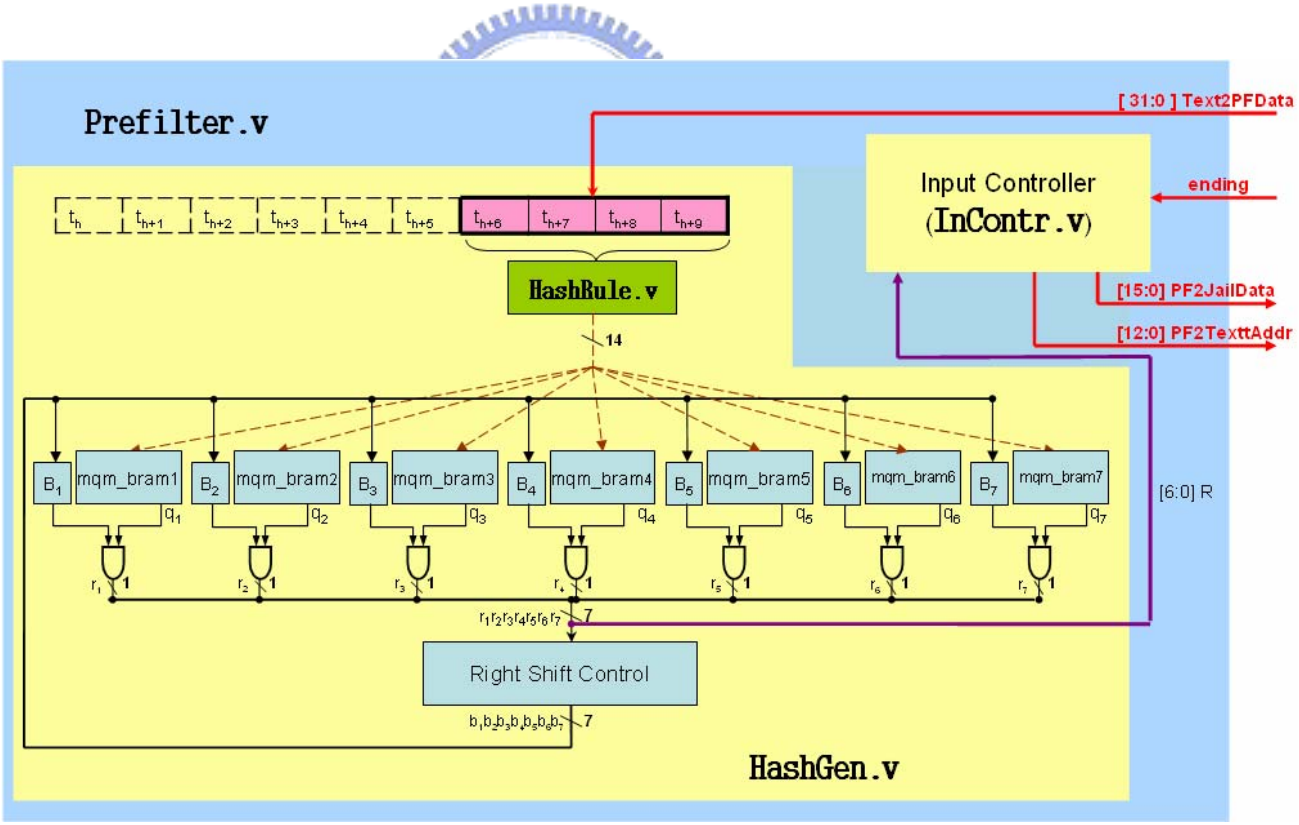

圖-4.7：預先過濾器實現於 FPGA 之架構圖

4.5.1. 雜湊產生器 (Hash Generator)

在 Prefilter 模組中，我們將接收到來自 TextRAM 模組送來的 4 個位元組(32 個位元，[31:0]Text2PFData)，Prefilter 模組先將其 32 個位元的資料送至 HashGen 模組中的 HashRule 模組，利用一雜湊函數使其產生 14 個位元的雜湊值，再將此 14 個位元回送至 HashGen 模組中作為 mqm_bram1~mqm_bram7 等 7 個 BlockRAM 的位址匯流排 (這 7 個 BlockRAM 的宣告型態為 RAMB16_S1，為單埠同步塊記憶體，這個型態每次所讀取出的資料為 1 個位元，如圖-4.8 所示。) 。mqm_bram1~mqm_bram7 會將此位址匯流排所指向的儲存空間內的資料輸出，即 $QB = q_1 \sim q_7$ ，並分別與主位元組列 $MB = b_1 \sim b_7$ 作逐位和運算(Bitwise AND operator)，得到 $R = r_1 \sim r_7$ 之結果。而圖-4.7 中的 Right Shift Control (RSC)區塊則會去判斷輸入 R 最右位元為 1 所發生的位置，再根據 3.2.4 節中所提的向右移動方式，決定出下一次 MB 內所儲存的值。HashGen 模組也會將 R 的結果輸出到 InContr 模組，使其可決定出下次要餵進來的 4 個位元組。

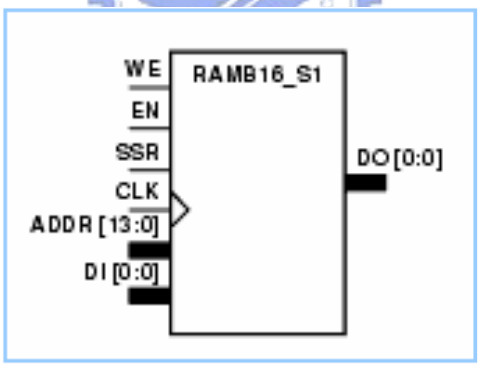

圖-4.8：mqm_bram1~mqm_bram7 的宣告型態為 RAMB16_S1

4.5.2. 輸入控制器(Input Controller)

InContr 模組接收到來自 HashGen 模組送來的輸入 R，我們將根據此資料分析出搜尋視窗可(向左)移動幾個位元組，並且決定出其下次要餵進來的 4 個位元組的起始位址(13 個位元，[12:0]PF2TextAddr)並送往 TextRAM 模組，TextRAM

模組內則會根據其撰寫之規則，依此接收到的位址帶出適當的 4 個位元組，並將送至 Prefilter 模組；InContr 模組所接收到的 R 的最低位元為 1 (R 為 7 個位元的向量，$R=[r_1r_2r_3r_4r_5r_6r_7]$，其宣告為 wire [6:0] R)，我們則須計算出此時搜尋視窗的起始位址(16 個位元，[15:0]PF2JailData)並送往 JailRAM 模組儲存之。若 TextRAM 模組已無或不足之位元組個數可以再餵入 PreFilter 模組，則 TextRAM 模組將利用一條宣告為 wire 的變數 ending 告知 PreFilter 模組應停止運作(此時 ending=1)。

第五章

實驗數據與結果

5.1 測試環境介紹

我們此次研究將在測試平台 Xilinx ISE 8.1i 下開發，所選用的硬體描述語言是 Verilog，並利用 ModelSim 作為跑程式的模擬器以及 Xilinx XST 作為程式的合成器(synthesizer)，並且選擇 ClamAV 在 2008/2 月所釋放出的原病毒碼(字樣)中的前 10,000 隻病毒作為測試。

我們在驗證程式的正確性主要分為三個步驟。首先，程式完成後，我們利用人工方式推導出幾個階段的結果並與 ModelSim 的結果比較；第二步驟，若人工方式推導出幾個階段的結果與 ModelSim 的結果相符合，則我們將利用 C++所撰寫的程式之結果與 ModelSim 的結果比較，若結果比對亦為相符合，則我們可以確定我們所撰寫出來的 RTL 程式碼是正確無誤的；最後，我們利用由 Xilinx 公司的一套軟體，ChipScope Pro8.1i，進行將 RTL 程式碼燒入 FPGA 板子後的驗證。ChipScope 軟體的功能就像一台軟體版的邏輯分析儀，它可以將欲調試(Debugging)的腳位或訊號抓取出來顯示於螢幕上一一檢視比對其結果是否與模擬時一致。

我們欲撰寫的程式架構如下所示，

- System 模組 (System.v)
  - AC 模組 (AC.v)
  - Prefilter 模組 (Prefilter.v)
    - ✓ InContr 模組 (InContr.v)

- ✓ HashGen 模組 (HashGen.v)
- ➢ TextRAM 模組 (TextRAM.v)
- ➢ JailRAM 模組 (JailRAM.v)

一開始將由 Prefilter 模組傳送搜尋視窗的起始(初始)位址給 TextRAM 模組進行抓取區塊大小(Block Size)內的資料(4 個位元組)。TextRAM 模組在接收到位址(wire [12:0] PF2TextAddr)後會計算出 text_bram 群(text_bram0~text_bram3)內每個 BlockRAM 之位址匯流排的相對應位址，text_bram0~text_bram3 的位址匯流排在接收到各自對應的位址後，在下一個時鐘(Clock)後會各自輸出為 1 個位元組(8 個位元)並作一適當的排列連結(concatenation)成為一個具 32 位元的資料匯流排，其資料匯流排將送至 Prefilter 模組內的 HashGen 模組。HashGen 模組利用一雜湊函數將 32 個位元轉成 14 個位元，並作為 mqm_bram 群(mqm_bram1~mqm_bram7)的位址匯流排的輸入；因此，在收到其 14 個位元的位址後，mqm_bram 群會在下一個時鐘後得到其 mqm_bram 群的輸出並將其輸出排列連結(R，為一 7 個位元之向量，每一個 BlockRAM 輸出值均為一個位元)。HashGen 模組內部會自行根據得到的 R 計算出下次主位元組列內的值(MB，為一 7 個位元之向量)。HashGen 模組亦會將得到的 R 送至 Prefilter 模組內的另一個模組，InContr 模組。InContr 模組會根據此接收到的 R 先行判定是否為一可疑字樣的發生，若是，則會計算出現在搜尋視窗的起始位址並將計算結果存入一 16 位元暫存器(reg [15:0] PF2JailData)；若否，則會將 16 個位元均設成 0，存入 PF2JailData 暫存器。InContr 模組也會根據接收到的 R 計算出下次搜尋視窗可(向左)移動的距離(wire [4:0] shift_bytes)，則一 13 位元暫存器，PF2TextAddr，會依 shift_bytes 換算出下次區塊大小的起始位址並送往 TextRAM 模組請求下次應傳送給 Prefilter 模組之區塊大小內的資料。若 TextRAM 模組已無或不足之位元組個數可以再餵入 PreFilter 模組，則 TextRAM 模組將利用一條宣告為 wire 的變數 ending 告知 PreFilter 模組應停止運作(此時 ending=1)。從上述可知，**當 TextRAM 模組接收到 13 位元的 PF2TextAddr 訊號，到下一次在接收到此訊號所需之週期為 3 個時鐘的時間**。

5.2 Prefilter 模組的修正版

在這，我們將針對改善 Prefilter 模組所抓到的可疑字樣的個數提出兩個版本，一個將以速度為導向，另一個將以功率為導向。圖-5.1 為一開始根據演算法所設計的版本，圖-5.2 是以速度為導向的 Prefilter 模組修正版，圖-5.3 是以功率為導向的 Prefilter 模組修正版。

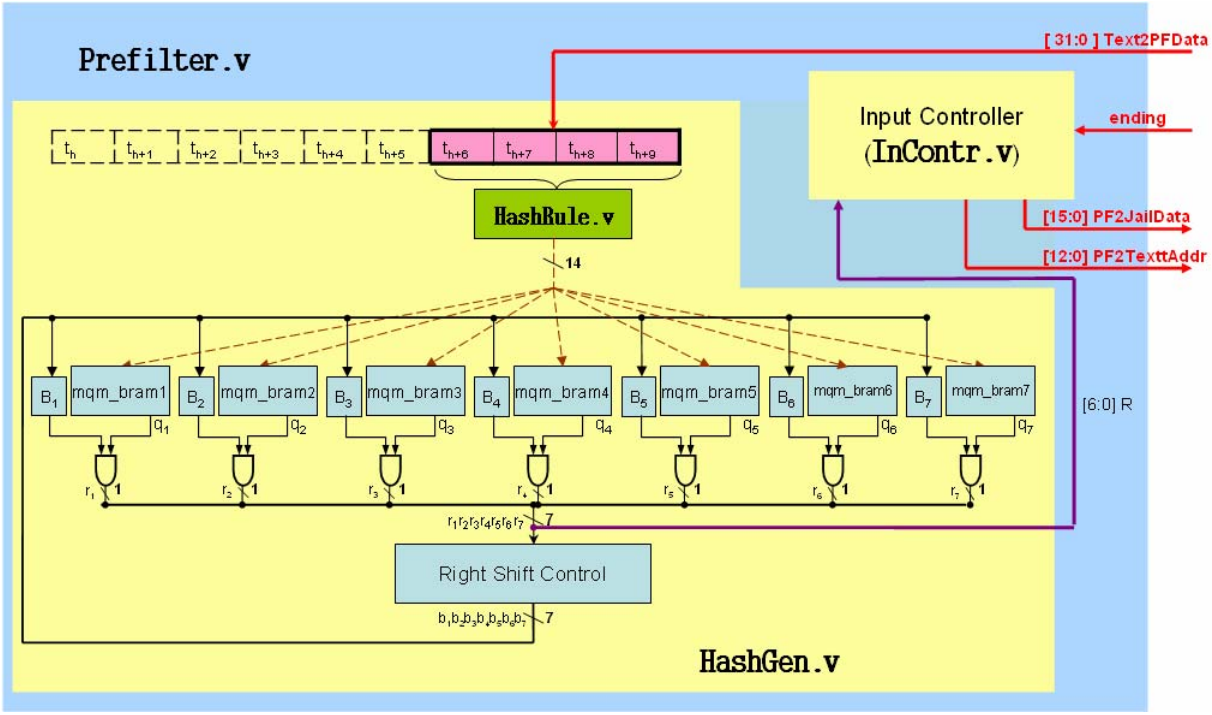

圖-5.1：PF1，原始的 Prefilter 模組的版本

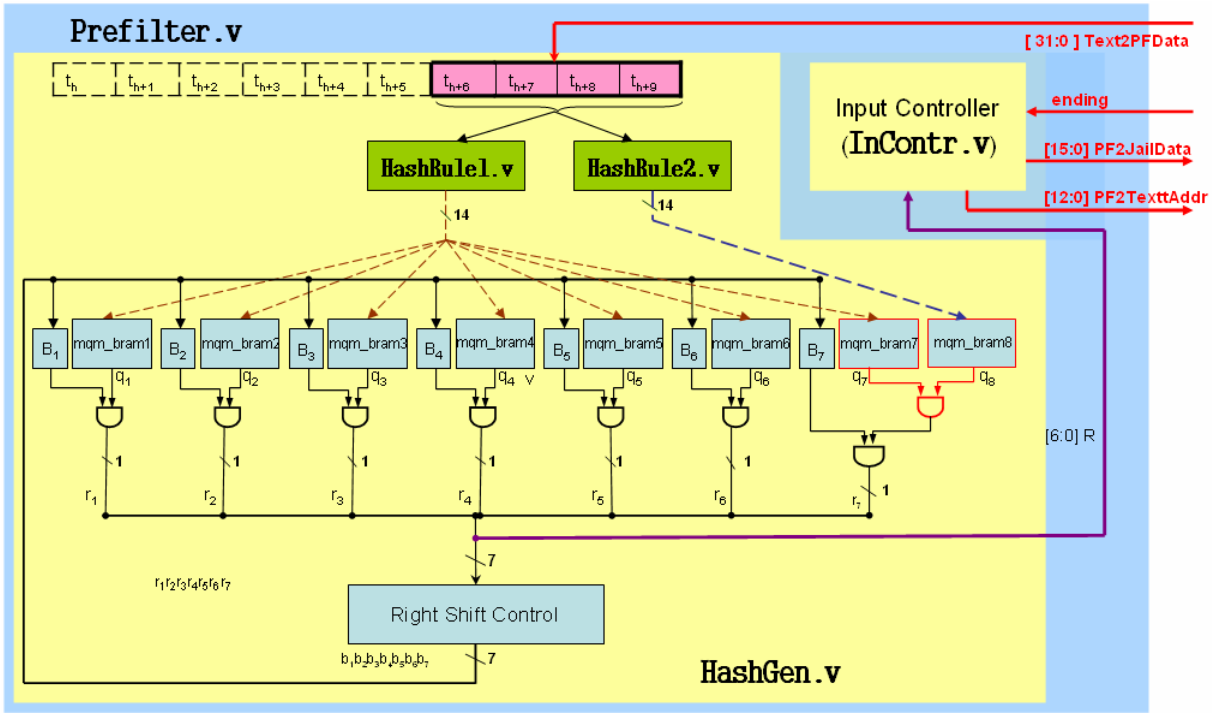

圖-5.2：PF2，以速度為導向的 Prefilter 模組修正版

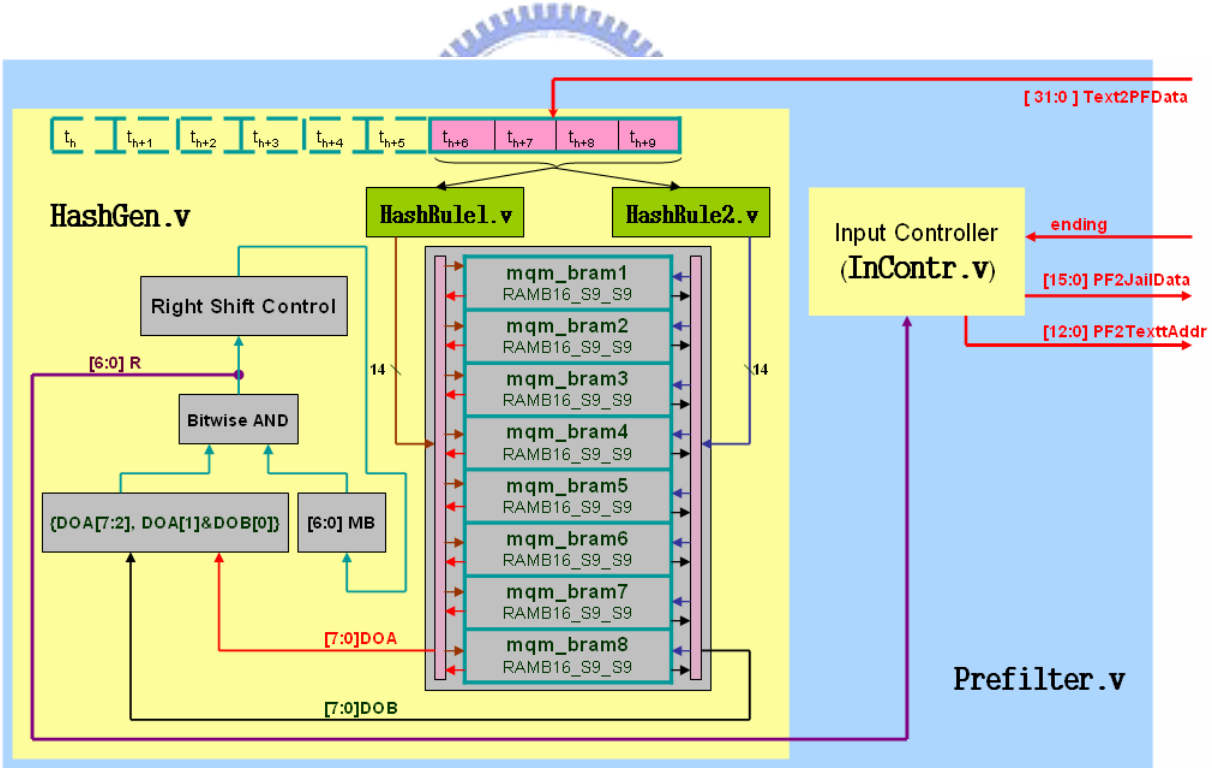

圖-5.3：PF3，以功率為導向的 Prefilter 模組修正版

在圖-5.2，我們多加入了一個 BlockRAM 並命名為 mqm_bram8。mqm_bram8 的宣告型態與 mqm_bram1～mqm_bram7 一樣，均為 RAMB16_S1 的單埠同步記憶體。它與 mqm_bram7 一樣是針對字樣的第 7 個位元組到第 10 個位元組，不同

的是它與 mqm_bram7 使用各自的雜湊函數，其目的為輔助字樣的第 7 個位元組到第 10 個位元組的偵測正確性。這也就是說，在多加入一個 mqm_bram8 後，因字樣的第 7 個位元組到第 10 個位元組獲得更精準的偵測判斷而使誤報率(false positive rate)變小，使其被判斷為可疑字樣的個數將獲得大幅的降低，這對於整體所設計之系統的生產量(Throughput)將有明顯的改善。

在圖-5.3，此設計的 MQM 儲存方式與前述 PF1 與 PF2 不同。PF1 與 PF2 由於設計方式使其每一個 BlockRAM 都處於永遠工作的狀態，直至 Prefilter 模組停止運作為止，這對於功率上的消耗佔有一定程度的比例。因此，我們又設計出另一種修正的版本，為了使功率上得到更佳的解。在圖中，我們亦宣告了 8 個 BlockRAM，其每一個的宣告型態均為 RAMB16_S9_S9，為一個雙埠同步記憶體，這是為了使我們每一個要得到 R 的值只需去讀取一個 BlockRAM 即可。由於我們所設計之預先過濾器所得到 R 值僅需 7 個位元，但我們所宣告之 BlockRAM 型態每次輸出值為 8 個位元，這比我們要的 R 值多 1 個位元；因此，我們一樣利用此位元作為輔助字樣的第 7 個位元組到第 10 個位元組能獲得更精準的偵測判斷而使誤報率(false positive rate)變小，使其被判斷為可疑字樣的個數將獲得大幅的降低。我們利用 HashRule1 模組的輸出作為 A 埠位址匯流排之輸入，得到高位的 7 個位元($r_7r_6r_5r_4r_3r_2r_1$)並利用 HashRule2 模組的輸出作為 B 埠位址匯流排之輸入，得到最低位元(r_0')，則 $R=\{r_7,r_6,r_5,r_4,r_3,r_2,r_1\cdot r_0'\}$，這也就是為什麼 BlockRAM 要宣告成『雙埠』的原因。故，PF3 每次在讀取 mqm_bram 群時最多僅需讀取 2 個 BlockRAM，這比 PF1 與 PF2 要少 4 倍，功率上節省了 75%。PF3 雖然可使功率獲得大幅的改善，但由於需付出一些控制電路來控制每次需讀取在 mqm_bram 群中的哪個 BlockRAM 以及要選擇哪些 BlockRAM 的輸出來作運算，這使得我們在時鐘頻率無法獲得像 PF1 與 PF2 一樣，最高僅可達到 53%，這使得預先過濾器部分的生產量降低約 50%，我們在下子節中會提出一些數據看到每一種預先過濾器的表現。

5.3 數據與結果

# of patterns	PF1 (Gbps)			PF1_NMB (Gbps)			$\frac{PF1-PF1\_NMB}{PF1\_NMB}$
	Min	Max	Aver.	Min	Max	Aver.	
1,000	2.756	2.800	**2.780**	3.199	3.292	**3.236**	**-14.1%**
2,000	2.525	2.569	**2.542**	2.800	2.895	**2.859**	**-11.1%**
3,000	2.339	2.411	**2.374**	2.510	2.573	**2.547**	**-6.8%**
4,000	2.191	2.232	**2.218**	2.242	2.306	**2.278**	**-2.6%**
5,000	2.084	2.100	**2.092**	2.009	2.069	**2.047**	**2.2%**
6,000	1.958	2.005	**1.981**	1.816	1.884	**1.842**	**7.6%**
7,000	1.874	1.920	**1.899**	1.683	1.727	**1.706**	**11.3%**
8,000	1.781	1.827	**1.809**	1.552	1.595	**1.566**	**15.5%**
9,000	1.716	1.750	**1.738**	1.440	1.485	**1.463**	**18.8%**
10,000	1.651	1.691	**1.679**	1.347	1.384	**1.367**	**22.9%**

表-5.1：PF1 與 PF1_NMB 的生產量之比較(隨機產生檔)

# of patterns	PF1 (Gbps)			PF1_NMB (Gbps)			$\frac{PF1-PF1\_NMB}{PF1\_NMB}$
	Min	Max	Aver.	Min	Max	Aver.	
1,000	2.806	2.882	**2.848**	3.258	3.358	**3.324**	**-14.3%**
2,000	2.574	2.675	**2.642**	2.948	3.025	**2.977**	**-11.2%**
3,000	2.436	2.520	**2.492**	2.680	2.766	**2.727**	**-8.6%**
4,000	2.313	2.374	**2.346**	2.359	2.465	**2.429**	**-3.4%**
5,000	2.216	2.269	**2.240**	2.170	2.295	**2.232**	**0.4%**
6,000	2.075	2.193	**2.134**	1.889	2.096	**2.010**	**6.2%**
7,000	1.980	2.097	**2.040**	1.791	1.959	**1.884**	**8.3%**
8,000	1.866	2.017	**1.950**	1.686	1.837	**1.775**	**9.9%**
9,000	1.825	1.945	**1.891**	1.567	1.740	**1.674**	**13.1%**
10,000	1.764	1.854	**1.815**	1.435	1.650	**1.564**	**16.2%**

表-5.2：PF1 與 PF1_NMB 的生產量之比較(window 執行檔)

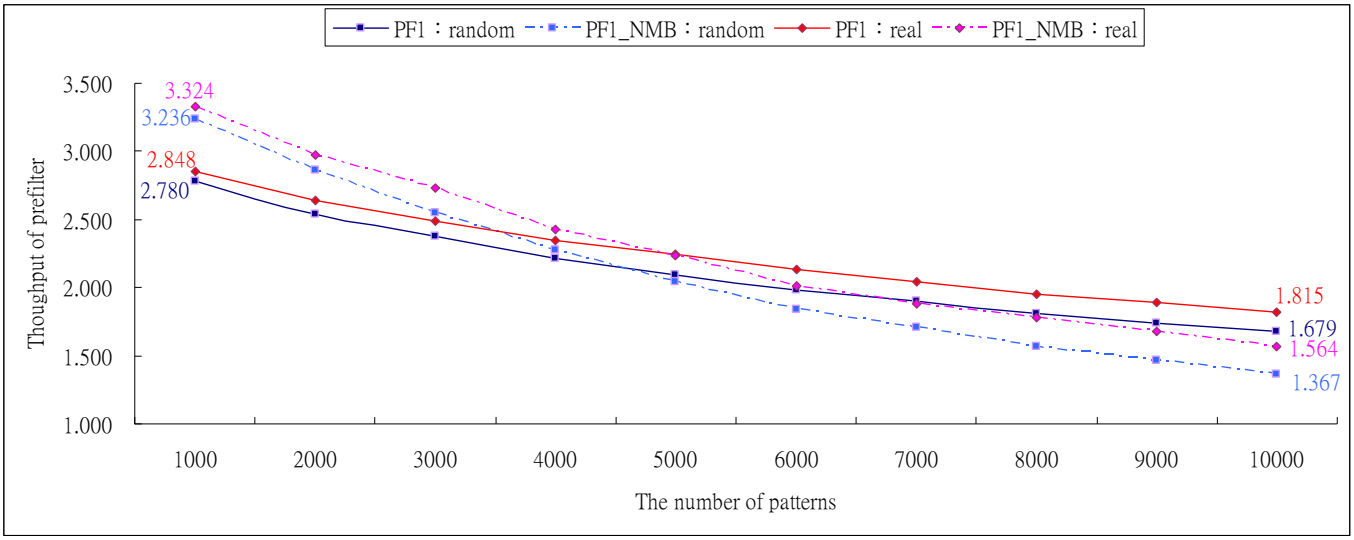

圖-5.4：PF1 與 PF1_NMB 的生產量表現

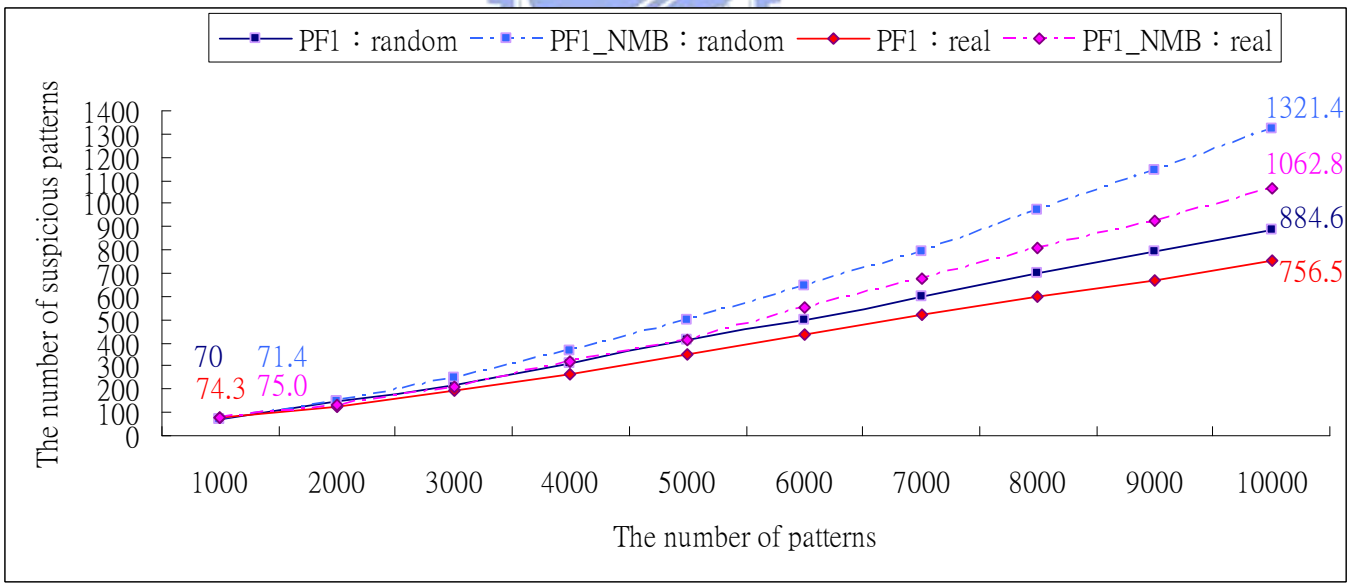

圖-5.5：PF1 與 PF1_NMB 所抓到的可疑字樣個數

# of patterns	Random files (PF1/PF1_NMB)		Real files (PF1/PF1_NMB)	
	The move times of search window	The number of shift bytes	The move times of search window	The number of shift bytes
1,000	1334.2/1358.8	6.130/6.021	1307.7/1326.7	6.256/6.167
2,000	1459.6/1537.6	5.606/5.322	1410.3/1484.7	5.800/5.509
3,000	1562.6/1726	5.236/4.740	1494.3/1616.3	5.474/5.061
4,000	1672.8/1930	4.890/4.238	1583.3/1816.7	5.166/4.504
5,000	1773.2/2147.8	4.612/3.808	1663.0/1988.7	4.920/4.114
6,000	1872.6/2386.4	4.368/3.428	1755.3/2221.7	4.661/3.686
7,000	1953.8/2577.2	4.187/3.174	1836.0/2366.0	4.456/3.460
8,000	2051.2/2807.4	3.988/2.914	1926.7/2509.0	4.246/3.263
9,000	2135.4/3004.4	3.831/2.723	1981.7/2665.7	4.129/3.072
10,000	2209.8/3217.2	3.702/2.543	2060.0/2869.7	3.971/2.857

表-5.3：PF1 與 PF1_NMB 之搜尋視窗平均移動次數和平均移動的位元組個數

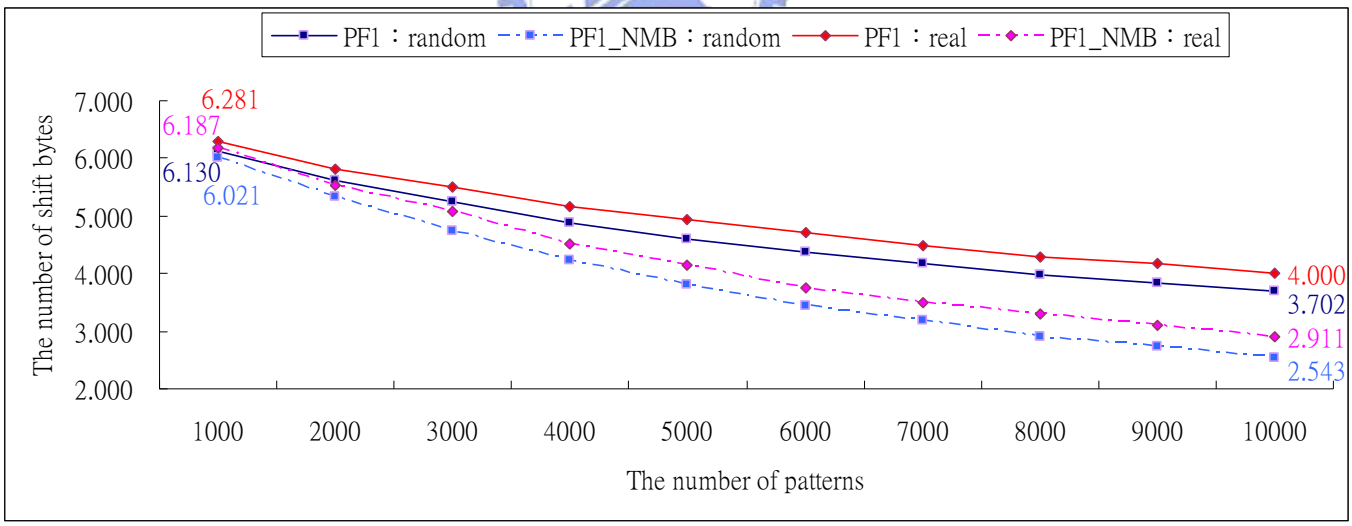

圖-5.6：PF1 與 PF1_NMB 的平均移動的位元組個數表現

# of patterns	PF2 (Gbps)			PF2_NMB (Gbps)			$\frac{PF2-PF2\_NMB}{PF2\_NMB}$
	Min	Max	Aver.	Min	Max	Aver.	
1,000	2.756	2.800	**2.780**	3.199	3.292	**3.236**	**-14.1%**
2,000	2.525	2.569	**2.542**	2.800	2.895	**2.859**	**-11.1%**
3,000	2.339	2.411	**2.374**	2.510	2.573	**2.547**	**-6.8%**
4,000	2.191	2.232	**2.218**	2.242	2.306	**2.278**	**-2.6%**
5,000	2.084	2.100	**2.092**	2.009	2.069	**2.047**	**2.2%**
6,000	1.958	2.005	**1.981**	1.816	1.884	**1.842**	**7.6%**
7,000	1.874	1.920	**1.899**	1.683	1.727	**1.706**	**11.3%**
8,000	1.781	1.827	**1.809**	1.552	1.595	**1.566**	**15.5%**
9,000	1.716	1.750	**1.738**	1.440	1.485	**1.463**	**18.8%**
10,000	1.651	1.691	**1.679**	1.347	1.384	**1.367**	**22.9%**

表-5.4：PF2 與 PF2_NMB 的生產量之比較(隨機產生檔)

# of patterns	PF2 (Gbps)			PF2_NMB (Gbps)			$\frac{PF2-PF2\_NMB}{PF2\_NMB}$
	Min	Max	Aver.	Min	Max	Aver.	
1,000	2.806	2.882	**2.848**	3.258	3.358	**3.324**	**-14.3%**
2,000	2.574	2.675	**2.642**	2.948	3.025	**2.977**	**-11.2%**
3,000	2.436	2.520	**2.492**	2.680	2.766	**2.727**	**-8.6%**
4,000	2.313	2.374	**2.346**	2.359	2.465	**2.429**	**-3.4%**
5,000	2.216	2.269	**2.240**	2.170	2.295	**2.232**	**0.4%**
6,000	2.075	2.193	**2.134**	1.889	2.096	**2.010**	**6.2%**
7,000	1.980	2.097	**2.040**	1.791	1.959	**1.884**	**8.3%**
8,000	1.866	2.017	**1.950**	1.686	1.837	**1.775**	**9.9%**
9,000	1.825	1.945	**1.891**	1.567	1.740	**1.674**	**13.1%**
10,000	1.764	1.854	**1.815**	1.435	1.650	**1.564**	**16.2%**

表-5.5：PF2 與 PF2_NMB 的生產量之比較(window 執行檔)

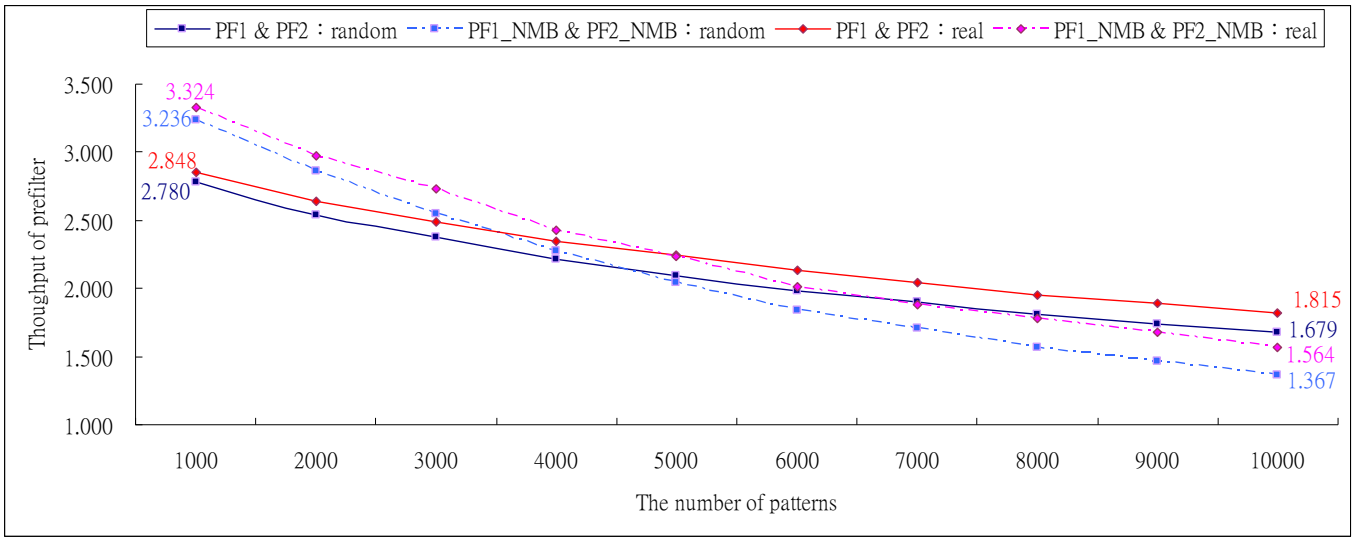

圖-5.7：PF2 與 PF2_NMB 的生產量表現

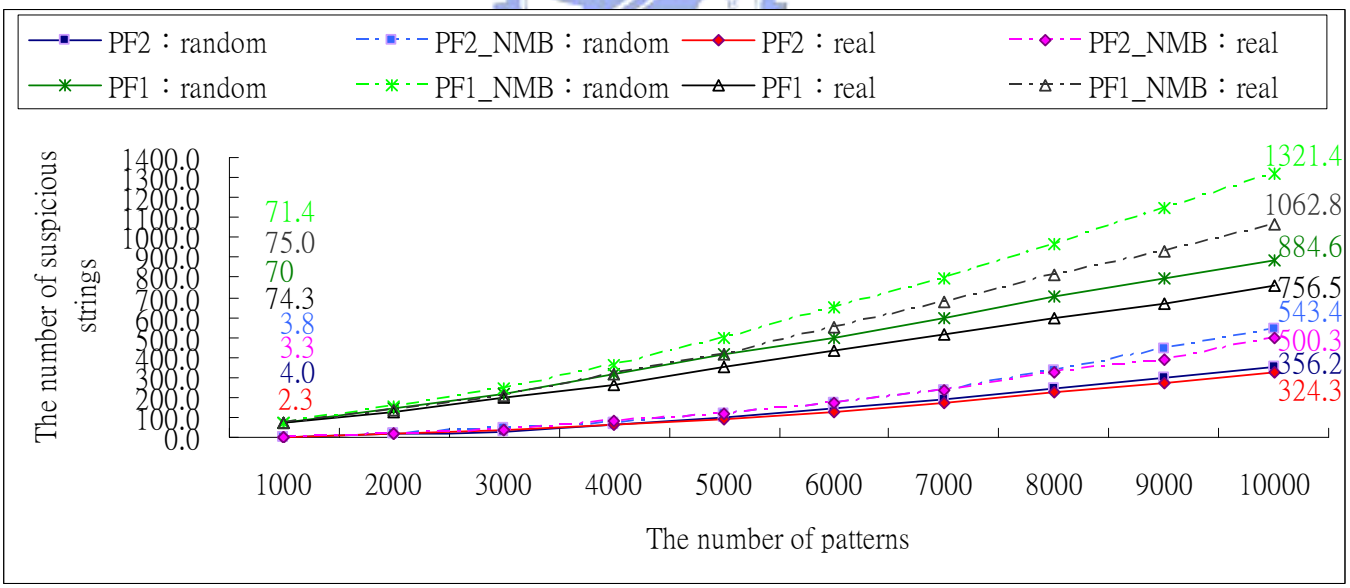

圖-5.8：PF2 與 PF2_NMB 所抓到的可疑字樣個數

# of patterns	Random files (PF2/PF2_NMB)		Real files (PF2/PF2_NMB)	
	The move times of search window	The number of shift bytes	The move times of search window	The number of shift bytes
1,000	1334.2/1358.8	6.130/6.021	1307.7/1326.7	6.256/6.167
2,000	1459.6/1537.6	5.606/5.322	1410.3/1484.7	5.800/5.509
3,000	1562.6/1726	5.236/4.740	1494.3/1616.3	5.474/5.061
4,000	1672.8/1930	4.890/4.238	1583.3/1816.7	5.166/4.504
5,000	1773.2/2147.8	4.612/3.808	1663.0/1988.7	4.920/4.114
6,000	1872.6/2386.4	4.368/3.428	1755.3/2221.7	4.661/3.686
7,000	1953.8/2577.2	4.187/3.174	1836.0/2366.0	4.456/3.460
8,000	2051.2/2807.4	3.988/2.914	1926.7/2509.0	4.246/3.263
9,000	2135.4/3004.4	3.831/2.723	1981.7/2665.7	4.129/3.072
10,000	2209.8/3217.2	3.702/2.543	2060.0/2869.7	3.971/2.857

表-5.6：PF2 與 PF2_NMB 之搜尋視窗平均移動次數和平均移動的位元組個數

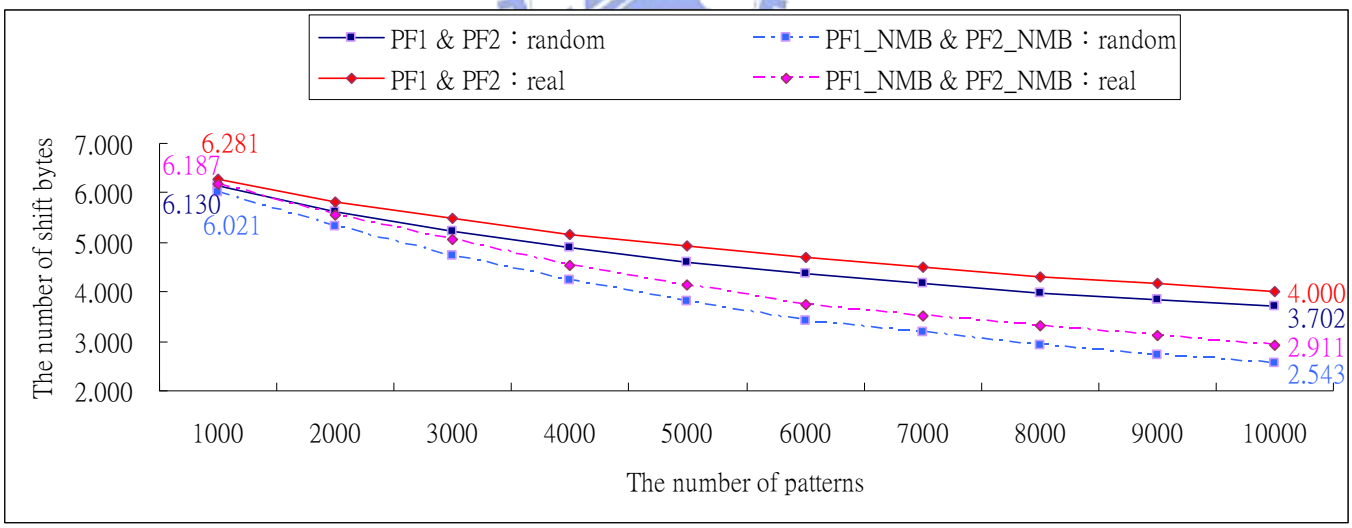

圖-5.9：PF2 與 PF2_NMB 的平均移動的位元組個數表現

我們所設計出的 PF1 版本與 PF2 版本可操作在時鐘頻率為 170 MHz，PF3 版本可操作在時鐘頻率為 84.5 MHz。圖-5.7 為關於 PF1 版本與 PF2 版本的生產量的表現。圖-5.8 為關於 PF1 版本與 PF2 版本所偵測的可疑字樣個數之總合比較，從圖中我們就可以很明顯地比較出優劣。。

第六章

結 論

我們可以看到原先依演算法所設計之預先過濾器可操作於 170 MHz 的頻率之上。在生產量方面，最高可近 2.9 Gbps(當字樣集合內僅有 1000 隻字樣)，最差也有近 1.7 Gbps(當字樣集合內僅有 10,000 隻字樣)，這個結果達到了我們預先設定之目標，這使得我們的每一個單位面積可提供的生產量最差亦有約 0.24 Gbps/BlockRAM；在偵測可疑字樣的個數方面，由於我們僅用一個雜湊函數去代表 MQM7 之結果，這使得誤報率居高不下，造成驗證器過於忙碌，使其整體系統生產量無法上來。因此，我們設計出了 PF2 版本，主要改善誤報率並盡量維持現有的生產量的表現。

PF2 版本可操作於 170 MHz 的頻率之上。在生產量方面，最高可近 2.9 Gbps(當字樣集合內僅有 1000 隻字樣)，最差也有近 1.7 Gbps(當字樣集合內僅有 10,000 隻字樣)，這個結果與 PF1 的生產量表現相同，但在偵測可疑字樣的個數方面，由於我們用了兩個雜湊函數去代表 MQM7 之結果，這使得誤報率大幅下降，最高的級差約 37 倍，這意味著所偵測可疑字樣的個數僅是 PF1 的 $\frac{1}{37}$ (在字樣集合內有 1000 隻字樣時，PF1 偵測可疑字樣的個數為 74，PF2 偵測可疑字樣的個數為 2)。

PF3 版本可操作於 84.5 MHz 的頻率之上。此版本為基於 PF2 的偵測可疑字樣的個數之表現下，改善其功率之消耗。若以 PF1 為基準，PF3 的功率表現上大幅降低了 75%，但付出了在生產量上的表現，約僅剩 51%。

參 考 文 獻

[1] D. Knuth, J. Morris and V. Pratt, “Fast pattern matching in strings,” TR CS-74-440, Stanford University, Stanford California, 1974

[2] R. S. Boyer and J. S. Moore. “A Fast String Searching Algorithm,” *Comm. of the ACM*, vol. 20, issue 10, pp.762-772, Oct. 1977.

[3] S. Wu and U. Manber, “A fast algorithm for multi-pattern searching," *Tech. Rep. TR94-17, Dept. Comput. Sci., Univ. Arizona*, May 1994.

[4] A. Aho and M. Corasick, “Efficient string matching: An aid to bibliographic search,” *Comm. of the ACM*, vol. 18, issue 6, pp.333-343, Jun. 1975.

[5] S. Dharmapurikar, P. Krishnamurthy, T. Sproull, and J. Lockwood, “Deep packet inspection using parallel bloom filters,” Symposium on High-Performance Interconnect (HotI), Stanford, CA, pp. 44-51, Aug. 2003.

[6] L. Tan and T. Sherwood, “A high throughput string matching architecture for intrusion detection and prevention,” *32nd Annual International Symposium on Computer Architecture*, pp. 112-122, 2005.

[7] R. Sidhu and V. K. Prasanna, “Fast regular expression matching using FPGAs,” IEEE Symposium on Field Programmable Custom Computing Machines (FCCM), Rohnert Park, CA, 2001.

[8] Clam anti virus signature database, www.clamav.net.

[9] T.H. Ptacek, T.N. Newsham, “Insertion, Evasion, and Denial of Service: Eluding Network Intrusion Detection”, Secure Networks Inc. Report, January 1998

[10] L. Tan and T. Sherwood, “Architectures for Bit-Split String Scanning in Intrusion Detection,” IEEE Micro, Vol.26, pp. 110-117, 2006

[11] N. Tuck, T. Sherwood, B. Calder, and G. Varghese, “Deterministic memory-efficient string matching algorithms for intrusion detection,” IEEE Infocom 2004, pp. 333-340.

[12] T. H. Lee and J. C. Liang, “A high-performance memory-efficient pattern matching algorithm and its implementation,” IEEE Tencon, Hong-Kong, 2006.

[13] L. Tan and T. Sherwood, “A high throughput string matching architecture for intrusion detection and prevention,” 32nd Annual International Symposium on Computer Architecture, pp. 112-122, 2005

[14] Y. Sugawara, M. Inaba and K. Hiraki, “Over 10Gbps string matching mechanism for multi-stream packet scanning systems,” Field Programmable Logic and Application, Vol. 3203, Sep. 2004, pp. 484-493.

[15] C. R. Clark, D. E. Schimmel. “Scalable Pattern Matching for High Speed Networks.” Proc. of 12th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'04), 2004

附　錄　一

各合成器合成(Synthesize)之結果

一、　使用 Xilinx ISE 8.1i 內建合成器(Synthesizer)，XST

1. PF1

Timing Summary:

Speed Grade: -6

Minimum period: 5.886ns (Maximum Frequency: 169.909MHz)

Minimum input arrival time before clock: 3.072ns

Maximum output required time after clock: 6.503ns

Maximum combinational path delay: No path found

Timing Detail:

All values displayed in nanoseconds (ns)

==========

Timing constraint: Default period analysis for Clock 'clk_p'

Clock period: 5.886ns (frequency: 169.909MHz)

Total number of paths / destination ports: 2026 / 181

Delay: 5.886ns (Levels of Logic = 16)

Source: hashgen/B_5 (FF)

Destination: input_contr/PF2TextAddr_10 (FF)

Source Clock: clk_p rising

Destination Clock: clk_p rising

Data Path: hashgen/B_5 to input_contr/PF2TextAddr_10

Cell:in->out	fanout	Gate Delay	Net Delay	Logical Name (Net Name)
FDPE:C->Q	4	0.374	0.536	hashgen/B_5 (hashgen/B_5)
LUT4:I3->O	1	0.313	0.506	input_contr/InContr_02_xo<0>18 (input_contr/InContr_02_xo<0>_map993)
LUT4:I1->O	1	0.313	0.418	input_contr/InContr_02_xo<0>28 (input_contr/InContr_02_xo<0>_map996)
LUT4_L:I2->LO	1	0.313	0.128	input_contr/InContr_02_xo<0>64 (input_contr/InContr_02_xo<0>_map1001)
LUT4:I2->O	1	0.313	0.506	input_contr/InContr_02_xo<0>109 (input_contr/addr_text<2>)
LUT2_D:I1->LO	2	0.313	0.000	input_contr/InContr__n0002<0>lut (N1460)
MUXCY:S->O	1	0.377	0.000	input_contr/InContr__n0002<0>cy (input_contr/InContr__n0002<0>_cyo)
MUXCY:CI->O	1	0.041	0.000	input_contr/InContr__n0002<1>cy (input_contr/InContr__n0002<1>_cyo)
MUXCY:CI->O	1	0.041	0.000	input_contr/InContr__n0002<2>cy (input_contr/InContr__n0002<2>_cyo)
MUXCY:CI->O	1	0.041	0.000	input_contr/InContr__n0002<3>cy (input_contr/InContr__n0002<3>_cyo)
MUXCY:CI->O	1	0.041	0.000	input_contr/InContr__n0002<4>cy (input_contr/InContr__n0002<4>_cyo)
MUXCY:CI->O	1	0.041	0.000	input_contr/InContr__n0002<5>cy (input_contr/InContr__n0002<5>_cyo)
MUXCY:CI->O	1	0.041	0.000	input_contr/InContr__n0002<6>cy (input_contr/InContr__n0002<6>_cyo)
MUXCY:CI->O	1	0.041	0.000	input_contr/InContr__n0002<7>cy (input_contr/InContr__n0002<7>_cyo)
MUXCY:CI->O	1	0.041	0.000	input_contr/InContr__n0002<8>cy (input_contr/InContr__n0002<8>_cyo)
MUXCY:CI->O	0	0.041	0.000	input_contr/InContr__n0002<9>cy (input_contr/InContr__n0002<9>_cyo)
XORCY:CI->O	1	0.868	0.000	input_contr/InContr__n0002<10>_xor (input_contr/_n0002<10>)
FDCE:D		0.234		input_contr/PF2TextAddr_10
Total		5.886ns (3.792ns logic, 2.094ns route)		
		(64.4% logic, 35.6% route)		

2. PF1_NMB

Timing Summary:

Speed Grade: -6

Minimum period: 4.968ns (Maximum Frequency: 201.303MHz)

Minimum input arrival time before clock: 3.063ns

Maximum output required time after clock: 6.049ns

Maximum combinational path delay: No path found

Timing Detail:

All values displayed in nanoseconds (ns)

==========

Timing constraint: Default period analysis for Clock 'clk_p'

Clock period: 4.968ns (frequency: 201.303MHz)

Total number of paths / destination ports: 1713 / 171

Delay: 4.968ns (Levels of Logic = 14)

Source: input_contr/PF2TextAddr_12 (FF)

Destination: input_contr/PF2TextAddr_10 (FF)

Source Clock: clk_p rising

Destination Clock: clk_p rising

Data Path: input_contr/PF2TextAddr_12 to input_contr/PF2TextAddr_10

Cell:in->out	fanout	Gate Delay	Net Delay	Logical Name (Net Name)
FDPE:C->Q	52	0.374	0.851	input_contr/PF2TextAddr_12 (input_contr/PF2TextAddr_12)
LUT3:I2->O	1	0.313	0.533	input_contr/InContr_01_xo<0>5 (input_contr/InContr_01_xo<0>_map933)
LUT4:I0->O	1	0.313	0.418	input_contr/InContr_01_xo<0>95 (input_contr/InContr_01_xo<0>_map951)
LUT3_D:I2->LO	2	0.313	0.000	input_contr/InContr__n0002<0>lut (N1453)
MUXCY:S->O	1	0.377	0.000	input_contr/InContr__n0002<0>cy (input_contr/InContr__n0002<0>_cyo)
MUXCY:CI->O	1	0.041	0.000	input_contr/InContr__n0002<1>cy (input_contr/InContr__n0002<1>_cyo)
MUXCY:CI->O	1	0.041	0.000	input_contr/InContr__n0002<2>cy (input_contr/InContr__n0002<2>_cyo)
MUXCY:CI->O	1	0.041	0.000	input_contr/InContr__n0002<3>cy (input_contr/InContr__n0002<3>_cyo)
MUXCY:CI->O	1	0.041	0.000	input_contr/InContr__n0002<4>cy (input_contr/InContr__n0002<4>_cyo)
MUXCY:CI->O	1	0.041	0.000	input_contr/InContr__n0002<5>cy (input_contr/InContr__n0002<5>_cyo)
MUXCY:CI->O	1	0.041	0.000	input_contr/InContr__n0002<6>cy (input_contr/InContr__n0002<6>_cyo)
MUXCY:CI->O	1	0.041	0.000	input_contr/InContr__n0002<7>cy (input_contr/InContr__n0002<7>_cyo)
MUXCY:CI->O	1	0.041	0.000	input_contr/InContr__n0002<8>cy (input_contr/InContr__n0002<8>_cyo)
MUXCY:CI->O	0	0.041	0.000	input_contr/InContr__n0002<9>cy (input_contr/InContr__n0002<9>_cyo)
XORCY:CI->O	1	0.868	0.000	input_contr/InContr__n0002<10>_xor (input_contr/_n0002<10>)
FDCE:D		0.234		input_contr/PF2TextAddr_10

Total 4.968ns (3.166ns logic, 1.802ns route)

(63.7% logic, 36.3% route)

3. PF2

Timing Summary:

Speed Grade: -6

Minimum period: 5.886ns (Maximum Frequency: 169.909MHz)

Minimum input arrival time before clock: 3.072ns

Maximum output required time after clock: 6.503ns

Maximum combinational path delay: No path found

Timing Detail:

All values displayed in nanoseconds (ns)

==========

Timing constraint: Default period analysis for Clock 'clk_p'

Clock period: 5.886ns (frequency: 169.909MHz)

Total number of paths / destination ports: 1949 / 181

Delay: 5.886ns (Levels of Logic = 16)

Source: hashgen/B_5 (FF)

Destination: input_contr/PF2TextAddr_10 (FF)

Source Clock: clk_p rising

Destination Clock: clk_p rising

Data Path: hashgen/B_5 to input_contr/PF2TextAddr_10

Cell:in->out	fanout	Gate Delay	Net Delay	Logical Name (Net Name)
FDPE:C->Q	4	0.374	0.536	hashgen/B_5 (hashgen/B_5)
LUT4:I3->O	1	0.313	0.506	input_contr/InContr_02_xo<0>18 (input_contr/InContr_02_xo<0>_map993)
LUT4:I1->O	1	0.313	0.418	input_contr/InContr_02_xo<0>28 (input_contr/InContr_02_xo<0>_map996)
LUT4_L:I2->LO	1	0.313	0.128	input_contr/InContr_02_xo<0>64 (input_contr/InContr_02_xo<0>_map1001)
LUT4:I2->O	1	0.313	0.506	input_contr/InContr_02_xo<0>109 (input_contr/addr_text<2>)
LUT2_D:I1->LO	2	0.313	0.000	input_contr/InContr__n0002<0>lut (N1460)
MUXCY:S->O	1	0.377	0.000	input_contr/InContr__n0002<0>cy (input_contr/InContr__n0002<0>_cyo)
MUXCY:CI->O	1	0.041	0.000	input_contr/InContr__n0002<1>cy (input_contr/InContr__n0002<1>_cyo)
MUXCY:CI->O	1	0.041	0.000	input_contr/InContr__n0002<2>cy (input_contr/InContr__n0002<2>_cyo)

MUXCY:CI->O	1	0.041	0.000	input_contr/InContr__n0002<3>cy (input_contr/InContr__n0002<3>_cyo)
MUXCY:CI->O	1	0.041	0.000	input_contr/InContr__n0002<4>cy (input_contr/InContr__n0002<4>_cyo)
MUXCY:CI->O	1	0.041	0.000	input_contr/InContr__n0002<5>cy (input_contr/InContr__n0002<5>_cyo)
MUXCY:CI->O	1	0.041	0.000	input_contr/InContr__n0002<6>cy (input_contr/InContr__n0002<6>_cyo)
MUXCY:CI->O	1	0.041	0.000	input_contr/InContr__n0002<7>cy (input_contr/InContr__n0002<7>_cyo)
MUXCY:CI->O	1	0.041	0.000	input_contr/InContr__n0002<8>cy (input_contr/InContr__n0002<8>_cyo)
MUXCY:CI->O	0	0.041	0.000	input_contr/InContr__n0002<9>cy (input_contr/InContr__n0002<9>_cyo)
XORCY:CI->O	1	0.868	0.000	input_contr/InContr__n0002<10>_xor (input_contr/_n0002<10>)
FDCE:D		0.234		input_contr/PF2TextAddr_10

Total 5.886ns (3.792ns logic, 2.094ns route)

(64.4% logic, 35.6% route)

4. PF2_NMB

Timing Summary:

Speed Grade: -6

Minimum period: 4.968ns (Maximum Frequency: 201.303MHz)

Minimum input arrival time before clock: 3.063ns

Maximum output required time after clock: 6.049ns

Maximum combinational path delay: No path found

Timing Detail:

All values displayed in nanoseconds (ns)

==========

Timing constraint: Default period analysis for Clock 'clk_p'

Clock period: 4.968ns (frequency: 201.303MHz)

Total number of paths / destination ports: 1712 / 171

Delay: 4.968ns (Levels of Logic = 14)

Source: input_contr/PF2TextAddr_12 (FF)

Destination: input_contr/PF2TextAddr_10 (FF)

Source Clock: clk_p rising

Destination Clock: clk_p rising

Data Path: input_contr/PF2TextAddr_12 to input_contr/PF2TextAddr_10

Cell:in->out	fanout	Gate Delay	Net Delay	Logical Name (Net Name)
FDPE:C->Q	52	0.374	0.851	input_contr/PF2TextAddr_12 (input_contr/PF2TextAddr_12)
LUT3:I2->O	1	0.313	0.533	input_contr/InContr_02_xo<0>5 (input_contr/InContr_02_xo<0>_map855)
LUT4:I0->O	1	0.313	0.418	input_contr/InContr_02_xo<0>95 (input_contr/InContr_02_xo<0>_map873)
LUT3_D:I2->LO	2	0.313	0.000	input_contr/InContr__n0002<0>lut (N868)
MUXCY:S->O	1	0.377	0.000	input_contr/InContr__n0002<0>cy (input_contr/InContr__n0002<0>_cyo)
MUXCY:CI->O	1	0.041	0.000	input_contr/InContr__n0002<1>cy (input_contr/InContr__n0002<1>_cyo)
MUXCY:CI->O	1	0.041	0.000	input_contr/InContr__n0002<2>cy (input_contr/InContr__n0002<2>_cyo)
MUXCY:CI->O	1	0.041	0.000	input_contr/InContr__n0002<3>cy (input_contr/InContr__n0002<3>_cyo)
MUXCY:CI->O	1	0.041	0.000	input_contr/InContr__n0002<4>cy (input_contr/InContr__n0002<4>_cyo)
MUXCY:CI->O	1	0.041	0.000	input_contr/InContr__n0002<5>cy (input_contr/InContr__n0002<5>_cyo)
MUXCY:CI->O	1	0.041	0.000	input_contr/InContr__n0002<6>cy (input_contr/InContr__n0002<6>_cyo)
MUXCY:CI->O	1	0.041	0.000	input_contr/InContr__n0002<7>cy (input_contr/InContr__n0002<7>_cyo)
MUXCY:CI->O	1	0.041	0.000	input_contr/InContr__n0002<8>cy (input_contr/InContr__n0002<8>_cyo)
MUXCY:CI->O	0	0.041	0.000	input_contr/InContr__n0002<9>cy (input_contr/InContr__n0002<9>_cyo)
XORCY:CI->O	1	0.868	0.000	input_contr/InContr__n0002<10>_xor (input_contr/_n0002<10>)
FDCE:D		0.234		input_contr/PF2TextAddr_10
Total		4.968ns (3.166ns logic, 1.802ns route) (63.7% logic, 36.3% route)		

5. PF3

Timing Summary:

Speed Grade: -6

Minimum period: 11.583ns (Maximum Frequency: 86.337MHz)

Minimum input arrival time before clock: 3.090ns

Maximum output required time after clock: 7.288ns

Maximum combinational path delay: No path found

Timing Detail:

All values displayed in nanoseconds (ns)

==========

Timing constraint: Default period analysis for Clock 'clk_p'

Clock period: 11.583ns (frequency: 86.337MHz)

Total number of paths / destination ports: 180092 / 324

Delay: 11.583ns (Levels of Logic = 22)

Source: textram/textram2 (RAM)

Destination: input_contr/PF2TextAddr_10 (FF)

Source Clock: clk_p rising

Destination Clock: clk_p rising

Data Path: textram/textram2 to input_contr/PF2TextAddr_10

Cell:in->out	fanout	Gate Delay	Net Delay	Logical Name (Net Name)
RAMB16_S9_S9:CLKA->DOA4	6	1.500	0.640	textram/textram2 (textram/textram2data_A<4>)
LUT3_L:I1->LO	1	0.313	0.000	textram/Text2PFData<12>1111_F (N1581)
MUXF5:I0->O	6	0.340	0.667	textram/Text2PFData<12>1111 (text2pfdata<12>)
LUT4:I0->O	3	0.313	0.495	hashgen/HashGen_05_xo<2>1 (hashgen/xo_hash1/_n0018)
LUT4:I2->O	15	0.313	0.761	hashgen/HashGen_025_xo<5>1_1 (hashgen/HashGen_025_xo<5>1)
LUT3:I2->O	1	0.313	0.440	hashgen/MQMOut<4>32_SW0 (N1360)
LUT4_L:I3->LO	1	0.313	0.000	hashgen/MQMOut<4>1261_F (N1625)
MUXF5:I0->O	8	0.340	0.678	hashgen/MQMOut<4>1261 (mqmout<4>)
LUT4:I1->O	1	0.313	0.418	input_contr/InContr_addr_text<2>_xor126_SW0 (N1382)
LUT4_L:I2->LO	1	0.313	0.128	input_contr/InContr_addr_text<2>_xor164 (input_contr/InContr_addr_text<2>_xor1_map563)
LUT4:I2->O	1	0.313	0.506	input_contr/InContr_addr_text<2>_xor1109 (input_contr/addr_text<2>)
LUT2_L:I1->LO	2	0.313	0.000	input_contr/InContr__n0003<0>lut (input_contr/N6)
MUXCY:S->O	1	0.377	0.000	input_contr/InContr__n0003<0>cy (input_contr/InContr__n0003<0>_cyo)
MUXCY:CI->O	1	0.041	0.000	input_contr/InContr__n0003<1>cy (input_contr/InContr__n0003<1>_cyo)
MUXCY:CI->O	1	0.041	0.000	input_contr/InContr__n0003<2>cy (input_contr/InContr__n0003<2>_cyo)
MUXCY:CI->O	1	0.041	0.000	input_contr/InContr__n0003<3>cy (input_contr/InContr__n0003<3>_cyo)
MUXCY:CI->O	1	0.041	0.000	input_contr/InContr__n0003<4>cy (input_contr/InContr__n0003<4>_cyo)
MUXCY:CI->O	1	0.041	0.000	input_contr/InContr__n0003<5>cy (input_contr/InContr__n0003<5>_cyo)
MUXCY:CI->O	1	0.041	0.000	input_contr/InContr__n0003<6>cy (input_contr/InContr__n0003<6>_cyo)
MUXCY:CI->O	1	0.041	0.000	input_contr/InContr__n0003<7>cy (input_contr/InContr__n0003<7>_cyo)
MUXCY:CI->O	1	0.041	0.000	input_contr/InContr__n0003<8>cy (input_contr/InContr__n0003<8>_cyo)

MUXCY:CI->O 0 0.041 0.000 input_contr/InContr__n0003<9>cy (input_contr/InContr__n0003<9>_cyo)

XORCY:CI->O 1 0.868 0.000 input_contr/InContr__n0003<10>_xor (input_contr/_n0003<10>)

FDCE:D 0.234 input_contr/PF2TextAddr_10

Total 11.583ns (6.850ns logic, 4.733ns route)

(59.1% logic, 40.9% route)

6. PF3_NMB

Timing Summary:

Speed Grade: -6

Minimum period: 11.530ns (Maximum Frequency: 86.734MHz)

Minimum input arrival time before clock: 3.087ns

Maximum output required time after clock: 7.288ns

Maximum combinational path delay: No path found

Timing Detail:

All values displayed in nanoseconds (ns)

==========

Timing constraint: Default period analysis for Clock 'clk_p'

Clock period: 11.530ns (frequency: 86.734MHz)

Total number of paths / destination ports: 162365 / 317

Delay: 11.530ns (Levels of Logic = 22)

Source: textram/textram0 (RAM)

Destination: input_contr/PF2TextAddr_10 (FF)

Source Clock: clk_p rising

Destination Clock: clk_p rising

Data Path: textram/textram0 to input_contr/PF2TextAddr_10

Cell:in->out	fanout	Gate Delay	Net Delay	Logical Name (Net Name)
RAMB16_S9_S9:CLKA->DOA1	4	1.500	0.602	textram/textram0 (textram/textram0data_A<1>)

LUT3_L:I1->LO 1 0.313 0.000 textram/Text2PFData<25>1111_F (N1568)

MUXF5:I0->O 8 0.340 0.705 textram/Text2PFData<25>1111 (text2pfdata<25>)

LUT4:I0->O 8 0.313 0.612 hashgen/HashGen_024_xo<1>1_SW0 (N1337)

LUT4:I3->O 4 0.313 0.514 hashgen/HashGen_024_xo<1>1_1 (hashgen/HashGen_024_xo<1>1)

LUT4:I2->O 1 0.313 0.440 hashgen/mqm_A<6>27 (hashgen/mqm_A<6>_map2167)

LUT4:I3->O 3 0.313 0.517 hashgen/mqm_A<6>112 (mqmout<6>)

LUT4_D:I3->LO 1 0.313 0.243 input_contr/InContr_addr_text<2>_xor128_SW0 (N1619)

LUT2:I0->O 1 0.313 0.440 input_contr/InContr_addr_text<2>_xor164_SW0 (N1374)

LUT4_L:I3->LO 1 0.313 0.128 input_contr/InContr_addr_text<2>_xor164 (input_contr/InContr_addr_text<2>_xor1_map2203)

LUT4:I2->O 1 0.313 0.506 input_contr/InContr_addr_text<2>_xor1108 (input_contr/addr_text<2>)

LUT2_L:I1->LO 2 0.313 0.000 input_contr/InContr__n0003<0>lut (input_contr/N6)

MUXCY:S->O 1 0.377 0.000 input_contr/InContr__n0003<0>cy (input_contr/InContr__n0003<0>_cyo)

MUXCY:CI->O 1 0.041 0.000 input_contr/InContr__n0003<1>cy (input_contr/InContr__n0003<1>_cyo)

MUXCY:CI->O 1 0.041 0.000 input_contr/InContr__n0003<2>cy (input_contr/InContr__n0003<2>_cyo)

MUXCY:CI->O 1 0.041 0.000 input_contr/InContr__n0003<3>cy (input_contr/InContr__n0003<3>_cyo)

MUXCY:CI->O 1 0.041 0.000 input_contr/InContr__n0003<4>cy (input_contr/InContr__n0003<4>_cyo)

MUXCY:CI->O 1 0.041 0.000 input_contr/InContr__n0003<5>cy (input_contr/InContr__n0003<5>_cyo)

MUXCY:CI->O 1 0.041 0.000 input_contr/InContr__n0003<6>cy (input_contr/InContr__n0003<6>_cyo)

MUXCY:CI->O 1 0.041 0.000 input_contr/InContr__n0003<7>cy (input_contr/InContr__n0003<7>_cyo)

MUXCY:CI->O 1 0.041 0.000 input_contr/InContr__n0003<8>cy (input_contr/InContr__n0003<8>_cyo)

MUXCY:CI->O 0 0.041 0.000 input_contr/InContr__n0003<9>cy (input_contr/InContr__n0003<9>_cyo)

XORCY:CI->O 1 0.868 0.000 input_contr/InContr__n0003<10>_xor (input_contr/_n0003<10>)

FDCE:D 0.234 input_contr/PF2TextAddr_10

Total 11.530ns (6.823ns logic, 4.707ns route)

(59.2% logic, 40.8% route)

二、 使用 Synplicity 公司的合成器，Synplify Pro 8.6.2

1. PF1

Performance Summary

****

Worst slack in design: 0.108

Starting Clock	Requested Frequency	Estimated Frequency	Requested Period	Estimated Period	Slack	Clock Type	Clock Group
System | clk_p	162.0 MHz	164.9 MHz	6.173	6.065	0.108	inferred	Inferred_clkgroup_0

==========

Clock Relationships

****

Clocks		rise to rise		fall to fall		rise to fall		fall to rise	
Starting	Ending	constraint	slack	constraint	slack	constraint	slack	constraint	slack
System|clk_p	System|clk_p	6.173	0.108	No paths	-	No paths	-	No paths	-

Note: 'No paths' indicates there are no paths in the design for that pair of clock edges.
'Diff grp' indicates that paths exist but the starting clock and ending clock are in different clock groups.

2. PF1_NMB

Performance Summary

****

Worst slack in design: -0.052

Starting Clock	Requested Frequency	Estimated Frequency	Requested Period	Estimated Period	Slack	Clock Type	Clock Group
System | clk_p	172.0 MHz	170.5 MHz	5.814	5.866	-0.052	inferred	Inferred_clkgroup_0

==========

Clock Relationships

****

Clocks		rise to rise		fall to fall		rise to fall		fall to rise	
Starting	Ending	constraint	slack	constraint	slack	constraint	slack	constraint	slack
System|clk_p	System|clk_p	5.814	-0.052	No paths	-	No paths	-	No paths	-

Note: 'No paths' indicates there are no paths in the design for that pair of clock edges.
'Diff grp' indicates that paths exist but the starting clock and ending clock are in different clock groups.

3. PF2

Performance Summary

****

Worst slack in design: 0.034

Starting Clock	Requested Frequency	Estimated Frequency	Requested Period	Estimated Period	Slack	Clock Type	Clock Group
System | clk_p	161.0 MHz	161.9 MHz	6.211	6.177	0.034	inferred	Inferred_clkgroup_0

Clock Relationships

****

Clocks		rise to rise		fall to fall		rise to fall		fall to rise	
Starting	Ending	constraint	slack	constraint	slack	constraint	slack	constraint	slack
System|clk_p	System|clk_p	6.211	0.034	No paths	-	No paths	-	No paths	-

Note: 'No paths' indicates there are no paths in the design for that pair of clock edges.
'Diff grp' indicates that paths exist but the starting clock and ending clock are in different clock groups.

4. PF2_NMB

Performance Summary

****

Worst slack in design: 0.084

Starting Clock	Requested Frequency	Estimated Frequency	Requested Period	Estimated Period	Slack	Clock Type	Clock Group
System | clk_p	161.0 MHz	163.2 MHz	6.211	6.128	0.084	inferred	Inferred_clkgroup_0

Clock Relationships

****

Clocks		rise to rise		fall to fall		rise to fall		fall to rise	
Starting	Ending	constraint	slack	constraint	slack	constraint	slack	constraint	slack
System|clk_p	System|clk_p	6.211	0.084	No paths	-	No paths	-	No paths	-

Note: 'No paths' indicates there are no paths in the design for that pair of clock edges.
'Diff grp' indicates that paths exist but the starting clock and ending clock are in different clock groups.

5. PF3

Performance Summary

****

Worst slack in design: -1.553

Starting Clock	Requested Frequency	Estimated Frequency	Requested Period	Estimated Period	Slack	Clock Type	Clock Group
System | clk_p	100.0 MHz	86.6 MHz	10.000	11.553	-1.553	inferred	Inferred_clkgroup_0

Clock Relationships

****

Clocks		rise to rise		fall to fall		rise to fall		fall to rise	
Starting	Ending	constraint	slack	constraint	slack	constraint	slack	constraint	slack
System|clk_p	System|clk_p	10.000	-1.553	No paths	-	No paths	-	No paths	-

Note: 'No paths' indicates there are no paths in the design for that pair of clock edges.
'Diff grp' indicates that paths exist but the starting clock and ending clock are in different clock groups.

6. PF3_NMB

Performance Summary

****

Worst slack in design: -1.190

Starting Clock	Requested Frequency	Estimated Frequency	Requested Period	Estimated Period	Slack	Clock Type	Clock Group
System | clk_p	100.0 MHz	89.4 MHz	10.000	11.190	-1.190	inferred	Inferred_clkgroup_0

Clock Relationships

****

Clocks		rise to rise		fall to fall		rise to fall		fall to rise	
Starting	Ending	constraint	slack	constraint	slack	constraint	slack	constraint	slack
System|clk_p	System|clk_p	10.000	-1.190	No paths	-	No paths	-	No paths	-

Note: 'No paths' indicates there are no paths in the design for that pair of clock edges.
'Diff grp' indicates that paths exist but the starting clock and ending clock are in different clock groups.