國立交通大學

電機學院　電信學程

碩 士 論 文

使用 FPGA 實現推測預先過濾器

Implementation of Speculation Pre-Filter

研 究 生：朱珮儀

指導教授：李程輝　教授

中 華 民 國 一百零一 年 一 月

使用 FPGA 實現推測預先過濾器

Implementation of Speculation Pre-Filter

研 究 生：朱珮儀　　Student：Pei-Yi Ju

指導教授：李程輝　　Advisor：Tsern-Huei Lee

國 立 交 通 大 學

電機學院　電信學程

碩 士 論 文

A Thesis

Submitted to 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

January 2012

Hsinchu, Taiwan, Republic of China

中華民國一百零一年一月

使用 FPGA 實現推測預先過濾器

學生：朱珮儀　　指導教授：李程輝

國 立 交 通 大 學

電機學院　電信學程 碩士班

摘要

近年來網際網路在網路入侵偵測、防毒方面或是網路負載偵測對速度及準確度上的需求越來越繁複，而在此方面經常使用的字串比對(pattern matching)也同時在處理速度上要求越來越快。在此 NTL 實驗室在字串比對理論中研發出一種以預先過濾器的概念增加字串比對的速度。預先過濾器的方法是利用預先濾過需求的關鍵字字串先去除不必要的資訊，讓後段硬體只需處理可能有用的資訊，如此一來使處理器的負擔減輕，相形之下處理的資訊越少處理的速度相對性就會變快。所以在此使用推測預先過濾器做為模型以硬體實現方式證明其實用上的可行性及優越的處理速度。除此之外以此推測預先過濾器做為基礎，改良硬體處理程序並且更進一步使用管線技術，利用同時處理多筆資訊的優點產生更快的資訊處理速度。在文中並同時實現非推測的預先過濾器並以此做為速度的比較基礎點來應證推測預先過濾器的實用性及硬體處理速度上的優越性。

Implementation of Speculation Pre-filter

Student：Pei-Yi Ju　　Advisor：Prof. Tsern-Huei Lee

Degree Program of Electrical and Computer Engineering
National Chiao Tung University

ABSTRACT

Pattern matching, finding patterns to gain wealthy information, is one of the most applied techniques in information retrieval and malware detection. Based on the purpose of improving pattern matching outcome, we review several most popular theories of pattern matching process in current technology, including pre-filter, a quickly excluder to eliminate the nullity part of patterns, which is a common technique to lead to efficient pattern matching result .

The Speculation pre-filter developed by NTL Lab is an efficient pre-filter that utilizes all previous query results by software programming. In the research, it is provided three different pre-filters implemented by Xilinx FPGA, containing Stateful pre-filter, Speculation pre-filter and Pipeline speculation pre-filter. Dual-port ram and pipeline are expected to accelerate pattern matching speed by multi-tasking function and enhance the performance of the hardware utility. Experimental result shows that the throughput in Speculation pre-filter is 1.5 times greater than Stateful pre-filter. Moreover, we discover that the throughput in pipe-line speculation pre-filter is 4 times faster than which in Speculation pre-filter, however, LUT in pipe-line speculation pre-filter only increases 2 times than which in Speculation pre-filter and no extra RAM spaces in pipe-line speculation pre-filter added. In the conclusion, pipe-line with pre-filters promises higher efficiency and performance in pattern matching.

致謝

研究所這幾年首先感謝我的指導教授李 程輝教授，於就讀期間的充分的指導與包容，李教授在專業知識與見解以及嚴謹的學術研究態度與平易近人的做人處事使我收穫良多，並於論文的方向給予關鍵性的指引與幫助，也感謝學校提供完善的學習研究環境，使得在學習過程無後顧之憂，在此謹誌以表達最深的謝忱。

也感謝 NTL 實驗室的各位同學、學長姐在期間對於這方面知識的切磋指教，由其感謝迺倫學姊在此方面的專長，並在百忙之中撥空指點，使得在此次實現設計中提供許多寶貴的意見。

最後感謝我的父母、丈夫及家人，在就讀期間給予的建議與鼓勵，可以在繁忙工作中保持對學問的熱忱。一路走來要感謝的人太多，難以一一答謝，但有他們的支持才賦予自我完成理想的信心與力量，僅以本文獻給所有我愛及愛我的家人朋友們。

珮儀

Jan. 2012

目　　錄

頁次

表目錄

圖目錄

第一章

簡介

1.1 研究背景

目前網際網路的傳輸速度以及應用發展持續成長，為了保持網路運作效率以及安全性的挑戰下，字串比對(Pattern matching)的技術在網路處理下，變的相當重要。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關鍵的角色。

在字串比對演算法中下列三項理論經常被參考，**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]*。當探討字串比對演算法時，將它們區分成，

以複雜度的觀點： 可分為『線性(linear)』/『子線性(sub-linear)』字串比對演算法。

AC 字串比對演算法是一種典型且常見的線性時間演算法，它將字串(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)，因此在最壞情況(worst case)的表現。BM 字串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，每次移動時，都跳躍(移動)一段長距離的位置，使得就時間複雜度而言優過於線性時間複雜度，稱此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 為字串長度；但在最壞情況下的表現就差強人意，時間複雜度為O(n*m)。

以可比對字串多寡的觀點： 可分為『單一(single)』/『多(multiple)』字串比對演算法。

BM 演算法為單一字串比對演算法，意指當輸入文字字串時，僅單一字串與其進行比對工作。AC 則為多字串比對演算法，意指當輸入文文字字串時，由多字串所建構成的有限狀態機會與其進行比對工作，因此，可能會在輸入文字字串掃描完畢後，同時偵測到多個字串。如上所述，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串比對中的搜尋引擎。

實現於軟體的字串比對因網際網路的快速成長，無法跟上日益增加的網路速度，使其無法有效即時地防堵網路上的惡意攻擊。所以以硬體化的方式實現字串比對的工作以解決速度上的問題。

1.2 研究動機

網路入侵檢測系統(Network Intrusion Detection System, NIDS)是目前普遍用作網路預防病毒以及網路保護的系統之一。這個系統有幾個常見的問題第一當網路的資料單位較大的時候其處理效率會隨著變低，意即系統會失掉部分來不及處理的攻擊行為偵測，而產生損失。第二在交換網路的時候若要做有效的監控則需處理大量網路資料。第三點當處於高速網路時若失掉部分封包則此系統會忽略掉有可能的網路攻擊行為。

且入侵偵測系統對於可疑字串的操作模式是屬於fail open，意指當入侵偵測系統失敗時，網路仍是會是流通的，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，防火牆對於可疑字串的操作模式是屬於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。

在上述第一個和第二個問題，在偵測上需快速且大量且正確率不下降的條件下處理資料。所以本文上述條件為目標。又基於軟體的入侵偵測系統在處理速度上遇到了瓶頸，所以在NTL 實驗室，李程輝 教授和黃 迺倫 學姐開發出一個 新的可延展之字串比對架構，此字串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，可延展之字串比對架構是現今的字串比對領域中的佼佼者。在有著優秀的字串比對演算法下，進一步地實現於硬體之上，並實際實現於FPGA。

第二章

相關工作

2.1 Aho-Corasick 演算法

Aho-Corasick為Alfred V. Aho 和Margaret J. Corasick 提出利用有限狀態機的方式有效的且同時比對複數字串的方法，目前在字串比對模型的理論中佔有很重要的地位，也由於相較其他理論可同時處理多筆字串而廣泛的應用於文字比對、網路入侵偵測、病毒偵測等，簡稱AC演算法。

關鍵字字串比對模型(pattern matching machine)為在輸入的文字字串集合中找出關鍵字字串並當作其輸出。AC 演算法由三個函數所組成分別為轉移函數(goto function)、輸出函數(output function)和失效函數(failure function)。假設關鍵字字串的集合為{he，she，his，hers}，且轉移函數如圖2.1 所示。定義轉移函數的初始狀態為”0”，在圖中顯示狀態有 10 個，從 0、1、…、9，轉移函數 g 依據當前狀態與輸入字元將輸入字元映射到轉移函數的某一狀態中或是映射到失效函數，例如當前狀態為”0”，輸入字元為 h 則轉移函數以 g(0,h)表示其輸出為”1”即下個狀態為”1”狀態，若當前狀態為”1”，輸入字元為 y，因 y 不屬於轉移狀態中任何一字元，所以 g(1,y)=fail。

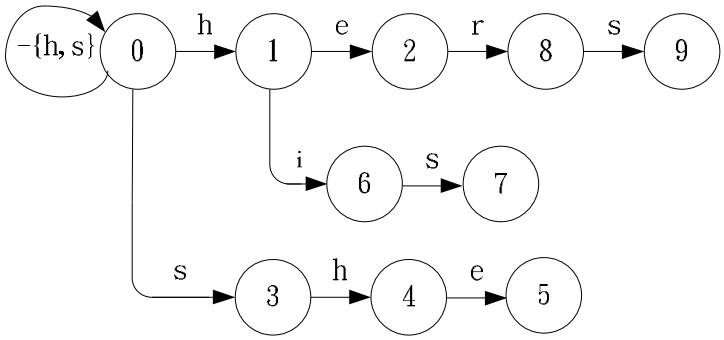


圖 2.1 轉移函數

建構轉移函數須先產生轉移圖表(goto graph)，以下建構使用關鍵字字串集合{he，she，his，hers}，在建立轉移圖表之初先定義初始狀態”0”，之後依序排入關鍵字字串字元，排入一關鍵字字串字元後就依序增加一狀態，先建構關鍵字字串集合中第一個關鍵字字串{he}，如圖 2.2 所示，圈圈內數字代表狀態，狀態與狀態之間的訊號稱為路徑，路徑上的字元為產生狀態改變的字元，若從狀態”0”走到狀態”2”可以拼出第一個關鍵字字串{he}，所以定義輸出 he 與狀態”2”做聯結。接下來建構第二個關鍵字字串{she}。由於”0”狀態到”2”狀態之間路徑不存在第二個關鍵字字串的第一個字元’s’，所以從初始狀態”0”建立新的路徑與狀態並依序排入第二關關鍵字字串字元，並定義輸出 she 與狀態”5”作聯結，建立第二關鍵字字串後的轉移圖表如圖 2.3。

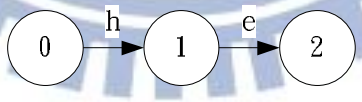


圖 2.2 轉移圖表一

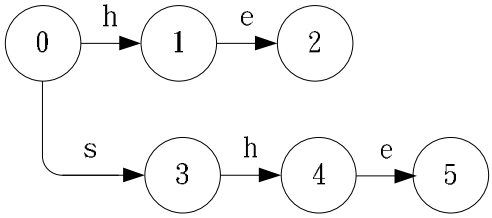


圖 2.3 轉移圖表二

接下來建立第三關鍵字字串{his}，由於’h’與第一關鍵字字串第一字元相同，所以已經建立在狀態”0”與狀態”1”的路徑上，因此由狀態”1”開始建構第三關鍵字字串。狀態”1”至狀態”2”的路徑上為’e’與第三關鍵字字串的第二字元不相同，所以在狀態”1”上建立分支並產生狀態”6”，狀態”1”與狀態”6”的路徑字元為’i’。順序排入關鍵字字串字元後得到輸出 his 與狀態”7”做聯結。上述結果表現於圖 2.4。

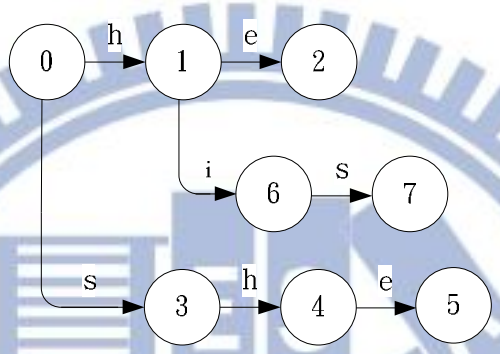


圖 2.4 轉移圖表三

依據上述規則建立第四關鍵字字串{hers}，第四關鍵字字串前兩個字元已建立在轉移圖表之中所以從狀態”2”開始建立新路徑及狀態，最後將輸出 hers 與狀態”9”做連結。其轉移圖表如 2.5 所示。因關鍵字字串集合的開頭字元集為’h’與’s’，若字元非上兩者，不會產生狀態上變化，所以在狀態”0”上加上一迴路路徑，即非’h’與非’s’字元只會在狀態”0”上產生自迴圈。圖 2.1 為最後完成的轉移函數。

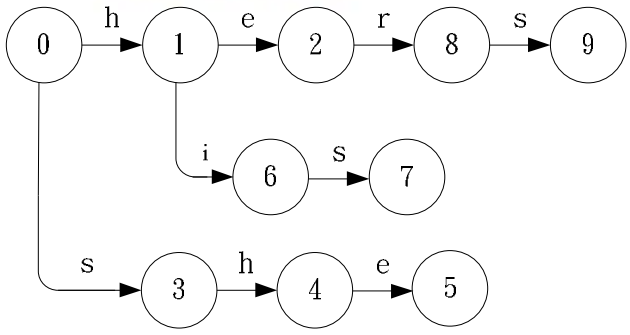


圖 2.5 轉移圖表四

失效函數 f 的產生是由轉移函數發出失效信號後將狀態映射到另一狀態的函數。假設一字串 u 可在轉移函數中找到一個最短的路徑使得從狀態”0”到狀態”S”可拼出字串 u，則稱 u 以狀態”S”表示，又字串 u 與 v 分別以狀態”S”與狀態”Q”表示，若 v 是 u 中最長的子集則 f(S’)=Q，如圖 2.1 所示範例則可推得 f(5)=2，表示狀態”5”的 she 包含了狀態”2”的 he。

i	1	2	3	4	5	6	7	8	9
f(i)	0	0	0	1	2	0	3	0	3

圖 2.6 失效函數

輸出函數 Output 則是將狀態映射到字串集合，其集合可為空集合。輸出函數包含的字串為其狀態可表示的字串，如狀態”5”可表示 she 與 he 所以其輸出函數為 Output(5)={she，he}。

S	*Output(S)*
2	{he}
5	{she,he}
7	{his}
9	{hers}

圖 2.7 輸出函數

對關鍵字字串集合產生三個函數之後，AC 演算法使用下面步驟來偵測字串，先設定”S”為目前狀態，y 為目前輸入字元且 y 屬於輸入文字字串集合 T。若 g(S,y)=Q 則演算法將目前狀態”S”轉換為狀態”Q”並輸入 T 中 y 的下一個字元。又若 Output(Q)≠∅(空集合)，則表示一個運算週期結束。假使 g(S,y)=fail，則演算法產生失效轉換，將失效函數產生 f(S)=R。演算法會將下個狀態設定為”R”並繼續運算下個輸入字元。

2.2 Wu-Manber 演算法

此演算法由Sun Wu 和Udi Manber 兩位先生於"A fast algorithm for multi-pattern searching"論文中共同提出。利用預先過濾器(pre-filter)與確認功能的觀念來處理字串偵測。

預先過濾器先將所有關鍵字字串取前 m 個字元產生新的關鍵字字串集合，m 為關鍵字字串中最短的字元數，所有關鍵字經過取統一長度後，可有效的利用硬體且若只先搜尋前 m 個關鍵字字串字元可先預先濾出可疑的字串，有可疑字串後再做下一步確認動作，可增加整體處理速度。此演算法以一次 m 字元處理輸入文字字串，稱為搜尋視窗，m 與關鍵字字串集最短長度相同。另外定義 k 為搜尋方塊，k 值小於 m。演算法中包含位移表(SHIFT table)、雜湊表(HASH table)、字串指標及字根表(PREFIX table)。位移表表示搜尋視窗在當前目標偵測後移動到下個可能的字串所需移動的位元數，假使位移表輸出為’1’時，可能的匹配產生，進一步查詢雜湊表與字根表並指向有可能偵測到的關鍵字字串。位移表可由下列的關係式產生。

```
for h← until N do SHIFT[h] ← m-k+1；
for j← until m-k+1 do
begin
    for i← until y do
       begin
          h←hash( p_i^j p_i^{j+1}...p_i^{j+k0-1} ) ;
          SHIFT[h] ← m-k+1-j;
       end
end
```

假設一關鍵字字串集合其 m=5，其中存在一個關鍵字字串{abcde}，將此關

鍵字字串以搜尋方塊大小 k 拆成 m-k+1 個子關鍵字字串{abc}、{bcd}、{cde}。以圖 2.8 為例若搜尋視窗恰好移動到輸入文字字串中，搜尋視窗處理輸入文字字串中五連續字元{fgcde}，搜尋視窗後三個位元{cde}則被搜尋方塊所選擇，可得 hash{cde}=h，又{cde}為子關鍵字字串集其中之一，所以可以從位移表中得到 SHIFT{h}=0，若 SHIFT{h}=0 則可進入雜湊表 HASH{h}=p 得到字串指標指向目前的搜尋視窗可能符合的關鍵字字串{adcde}。

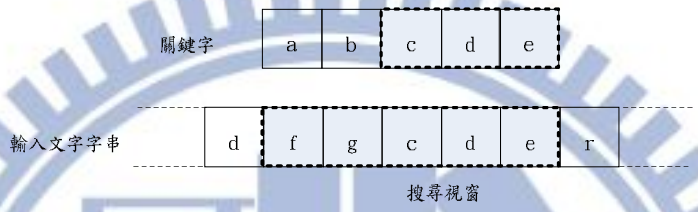


圖 2.8 視窗位移圖一

若搜尋視窗處理輸入文字字串中五連續字元為{fcbcd}如圖 2.9，搜尋視窗後三個位元{bcd}則被搜尋方塊所選擇，可得 hash{bcd}=h，又{bcd}為子關鍵字字串集其中之一，從圖中可以觀察得到若觀察視窗位移一個位元有可能可以搜尋到關鍵字字串{abcde}，所以查位移表可得 SHIFT{h}=1，由於目前搜尋視窗未找到有可能的關鍵字字串，所以先進行視窗位移而未進入到雜湊表查詢。

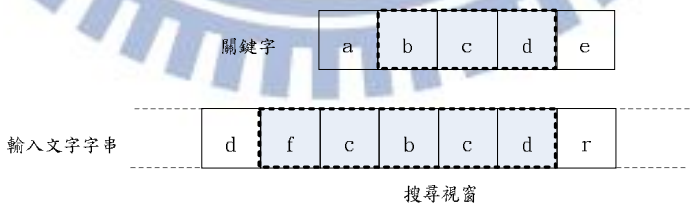


圖 2.9 視窗位移圖二

若搜尋視窗處理輸入文字字串中五連續字元為{xyzuv}如圖 2.10，搜尋視窗後三個位元{zuv}則被搜尋方塊所選擇，可得 hash{zuv}=h，若{zuv}不符合任一子關鍵字字串集其中之一，所以可以從位移表中得到 SHIFT{h}=0，若 SHIFT{h}=0

則代表目前搜尋視窗無可疑關鍵字字串存在，此時搜尋視窗位移 m-k+1 位元，如圖 2.10。

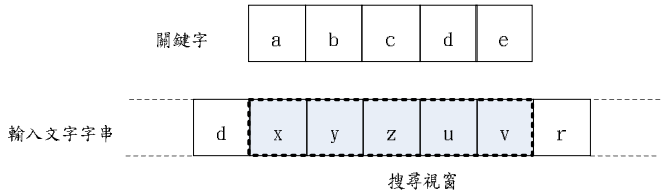


圖 2.10 視窗位移圖三

總結來說演算法先取搜尋方塊字串，將字串與輸入 hash 函數得到一對應值，將其值查位移表可得視窗位移數，若發生位移表為 0 時，代表目前搜尋視窗可能有符合的關鍵字字串存在，此時查詢雜湊表產生關鍵字字串指標，由字串指標查詢有可能的關鍵字字串。

2.3 狀態字串比對(Stateful pattern matching)

此字串比對的方法由 Tsern-Huei Lee 和Nai-Lun Huang於" A Pattern Matching Scheme with High Throughput Performance and Low Memory Requirement"論文中提出，其架構將比對字串分為兩個步驟，第一個步驟先將輸入文字字串放進預先過濾器找出輸入文字字串中可能發生關鍵字字串的位址，第二步再將可能發生關鍵字字串的部分放入驗證器做驗證，驗證可疑的文字字串部分是否真實發生關鍵字字串。預先過濾器的步驟可先將所有輸入字元快速的掃瞄過，由於過濾器不比對完整的關鍵字字串只比對關鍵字字串的部分字元，然而比對的字元越少硬體的處理速度就會越快，但是比對後得到的結果為”可能”匹配的關鍵字字串，所以必須將搜尋到的部分加上驗證器驗證以避免找出的字串非完全吻合關鍵字字串。

2.3.1 成員詢問模塊(Membership Query Module)

首先說明預先過濾器，定義搜尋視窗(search window)W，其視窗長度(window length)為m、搜尋方塊(block)其方塊長度(block size)為k。假設關鍵字字串集合包含N個關鍵字字串，為了系統處理速度將所有關鍵字字串只取前m個字元，接下來再將各個關鍵字字串分為m-k+1組，每組k個位元，並將結果儲存到不同的成員詢問模塊(Membership Query Module, MQM)。

以關鍵字字串{b9c0012e8a272e3226dd022e882743e2f2c3}為例，(此字串取之於ClamAV，以16 進制表示)，設定m=6，k=3，所以取出關鍵字字串前六個字元{ b9c0012e8a27 } ，以k個位元為一組分別儲存在不同的成員詢問模塊中，如上例

♦第一個子字串 (Sub-pattern) 為第一個位元至第三個位元，『*b9c001*』，儲存至第一個成員詢問模塊(MQM1)。

♦第二個子字串為第二個位元至第四個位元，『*c0012e*』，儲存至第二個成員詢問模塊(MQM2)。

♦第三個子字串為第三個位元至第五個位元，『*012e8a*』，儲存至第三個成員詢問模塊(MQM3)。

♦第四個子字串為第四個位元至第六個位元，『*2e8a27*』，儲存至第四個成員詢問模塊(MQM4)。

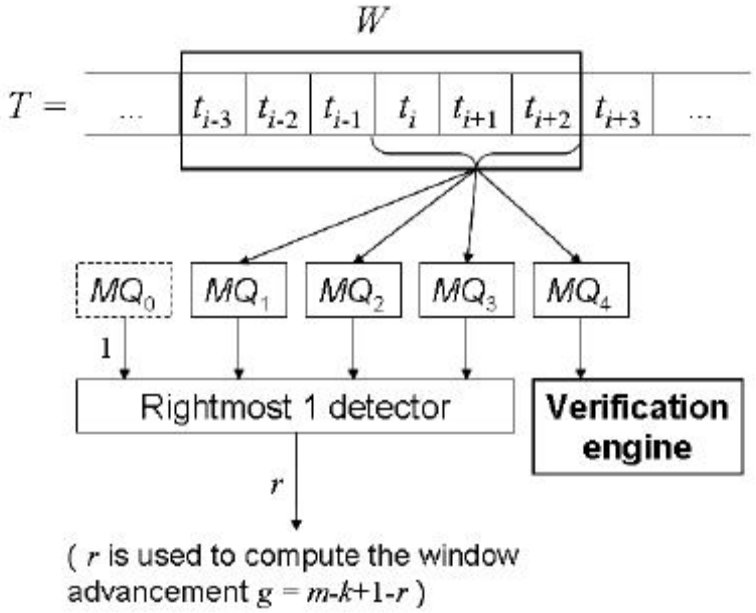


圖 2.11 m=6，k=3 的預先過濾器

承上所述若在關鍵字字串集合中第 i 個關鍵字字串 P_i 取前 m 個字元則 $P_i = p_i^1 p_i^2 ... p_i^m$，則子字串 $p_i^1 p_i^2 ... p_i^k$ 儲存在第一個成員詢問模塊，$p_i^2 p_i^3 ... p_i^{k+1}$ 儲存在第二個成員詢問模塊，以此類推 $p_i^{m-h+1} p_i^{m-k+2} ... p_i^m$ 儲存在第 m-k+1 個成員詢問模塊中。

成員詢問模塊在此預先過濾器中扮演雜湊陣列(hash array)的角色，成員詢問模塊中的的子字串並非直接存入成員詢問模塊中，而是將子字串先經過雜湊函數(hash function)處理，得到一個雜湊值(hash value)，此雜湊值視為一個位址，並將位址指向的內容設定為 1，所以當成員詢問模塊的回饋值為 1 時，代表搜尋視窗可能有關鍵字字串的部分內容存在，利用雜湊的儲存方式可降低儲存空間，但是當雜湊函數設計不夠精確時，例如多個關鍵字字串有相同的子字串時有可能會指向相同的雜湊值而產生誤報。

2.3.2 最右位元偵測器(Rightmost Bit Detector)

得到成員詢問模塊之後，將成員詢問模塊與當前搜尋視窗後k個位元做運算

並將其結果命名為MQM，如圖2.11的例子m=6，k=3的情形下可以得到MQM1、MQM2、MQM3、MQM4，此四個位元當作與搜尋方塊比對子關鍵字字串的結果。若{MQM1,MQM2,MQM3,MQM4}={0010}如圖2.12所示，MQM3為1代表 $\{t_{h+3}t_{h+4}t_{h+5}\}$ 可能符合某關鍵字字串Pi的第三到第五字元，所以將搜尋視窗與可能的關鍵字字串位置對齊即下個搜尋視窗為$\{t_{h+1}t_{h+2}t_{h+3}t_{h+4}t_{h+5}t_{h+6}\}$即移動一個位元。

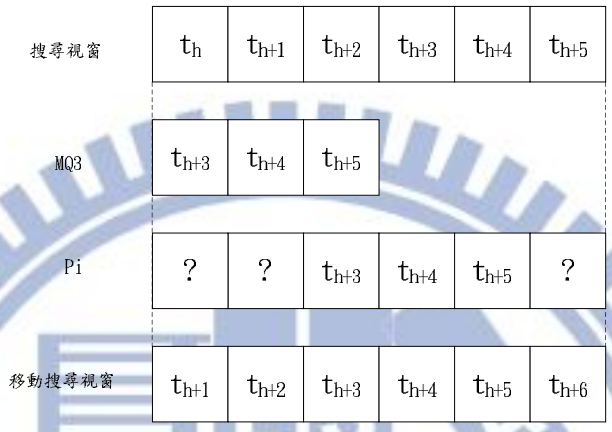


圖 2.12 最右位元偵測器一

若{MQM1,MQM2,MQM3,MQM4}={1100}如圖2.13所示，MQM1為1代表$\{t_{h+3}t_{h+4}t_{h+5}\}$可能符合某關鍵字字串Pi的第一到第三字元，MQM2為1代表$\{t_{h+3}t_{h+4}t_{h+5}\}$可能符合某關鍵字字串Pj的第二到第四字元，目前有兩個位移的選擇，然而若移動較長的距離後可能會有遺漏關鍵字字串未被搜尋到，所以將搜尋視窗位移選擇移動位元數較少的，所以下個搜尋視窗為$\{t_{h+2}t_{h+3}t_{h+4}t_{h+5}t_{h+6}t_{h+7}\}$即移動兩個位元如圖2.13。

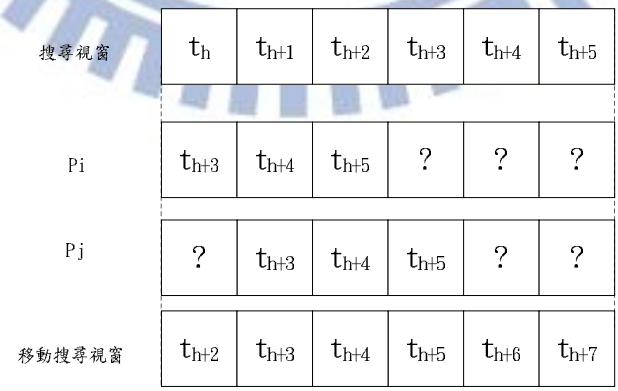


圖 2.13 最右位元偵測器二

若{MQM1,MQM2,MQM3,MQM4}={1001}如圖2.14所示，MQM1為1代表$\{t_{h+3}t_{h+4}t_{h+5}\}$

可能符合某關鍵字字串Pi的第一到第三字元，MQM4為1代表{$t_{h+3}t_{h+4}t_{h+5}$}可能符合某關鍵字字串Pj的第四到第六字元，此時將{t_h}的位址儲存供驗證器驗證，除此之外因MQM1={1}為了不遺漏可能的關鍵字字串所以將搜尋視窗移動到{$t_{h+3}t_{h+4}t_{h+5}t_{h+6}t_{h+7}t_{h+8}$}即移動三個位元。

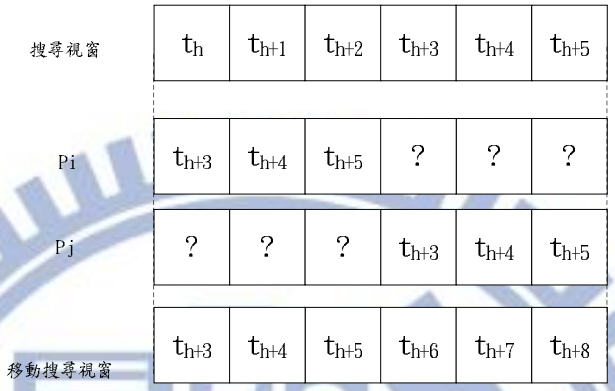


圖 2.14 最右位元偵測器三

若{MQM1,MQM2,MQM3,MQM4}={0000}如圖2.15所示表示目前的搜尋視窗沒有符合的關鍵字字串，所以移動搜尋視窗到{$t_{h+4}t_{h+5}t_{h+6}t_{h+7}t_{h+8}t_{h+9}$}，移動了四個位元而最大的移動距離為m-k+1=6-3+1=4。

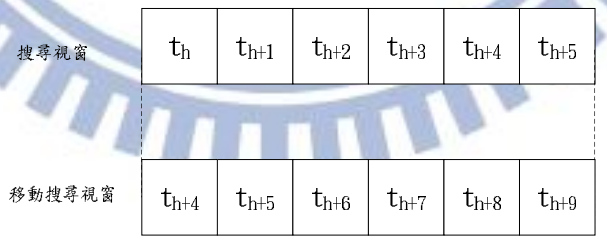


圖 2.15 最右位元偵測器四

歸納來說，若搜尋視窗同時有多個選擇位移位元數，為了避免過大的位移造成關鍵字字串搜尋的疏漏，所以以位移數以最少的位移為原則。若MQM1為1可位移3個位元，MQM2為1可位移2個位元，MQM3為1可位移1個位元，若MQM4為1代表有可疑字串發生，{MQM1,MQM2,MQM3,MQM4}為全零則位移4個位元。若要選擇最小位

移則{MQM1,MQM2,MQM3,MQM4}中存在多個1時，選擇最右邊的1作為位移選擇。

2.3.3 主位元組列(Master Bitmap)

主位元組列(Master Bitmap)如圖 2.16 所示，命 MB={mb1,mb2,mb3,mb4}，其內容為上一個運算周期的成員詢問模塊經過未移之後的值。其功用在於可以二次確認經過位移後的搜尋方塊是否真存在關鍵字字串集，經過二次確認後可降低搜尋關鍵字字串的錯誤率並增加視窗位移速度亦即增加關鍵字字串搜尋的速度。

假設存在一輸入文字字串，且搜尋視窗內容為{abcdef}，若存在一關鍵字字串{defccc}，則在此視窗的成員詢問模塊 MQ={0110}，所以依據節所述規則移動搜尋視窗使得下一個搜尋視窗內容為{def???}，若將上一運算週期的 MQ 做相同的向右位移並使位移後的空缺補 1 後存入 MB 暫存得到 MB={1011}，假若目前的搜尋視窗{def???}的 MQ={0100}若不考慮 MB，依據 MQ 值位移為 2 位元，但若加入主位元組列後，因為 MB={1011}，表示位移前視窗不存在符合第二到第四位元的關鍵字字串，所以即便位移後的 MQ={0100}，也只代表目前視窗為部分符合關鍵字字串但是與關鍵字字串沒有完全符合，所以將 MQ 與 MB 作邏輯的 AND 得到{0000}意即下一個位移值為 4 個位元。

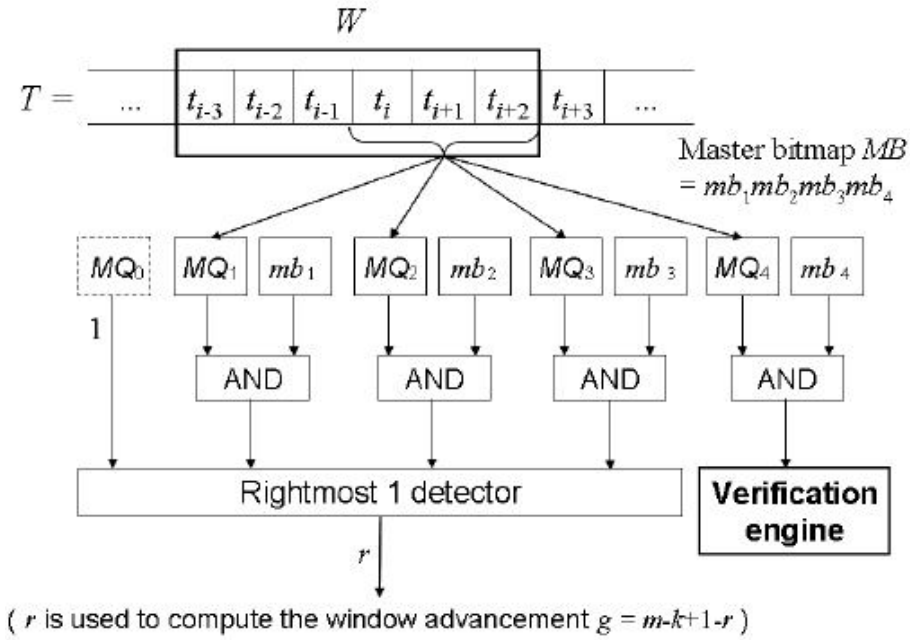


圖 2.16 主位元組列

圖 2.17 為無主位元組列的視窗位移狀態，若此時 MQM={1010}，則 MQ1、MQ3 子字串可能存在於搜尋視窗於是移動搜尋視窗 1 位元，位移後假設 MQM={0010} 則 MQ3、MQ4 子字串可能存在於搜尋視窗於是移動搜尋視窗 1 位元。圖 2.18 為有主位元組列的視窗位移狀態，同上例 MQM={1010}，位移搜尋視窗 1 位元並同時產生 MB 即 MQM 向右位移後，再空缺上補 1 則得 MB={1101}，位移後視窗的 MQM={0010}，則 MQM ⊕ MB={0000}，所以第二次位移可移動四個位元。

圖 2.17 無主位元組列的搜尋視窗移位

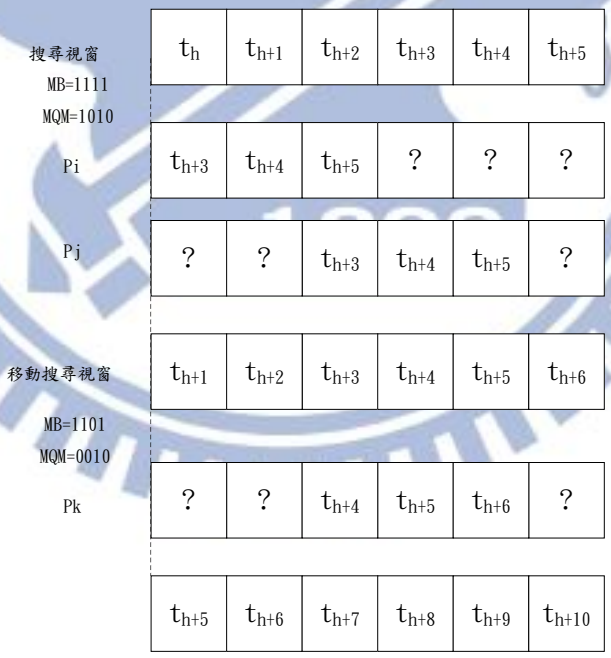


圖 2.18 主位元組列的搜尋視窗移位

第三章

推測預先過濾器

3.1 推測預先過濾器架構及介紹

此架構由李 程輝教授以及黃 迺倫學姊發表之論文”Speculation Pre-Filter for Hardware Accelerated Pattern Matching”中提出，在基於 Aho-Corasick 可在同一時間作多筆字串比對的理論基礎，也由於世代變遷需要更快的處理速度，進而使用有限硬體實現產生更大的字串比對量。

此架構使用狀態字串比對(Stateful pattern match)的方法並可運用到所有的成員詢問模塊(previous query results)，且利用兩個資料方塊(data block)來增加搜尋視窗(search window)的平均位移量，另外因為使用了兩個資料方塊(data block)，所以需使用雙埠記憶體來，也由於使用雙埠記憶體在增加最少的硬體空間下產生較大的資料處理量。

3.2 前級詢問結果(Previous Query Result)

L 為一整數值，其值小於等於最小的比對字串的長度，假設比對中的第 i 個字串 $Pi = p_i^1 p_i^2 ... p_i^L, 1 \le i \le y$，表示為第 i 個字串的長度 L 的(prefix)，另外定義輸入文字字串 $T = t_1 t_2 .. t_n$ (Text string)。

此預先過濾器擁有 L-K+1 個成員詢問模組(membership query modules)，且各模組都有其各自的儲存空間。Q 為輸入字串與詢問模組比較後的結果

$Q_i = q^i_{L-K} q^i_{L-K-1} ... q^i_0$。R 為狀態暫存器 $R = r_{L-K} r_{L-K-1} ... r_0$。R 的初始值為全 1，且 R 為 L-K+1 個字元以 $R = 1^{L-K+1}$ 表示，假使當前的搜尋視窗 $W = t_{i+1} .. t_{i+L}$。產生 Q 值後與目前暫存的 R 做每個字元的單一 AND 邏輯合成產生 R’ 即 R’ = Q ⊕ R。若 $r'_0 = 1$ 則輸入字串的第 i+1 個位置有可能的關鍵字字串產生。若 $r'_i = 0$，i 為 1 到 L-K 所有都為 0，則搜尋視窗位移 L-K+1 個位元。若 $r'_d = 1$，$d \in \{1,...,L-K\}$ 且 $r'_i = 0$，$1 \le i \le d$，則搜尋視窗將位移 d 個位元。假若搜尋視窗位移量為 g 個位元則同步將 R’ 右移 g 個位元，並在移位後的空缺上補 1 產生新的狀態暫存器 R，其表示法為 $R = 1^g \mid R' >> g$。

3.3 搜尋視窗位移量

在這個過濾器中存在兩組搜尋方塊，而當有兩組搜尋方塊時，兩組方塊與搜尋視窗的相對位置影響搜尋視窗可移位的總位元數。第一個搜尋方塊的位址為 j，第二個搜尋方塊的位址為 k，其相對位置可分為四種。

第一種如圖 3.1 所示，$j > i+L-K+1$，$k < i+1$ 或 $k > i+L-K+1$，$j < i+1$。當第一個搜尋方塊(B1)與第二個搜尋方塊(B2)都不完全在搜尋方塊內則兩個方塊產生的 Q 值無法與目前搜尋視窗的關鍵字字串產生關聯。

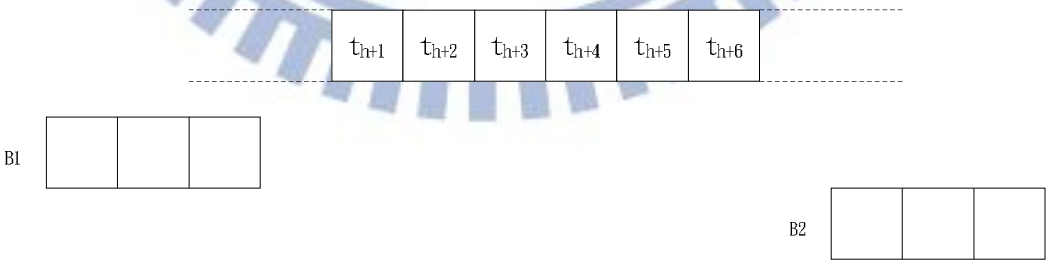


圖 3.1 雙搜尋方塊的相對關係一

第二種情況，如圖 3.2 所示，$j < i+1$，$i+1 \le k \le i+L-K+1$。其中之一的搜尋方塊在搜尋視窗內。若第一個搜尋方塊(B1)不在搜尋視窗內，而第二搜尋視窗位於搜尋視窗內，此時第二搜尋視窗的成員詢問模組比較結果為有效結果，可

反映目前搜尋視窗的位移狀態，假設視窗位移量g被 $R \oplus 1^{i+L-K+1-k} q^2_{L-K} \ldots q^2_{i+L-K+1-k}$ 所決定，而在視窗位移後的新狀態暫存器結果為 $R = 1^g \mid (R \oplus 1^{i+L-K+1-k} q^2_{L-K} \ldots q^2_{i+L-K+1-k}) >> g$ 。

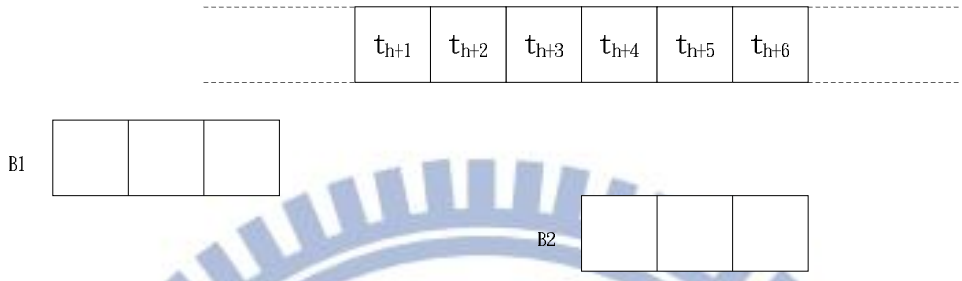


圖 3.2 雙搜尋方塊的相對關係二

第三種情況，如圖 3.3 所示，$j \geq i+1$，$k \leq i+L-K+1$。兩個搜尋方塊都在搜尋視窗的範圍之內。第分成兩次位移來看。第一個搜尋方塊若產生視窗位移量為$g1$，則假若$g1 \geq k-i$則經過第一次的視窗位移後，第二搜尋方塊就會在新的搜尋視窗之外，此時第二搜尋方塊會如第一種情況對新視窗位移無貢獻。所以先假設 $g1 < k-i$ ，$g1$ 由 $R' = R \oplus 1^{i+L-K+1-j} q^1_{L-K} \ldots q^1_{i+L-K+1-j}$ 產生，且 $r'_{g1-1} = \ldots = r'_1 = 0$ ，$r'_{g1} = 1$ ，第二搜尋方塊產生的位移量為 $g2$ 由 $R'' = (1^{g1} \mid R' >> g1) \oplus 1^{i+L-K+1-k+g1} q^2_{L-K} \ldots q^2_{i+L-K+1-k+g1}$ 產生，所以總位移量 $g = g1 + g2$ 。

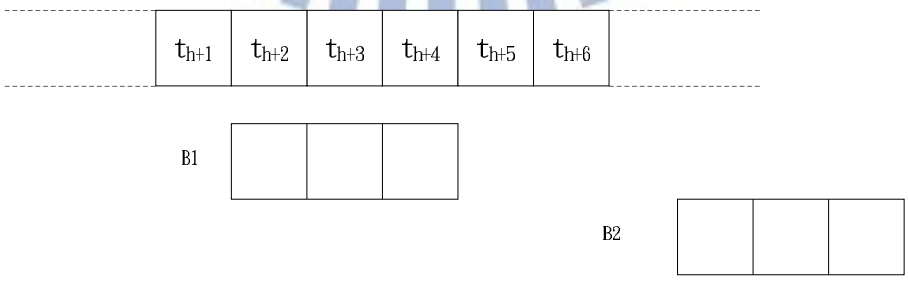


圖 3.3 雙搜尋方塊的相對關係三

第四種情況，如圖 3.4 所示，$i+L-K+1 \geq j \geq i+1$，$k > i+L-K+1$。兩個搜尋方塊都在搜尋視窗的範圍之內。第一個搜尋方塊若產生視窗位移量為 d，則假若 $k > i+L-K+d+1$ 則經過第一次的視窗位移後，第二搜尋方塊就會在新的搜尋視窗之外，此時第二搜尋方塊會如第一種情況對新視窗位移無貢獻。所以先假設 $k \leq i+L-K+d+1$，視窗位移量為 g+s，其中 g 從 $\overline{R} = (1^s \mid R \oplus 1^{i+L-K+1-j} q^1_{L-K} \ldots q^1_{i+L-K+1-j} >> s) \oplus 1^{i+L-K+1-k+s} q^2_{L-K} \ldots q^2_{i+L-K+1-k+s}$ 產生，且 s 滿足 $k-i-L+K-1 \leq s \leq d$。

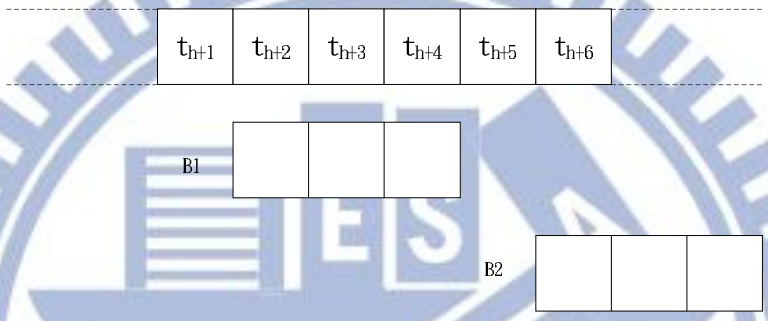


圖 3.4 雙搜尋方塊的相對關係四

以搜尋視窗的最大位移量來說，第一種情況無法產生視窗位移量，而第二種在一搜尋方塊在搜尋視窗之外的情況下只會有一搜尋方塊對位移有幫助所以會與單一搜尋方塊相同，最大的視窗位移量為 L-K+1。第三種情況若 $g1 \geq k-i$ 最大視窗位移視為與單一搜尋方塊相同所以為 L-K+1，若 $g1 < k-i$ 又 $k \leq i+L-K+1$，所以 $g1 < L-K+1$，g1 最大值為 L-K。若以 $g1 = L-K$，$k = i+L-K+1$ 則最大的位移量為第一搜尋方塊位移 L-K 加上第二搜尋方塊位移 K，總共可位移 L 位元。第四種情況，若第一搜尋方塊為 $j = i+L-K+1$ 則第一搜尋方塊提供的位移為 L-K+1，又假設第二搜尋方塊為有效則可提供 K 個位移兩者加總可提供 L-K+1+K=L+1 的位移。

經過推論若要達到最大的視窗位移量，需要用到第四種情況，$i+L-K+1 \geq j \geq i+1$ 當 $j = i+L-K+1$ 時第一個搜尋方塊會有最大的位移量，若經過第一個搜尋方塊位移視窗 s 位元之後，若第二搜尋方塊也同樣在位移後的

視窗後 k 個位元上，即 $k = i + L - K + 1 + s$，$1 \le s \le L - K$ 且 $r_s = 1$ 則會有最大視窗位移量 s+g。第一個搜尋方塊產生的位移量為 s，由 $R \oplus Q1$ 經過最右位元偵測器後決定，而 g 則由第二個詢問模塊 Q2 與產生的 $\overline{R}$ 經最右位元偵測器決定 $\overline{R} = (1^s \mid (R \oplus Q1) >> s) \oplus Q2$ 。

3.4 推測預先過濾器的操作

定義一個推測預先過濾器的運算週期，假設 R 為第 M-1 次運算週期的狀態記憶體，當第 M 次運算時讀入搜尋視窗大小的輸入文字字串，而後輸入成員詢問模塊得到 Q1、Q2，為第 M 次詢問模塊的結果。若 $q_0^1 = 1$ 則此搜尋視窗出現可疑關鍵字字串，記憶目前搜尋視窗位址並輸出給驗證器。運用 $R \oplus Q1$ 得到第一個視窗位移的結果 g，此時將狀態記憶體更新並同步位移 g 位元，更新後的狀態記憶體與 Q2 做運算 $\overline{R} = (1^s \mid (R \oplus Q1) >> s) \oplus Q2$，產生第二搜尋方塊可產生的位移 sum，同時若 $\overline{r}_0 = 1$ 則此時發生可疑關鍵字字串，記憶位移 sum 後的搜尋視窗位址並輸出給驗證器。最後將 $\overline{R}$ 做向右’ sum’ 位元的位移並更新 R，$R = 1^g \mid \overline{R} >> g$ 。得到視窗位移量後需更新搜尋視窗即更新搜尋視窗位址，此時完成以上動作稱為一個運算週期。

將運算週期依據運算順序分為下列的步驟，並先給定運算初始值 R 初始值為全 1，$R = 1^{L-K+1}$，第一搜尋方塊位移量初始為 0，g=0、sum=s’=1，第一搜尋方塊位址 ADDR1=L-K+1，第二搜尋方塊位址 ADDR2=L-K+2。

第一步驟將初始位址 ADDR1、ADDR2 輸入儲存輸入文字字串的雙埠記憶體，產生 B_{ADDR1} 與 B_{ADDR2} 即為輸入文字字串其長度為設定的搜尋方塊大小 K，其內容為搜尋方塊所檢視的內容。

第二步驟，根據輸入的 B_{ADDR1}、B_{ADDR2} 與建立好的雜湊表產生雜湊位址並產生詢問模塊的結果 Q1、Q2。

第三步驟將 Q1 與狀態記憶體做個別位元 AND 邏輯運算並更新狀態記憶體即 $R = (1^{s'} \mid (R \oplus Q1) >> s') \oplus Q2$，若此時 $r_0 = 1$ 則將此時搜尋視窗的位址 ADDR1-L+K 為可疑關鍵字字串位址將其輸出並儲存。

第四步驟，依據 R 值，查詢 lookup 表產生 g、sum 和 s’ ，若此時 $r_0 = 1$ 則 ADDR2-L+K 位址有可能的關鍵字字串產生，同一時間使 ADDR1=ADDR2。

第五步驟，將狀態記憶體 R 更新為 $R = 1^g \mid R >> g$，經位移後的搜尋視窗位址為 ADDR2-L+K+g+sum，並產生新的第一搜尋方塊位址 ADDR1=ADDR2+g，第二搜尋方塊位址為 ADDR2=ADDDR2+sum。

以上步驟完成後其流程如圖 3.5 所示，重複運算周期直至所有輸入文字字串完成偵測動作。若從時序上則如圖 3.6 所示。將流程上的五個步驟分為五個相位如表 3.1 所示。

表 3.1 推測預先過濾器相位表

1	• Read the blocks B_{ADDR1} and B_{ADDR2}
2	• Read the query reports Q_1 and Q_2
3	• $R = (1^{s'} \mid (R \oplus Q_1) >> s') \oplus Q_2$ • Identify the text position $ADDR1 - L + K$ as the starting position of a suspicious pattern occurrence if $q_0^1 = 1$
4	• Consult the look-up table according to the current content of R to read the corresponding values of g, sum, and s' • Identify the text position $ADDR2 - L + K$ as the starting position of a suspicious pattern occurrence if r_0 =1 • $ADDR1 = ADDR2$
5	• $R = 1^g \mid R >> g$ • $ADDR1 = g + ADDR1$ • $ADDR2 = sum + ADDR2$

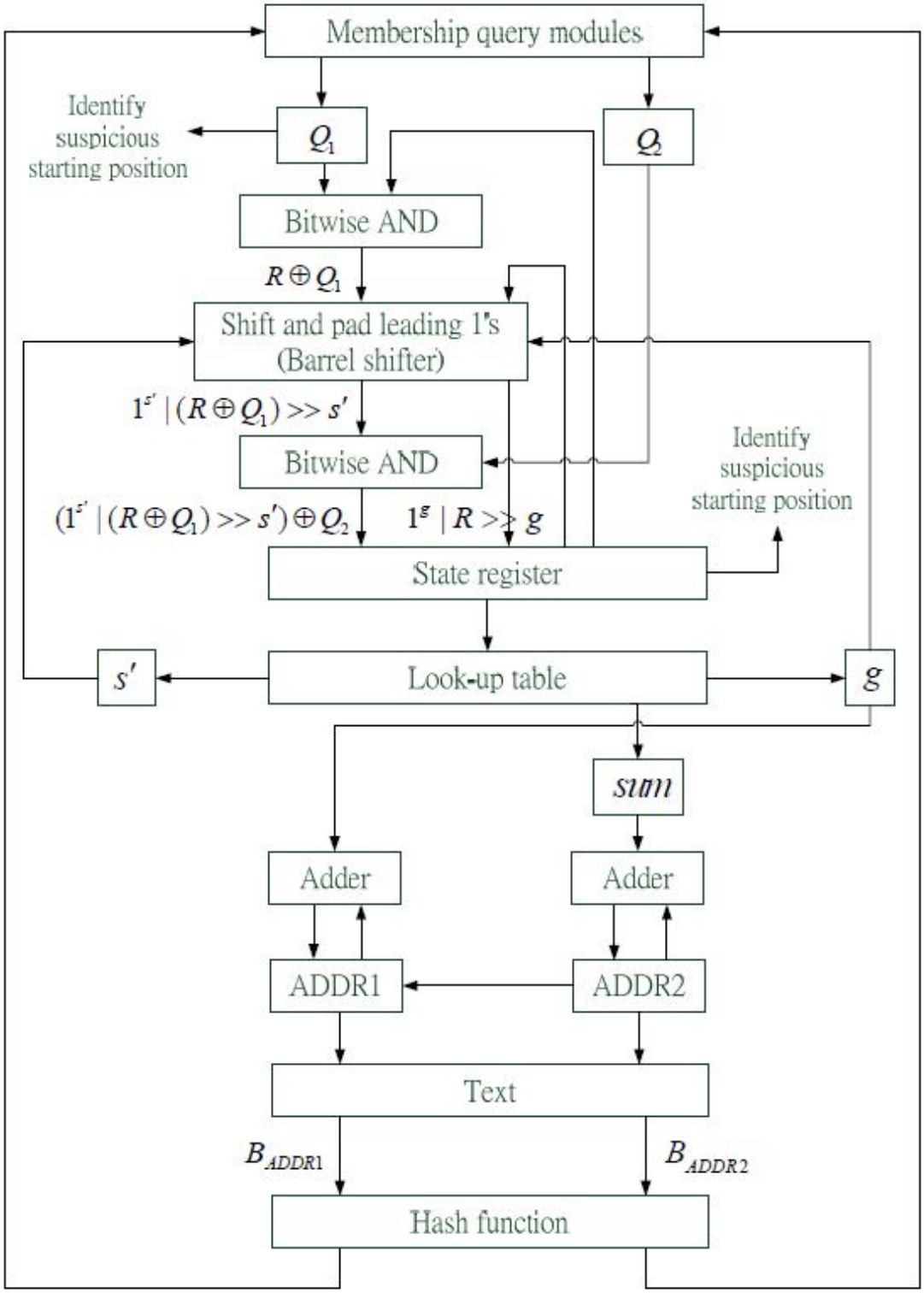


圖 3.5 推測預先過濾器流程圖，引用自 ref[4] Fig1

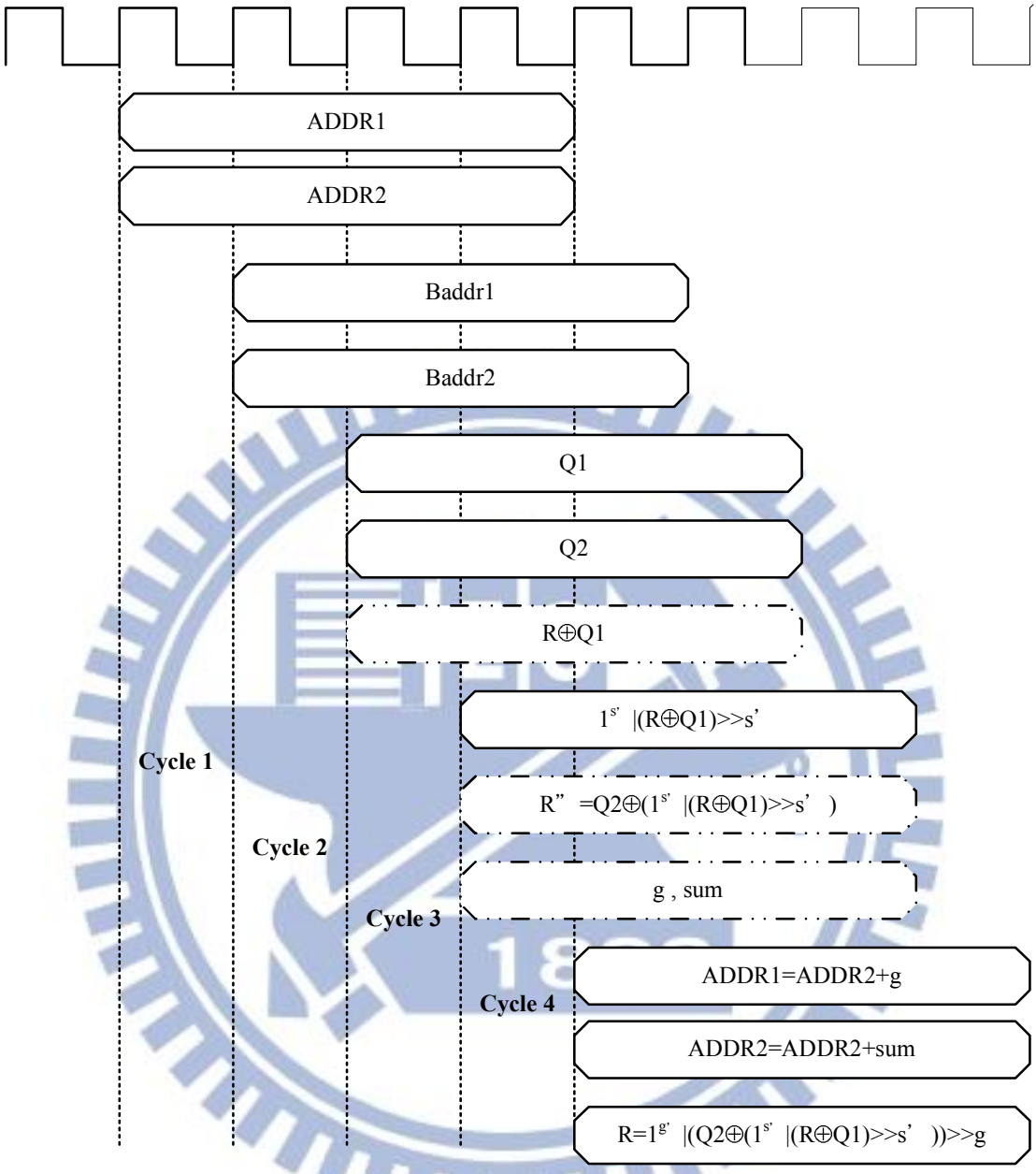


圖 3.6 推測預先過濾器時序圖

第四章

管線推測預先過濾器

4.1 管線推測預先過濾器的架構

使用推測預先過濾器的架構，以增加有限的硬體資源、記憶體下，來增加關鍵字字串的搜尋的速度，這裡使用管線(pipeline)的方法。

在推測預先過濾器中，一個運算週期包含五個相位，第一個相位是將輸入文字字串從記憶體中讀出以 RD 表示，第二個相位是將詢問模塊的結果輸出以 QR 表示，第三個相位是將搜尋視窗做第一次的位移以 SH1 表示，第四相位是將搜尋視窗做第二次位移以 SH2 稱之，第五相位是產生新的搜尋方塊位址以 NA 表示。一個搜尋視窗經過一個運算週期如圖 4.1 所示，第一步先經過 RD 相位，此時使用雙埠記憶體做讀取動作，RD 相位產生輸出資料後把結果放到 QR 相位，此時 RD 使用的記憶體會在等待狀態。QR 相位使用記憶雜湊位址的雙埠記憶體，利用輸入搜尋方塊內容做讀取動作，讀取後輸出詢問模塊的結果。每個相位使用的運算單元與記憶體在其單位功能完成後就會處於預備等待狀態，等待下個運算週期到來。為了使每個相位的單位硬體都可以全速工作以達到增加運算速度的功效，所以使用管線的概念。

為做管線將輸入文字字串 $T = t_1 t_2 t_3 .. t_{100}$ 分為多個子文字字串集合，下述例子將分為四個子集合，四個子集合分別為 $T_1 = t_1 t_2 t_3 .. t_{32}$ 、$T_2 = t_{26} t_{27} t_{28} .. t_{57}$ 、$T_3 = t_{51} t_{52} t_{53} .. t_{82}$ 、$T_4 = t_{76} t_{77} t_{78} .. t_{100}$ 。在分割子集合字串時，為避免因分割而導致分割集合邊緣的字串中有可疑的關鍵字字串因被中斷而被偵測機制忽略，所以在

分割的子集合邊緣會多加上 L-K+1 個字元，此例分割是使用 L=10、K=4，所以第一個子集合T_1 是原輸入文字字串 100 個字元除四後再加上 L-K+1=7 總共包含 32 個字元。如圖 4.2 所示，RD 相位在 cycle 1 處理T_1 的輸入位址並輸出T_1搜尋方塊資料，cycle 2 處理T_2 的輸入位址並輸出T_2 搜尋方塊資料，cycle 3 處理T_3 的輸入位址並輸出T_3 搜尋方塊資料。相較圖 4.1，RD 相位在 cycle 2 到 cycle 5 在等待狀態，硬體使用率就增加了五倍。從另一個觀點來看，在 cycle 2 時，RD 相位的硬體處理T_2 子集合的字串讀取，QR 處理T_1 子集合的詢問模組，cycle 3 時 RD 相位的硬體處理T_3子集合的字串讀取，QR 處理T_2 的詢問模組，SH1 處理T_1 的第一視窗位移。若使用管線方式，硬體以平行處理四組子集合輸入字串，除了在運算執行初始與結束期間硬體會有等待情形外，其餘時間皆可充分利用硬體運算。

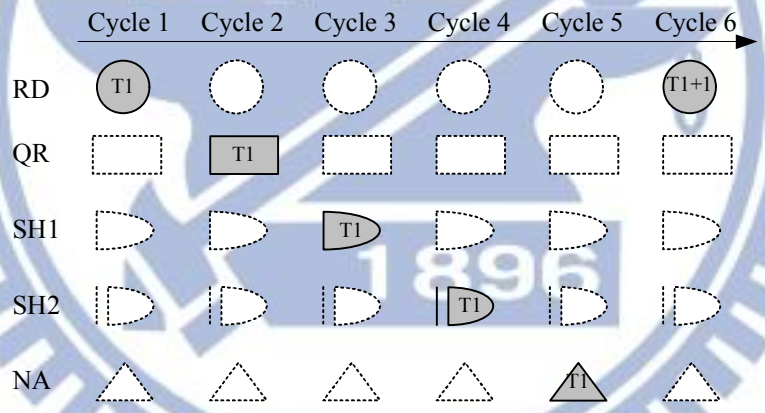


圖 4.1　無管線的相位圖

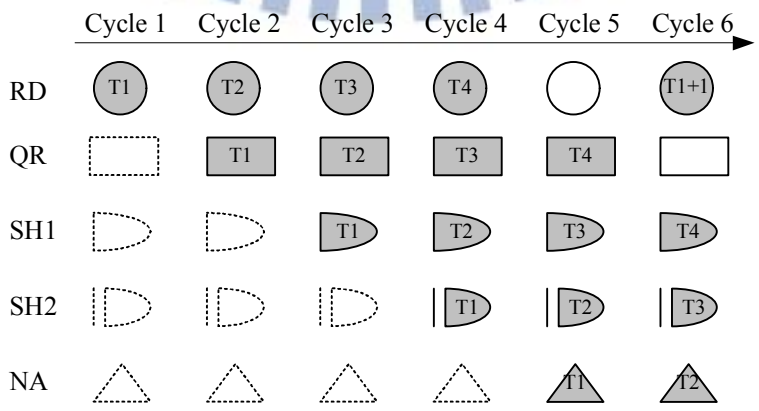


圖 4.2　有管線的相位圖

4.2 管線推測預先過濾器的危障(Hazard)

如圖 4.3，第三相位 SH1 產生的輸出為 $\overline{R} = (1^s \mid (R \oplus Q1) >> s) \oplus Q2$，此時產生的 $\overline{R}$ 屬於 T_1 按照理論當作搜尋視窗在 cycle 4 產生 $\overline{R}$ 的輸入，但實際上在 cycle 4 時，SH1 硬體正在處理 T_2 的第一視窗位移，其硬體暫存器中儲存為 T_2 視窗位移量，若將此時的輸入當作 T_1 第四相位輸入處理會產生錯誤。另一方面，第四相位時產生 T_1 的第二位移量，此值應作下一個相位，相位五 NA 取做運算產生下一個 T_1 的新位址，但是在運行到相位五時，SH2 的硬體暫存器儲存為 T_2 的第二位移量，若同時用作產生 T_1 的新位址，會發生錯誤。在上述兩種情況下管線運作會因為取出錯誤的數值而發生錯誤偵測及錯誤視窗位移，即管線的資料危障(hazard)。

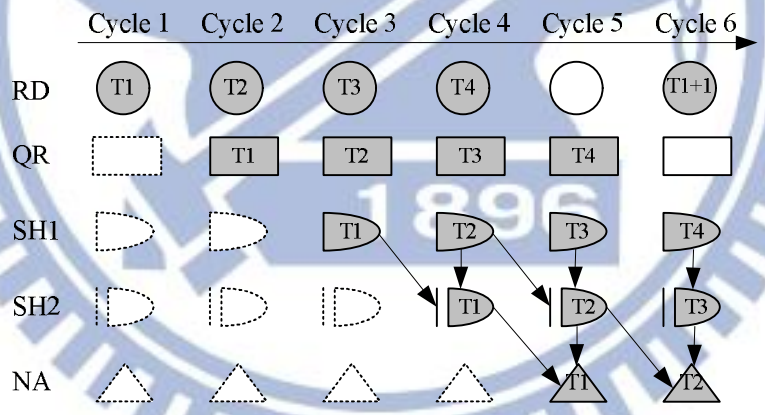


圖 4.3 使用管線時發生的錯誤

為解決管線產生的危障問題，在 SH1 與 SH2 輸出增加狀態暫存器，從原本無管線(Pipeline)的一組狀態暫存器，若如本例使用四組管線(pipeline)則增加為四組狀態暫存器。以搜尋視窗長度 10 位元，搜尋方塊長度 4 位元的情形下，一組狀態暫存器為 7bits，在 SH1 與 SH2 各增加四組狀態暫存器則為 56bits。如圖 4.4 所示，Reg1 記憶 SH1 產生的當前 T_1 狀態暫存 $\overline{R_1}$，Reg2 記憶 SH1 產生的當前

T_2 狀態暫存 $\overline{R_2}$，Reg3 記憶 SH1 產生的當前 T_3 狀態暫存 $\overline{R_3}$，Reg4 記憶 SH1 產生的當前 T_4 狀態暫存 $\overline{R_4}$。Reg21 記憶 SH2 產生的當前 T_1 狀態暫存 R1，Reg22 記憶 SH2 產生的當前 T_2 狀態暫存 R2，Reg23 記憶 SH2 產生的當前 T_3 狀態暫存 R3，Reg24 記憶 SH2 產生的當前 T_4 狀態暫存 R4。以增加可容忍的硬體解決因管線產生的危障(Hazard)問題。

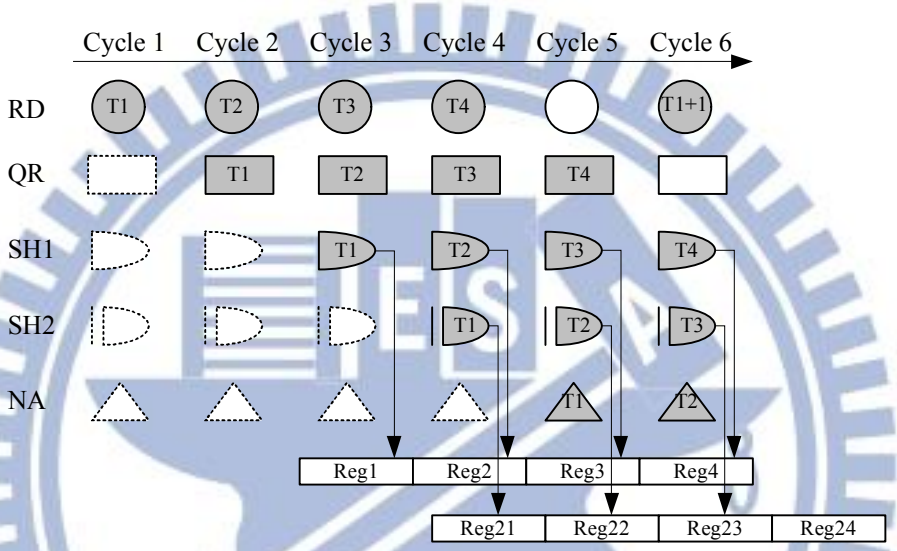


圖 4.4 管線推測預先過濾器錯誤的解決示意圖

4.3 管線推測過濾器的操作

在管線推測預先過濾器操作中，使用與推測預先過濾器相同的流程圖，因其操作過程是相同的，使用管線的目的在於充分利用已有的硬體來達到高速的目的。雖在操作上相似但是在時序上卻略有不同，因為有管線的預先過濾器在同一時間內處理多個子集，不同的相位硬體單元在長時間視為全速運算，如圖 4.5 所示。

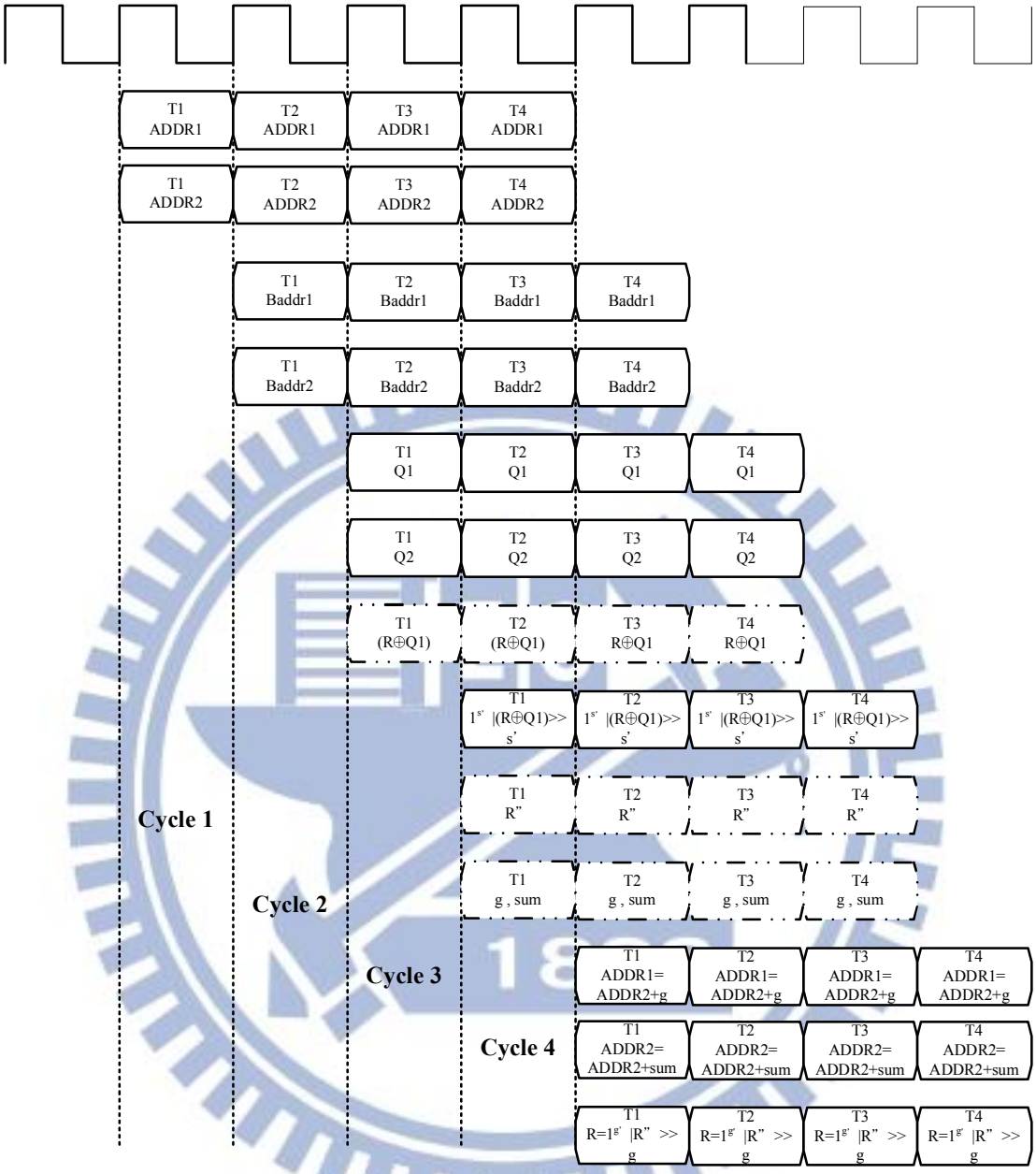


圖 4.5 管線推測預先過濾器時序圖

第五章

FPGA 的實現

5.1 推測預先過濾器方塊圖

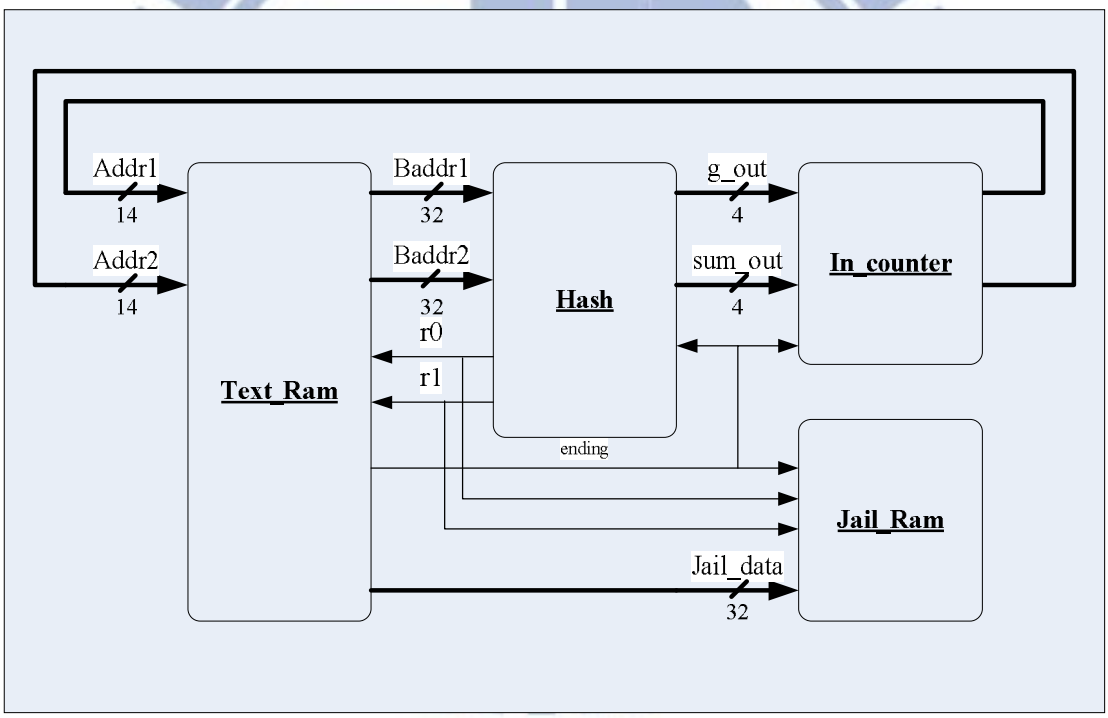


圖 5.1 推測預先過濾器方塊圖

圖 5.1 為推測預先過濾器的實現的方塊圖，其中分為四個部份，分別為輸入字串記憶單元(Text_Ram)、雜湊計算單元(Hash)、輸入位址計算單元(In_counter)和可疑位址儲存記憶單元(Jail_Ram)。在此的實現模型中將以隨機產生的 65536 bits 當作輸入文字字串的測試字串而搜尋視窗大小設定為 10bytes，搜尋方塊

設定為 4bytes，承前幾章的理論成員詢問模塊數量為 10-4+1=7 個。方塊圖中的訊號如表 5.1 所示。

表 5.1 推測預先過濾器方塊圖之訊號表

訊號	工作
Addr1[13:0]	第一組搜尋視窗所搜尋的資料在Ram中的位址指標
Addr2[13:0]	第二組搜尋視窗所搜尋的資料在Ram中的位址指標
Baddr1[31:0]	第一組搜尋視窗搜尋的資料
Baddr2[31:0]	第二組搜尋視窗搜尋的資料
g_out[3:0]	第一搜尋視窗所需前進的位元數
sum_out[3:0]	第二搜尋視窗所需前進的位元數
Jail_data[31:0]	經搜尋後有可能發生符合字元的位址指標
r0	query 1發生嫌疑位址指標的時間
r1	query 2發生嫌疑位址指標的時間
ending	搜尋完所有資料時發出完成的訊號

5.1.1 輸入字串記憶單元的產生與設計

先產生測試之用的字串，其產生方式為透過 c++隨機產生 0-255 的值，以 16 進制儲存，總共產生 65536 bits，並依其順序將測試字串編號，因實作中二位元運算計數的初始值為全零，所以將第一個字元的資料編號為 0，第二筆字元的資料編號為 1，用此原則最後一筆字元資料編號為 8191。由於 ClamAV 所釋放出的公開病毒碼中，最長的長度為 10 個字元，所以依據此原因將推測預先過濾器實現中所用的搜尋視窗長度(L)設定為 10 字元，而搜尋視窗區塊大小(K)設定為 4 字元。

推測預先過濾器中須同時存取視窗區塊大小的位元組做後續比對之用，在這裡視窗區塊大小設定為 4，因此設計中將儲存測試字串的記憶體分為四個部份分別稱為 Ram_0、Ram_1、Ram_2 與 Ram_3 並各別儲存編號為 4n、4n+1、4n+2 和 4n+3 的測試字串，n=0,1,…,2047，如 圖 5.2 所示。實際實現中在此區塊使用了四個長度為 8bits、深度為 2048 的雙埠隨機記憶體(Dual-port RAM)。因為使用到兩組搜尋視窗在不增加記憶體使用量的情況下選擇雙埠隨機記憶體，雙埠隨機記憶體提供兩組位址指標可同時從記憶體中取出兩筆資料，兩組位址指標對應如圖 5-1 的訊號 Addr1 和 Addr2。Addr1 和 Addr2 各為 14 bits，前 3 bits 指向目前資料位於 Ram_0、Ram_1、Ram_2 或 Ram_3 的記憶體中，後 11 bits 則指向該記憶體中的實際位址。

記憶體架構分為四個部份，而位址指標指向的是搜尋視窗方塊中的第一筆資料，根據第一筆資料的位址可依據下列規則產生第二至四筆資料的位址，例如若搜尋視窗方塊中第一筆資料位於 Ram_0 第 y 個位置，則第二筆資料位於 Ram_1 第 y 個位置，若第一筆資料位於 Ram_3 第 y 個位置，則第二筆資料會位於 Ram_0 第 y+1 個位置，此範例如 圖 5.3 所示。根據上例推得第二筆資料的位址第 13 和 12 的 bit 為第一個位址第 13 和 12 的 bit 的值加上 1，其值的溢位則加入第 14 個位元視為記憶體實際位址是否加 1 的依據，從另一個觀點來看無論位址為何在此

種架構下，四個記憶體都會同時取出資料，由 Ram_0 產的資料稱 D0，Ram_1 產生的資料稱為 D1，Ram_2 產生的資料稱為 D2，Ram_3 產生的資料稱為 D3，若第一筆資料產生的位址在 Ram_0 則 Text_Ram 輸出的資料 Baddr={D0,D1,D2,D3}，若第一筆資料產生的位址在 Ram_1 則 Text_ram 輸出的資料 Baddr={D1,D2,D3,D0}，若第一筆資料產生的位址在 Ram_2 則 Text_ram 輸出的資料 Baddr={D2,D3,D0,D1}，若第一筆資料產生的位址在 Ram_3 則 Text_ram 輸出的資料 Baddr={D3,D0,D1,D2}，對四個記憶體輸出的資料作排序的動作以產生正確的搜窗方塊資料，上述的動作稱之為搜尋視窗方塊的排序。排序後產生如圖 5.1 的 Baddr1 和 Baddr2 兩組 32bits 的搜尋視窗方塊資料。

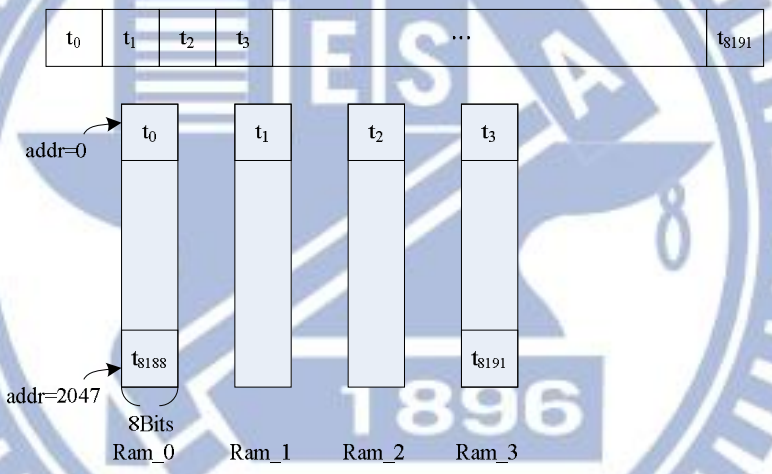


圖 5.2 測試字串的儲存

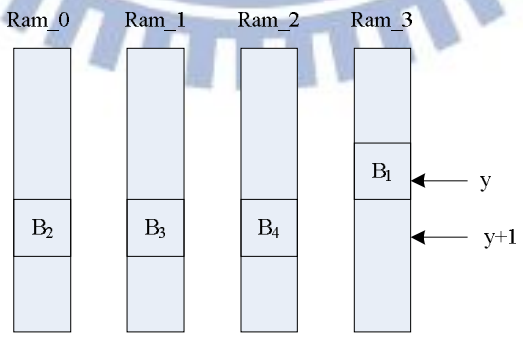


圖 5.3 搜尋視窗的資料讀取

搜尋視窗方塊的排序實現是利用 barrel shifter，其優點在於使用組合邏

輯產生重新排列後的資料可減少 flip flop 的使用量，雖然在資料變換的瞬間會有不穩定的暫態產生，但是不會影響資料的讀取，因為資料變換且不穩定的區域並非下個資料存取的區域，所以可容任資料暫時的不穩定性。

表 5-1 為資料位移狀態表，產生資料重組的控制位元為 Addr 第 13 和第 12 位元，其位移結果如表右的 Data 所示。 表 5-2 為控制位元產生的實際位移動作。圖 5.4 則為搜尋視窗方塊的排序利用 barrel shifter 的實現，在這裡使用八組二選一的多工器，多工器的輸入為 8 位元，虛線左方的多工器由 Addr[11]控制，虛線左方則由 Addr[12]控制。

表 5-2 資料位移狀態表

Operation Addr[12:11]	Data
00	{D0, D1, D2, D3}
01	{D1, D2, D3, D0}
10	{D2, D3, D0, D1}
11	{D3, D0, D1, D2}

表 5-3 資料位移動作表

2 bits opcode Addr[12:11]		Operation
0	0	無位移
0	1	往左位移1
1	0	往右位移2
1	1	往右位移1

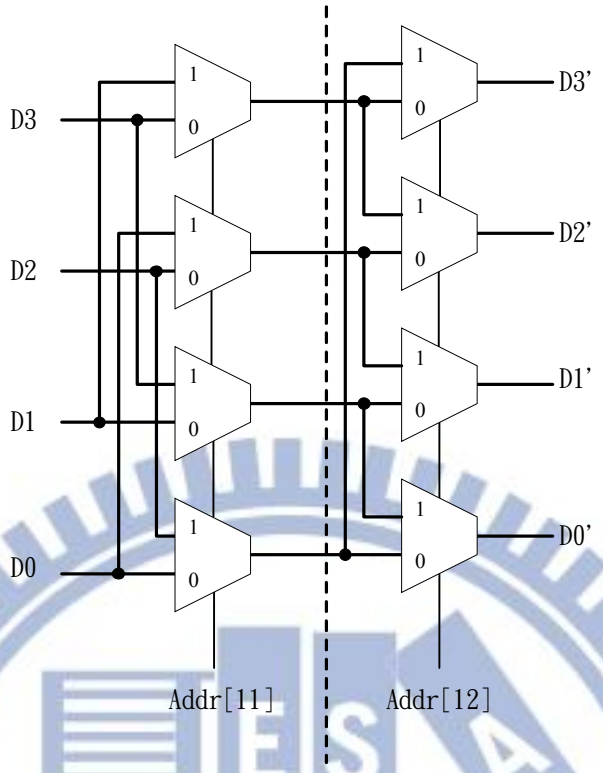


圖 5.4 搜尋視窗方塊排序的實現

5.1.2 雜湊計算單元的產生與設計

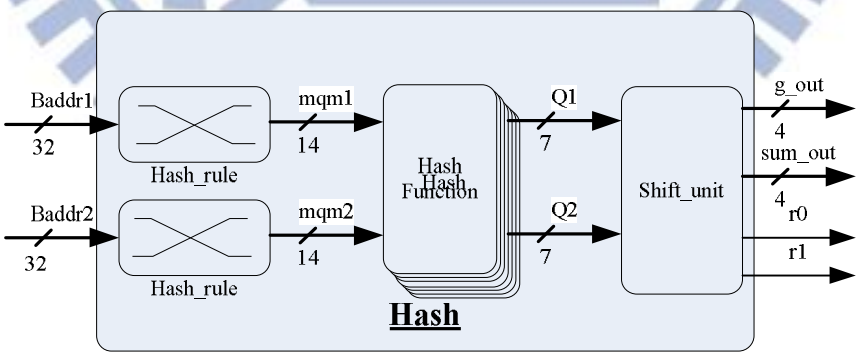


圖 5.5 雜湊計算單元方塊圖

雜湊計算單元主要執行雜湊功能計算以及產生視窗位移量的工作，在實現上將其分為三個功能方塊，分別為 Hash rule、Hash Function 和 Shift_unit。

Hash rule 其功能先產生需要找尋的字串建立的雜湊規則表，此表為查詢表(lookup table)，當有輸入資料時查詢雜湊規則表產生相對應的雜湊值，並將其值輸出，此動作為邏輯運算。

Hash Function 由七個長度為 1 位元、深度為 2048 的雙埠隨機記憶體(Dual-port RAM)組成，雙埠的目的是為了同時存取兩組搜尋視窗產生的雜湊值當作記憶體的找尋位址，由雜湊值的位址各自從七個記憶體中各查出 1 位元作為 Q1 與 Q2 的內容資料。

Shift_Unit 依據推測預先過濾器的視窗位移規則，當 mqm1 的最後一個位元為 1 時代表此時的偵測視窗有可能有符合的字串，此時 r0 輸出為高準位，代表嫌疑字串的事件發生。依據是窗位移規則建立一個查詢表(lookup table)其內容描述當 mqm1 第二個位元為 1 時，位移量為 1 位元，第三位元為 1 且第二位元為 0 時，位移量為 2 位元，以此類推當第七位元為 1 且其餘位元為 0 時，位移量為 6，當 mqm1 第七至第二位元皆為 0 時，位移量訂為 7，代表此偵測視窗無嫌疑字串。mqm1 依據上述規則查表後進行位移，並於位移後與暫存的 R 做 bit wise and 產生新的暫存 R’ 值。另一方面 mqm1 和 mqm2 是同時產生所以也同時進行以上步驟，意即 mqm2 最後一位元，即第一個位元為 1 時，代表第二搜尋視窗有嫌疑字串產生此時 r1 輸出為高準位，此時先將 mqm2 和 R’ 產生新的暫存 R”，依據 R” 的結果除了第一個位元外最低位元的 1 作為 g_out 的輸出，次低位元的 1 作為 sum_out 的輸出，例如 R”={0,0,1,0,1,0,0}則最低位元的 1 為第三個位元，所以 g_out=2，次低位元的 1 為第五個位元，所以 sum_out=4，若 R” 中只有一個 1 時，sum_out 則為 7，若 R” 中無 1 則 g_out 為 7，sum_out 為 8。依據以上的規則也將建立一個 lookup table 以作查表之用，g_out 輸出代表 Addr_1 的位移量，sum_out 代表 Addr_2 的位移量，r0、r1 代表發現嫌疑字串的訊號作為後續儲存嫌疑視窗位置之用。

圖 5-6 為 Shift_unit 的時序圖，圖中 CLK 為系統時脈，此實現中皆以時脈

正緣觸發，無穿插負緣觸發以避免產生較長的 set up time 與 hold time，使得模擬後的速度會較快。當 Addr1 或 Addr2 產生變化的時候，即新位址產生時會從記憶體中取出新的 Baddr 資料，由 mqm1 產生 Q1 也為從記憶體取出資料的動作，所以產生的輸出變化都會在下一時脈發生，因此 Q1、Q2 為 Baddr1 與 Baddr2 推遲一時脈產生變化。Q1 與 R 做 bit wise and 動作為組合邏輯所以會與 Q1 值同時產生變化，產生 R”有經過位移是經過 LATCH 的動作所以會在 Q1、Q2 下一時脈發生變化，R”與 Q2 的 bit wise and 動作同為組合邏輯所以會與 R”值同時產生變化，在此同時由於產生 g 與 sum 為查表動作，在此實現上也為組合邏輯動作所以也會與 R”同時產生。經過上面程序之後 R”經過位移後將值儲存於 R 中，其新值將在下一運算週期中使用。

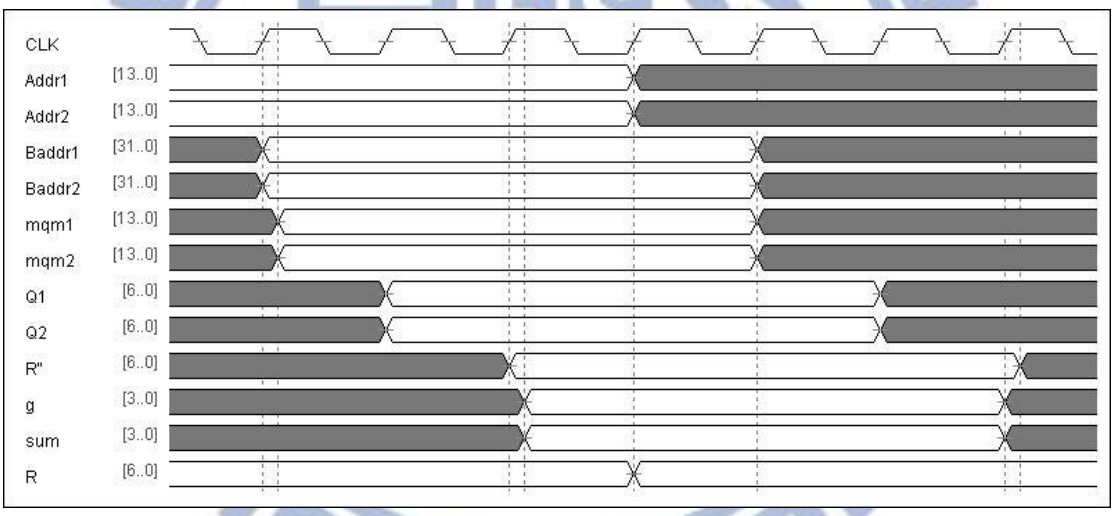


圖 5.6 Shift_unit 時序圖

依據上述經過實現的時序圖描述後，將推測預先過濾器的相位(phase)因應部份使用組合邏輯而做相位(phase)的重組。原 Phase 3 中暫存 R 和 Q1 做(bit wise and)位移後與 Q2 的步驟可同時用組合邏輯將 g , sum 的合成出，因此可排在相同 phase。而原 Phase 3 中根據 $q_0^1 = 1$ 讀出嫌疑字串位址因與後續運算無關也無影響，所以可同步合併於 Phase 2。重組後如 表 5.1 所示 phase 1 為將資料從記憶體內讀出，phase 2 為產生兩組詢問模組，並同時讀取 Q1 以及產生 r0 訊號來提供 Jail_data 方塊做儲存可疑字串位址之用。Phase 3 將 Q1 和暫存 R

做(bit wise AND)根據查表產生位移量並在位移後與 Q2 做(bit wise AND)，此結果稱為 R”。在依據 R”讀出最小位元與次小位元的”1”，並查表產生 g 與 sum。Phase 4 將 R”位移 g 後儲存至暫存 R，作為下一個運算週期之用，另外將上一個 Phase 產生的位移量 g， sum 與 Addr_1、Addr_2 做運算，產生新的位址做下一運算週期讀出 Text 資料之用。

表 5-4 相位表

phase	*action*
1	● Read the B_{ADDR1} and B_{ADDR2}
2	● Read the query reports Q1 and Q2 ● If q_0^1=1 ADDR1-L+K as the starting position of a suspicious
3	● $R''=(1^{s'}\|R\oplus Q1)>>s') \oplus Q2$ ● Consult g,sum,s' ● If r''_0=1 ADDR2-L+K as the starting position of a suspicious
4	● $R=(1^{g'}\|R''>>g')$ ● ADDR1=g+ADDR1 ● ADDR2=sum+ADDR1

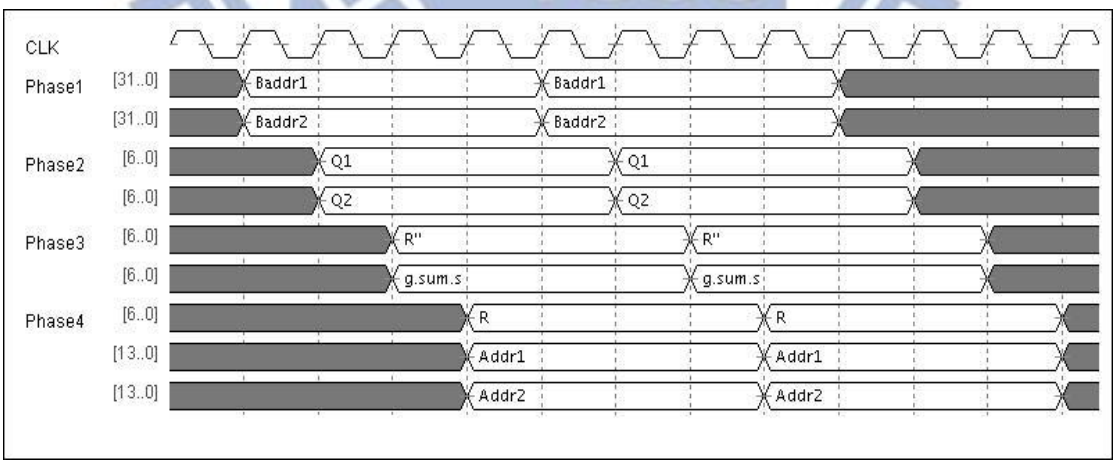


圖 5.7 相位圖

5.1.3 輸入位址計算單元的產生與設計

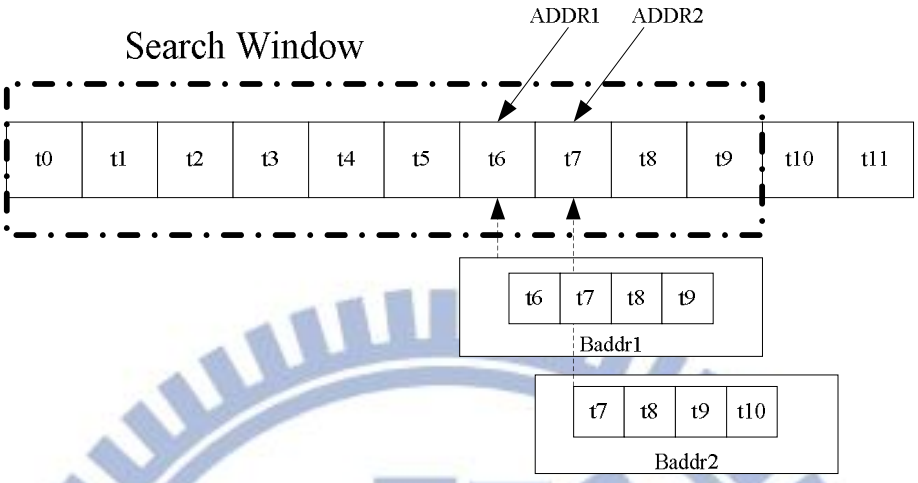


圖 5.8 初始搜尋視窗示意圖

如圖 5.8 所示在初始的搜尋視窗在 L=10 的狀況下，視窗內容包含第一到第十個字元，而兩個搜尋方塊(K)則各包含第七到第十個字元與第八到第十一字元，在初始設定定義第一個搜尋方塊位於 L-K+1，第二個搜尋方塊位於 L-K，相差一字元且第二搜尋方塊部分位於搜尋視窗之外，如圖 5.8 所示。而後 Addr1 與 Addr2 的值則根據理論淨位移分別為 s’+g 與 sum，而在實現上因輸入文字字串分為四部分儲存，所以會跟據目前 Addr2 存在哪一個(column)，而產生增加 1 個位移量或 0 個位移量。在實現中當搜尋方塊位於 Text 最後三個字元位置時判定目前實驗數據已完全處理完畢，將輸出訊號 ending 從低準位設定到高準位，此訊號為高準位時所有的運作就會停止，所有的值都會維持當前裝態。另外輸出訊號 ending 低準位的持續時間也可視作整個預先過濾器的整體運算時間。

5.1.4 可疑位址儲存記憶單元的產生與設計

此方塊為儲存 Text 中經前面功能偵測的嫌疑字串的位址，方塊的輸入訊號有四，一者為 r0 用來通知此方塊，第一個搜尋方塊找到嫌疑字串，二者用來通知第二搜尋方塊搜尋到嫌疑字串，三者為系統通知停止運作的 ending 訊號以及

嫌疑位址，由於 r0、r1 在不同的時脈偵測出來所以不會同時為 1，所以在進入此方塊後作邏輯 and 用來當作儲存嫌疑位址記憶體寫入的觸發訊號，也因為只需一組觸發訊號所以此處的記憶體使用單埠隨機記憶體，嫌疑位址為 Addr1 或 Addr2 的直接輸出，然而此位址為搜尋方塊的位址，若要記憶巡視窗的位置則需再扣掉 L-K，在本文中即為 6，扣掉之後再將其值儲存至記憶體中。在輸入訊號 ending 訊號為高準位時，將不再儲存輸入訊號，並停止記憶體位址的增加，由於記憶體初始位址值為 0，所以最後記憶體位址停止的值即為此流程找到的嫌疑字串的總數目。

5.2　管線推測預先過濾器 推測預先過濾器方塊圖

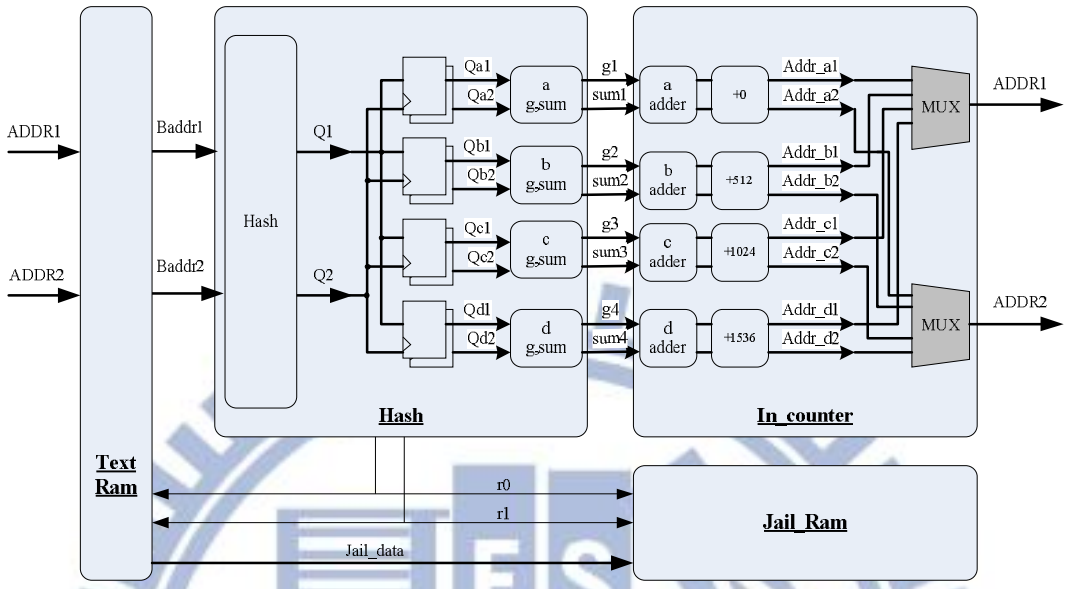


圖 5.9 Pipeline 推測預先過濾器方塊圖

在此實現中將輸入文字字串分為四個部分以 pipeline 的概念，在四個相位中分別處理四個部份的個別功能，在此情況下可以不增加記憶體的條件下產生搜尋可疑字串的功能，其方塊圖如圖 5.9 所示，目前使用到記憶體的部分為輸入文字字串記憶單元(Text Ram)、雜湊計算單元(Hash function)以及可一字串位址記憶單元(Jail Ram)，所以這三部分的輸入訊號和方塊功能與推測預先過濾器相同，而雜湊計算單元內與輸入位址計算單元為避免危障(hazard)產生而多出八組暫存記憶體。

5.2.1 管線推測預先過濾器 輸入字串記憶單元的產生與設計

輸入文字字串的儲存方式與推測預先過濾器相同，如圖 5.10 所示，為了同時取出四個字元的搜尋方塊資料所以以雙埠記憶體的架構，將輸入文字字串分為四個部份儲存。由於使用管線(Pipeline)架構時也將輸入文字字串分成四個部份

處理，在實體儲存輸入文字字串上並無異變，但在個別取出四個輸入文字字串的子集合的搜尋方塊資料時，會如圖 5.10 所示般存取不同的位置，其實際位址在輸入位址計算單元中產生。雖然資料存取架構與推測預先過濾器無異但是因為使用管線(pipeline)的緣故，使得資料存取速度較無管線設計的預先過濾器快上四倍。圖 5.11 為推測預先過濾器讀出搜尋方塊資料時的時脈圖，從圖中可顯示每四個時脈才會對雙埠記憶體存取一次。5.12 為管線推測預先過濾器讀取搜尋方塊資料的時脈圖，因為使用管線所以使得每個時脈都會有資料被讀取，其讀取會依照輸入文字字串子集合順序做讀出處理。四個輸入文字子集合處理偵測可疑字串的行為是獨立的所以在正常情況下有可能會發生其中某一子集合先行處理完後產生 ending 訊號，此時當前的搜尋視窗位址會維持在 ending 時的狀態直至所有子集合處理完成。

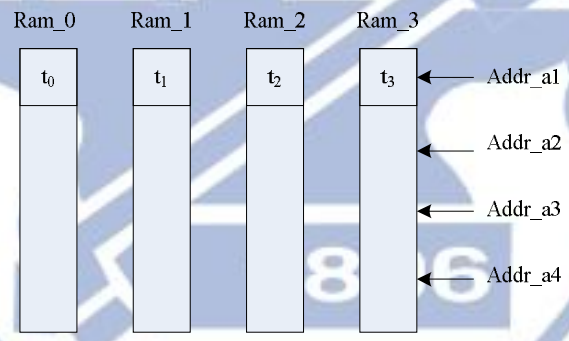


圖 5.10 管線推測預先過濾器輸入字串記憶單元

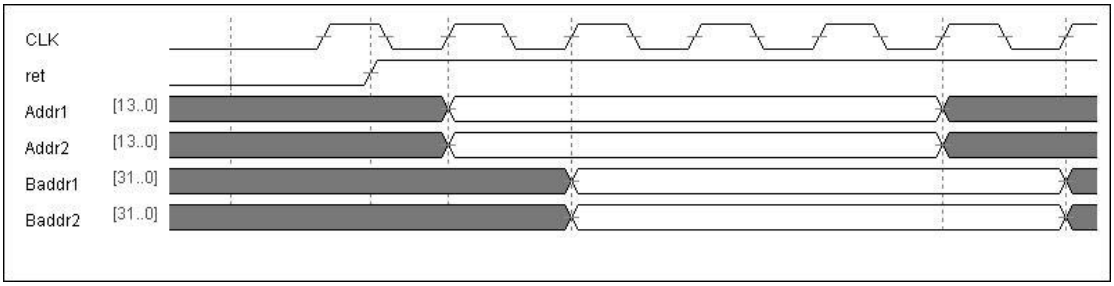


圖 5.11 推測預先過濾器輸入字串記憶單元的時脈圖

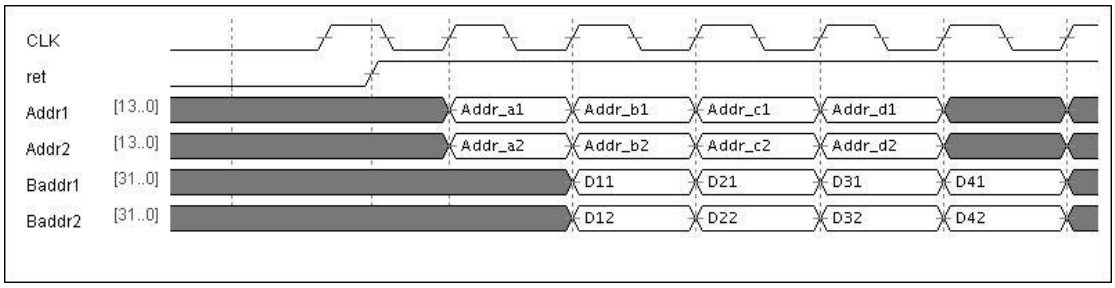


圖 5.12 管線推測預先過濾器推測預先過濾器輸入字串記憶單元的時脈圖

5.2.2 管線雜湊計算單元的產生與設計

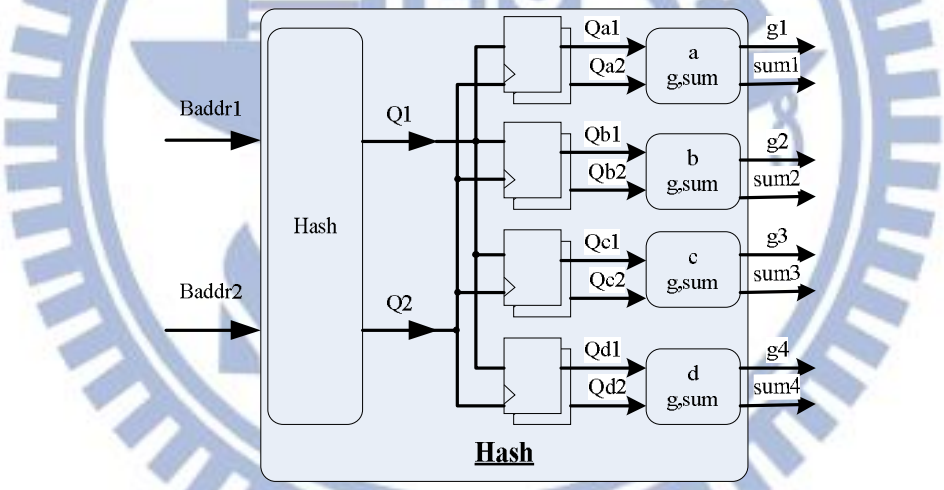


圖 5.13 管線推測預先過濾器雜湊計算單元方塊圖

雜湊計算單元分為兩個部份其一為詢問模塊結果產生單元，其二為計算位移量。雜湊值產生單元其中的雜湊表與推測預先過濾器無異使用相同的尋找字串。與前者不同的是每個時脈都會產生新的詢問模塊結果，每產生新的詢問模塊結果都會將其值儲存在對應的子集合所在的暫存記憶體分別為第一子集合的 Qa1、Qa2，第二子集合的 Qb1、Qb2，第三子集合的 Qc1、Qc2，第四子集合的 Qd1、Qd2。個別依照相同的位移準則產生個別子集的視窗位移量 g 與 sum，提供給輸入位址

計算單元產生個別的搜尋視窗位址。

5.2.3 管線預先過濾器 輸入位址計算單元的產生與設計

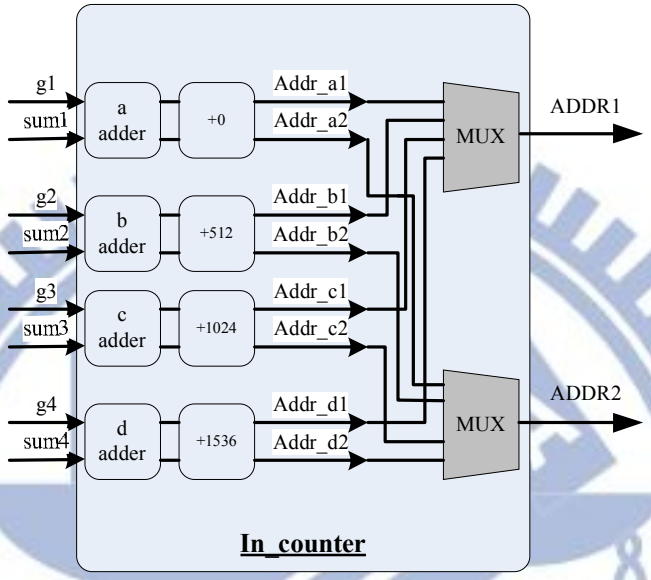


圖 5.14 管線推測預先過濾器位址產生器

在此方塊中存在四組位址產生器，四組產生器使用相同模組，因為四組位址只有在絕對位址上的差異，但是執行相同工作，所以將其在絕對位址上的差異移至外部使得四組產生器功能完全相同只須依據外部位移數據不同產生不同位址即可。四組位址在絕對位址上的範圍分別為位址 0~514，位址 512~1026，1024~1538 以即 1536~2050。位址產生器產生的範圍則為 0~514，將位址產生器之結果與絕對差值相加總即為在 Text 上的絕對位址，此作法不但模組化使程式簡單化也可減少各位址產生器的合成邏輯，因為若產生 1536~2050 的位址需 13 位元，但是產生 0~514 位址只需 11 個位元。最後一組的位址產生器在加入絕對差值之後會產生超出 Text 位址的最大值，所以在此會做偵測，若位址將要超過 Text 最大值時會將此方塊 endind 設為高準位以免誤動作。

依據 Pipeline 的 phase 分法，依據不同 phase 使用多工器選擇的方式選擇

目前需處理的位址，例如在 phase 4 時 Addr1 與 Addr2 分別為 Addr_a1 與 Addr_a2，phase1 時則分別為 Addr_b1 與 Adddr_b2，phase 2 時分別為 Addr_c1 與 Addr_c2，phase 3 時則分別為 Addr_d1 與 Addr_d2。

5.2.4 管線預先過濾器 可疑位址儲存記憶單元的產生與設計

在此方塊中有 r0 與 r1 訊號作為通知有嫌疑字串事件發生，在推測預先過濾器中 r0 與 r1 在不同 phase 產生，所以可將兩者做加總表示嫌疑發生的總和，但是在 pipeline 架構中若第一子集 Qa2 偵測到可疑字串產生 r1，而同一時間第四子集有可能因 Qd1 同時偵測到可疑字串使得 r0 同時被觸發，所以在管線(pipeline)架構中 r0 與 r1 訊號需各別處理。將儲存可疑位址的記憶體分為兩部份，第一部分儲存 r0 偵測到的可疑位址，第二部份儲存 r1 偵測到的可疑位置。可疑位址的總和為兩部份的數量總和。

第六章

實驗模擬與結果

6.1 實驗環境

此次實現使用XILINX Virtex4系列，此系列結合先進的矽片組合模塊架構使可以有許多靈活的應用，大大增強了可編譯邏輯設計能力，Virtex-4 系列使用90耐米12吋晶圓技術。其硬體包含數位時脈管理方塊，智慧的隨機記憶體，除固定的隨機記憶體外還提供離散的隨機記憶體，且可提供DDR與DDR2的記憶體高速介面，外部輸入輸出埠可提供1.5伏及3.3伏，內部以1.2伏低電壓做運作。

表 6.1 XILINX Virtex4 系列

Device	Configurable Logic Blocks (CLBs)[1]				XtremeDSP Slices[2]	Block RAM		DCMs	PMCDs	PowerPC Processor Blocks	Ethernet MACs	RocketIO Transceiver Blocks	Total I/O Banks	Max User I/O
	Array[3] Row x Col	Logic Cells	Slices	Max Distributed RAM (Kb)		18 Kb Blocks	Max Block RAM (Kb)							
XC4VLX15	64 x 24	13,824	6,144	96	32	48	864	4	0	N/A	N/A	N/A	9	320
XC4VLX25	96 x 28	24,192	10,752	168	48	72	1,296	8	4	N/A	N/A	N/A	11	448
XC4VLX40	128 x 36	41,472	18,432	288	64	96	1,728	8	4	N/A	N/A	N/A	13	640
XC4VLX60	128 x 52	59,904	26,624	416	64	160	2,880	8	4	N/A	N/A	N/A	13	640
XC4VLX80	160 x 56	80,640	35,840	560	80	200	3,600	12	8	N/A	N/A	N/A	15	768
XC4VLX100	192 x 64	110,592	49,152	768	96	240	4,320	12	8	N/A	N/A	N/A	17	960
XC4VLX160	192 x 88	152,064	67,584	1056	96	288	5,184	12	8	N/A	N/A	N/A	17	960
XC4VLX200	192 x 116	200,448	89,088	1392	96	336	6,048	12	8	N/A	N/A	N/A	17	960

此實現使用此系列的XC4VX15，包裝為SF363，速度為-12。此硬體提供13824個邏輯細胞，及96個離散隨機記憶體，48個大小為18K的方塊隨機記憶體及四個數位時脈管理方塊。其硬體資源的分割與計算方式以CLB個數計算，一個CLB包含四個Slices，一個Slices包含兩個四輸入的LUT、兩個儲存元件、兩個廣義功能的mux及溢位與算術邏輯，而LUT與儲存元件合併稱之為邏輯細胞。

程式寫作平台使用XILINX發展的XILINX ISE Design Suite 12.2，Synthesis為XILINX提供的XST，使用語言為Verilog。程式模擬平台為Active-HDL7.2。

6.2 實驗結果

實驗結果以完成狀態的程式以 Active-HDL7.2 程式進行模擬，以模擬結果時脈圖檢驗功能正確性。模擬分別對三個程式分別為”狀態預先過濾器”、”推測預先過濾器”以及”管線推測預先過濾器”，三種尋找字串方程做驗證。此系統只存在兩個輸入一為系統時脈，二為系統重置訊號，其他變數如輸入文字字串、尋找字串皆以相同內容視為輸入常數。三種預先過濾器會以各自程序尋找符合字串，因尋找字串以及輸入文字字串相同所以在驗證功能正確後即可比較三種預先過濾器在相同系統時脈下處理資訊的速度。另一方面因三種預先過濾器的硬體處理程序不同而產生硬體合成上的差異，再依照各自程式以其硬體可產生的最高系統時脈速度做模擬產生第二種比較結果，也就是通過量(Throughput)。若硬體可產生的系統時脈越高且相同資料量下處理時間越快就會產生越高的通過量。

6.2.1 “狀態預先過濾器”實驗模擬時序圖

系統時脈設定為 125MHz，模擬初始開始時間為 50ns，前置 50ns 為系統重置時間。此字串偵測系統運作至 70.596us 完成所有字串偵測。總共運算時間為 70.546us。如圖 6.1 所示。

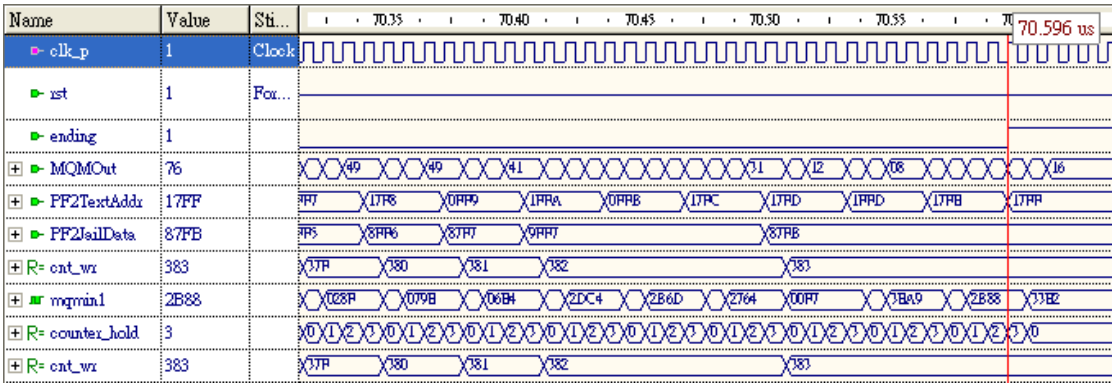


圖 6.1 “狀態預先過濾器”模擬時序結果圖一

系統經 implement 後得到系統運作的速度為 162MHz，以此速度重新模擬後得到運作時間為 54.477us，經計算後此系統的資料通過量為 1203M bits/s，如圖 6.2 所示。

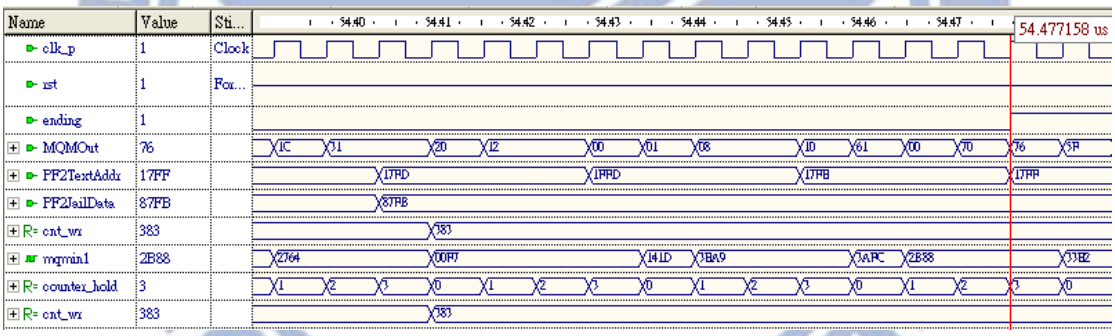


圖 6.2 “狀態預先過濾器”模擬時序結果圖二

6.2.2 “推測預先過濾器”實驗模擬時序圖

系統時脈設定為 125MHz，模擬初始開始時間為 50ns，前置 50ns 為系統重置時間。此字串偵測系統運作至 45.172us 完成所有字串偵測。總共運算時間為 45.122 us。如圖 6.3 所示。

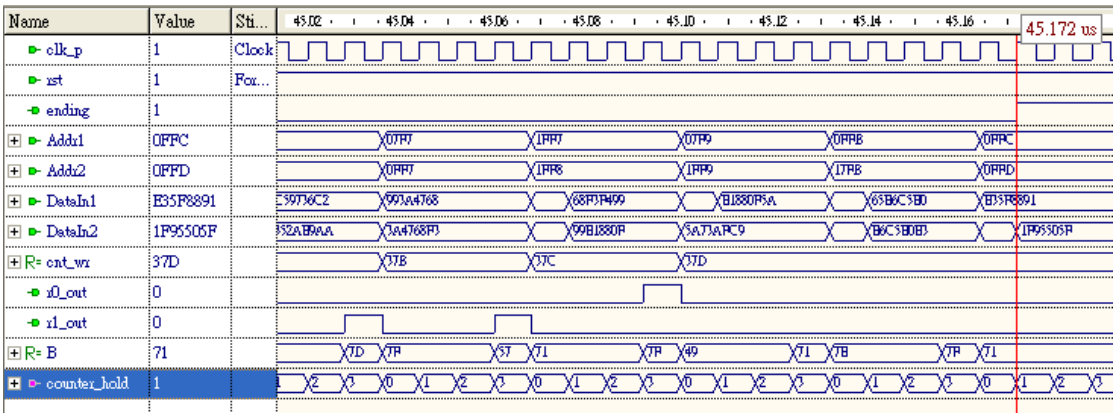


圖 6.3 “推測預先過濾器” 模擬時序結果圖一

系統經 implement 後得到系統運作的速度為 150MHz，以此速度重新模擬後得到運作時間為 37.652us，經計算後此系統的資料通過量為 1742Mbits/s，如圖 6.4 所示。

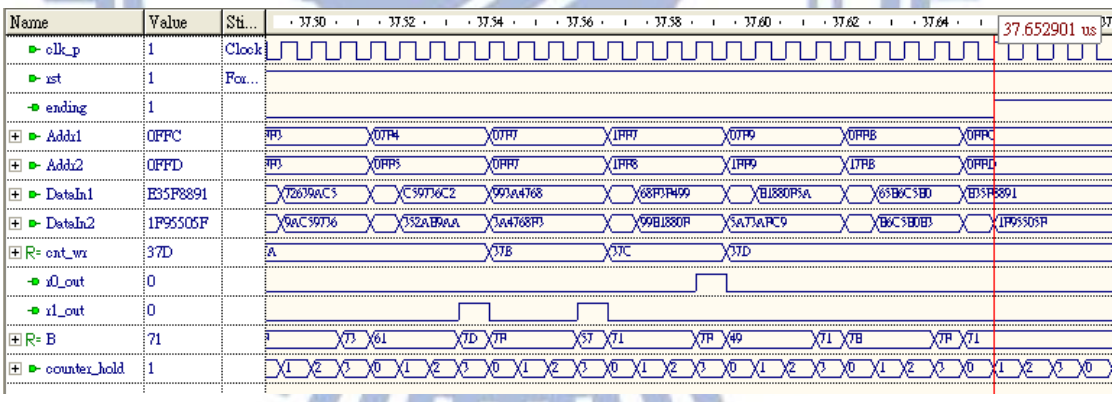


圖 6.4 “推測預先過濾器” 模擬時序結果圖二

6.2.3 “管線推測預先過濾器” 實驗模擬時序圖

系統時脈設定為 125MHz，模擬初始開始時間為 48ns，前置 48ns 為系統重置時間。此字串偵測系統運作至 14.26us 完成所有字串偵測。總共運算時間為

14.212us。如圖 6.5 所示。

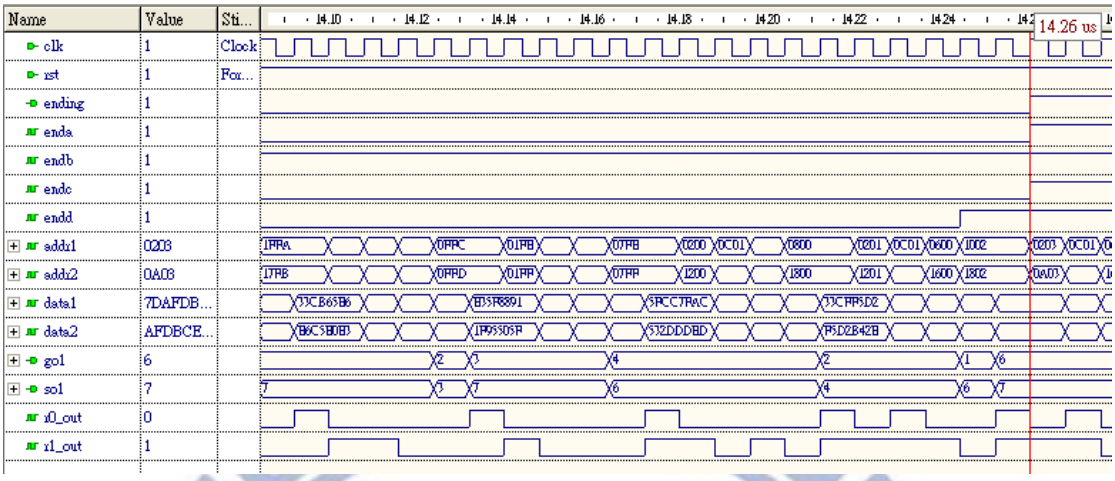


圖 6.5 “管線推測預先過濾器“模擬時序結果圖一

系統經 implement 後得到系統運作的速度為 179MHz，以此速度重新模擬後得到運作時間為 9.918us，經計算後此系統的資料通過量為 6639Mbits/s，如圖 6.6 所示。

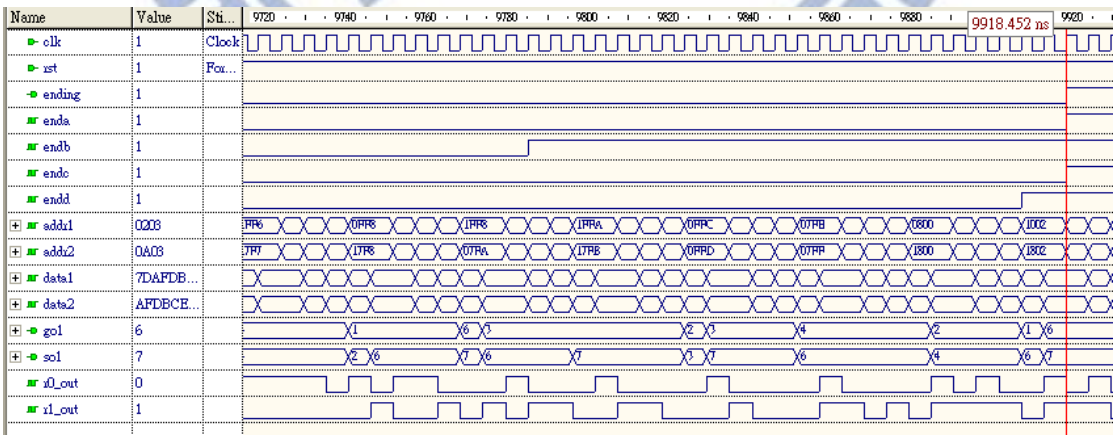


圖 6.6 “管線推測預先過濾器“模擬時序結果圖二

第七章

結論

在此篇論文中，以硬體FPGA分別實現了”狀態預先過濾器”與”推測預先過濾器”以及”管線推測預先過濾器”以期達到預期的高速處理偵測字串的目的。在驗證實驗中使用了”推測預先過濾器”中所有的詢問模塊，且利用雙埠記憶體的特性簡單的儲存輸入文字字串，而後進一步以管線(pipeline)方式增加”推測預先過濾器”的處理速度，雖然付出的些許暫存記憶體的代價，但是以實驗結果來看，佔用硬體空間最大的雙埠記憶體沒有因此增加，而整體運算速度卻增加了三倍之多，所以結果如預期般對偵測字串系統有相當大的幫助，實驗結果整理如表7-1與表7-2。

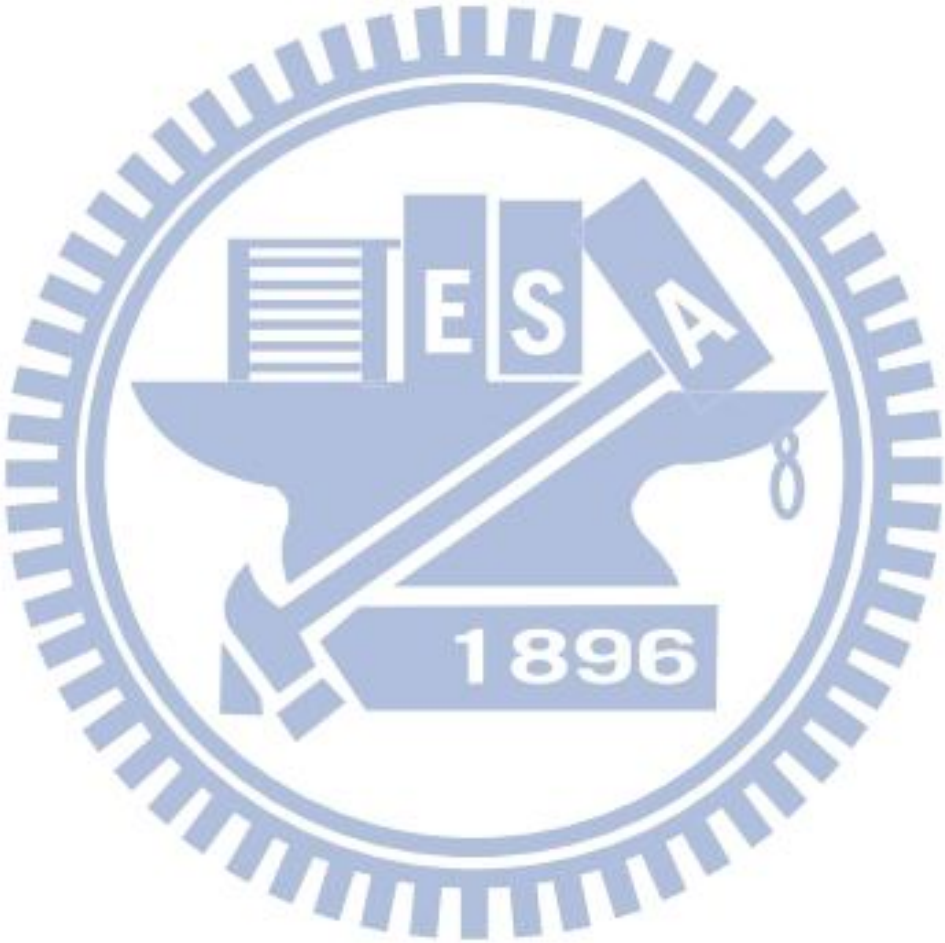
“推測預先過濾器”比”狀態預先過濾器”在處理相同大小的輸入文字字串中，速度將成長一點五倍。而”管線推測預先過濾器”則處理相同大小的輸入文字字串時速度將近五倍。經硬體實現後證明此架構的字串比對系統有助於處理更大量資料或傳輸速度更快的字串比對系統。

表 7-1 硬體空間比較表

	Number of Slice Flip Flops	*Number of 4-input LUTs*	*Number of FIFO16/RAMB16s*	*Clock Speed*	*Throughtput*
狀態預先過濾器	57	244	12	162M	1203bits/s
推測預先過濾器	74	362	12	150M	1742bits/s
管線推測預先過濾器	278	810	12	179M	6639bits/s

表 7-2 雙埠記憶體使用比較表

	Hushgen	*Jail_ram*	*Text*	*Total*
狀態預先過濾器	7	1	4	12
推測預先過濾器	7	1	4	12
管線推測推測預先過濾器	7	1	4	12



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. Wu and U. Manber, "A fast algorithm for multi-pattern searching," TR-94-17, 1994.

[6] P. C. Lin, Y. D. Lin, Y. C. Lai, Y. J. Zheng, and T. H. Lee, "Realizing a sub-linear time string-matching algorithm with a hardware accelerator using Bloom filters," IEEE Trans. VLSI Syst., vol. 17, no. 8, pp. 1008–1020, Aug. 2009.

[7] T. H. Lee and N. L. Huang, "A Pattern Matching Scheme with High Throughput Performance and Low Memory Requirement," submitted for publication.

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

[9] Virtex-II Pro and Virtex-II Pro X platform FPGAs: complete data sheet, Xilinx Inc., Oct. 2005

[10] "Barrel Shifter," Field Programmable Gate Array Application Note, Atmel Corp.

[11] P. Gigliotti, "Implementing Barrel Shifters Using Multipliers," XAPP195 (v1.1), Xilinx Inc., Aug. 17, 2004.

[12] Virtex-II Pro and Virtex-II Pro X platform FPGAs: complete data sheet, Xilinx Inc., Oct. 2005.

[13] 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.

[14] 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.

[15] 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.

附錄一

Implementationr Result List

- State-ful Pre-Filter

Design Summary

Number of errors: 0

Number of warnings: 27

Logic Utilization:

Number of Slice Flip Flops: 57 out of 12,288 1%

Number of 4 input LUTs: 244 out of 12,288 1%

Logic Distribution:

Number of occupied Slices: 143 out of 6,144 2%

Number of Slices containing only related logic: 143 out of 143 100%

Number of Slices containing unrelated logic: 0 out of 143 0%

*See NOTES below for an explanation of the effects of unrelated logic.

Total Number of 4 input LUTs: 262 out of 12,288 2%

Number used as logic: 244

Number used as a route-thru: 18

The Slice Logic Distribution report is not meaningful if the design is

over-mapped for a non-slice resource or if Placement fails.

Number of bonded IOBs: 19 out of 240 7%

Number of BUFG/BUFGCTRLs: 1 out of 32 3%

Number used as BUFGs: 1

Number of FIFO16/RAMB16s: 12 out of 48 25%

Number used as RAMB16s: 12

Average Fanout of Non-Clock Nets: 3.61

Peak Memory Usage: 231 MB

Total REAL time to MAP completion: 7 secs

Total CPU time to MAP completion: 4 secs

Utilization by Hierarchy

| Module | Slices | Slice Reg | LUTs | LUTRAM | BRAM/FIFO |

+----------+----------+----------+----------+----------+----------+

| System / | 13/165 | 2/57 | 22/262 | 0/0 | 0/12 |

| +hashgen | 9/24 | 5/5 | 14/75 | 0/0 | 7/7 |

| ++hash1 | 33/33 | 0/0 | 61/61 | 0/0 | 0/0 |

| +incontr | 37/37 | 28/28 | 52/52 | 0/0 | 0/0 |

| +jailram | 12/12 | 20/20 | 3/3 | 0/0 | 1/1 |

| +textram | 61/61 | 2/2 | 110/110 | 0/0 | 4/4 |

Timing summary

Timing errors:0 Score:{} (Setup/Max:0,Hold:0)

Design statistics:

Minimum periord: 6.136ns (Maximum frequency:162.972Mhz)

- Speculation Pre-Filter

Design Summary

Number of errors: 0

Number of warnings: 13

Logic Utilization:

Number of Slice Flip Flops: 74 out of 12,288 1%

Number of 4 input LUTs: 362 out of 12,288 2%

Logic Distribution:

Number of occupied Slices: 213 out of 6,144 3%

Number of Slices containing only related logic: 213 out of 213 100%

Number of Slices containing unrelated logic: 0 out of 213 0%

*See NOTES below for an explanation of the effects of unrelated logic.

Total Number of 4 input LUTs: 392 out of 12,288 3%

Number used as logic: 362

Number used as a route-thru: 30

The Slice Logic Distribution report is not meaningful if the design is

over-mapped for a non-slice resource or if Placement fails.

Number of bonded IOBs: 19 out of 240 7%

Number of BUFG/BUFGCTRLs: 1 out of 32 3%

Number used as BUFGs: 1

Number of FIFO16/RAMB16s: 12 out of 48 25%

Number used as RAMB16s: 12

Average Fanout of Non-Clock Nets: 3.32

Peak Memory Usage: 181 MB

Total REAL time to MAP completion: 9 secs

Total CPU time to MAP completion: 8 secs

Utilization by Hierarchy

| Module | Slices | Slice Reg | LUTs | LUTRAM | BRAM/FIFO |

System2/	24/245	2/74	35/392	0/0	0/12
+hashgen2	18/73	7/7	26/192	0/0	7/7
++hash1	29/29	0/0	54/54	0/0	0/0
++hash2	26/26	0/0	49/49	0/0	0/0
+incontr2	37/37	40/40	42/42	0/0	0/0
+jailram2	12/12	20/20	4/4	0/0	1/1
+textram2	99/99	4/4	182/182	0/0	4/4

Timing summary

Timing errors:0 Score:{} (Setup/Max:0,Hold:0)

Constrains cover 18008 paths, 0 nets, and 1666 connections

Design statistics:

Minimum periord: 6.665ns (Maximum frequency:150.038Mhz)

- Pipeline Speculation Pre-Filter

Design Summary

Number of errors: 0

Number of warnings: 1

Logic Utilization:

Number of Slice Flip Flops: 278 out of 12,288   2%

Number of 4 input LUTs: 810 out of 12,288   6%

Logic Distribution:

Number of occupied Slices: 515 out of   6,144   8%

Number of Slices containing only related logic: 515 out of   515 100%

Number of Slices containing unrelated logic: 0 out of   515   0%

*See NOTES below for an explanation of the effects of unrelated logic.

Total Number of 4 input LUTs: 908 out of 12,288   7%

Number used as logic: 810

Number used as a route-thru: 98

The Slice Logic Distribution report is not meaningful if the design is

over-mapped for a non-slice resource or if Placement fails.

Number of bonded IOBs: 19 out of   240   7%

Number of BUFG/BUFGCTRLs: 1 out of   32   3%

Number used as BUFGs: 1

Number of FIFO16/RAMB16s: 12 out of   48   25%

Number used as RAMB16s: 12

Average Fanout of Non-Clock Nets: 3.78

Peak Memory Usage: 183 MB

Total REAL time to MAP completion: 13 secs

Total CPU time to MAP completion: 9 secs

Utilization by Hierarchy

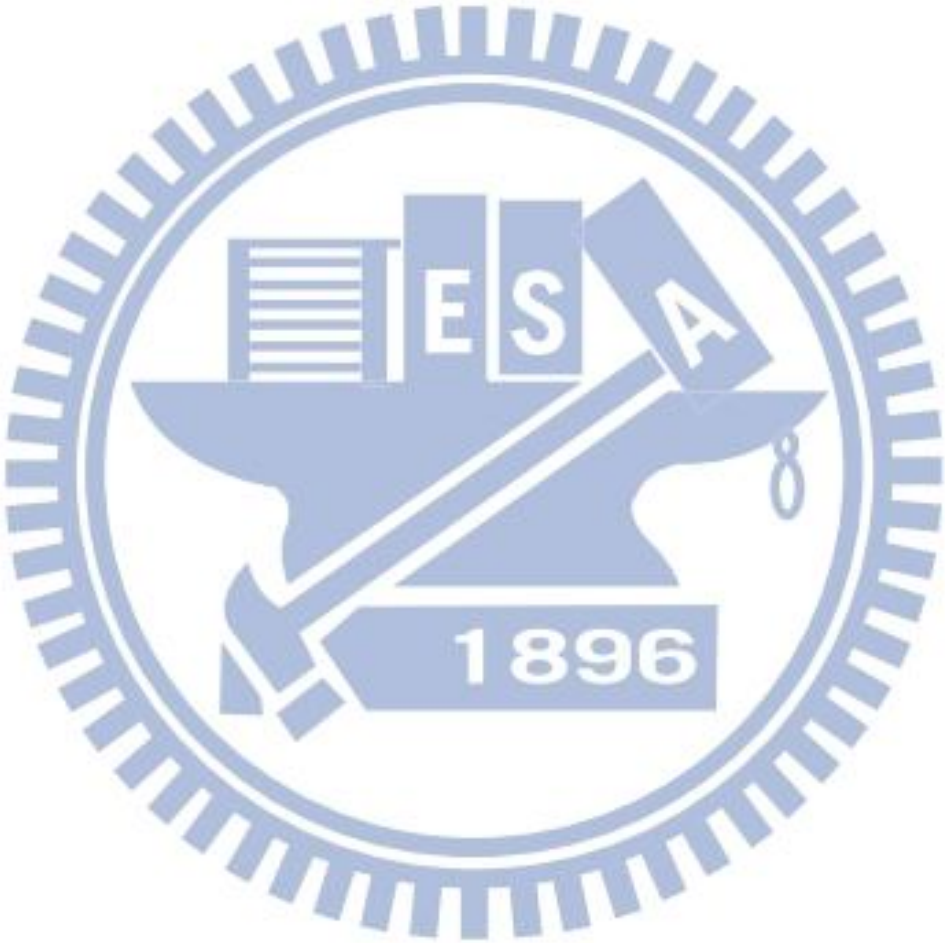
Module	Slices	Slice Reg	LUTs	LUTRAM	BRAM/FIFO
System3/	58/607	0/278	98/908	0/0	0/12
+jailram_3v	49/49	35/35	62/62	0/0	1/1
+hashgen_3v	91/142	76/76	95/192	0/0	7/7
++hash1	26/26	0/0	48/48	0/0	0/0
++hash2	25/25	0/0	49/49	0/0	0/0
+pipe_gen	204/358	7/167	372/556	0/0	4/4
++p5_incontr_3v	38/38	40/40	46/46	0/0	0/0
++p6_incontr_3v	38/38	40/40	46/46	0/0	0/0
++p7_incontr_3v	38/38	40/40	46/46	0/0	0/0
++p8_incontr_3v	38/38	40/40	46/46	0/0	0/0

Timing summary

Timing errors:0 Score:{} (Setup/Max:0,Hold:0)

Design statistics:

Minimum periord: 5.588ns (Maximum frequency:178.95Mhz)



附錄二

中英對照

B.

(block) 搜尋方塊

(block size) 方塊長度

C.

(character) 字元

D.

(data block) 資料方塊

F.

(failure function) 失效函數

G.

(goto graph) 轉移圖表

(goto function) 轉移函數

H.

(hash array) 雜湊陣列

(hash function) 雜湊函數

(hash table) 雜湊表

(hash value) 雜湊值

(hazard) 危障

M.

(Master Bitmap) 主位元組列

(Membership Query Module) 成員詢問模塊

P.

(pattern) 字串

(pattern matching machine) 關鍵字串比對模型

(Pipeline Speculation Pre-Filter) 管線推測預先過濾器

(pre-filter) 預先過濾器

(PREFIX table) 字根表

(Previous Query Result) 前級詢問結果

O.

(output function) 輸出函數

R.

(Rightmost Bit Detector) 最右位元偵測器

S.

(search window) 搜尋視窗

(SHIFT table) 位移表

(Speculation Pre-Filter) 推測預先過濾器

(Stateful pattern matching) 狀態字串比對

T.

(Text string) 輸入文字字串

(Throughput) 通過量

W.

(window length) 視窗長度

(window shift) 視窗位移