國立臺灣師範大學
資訊工程研究所碩士論文

指導教授：　黃　文　吉　博士

基於 RBF 實現紋理辨識之硬體架構

Radial Basis Function Hardware Architecture for Texture Classification

研究生：　范　哲　誠　撰

中華民國　一百零一　年　八　月

國立臺灣師範大學資訊工程研究所
博碩士學位論文

基於 RBF 實現紋理辨識之硬體架構
Radial Basis Function Hardware Architecture for Texture Calssification

經考試合格特此證明

審查教授：

尤信程

黃仲陵

指導教授：

系主任：李忠謀

中華民國　一百零一　年　六　月

中文摘要

本論文提出以 Recursive Least Mean Square 為基礎，結合 Fuzzy *c*-Means 分群演算法實作出 Radial Basis Function 類神經網路之紋理圖辨識系統。在本論文中，Fuzzy *c*-Means 計算紋理圖的質量中心點，Recursive Least Mean Square 計算類神經網中的權重係數，希望利用硬體的特性來實現快速運算、低資源消耗、低功率消耗以及擁有良好的效能之硬體架構。

最後我們所提出的硬體架構會在以 FPGA 為基礎的可程式化系統晶片設計(System On a Programmable Chip，SOPC)之平台上作實際的效能測試。根據使用不同的紋理圖作為測試資料，實驗結果顯示本架構對於紋理圖辨識有良好的分類正確率，且此硬體架構提供了日後高度的延伸性。

關鍵字:可程式化系統晶片、資料分群、FCM演算法、Recursive Least Mean Square、紋理圖辨識、系統程式晶片設計。

Abstract

This paper presents a real time RBF training hardware architecture for texture recognition which is based on recursive least mean square method and fuzzy c-means algorithm. We use fuzzy c-means algorithm to calculate centers in the hidden layer and use recursive least mean square method to estimate connecting weights in the output layer. Experimental results show that the proposed architecture is a effective hardware for real time training with low computational cost, low power consumption and high performance.

Keywords: *FPGA, data clustering, FCM algorithm, Recursive Least Mean Square, texture recognition, system on programmable chip.*

誌謝

首先，我要感謝 黃文吉 老師在這兩年間不辭辛勞地指導，無論是在專業技術或是待人處世上都獲益良多，我永遠不會忘記老師的諄諄教誨，讓我在這段學習的時間有很多的成長與體驗，並使我在人生的旅途中留下一段深刻又美好的回憶，老師 非常感謝您！

此外，感謝已畢業的學長姐：銘彥、士彰、斯閔、宗懋、侃翰及坤宏，還有其他許多的學長姐們，謝謝你們的教導，很榮幸能當你們的學弟。還有多媒體通訊暨系統晶片實驗室的同學偉豪、陳昊、嘉翎、子昕和浩聲，和其他實驗室的同學們，這兩年生活的所有點滴將成為我在碩士生涯的美好回憶。另外也感謝可愛的學弟們建廷、清志、國璿、聖穎、任軒和翰逸，每當我需要你們的時候伸出援手。很榮幸能來到國立臺灣師範大學，並在多媒體通訊暨系統晶片實驗室這個大家庭，和大家一起學習和成長。

最後，還要感謝我的父母，因為有父母的鼓勵和支持，才能讓我無後顧之憂的完成碩士學位，以及我的女友和好朋友們，在我為課業煩惱的時候，有你們陪我解悶，陪我度過研究上的難關，謝謝大家，在此將此論文獻給我最親愛的大家。

范哲誠 謹誌

台灣師大資訊工程研究所

中華民國一百零一年八月

目 錄

附圖目錄

附表目錄

第一章 緒論

本章節主要說明本論文的研究背景、動機以及目的。最後會簡略介紹各章節重要的內容架構。

1.1 研究背景

類神經網路(Artificial Neural Network 或譯為人工神經網路)是一種模擬大腦神經突觸連接的結構，它運用大量的神經元互相連接來進行資訊處理，並且模仿生物神經網路的方式。主要的運作方式為利用已知的輸入與輸出資料所組成的範例，建立系統模式(即輸入與輸出資料間的關係模式)，稱之為訓練過程，然後利用此系統模式從事學習、預測、決策、分類等工作，稱之為測試過程。

類神經網路架構由是輸入層、隱藏層、輸出層所構成，每一層的原點代表一個神經元。其中輸入層是輸入訓練資料或測試資料，隱藏層是把由輸入層得到的資料做運算，對於複雜的網路會有多個隱藏層，而輸出層是接收隱藏層的運算結果。除此之外，類神網路可以分為前饋式類神經網路(Feedforword Neural Network)和回饋式類神經網路(Feedback Neural Network)，所謂前饋式指的是連結方式為單一向前傳遞連結，輸出訊號只會向前，而回饋式至少會含有一回饋迴圈，神經元會各自將其輸出之訊號回傳給同一層中的其他神經元或前一層中的神經元。

幅狀基底函數網路(Radial Basis Function,或譯為半徑式函數網路)是屬於類神經網路中的前饋式類神經網路，其特點是可以用來解決高維度或非線性系統的數學函數，並且可以大量減少學習時間，以函數逼近的方式找出輸入和輸出之間的關係。因此，我們運用 RBF 類神經網路之架構建立紋理圖辨識系統。

Radial basis function (RBF)網路對於生活上的應用是非常有用的，原因在於它可以使用簡單的拓樸結構去映射複雜的非線性函數。一個基本的 RBF 網路架構由一層輸出層，一層有 nonlinear kernel 的隱藏層，和一層線性的輸出層所組成，而 nonlinear kernel 是使用 Gaussian function。

在 RBF 網路中，主要的計算目標在於 Gaussian kernel 中的質量中心點最佳化和神經元之間的權重最佳化，而計算方式是以 two-staged 學習策略來實行。在第一個 stage，質量中心的分析用來計算適當的質量中心點的值。在第二個 stage，監督式最佳化過程(supervised optimization procedures)用來最佳化 RBF 網路中連接輸出層的權重。

一個有效找尋質量中心點的分群方法是 *K*-means 演算法[6]。由於 *K*-means 演算法的分群結果對於初始質量中心點的選擇是很敏感的，如果選擇到不適當的初始質量中心點，*K*-means 的分群計算效果會不佳。除此之外，*K*-means 演算法對於大量的訓練向量集合之計算複雜度是很高的。而 Fuzzy *C*-means (FCM)演算法

和它的延伸[7, 11]對於尋找質量中心點是非常有效率的。FCM 對於分群採取模糊分割法，它允許訓練向量同時屬於多個質量中心點，並且與每個質量中心點產生不同的 membership coefficients。盡管 FCM 也是遞迴演算法，但 FCM 的分群效果對於初始質量中心點的選擇影響是較少的。然而模糊分群法包含 membership coefficients 的運算，隨著分群的數目變多，membership coefficients 的運算會需要大量的計算。

為了估計連接輸出層的權重，最小平方法(Least Mean Square, LMS)是常見被使用的技術，然而基礎的 LMS 方法在 RBF 網路的隱藏層中包含反矩陣運算。當隱藏層神經元的數目變多或訓練向量集合變大，反矩陣運算會成為艱鉅的任務，因此反矩陣運算的需求可以藉由 recursive LMS 來實行。但由於 recursive LMS 有較高的計算複雜度，尤其是對於大量的訓練集合與當隱藏層神經元數目變多時，仍需要大量的矩陣乘法運算。

1.2 研究動機

RBF 訓練已經有許多研究成果，在[12, 3, 10]中專注於減少質量中心點的訓練時間。在[12]中的演算法使用減法分群法(subtractive clustering)，在[3]中為修改基本的 *K*-means 演算法，在[10]中提到以逐步調整的方式來更新質量中心點。在[13]中提到以逐步調整的方式更新連接輸出層的權重，而這些所有快速的演算法皆以軟體實現，因此只可以顯示出適度的加速效果。除此之外，在[10, 13]提到的逐步調整的演算法，如果沒有適當的選擇學習速率(Learning rate)會嚴重地降低訓練的效果。

以類比電路實現 RBF 訓練過程[5, 9]已經發現可以有效的減少計算時間，然而這些架構很難直接使用在數位設備上。RBF 的數位硬體架構在[14]中只實現出 RBF 類神經網路的計算過程，而隱藏層中的質量中心點訓練和連接輸出層的權重訓練則以軟體的方式實現。以 RBF 為基礎的應用實現在嵌入式系統中[4, 8]也是只實現 RBF 類神經網路的計算過程。

在[2, 15]中提出 RBF 訓練的數位硬體架構，然而在[2]中並沒有包含質量中心點的訓練，權重訓練則是以逐步調整的方式為基礎。在[8]中提出質量中心點的訓練架構和權重的訓練架構，所有的訓練以逐步調整的方式運行。雖然逐步訓練適

合硬體實現，但其執行效果還是依靠學習速率(learning rate)的選擇而定。學習速率的值在有限經度的硬體實現中會被截短，而對於 RBF 訓練學習速率的截短會導致較差的區域最佳化，這樣的結果相似於選擇不適當的學習速率。

1.3 研究目的

本論文的目標是對於即時的 RBF 訓練呈現出一個新穎的硬體架構，此架構可分為兩部分：FCM 電路和 recursive LMS 電路。FCM 電路用來訓練隱藏層中的質量中心點，而 recursive LMS 電路用來訓練連接輸出層的權重。除此之外，FCM 電路和 recursive LMS 電路都是不需要學習速率的數位電路。

FCM 電路的特點是低記憶體消耗和高速計算。在此電路中，為了避免需要大量的儲存空間，更新 membership matrix 與更新群集質量中心點的運算過程合併成一個步驟。除此之外，FCM 以新穎的管線化(Pipeline)架構來加強訓練時的吞吐量。在 FCM 的計算過程中，更新過程分為三個步驟: pre-computation, membership coefficients updating, 和 center updating。Pre-computation 主要是計算不同的 membership coefficients 相同的部分，即為 membership coefficients 的分母，這個計算過程有助於降低 membership coefficients updating 的計算複雜度。Membership coefficients updating 以質量中心點集合與 pre-computation 的計算結果為基礎，計算新的 membership coefficients。Center updating 使用 membership coefficients updating 的計算結果來更新質量中心點，而新的質量中心點會在之後的 RBF 訓練過程中使用。

Recursive LMS 電路使用 FCM 電路更新的質量中心點來計算新的權重。由於 recursive LMS 演算法包含到大量的矩陣計算，我們對於平行乘法和平行加法提出有效率的區塊運算(block computation)來加快矩陣計算的速度。區塊的維度等於隱藏層的節點數目，以便於連接輸出層的權重可以同時更新。為了使區塊運算計算方便，我們設計暫存器(buffer)用來儲存 recursive LMS 演算法的中繼結果，並且可以水平移動或垂直移動，以便於矩陣的行和列可以輕易的存取。為了降低 area cost，所有矩陣運算分享相同的區塊運算電路。因此對於 recursive LMS，區塊運算電路有高速計算和低 area cost 的優點。

為了實現本論文所提出的架構之效果，紋理圖辨識系統建立在可程式化系統晶片(System on Programmable Chip, SOPC)的平台上。此系統由所提出的架構，NIOSS II 處理器[1]，DMA controller，和 SDRAM 所組成。此架構採取即時 RBF 訓練(online RBF training)並把訓練向量儲存在 SDRAM 上。DMA controller 用來傳送訓練向量，而軟核處理器(softcore processor)只用來協調 SOPC 系統，它並不參與 RBF 訓練過程。與此系統相對應的軟體在 Intel I5 CPU 執行比較後，我們所提出的系統對於大量的訓練集合可以大幅降低計算時間。

1.4 全文架構

本篇論文共分為五章，以下為各章的內容概述：

【第一章】緒論

說明本論文的研究背景、研究動機、研究目的和全文的架構。

【第二章】理論基礎與技術背景

簡介基本 RBF 類神經網路架構、fuzzy c-means 分群演算法、recursive LMS 的基礎理論與技術背景。

【第三章】RBF 硬體系統架構實現

詳細說明本論文所提出的 RBF 基礎電路，及其內部各單元架構。

【第四章】實驗結果與數據探討

呈現本論文所提出的電路設計成果數據、以及討論。

【第五章】結論

對本論文做最後的總結。

第二章 理論基礎與技術背景

本章將探討本論文所用到的理論基礎與技術背景。在本章的第一節會介紹 RBF 類神經網路架構，第二節會介紹 fuzzy *c*-means 分群演算法，第三節會介紹 recursive LMS 演算法，最後則是介紹 SOPC 系統整合設計，讓讀者對本論文能有初步的了解與認識。

2.1 RBF Networks

首先，這部分先說明 RBF 網路基本的架構。圖 2.1 顯示典型的 RBF 網路架構，由一層輸入層，一層隱藏層，一層輸出層組成。輸入層包含 *n* 個來源節點(source node)，其中 *n* 是輸入向量 **x** 的維度，而隱藏層由 *c* 個神經元組成，每個神經元伴隨著 kernel function。在 RBF 中使用的 kernel function 是 Gaussian kernel。假設 ϕ_i 代表第 *i* 個神經元的 Gaussian kernel，它的定義如下 ：

$$\phi_i(\mathbf{x}) = \exp\left(-\frac{1}{2\sigma^2}\left\|\mathbf{x}-\mathbf{v}_i\right\|^2\right) \tag{1}$$

$\mathbf{v}_i$ 在公式(1)中是連結第 *i* 神經元的質量中心點。**x** 和 $\mathbf{v}_i$ 都有相同的維度 *n*。σ^2 在公式(1)中稱為 Gaussian kernel 半徑。假設在這篇論文裡所有的 kernel 都有相同的半徑，輸出層只有一個神經元，讓 *y* 為輸出層的神經元，它的定義為:

$$y = \sum_{i=1}^{c} w_i \phi_i(\mathbf{x}) \tag{2}$$

w_i 稱為連接第 i 個隱藏層神經元和輸出神經元的權重。RBF 訓練通常包含質量中心 $\mathbf{v}_i$ 的訓練和權重 w_i 的訓練，其中 $i = 1,..., c$ 。

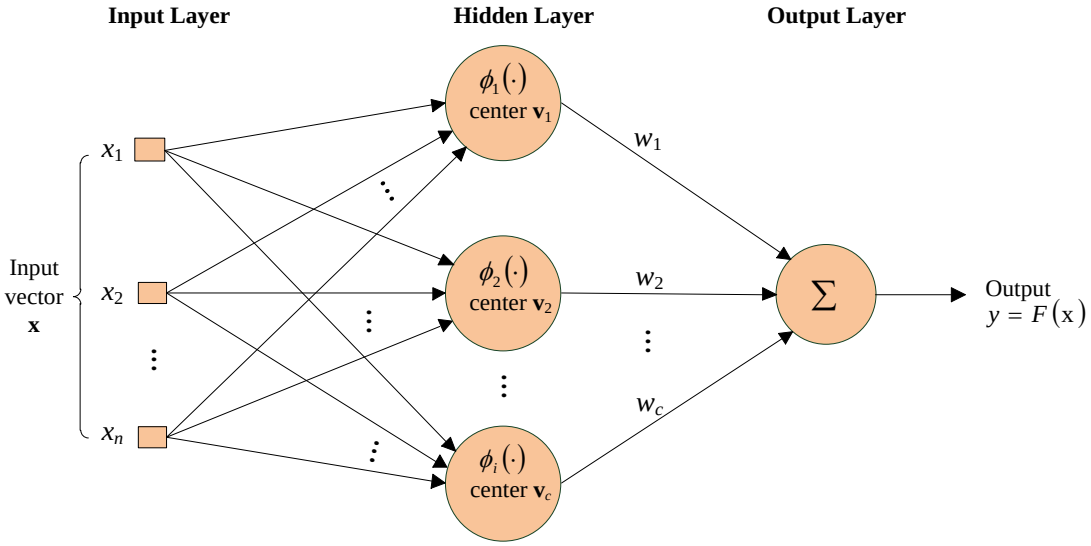

圖 2.1　RBF 網路架構圖

2.2 Fuzzy *C*-Means 演算法

FCM 可以有效地用來訓練質量中心點。假設 $X = \{\mathbf{x}_1,...,\mathbf{x}_t\}$是 RBF 訓練過程中的訓練向量集合，$t$ 是訓練向量的個數。FCM 藉由分割 $\boldsymbol{X}$ 成 c 群來計算 $\mathbf{v}_i$, i = 1,...,c，然後 $\mathbf{v}_i$ 為群集 i 的質心。而 FCM 需要最小化下列目標函式：

$$J = \sum_{i=1}^{c}\sum_{k=1}^{t} u_{i,k}^{m}\left\|\mathbf{x}_k - \mathbf{v}_i\right\|^2, \tag{3}$$

其中 $u_{i,k}$ 為 $\mathbf{x}_k$ 在類別 i 的 membership，$m > 1$ 指的是 FCM 的模糊程度。目標函式 J 在 FCM 中藉由 two-step 的遞迴步驟達最小化。在第一個步驟，質量中心點 $\mathbf{v}_1, \ldots,$ $\mathbf{v}_c$ 是固定的，並且 membership matrix $\left\{u_{i,k}, i = 1,..., \ c, k = 1,... \ t\right\}$ 由下列函式計算 ：

$$u_{i,k} = \left(\sum_{j=1}^{c}\left(\left\|\mathbf{x}_k - \mathbf{v}_i\right\| / \left\|\mathbf{x}_k - \mathbf{v}_j\right\|\right)^{2/(m-1)}\right)^{-1}. \tag{4}$$

在第一個步驟後，membership matrix 就會被固定，然後新的質量中心點 $\mathbf{v}_i$ 會以下列函式計算：

$$\mathbf{v}_i = \left(\sum_{k=1}^{t} u_{i,k}^{m}\mathbf{x}_k\right) / \left(\sum_{k=1}^{t} u_{i,k}^{m}\right) \tag{5}$$

FCM 遞迴運算會執行到 J 值收斂。從公式(3)和公式(5)得知，在目標函式和質量中心點的計算過程中，membership matrix 必須被儲存。因此當 t 和 c 的乘積越大，membership matrix 的大小也會提高，則 FCM 要求的儲存空間在硬體實作上會增加困難度。

2.3 Recursive Least Mean Square 演算法

權重的訓練也是以訓練向量集合 $\boldsymbol{X} = \{\mathbf{x}_1,...,\mathbf{x}_t\}$ 為基礎。假設輸入第 k 個訓練向量 $\mathbf{x}_k \in \boldsymbol{X}$，則 y_k 是 RBF 網路的輸出值。換句話說，從公式(2)：

$$y_k = \sum_{i=1}^{c} w_i \cdot \phi_i(\mathbf{x}_k)$$

定義：

$$\mathbf{a}_k = [\phi_1(\mathbf{x}_k)\phi_2(\mathbf{x}_k)... \phi_c(\mathbf{x}_k)]^T \tag{6}$$

$$\mathbf{w} = [w_1 w_2 ... w_c]^T \tag{7}$$

並且定義 y_k 為兩者相乘：

$$y_k = \mathbf{a}_k^T \mathbf{w} \tag{8}$$

除此之外，假設：

$$\mathbf{y} = [y_1, y_2 \cdots y_t]^T \tag{9}$$

是 $\boldsymbol{X}$ 中所有訓練向量相對應的 RBF 網路輸出，並且：

$$\mathbf{A} = \begin{bmatrix} \mathbf{a}_1^T \\ \mathbf{a}_2^T \\ \cdots \\ \mathbf{a}_t^T \end{bmatrix} \tag{10}$$

從公式(8)和(10)，可以知道：

$$\mathbf{y} = \mathbf{A}\mathbf{w} \ . \tag{11}$$

在[6]中可以知道 $\mathbf{w}$ 的 LMS 計算結果是：

$$\mathbf{w} = (\mathbf{A}^T \mathbf{A})^{-1} \mathbf{A}^T \mathbf{y} \ . \tag{12}$$

從公式(12)可以知道，計算 **w** 的過程包含反矩陣運算和矩陣乘法運算。當訓練向量的數量 t 或者質量中心點的數量 c 變大時，在硬體上執行 **w** 的 LMS 運算是困難的。因此，相對於 LMS 計算反矩陣的方法，一個有效的選擇是 recursive LMS，給定訓練向量集合 $\boldsymbol{X}$，運用 recursive LMS 做逐步計算，而不用公式(12)一次算完 **w** 矩陣。

假設依序輸入訓練向量 $\mathbf{x}_1,...,\mathbf{x}_{k-1}$，並且 $\mathbf{x}_1,...,\mathbf{x}_{k-1}$ 的相對應輸出 $y_1,...,y_{k-1}$，則定義：

$$\mathbf{y}_{k-1} = [y_1, y_2 \cdots y_{k-1}]^T \tag{13}$$

基於 $\mathbf{x}_1,...,\mathbf{x}_{k-1}$，矩陣 **A** 的前$(k-1)$列是可用計算獲得。則我們讓：

$$\mathbf{A}_{k-1} = \begin{bmatrix} \mathbf{a}_1^T \\ \mathbf{a}_2^T \\ \cdots \\ \mathbf{a}_{k-1}^T \end{bmatrix}, \tag{14}$$

是矩陣 **A** 的前$(k-1)$列。因此，基於前面定義的 $\mathbf{A}_{k-1}$ 和 $\mathbf{y}_{k-1}$，**w** 的 LMS 計算由公式(12)可以標記為 $\mathbf{w}_{k-1}$：

$$\mathbf{w}_{k-1} = (\mathbf{A}_{k-1}^T \mathbf{A}_{k-1})^{-1} \mathbf{A}_{k-1}^T \mathbf{y}_{k-1}.$$

假設一對新的資料$(\mathbf{x}_k, y_k)$為已知，則我們使用 recursive LMS 和 $\mathbf{w}_{k-1}$ 計算得到 $\mathbf{w}_k$，而不是用公式(12)重新計算 $\mathbf{w}_k$，定義：

$$\mathbf{P}_{k-1} = (\mathbf{A}_{k-1}^T \mathbf{A}_{k-1})^{-1} \tag{15}$$

然後 $\mathbf{P}_k$ 可以表示為：

$$\mathbf{P}_k = \mathbf{P}_{k-1} - \frac{\mathbf{P}_{k-1}\mathbf{a}_k\mathbf{a}_k^T\mathbf{P}_{k-1}}{1+\mathbf{a}_k^T\mathbf{P}_{k-1}\mathbf{a}_k} \tag{16}$$

並且 $\mathbf{w}_k$ 可以表示為:

$$\mathbf{w}_k = \mathbf{w}_{k-1} + \mathbf{P}_k\mathbf{a}_k(y_k - \mathbf{a}_k^T\mathbf{w}_{k-1}) \tag{17}$$

為了初始化演算法，設置：

$$\mathbf{P}_0 = \lambda^{-1}\mathbf{I}, \tag{18}$$

$$\mathbf{w}_0 = 0, \tag{19}$$

在這裡，$\mathbf{P}_0$ 的初始化為 λ 乘上一個單位矩陣，$\mathbf{w}_0$ 為零矩陣。其中，在這裡 λ 為一個小的正數。

2.4 SOPC 系統整合設計

進幾年來，隨著科技進步，電子產品功能日益複雜，但開發週期越來越短，如何在短時間內開發出良好的產品並且符合客戶需求是個重要的議題，為了在競爭的壓力下生存，必須採取更有效率的設計方法。以 Altera 提供的 SOPC 系統為實驗平台，燒錄在可程式化邏輯閘陣列(Field Programmable Gate Array, FPGA)。由於 FPGA 是一種可以重複改變組態的電路，有可以讓使用者進行編譯的邏輯閘元件，其邏輯閘特性，可依設計者的需要加以改變，並提供各種基本功能，因此完成的電路設計可以經過簡單的綜合與布局，快速的燒錄在 FPGA 上進行測試，適用於程式在開發時必須不斷變更設計的應用。因此有效加速程式完成的時間，提升整體的競爭力，大大的降低設計的時間與成本。

根據不同使用者的需求，Altera 公司開發出許多不同系列的 FPGA 開發板，而本研究是在 NIOS development kit 中的 Cyclone III EP3C120 系統開發板上實現我們所提出的硬體電路。在 NIOS 系統中提供了一套專門給 NIOS 處理器使用的匯流排(Avalon bus)，讓使用者設計的電路視為一個客製化電路(custom logic circuit)，透過此匯流排上的各個訊號線，將電路掛載在 Avalon bus 上與整個系統溝通，如圖 2.2 所示。除此之外，Cyclone III 開發版上也提供 DDRII SDRAM、Flash memory、I/O 裝置等，供開發人員使用。而整個系統的設計流程如圖 2.3 所示。

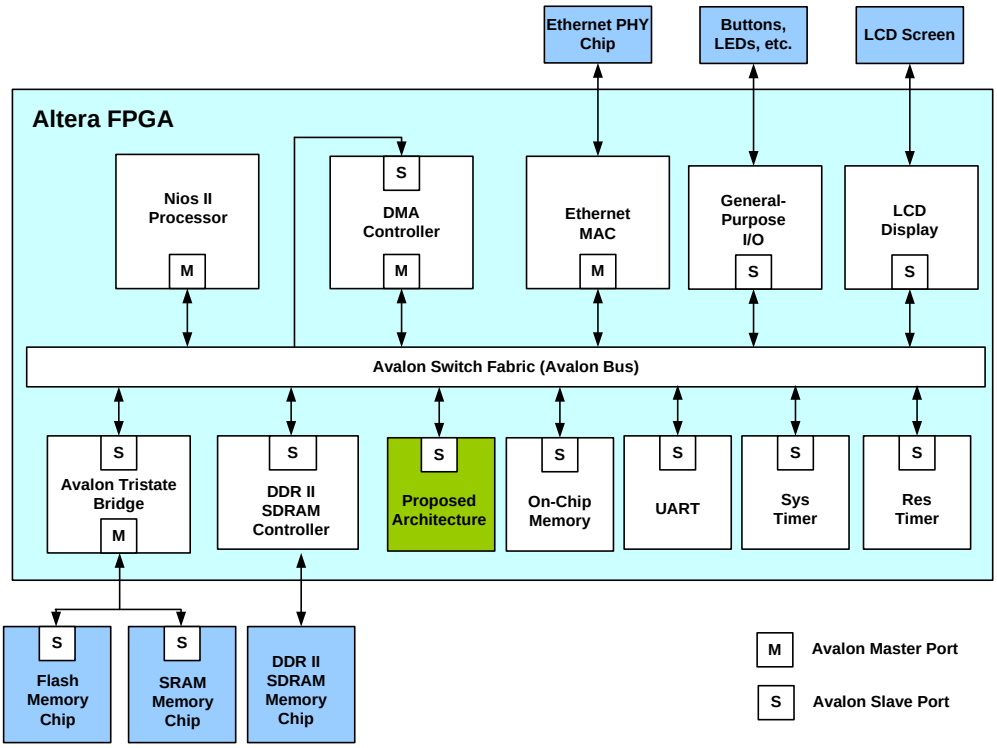

圖 2.2　SOPC 系統架構圖

NIOS 系統擁有下列優點：

1. Altera 公司所提供的 NIOSⅡ IDE 是視窗介面的軟體開發工具，設計者可以在上面編寫程式碼、編譯、除錯及觀察程式執行結果。此開發工具除了包含 C 語言的函式庫外，還包含 HAL(hardware abstraction layer)函式庫。其中 HAL 函式庫提供設計者一個呼叫系統相關裝置的 API(Application Program Interface)，如圖 2.3 所示，所以設計者可以撰寫 C 語言程式並透過 HAL API 呼叫來讓特定的裝置運作。

2. 提供 DMA(direct memory access)機制讓設計者可加快資料在記憶體與系統周邊元件的傳輸速度且不佔用 CPU 資源。

3. 開發人員可依自己的需求增減電路的功能，製作一個專屬的系統。如需增減裝置時，只要重新 build 系統並 compile 成硬體檔之後燒入板子即可。具有彈性大與快速建置的優勢。

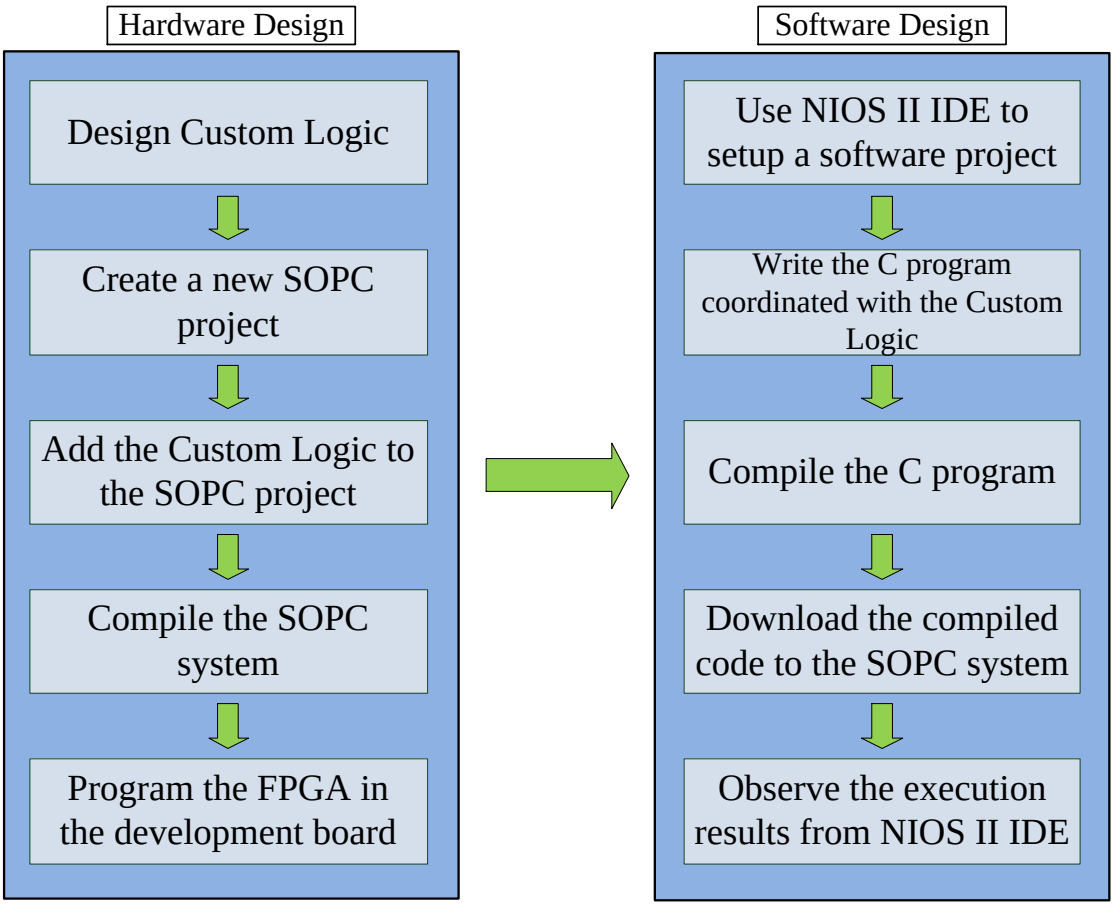

圖 2.3　軟硬體共同設計圖

第三章 RBF 硬體系統架構實現

在本章節中，我們將介紹本論文所提出的 RBF 硬體系統架構，並且將詳細介紹各單位元件電路以及如何實現快速運算與達到低硬體資源消耗的相關過程。

3.1 簡介

如圖 3.1，RBF 訓練可以分為兩個元件：FCM Unit 和 recursive LMS Unit。FCM Unit 的運算目標是給予訓練向量集合 $\boldsymbol{X}$，計算出新的質量中心點 $\mathbf{v}_i$, $i=1,\ldots,c$。而 recursive LMS Unit 的運算目標則是以 FCM Unit 計算出來的質量中心點和訓練向量集合 $\boldsymbol{X}$ 為基礎，找出權重 w_i, $i=1,\ldots,c$。

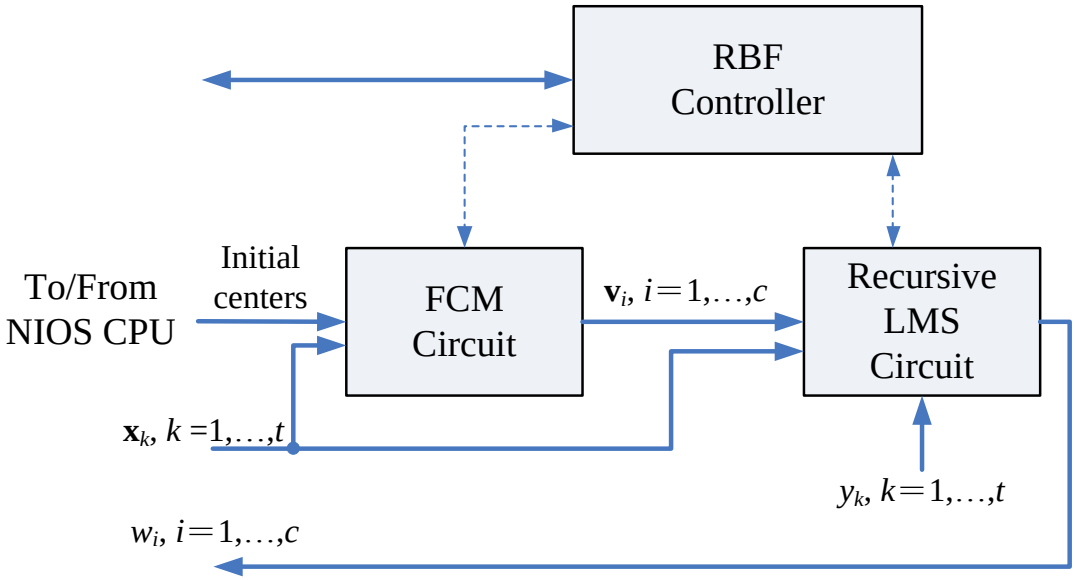

圖 3.1 RBF 訓練硬體電路架構圖

3.2 FCM unit

圖 3.2 顯示 FCM Unit 的架構，包含六個子元件：pre-computation unit、membership coefficients updating unit、center updating unit、cost function computation unit、FCM memory unit 和 control unit。接下來的小節將會詳細介紹每個元件的內部電路操作。

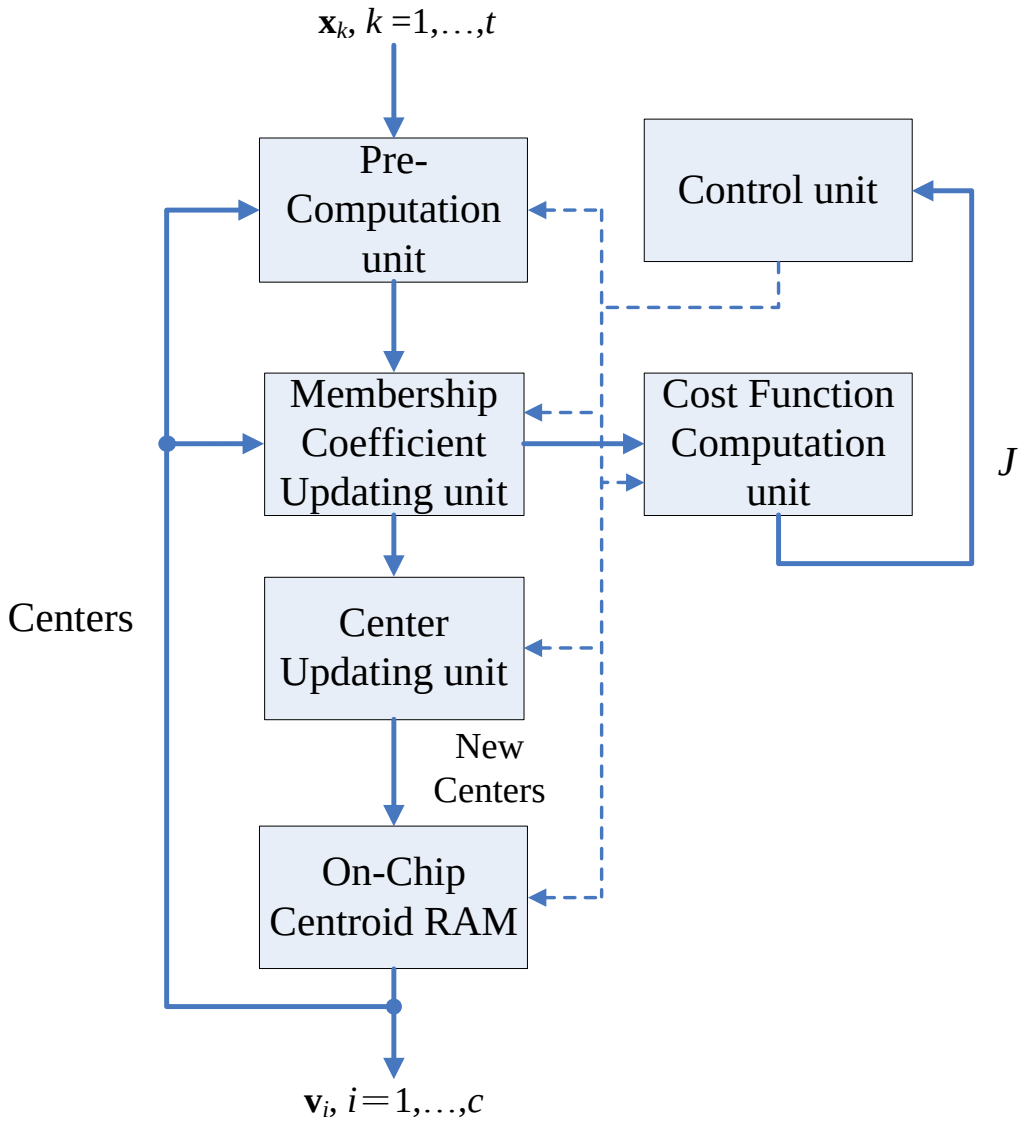

圖 3.2 FCM Unit 硬體電路架構圖

3.2.1 Pre-computation unit

Pre-computation unit 的目的是用來降低 membership coefficients updating 的計算複雜度。如公式(4)所示，$u_{i,k}$ 可以改寫成：

$$u_{i,k} = \left\|\mathbf{x}_k - \mathbf{v}_i\right\|^{-2/(m-1)} R_k^{-1}, \tag{20}$$

其中 R_k 為：

$$R_k = \sum_{j=1}^{c} (1/\left\|\mathbf{x}_k - \mathbf{v}_j\right\|^2)^{1/(m-1)} . \tag{21}$$

給定訓練向量 $\mathbf{x}_k$ 和質量中心點 $\mathbf{v}_1,\ldots,\mathbf{v}_c$，membership coefficients $u_{1,k},\ldots,u_{c,k}$ 有相同的 R_k。因此，藉由在 Pre-computation unit 計算 R_k，再傳遞至 membership coefficients updating unit 時可以降低在計算 membership coefficients 時的複雜度。為了簡化，在設計過程中我們預設 $m=2$，因此 R_k 可以被視為 $1/\left\|\mathbf{x}_k - \mathbf{v}_j\right\|^2$ 的總和。

計算 R_k 的架構顯示在圖 3.3，此圖片可以分為兩個階段。第一個 stage 計算 $\left\|\mathbf{x}_k - \mathbf{v}_i\right\|^2$，第二個 stage 的第一部分計算 $\left\|\mathbf{x}_k - \mathbf{v}_i\right\|^2$ 的倒數，第二部分累加第一部分的結果成為 $\sum_{j=1}^{i-1} 1/\left\|\mathbf{x}_k - \mathbf{v}_j\right\|^2$。

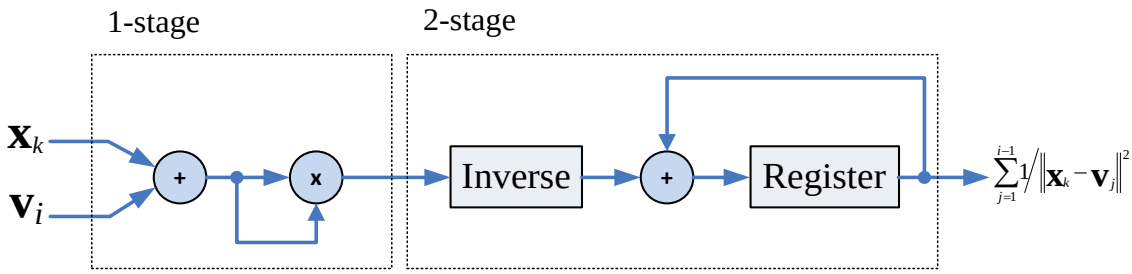

圖 3.3 Pre-computation Unit 硬體電路架構圖

3.2.2 Membership updating unit

根據公式(20)，Membership updating unit 使用 Pre-computation unit 的計算結果來計算 membership coefficients。圖 3.4 顯示 Membership coefficients updating unit 的架構。從圖 3.4 可以觀察出，給定訓練向量 $\mathbf{x}_k$，membership coefficients computation unit 計算 $u_{i,k}^2$，$i=1,\ldots,c$，一次計算一個。這部分的電路可以分為兩個 stage，第一個 stage 和第二個 stage 都是管線化的架構，第一個 stage 計算 $\|\mathbf{x}_k - \mathbf{v}_i\|^2 R_k$，第二個 stage 計算 $u_{i,k}^2$。

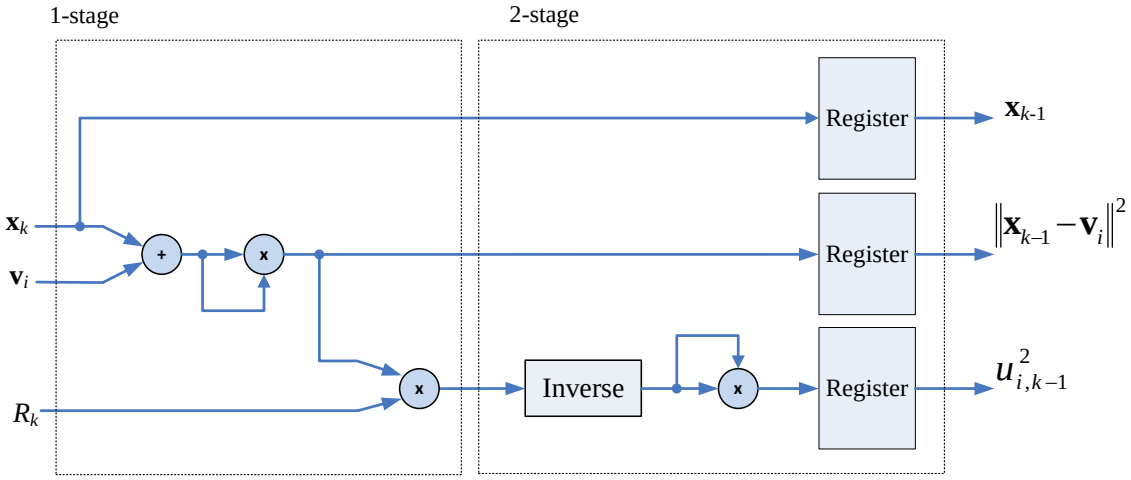

圖 3.4　Membership updating unit 硬體電路架構圖

3.2.3 Center updating unit

Center updating unit 逐步計算每個群集的質量中心點。逐步計算的優點在於質量中心點的計算不需要儲存所有的 membership coefficients。我們定義第 i 個群集逐步計算至 $\mathbf{x}_k$：

$$\mathbf{v}_i(k) = (\sum_{n=1}^{k} u_{i,n}^{m}\mathbf{x}_n)/(\sum_{n=1}^{k} u_{i,n}^{m}). \tag{22}$$

當 $k=t$ 時，$\mathbf{v}_i(k)$則與公式(5)中的 $\mathbf{v}_i$ 是相同的。

Center updating unit 的架構如圖 3.5 所示，它包含一個乘法器，一個 Intermediate on-chip RAM 和一個除法器。圖 3.6 顯示 Intermediate on-chip RAM 的架構，有兩組 ACC(Accumulator)在 array 中。第一組中的第 i 個 ACC 儲存 $\sum_{j=1}^{k-1}\mathbf{x}_j u_{i,j}^2$ 累加的結果，第二組中的第 i 個 ACC 儲存 $\sum_{j=1}^{k-1}u_{i,j}^2$ 累加的結果。如圖 3.6 所示，array 的 output 用來計算 $\mathbf{v}_i(k)$。

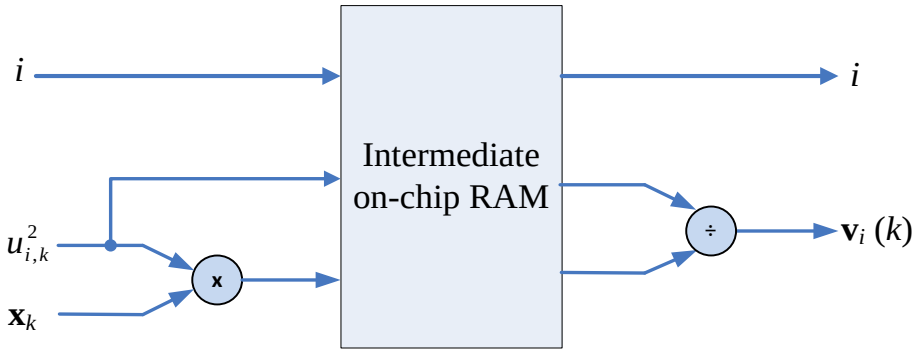

圖 3.5 Center updating unit 硬體電路架構圖(一)

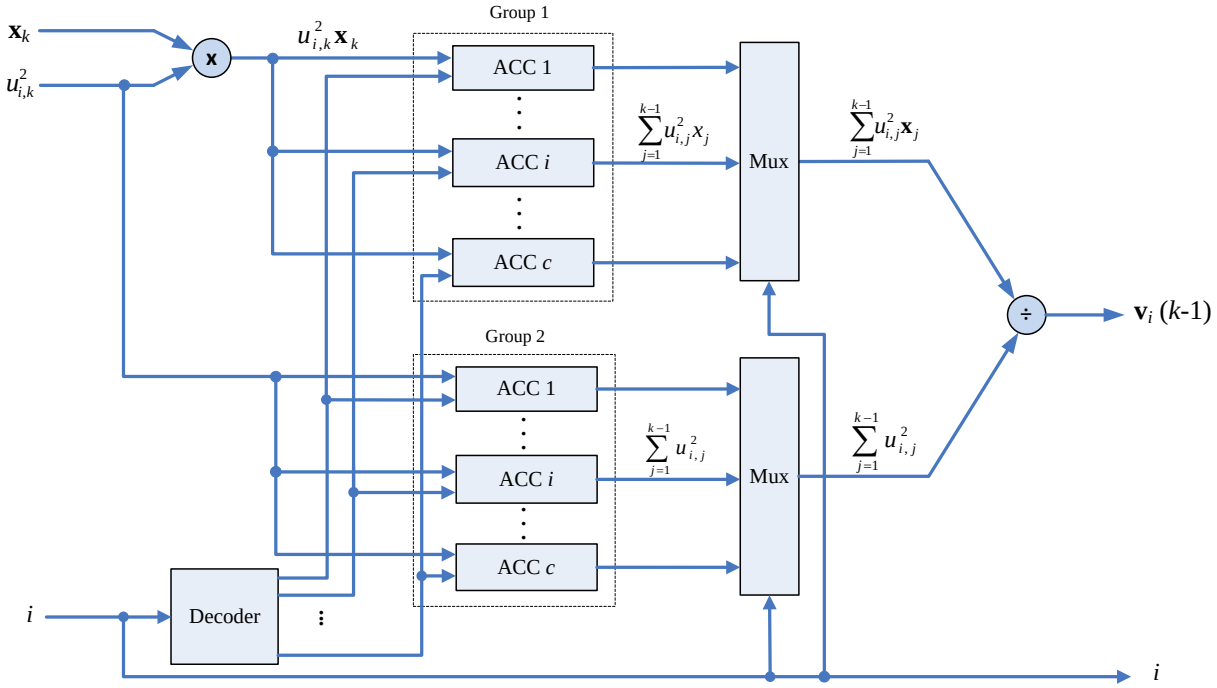

圖 3.6 Center updating unit 硬體電路架構圖(二)

3.2.4 Cost function computation unit

計算方式相似於 Center updating unit，Cost function computation unit 以逐步計算的方式來計算 cost function J。定義累加的 cost function $J(i, k)$為：

$$J(i,k) = \sum_{z=1}^{k} \sum_{j=1}^{i} u_{j,z}^{2} \left\| \mathbf{x}_z - \mathbf{v}_j \right\|^2 . \tag{23}$$

如圖 3.7 所示，此電路從 membership coefficients updating unit 收到$u_{i,k}^2$和 $\left\| \mathbf{x}_k - \mathbf{v}_i \right\|^2$ 之後，從公式(23)中可以知道乘積 $u_{i,k}^2 \left\| \mathbf{x}_k - \mathbf{v}_i \right\|^2$ 開始累加計算 $J(i, k)$。

當 $i=c$ 和 $k=t$ 時，$J(i,k)$就是最後的計算結果，也就是與公式(3)中的 cost function J 是相同的。因此，當所有的訓練向量傳遞完畢時，此元件的輸出即為 J 值。

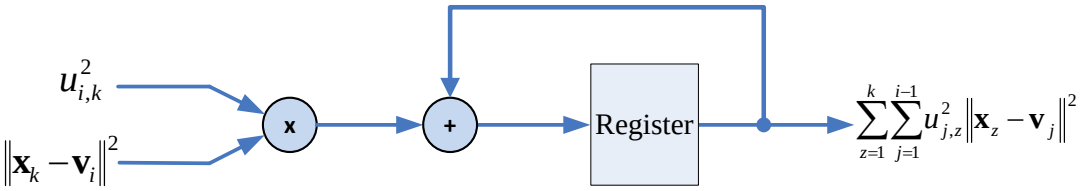

圖 3.7　Cost function computation unit 硬體電路架構圖

3.2.5 FCM Memory Unit

這個單元用來儲存 FCM 群集的質量中心點。在 on-chip centroid memory unit 中分為兩個區塊，即 Memory Bank 1 和 Memory Bank 2。Memory Bank 1 儲存目前使用的質量中心點 $\mathbf{v}_1,\ldots\mathbf{v}_c$，Memory Bank 2 儲存的是由 center updating unit 更新的質量中心點。為了計算出 membership coefficients，只有儲存在 Memory Bank 1 的質量中心點會被傳遞到 pre-computation unit 和 membership updating unit。來自 center updating unit 更新過的質量中心點會儲存在 Memory Bank 2。特別注意的是，Memory Bank 2 中的質量中心點在所有的訓練向量 $\mathbf{x}_k$, $k = 1,\ldots,t$，被傳遞結束前不會被更新取代到 Memory Bank 1 的質量中心點。

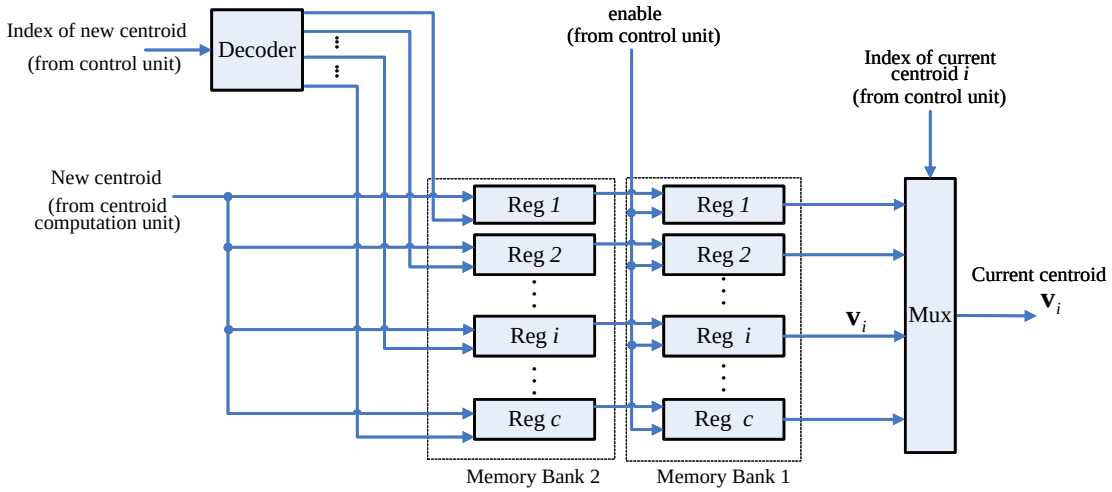

圖 3.8　FCM memory unit 硬體電路架構圖

3.3 Recursive LMS Unit

Recursive LMS 元件架構如圖 3.9，由 kernel Gaussian computation unit、memory unit、matrix computation unit 和 control unit 組成。

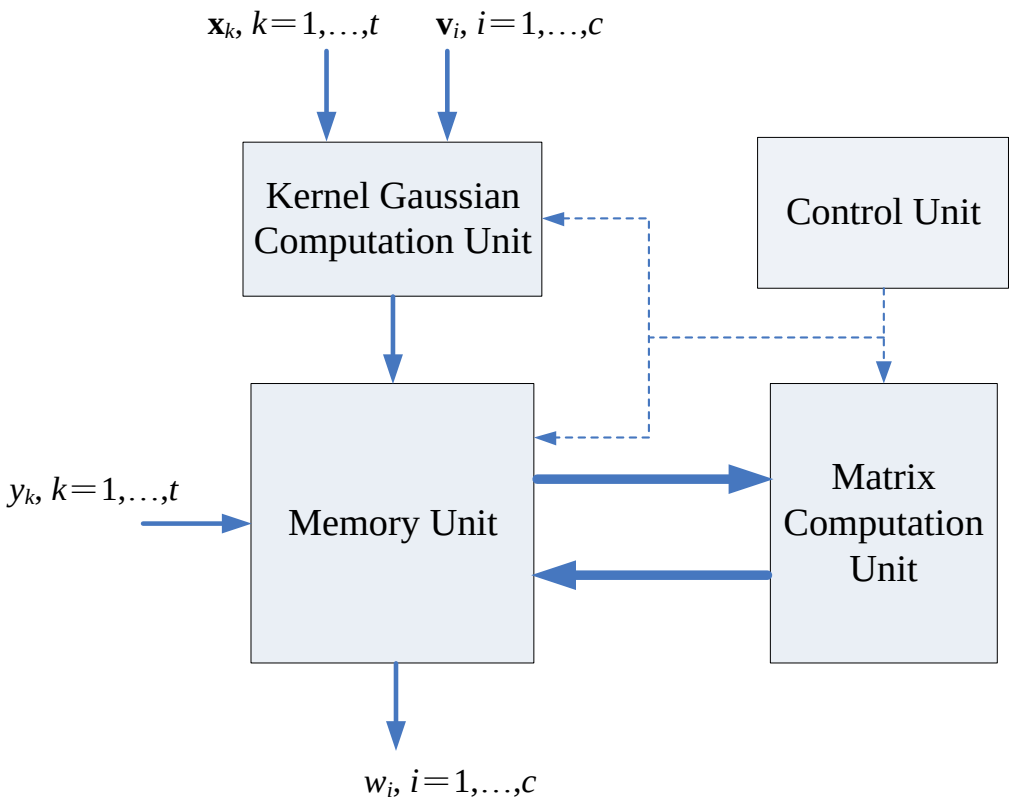

圖 3.9　Recursive LMS 硬體電路架構圖

3.3.1 Kernel Gaussian Computation Unit

Kernel Gaussian computation unit 的目的在於計算由公式(1)中定義的 $\phi_i(\mathbf{x})$。給定 $\mathbf{x}_k$ 和 $\mathbf{v}_1,\ldots\mathbf{v}_c$，kernel Gaussian 計算 $\phi_1(\mathbf{x}_k),\ldots, \phi_c(\mathbf{x}_k)$，並且以順序的方式產生 $\mathbf{a}_k$。圖 3.10 顯示 kernel Gaussian computation unit 的架構。架構中除了加法器和乘法器，

還包含了計算指數函數(exponential function)的電路，此電路由 Altera Floating Point Exponent (ALTFP_EXP) Megafunction 來實現。

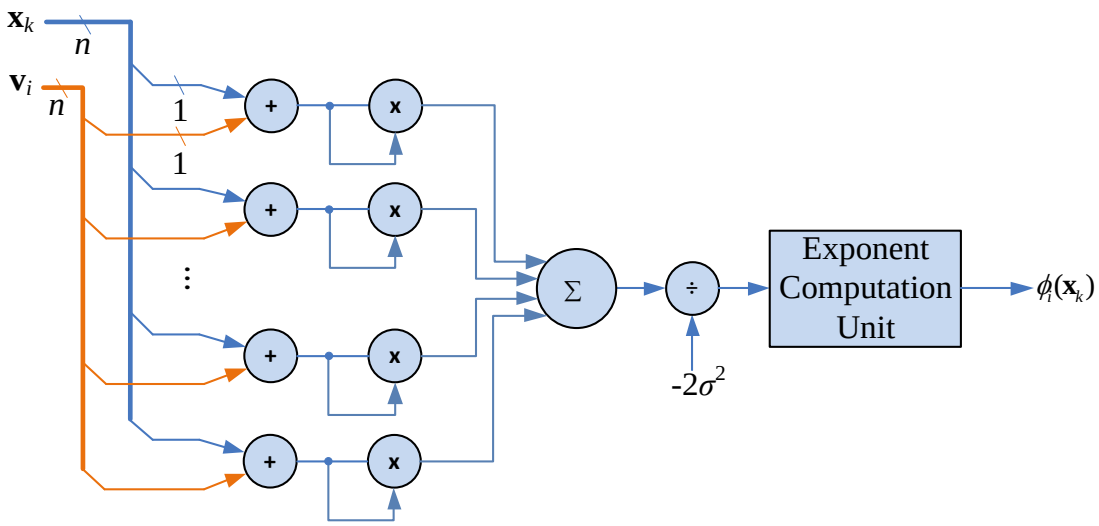

圖 3.10 kernel Gaussian computation unit 硬體電路架構圖

3.3.2 Memory Unit

Memory unit 用來儲存在公式(16)(17)中顯示的 recursive LMS 計算過程中所需要的值。如圖 3.17 所示，Memory unit 有 8 個 buffers(Buffers A, Y, P, W, D, G, H, 和 S)。假設 $\mathbf{x}_k$ 是目前計算中的訓練向量，Buffer A 儲存來自 kernel Gaussian computation unit 的計算結果 $\mathbf{a}_k$，Buffer Y 儲存 y_k。Buffer P 和 Buffer W 儲存由先前輸入訓練向量的計算結果 $\mathbf{P}_{k-1}$ 和 $\mathbf{w}_{k-1}$。Matrix computation unit 以 $\mathbf{a}_k$，y_k，$\mathbf{P}_{k-1}$ 和 $\mathbf{w}_{k-1}$ 為基礎，計算新的 $\mathbf{P}_k$ 和 $\mathbf{w}_k$。而計算過程中的中繼結果儲存在 Buffers T,G,H 和 S。為了下個訓練向量 $\mathbf{x}_{k-1}$ 的之後操作，$\mathbf{P}_k$ 和 $\mathbf{w}_k$ 儲存在 Buffer P 和 Buffer W。另外，Buffer T 和 Buffer Y 由一個暫存器所組成，Buffer W, H, G 與 Buffer A 相

同，由 c 個暫存器組成。Buffer P 和 Buffer S 由 c^2 個暫存器組成。其中，Buffer H 和 Buffer S 可以給予不同的訊號使之向右位移或向左循環，而 Buffer P 除了可以向右位移向左循環之外，也可以向下位移或向上循環。

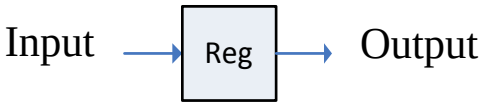

圖 3.11　Buffer T 和 Buffer Y 硬體架構圖

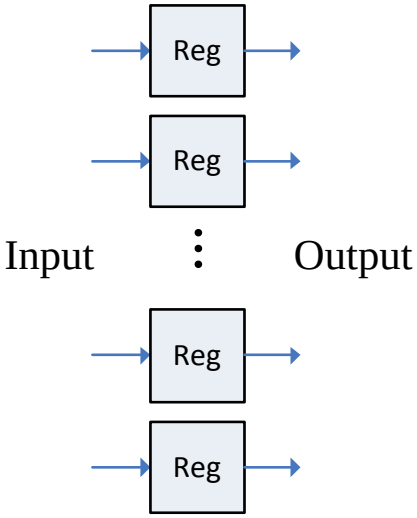

圖 3.12　Buffer W 和 Buffer A 硬體架構圖

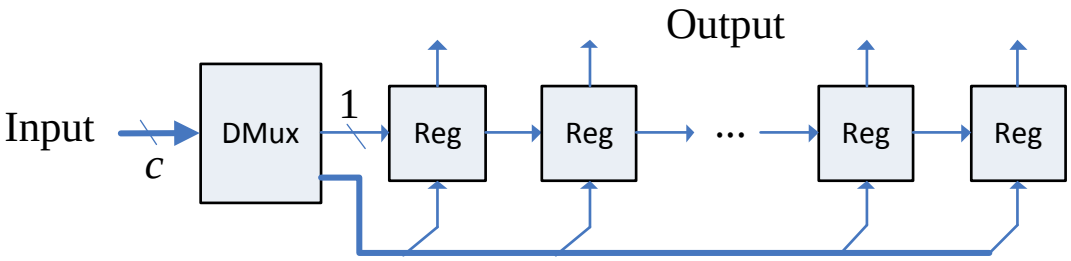

圖 3.13　Buffer G 硬體架構圖

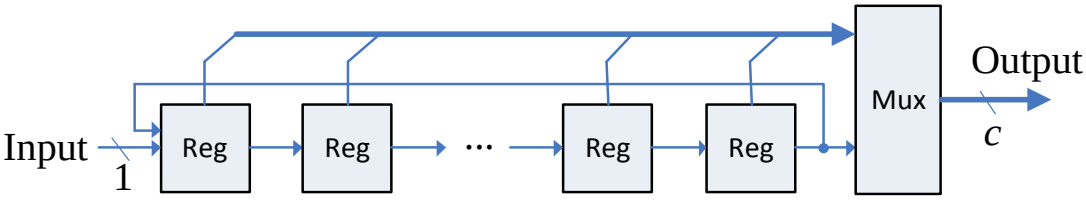

圖 3.14 Buffer H 硬體架構圖

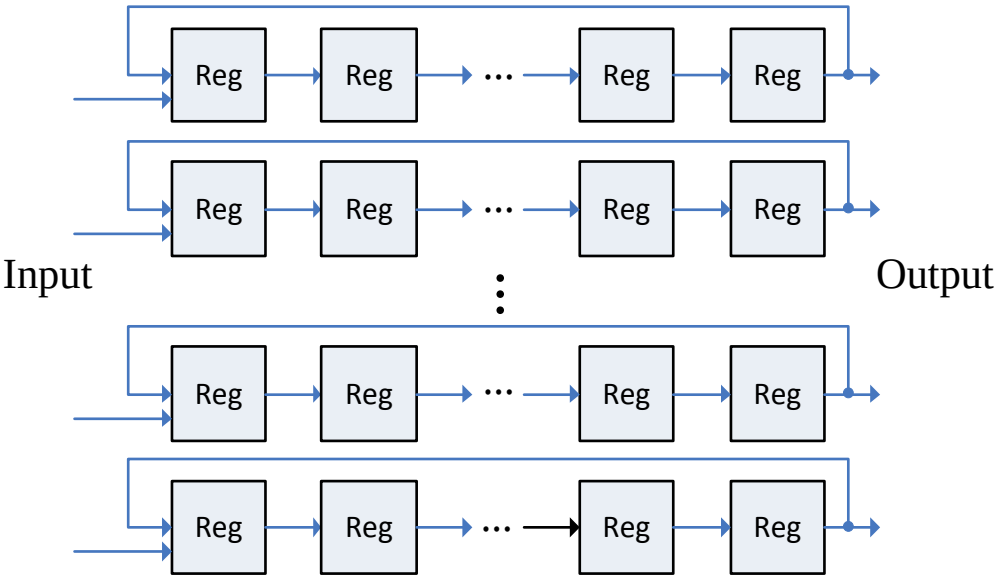

圖 3.15 Buffer S 硬體架構圖

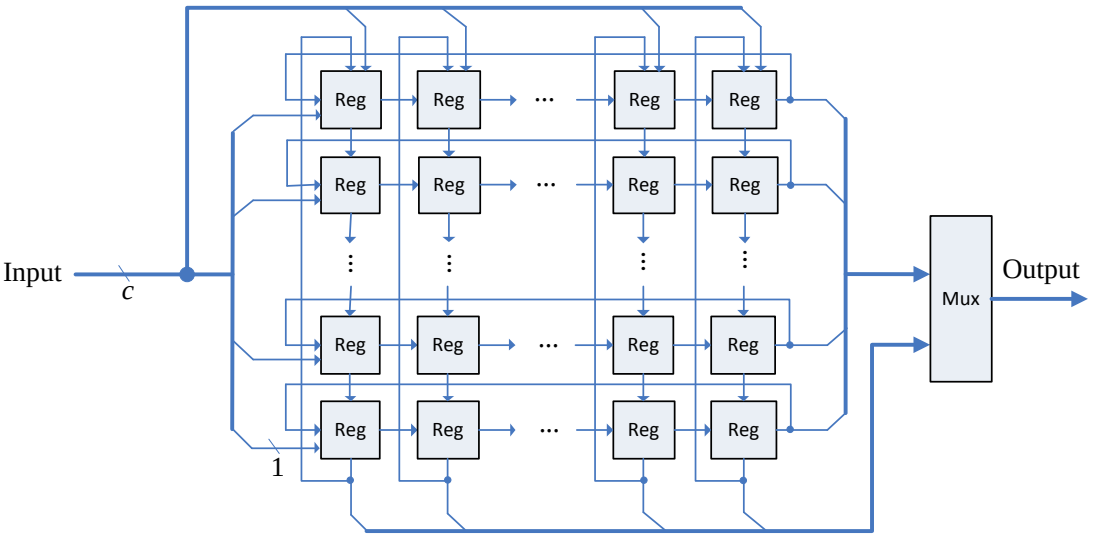

圖 3.16 Buffer P 硬體架構圖

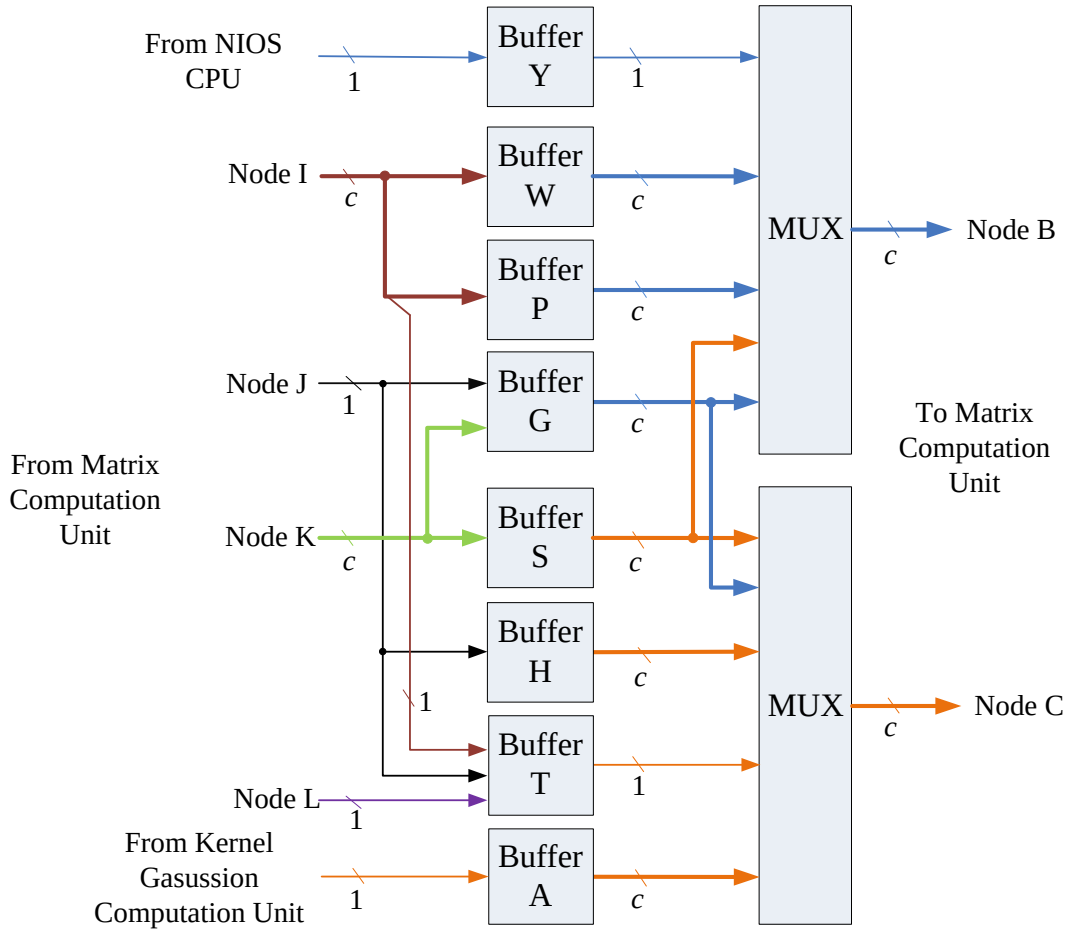

圖 3.17 Memory unit 硬體電路架構圖

表 3.1 Buffer 功能表

	Input		Ouput		Circuit Architecture
	Number of input ports	Input attributes	Number of output ports	Output attributes	
Buffer Y	1	Serial in	1	Serial out	SISO
Buffer W	*c*	Parallel in	*c*	Parallel out	PIPO
Buffer P	*c*	Parallel in	*c*	Parallel out	PIPO
Buffer G	1	Serial in	*c*	Parallel out	SIPO or PIPO
	c	Parallel in			
Buffer S	*c*	Parallel in	*c*	Parallel out	PIPO
Buffer H	1	Serial in	1	Serial out	SISO or SIPO
			c	Parallel out	
Buffer T	1	Serial in	1	Serial out	SISO
Buffer A	1	Serial in	*c*	Parallel out	SIPO

3.3.3 Matrix Computation Unit

Matrix computation unit 包含 *c* 個 2-input 乘法器，*c* 個 2-input 加法器，一個 *c*-input 加法器和一個 inverse operator，如圖 3.18。因此，matrix computation unit 可以平行計算 *c* 個乘法和 *c* 個加法。除此之外，此電路有四種運作模式，Mode 1 使用 *c* 個 2-input 加法器執行 *c* 個平行加法，如圖 3.19。Mode 2 使用 *c* 個 2-input 乘法器執行 *c* 個平行乘法和加法，如圖 3.20。Mode 3 使用 *c* 個 2-input 乘法器平行執行 *c* 個乘法，如圖 3.21，Mode 4 則是執行倒數運算，如圖 3.22。

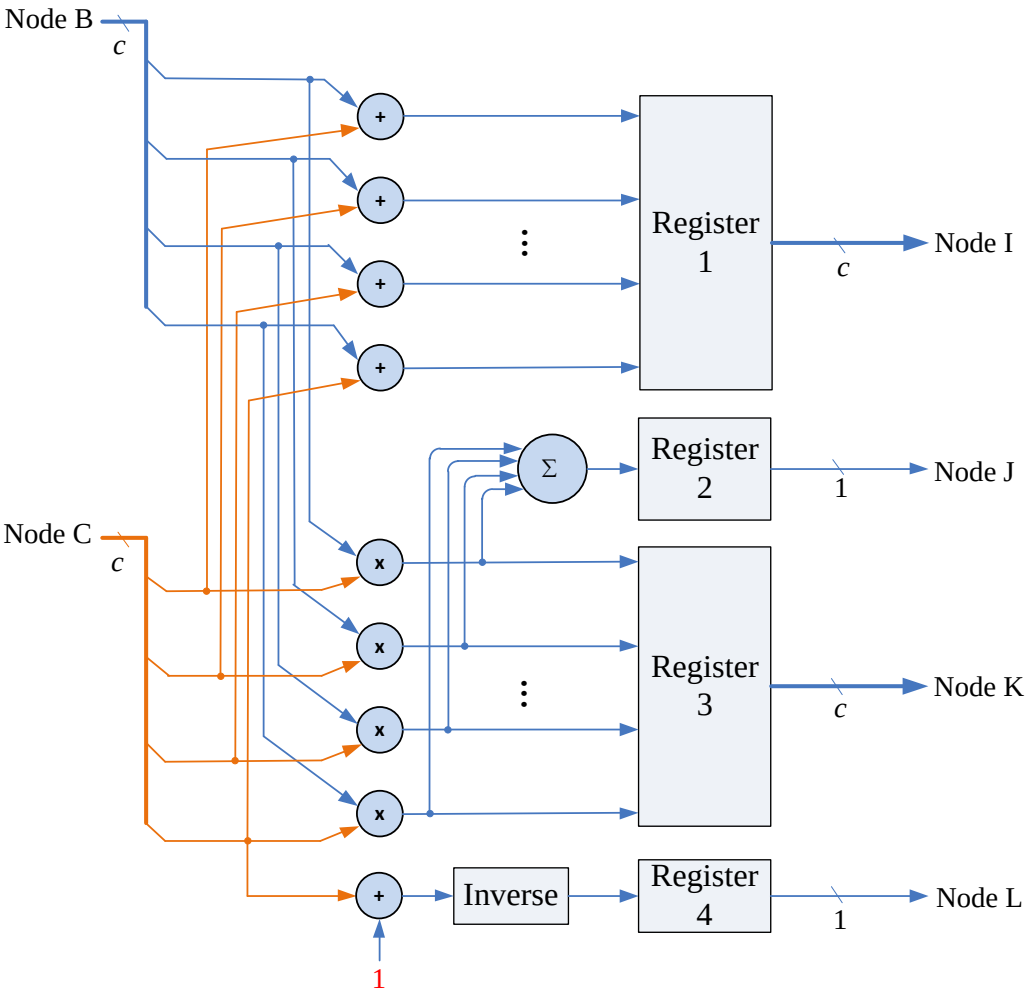

圖 3.18 Matrix computation 硬體電路架構圖

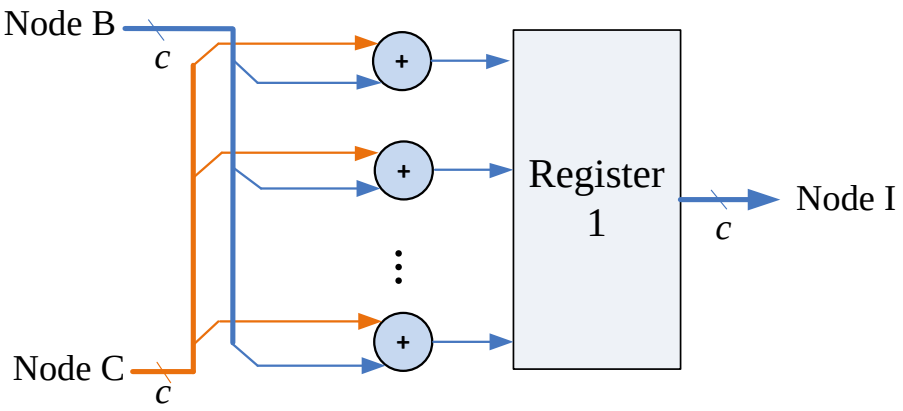

圖 3.19 Mode 1 硬體電路架構圖

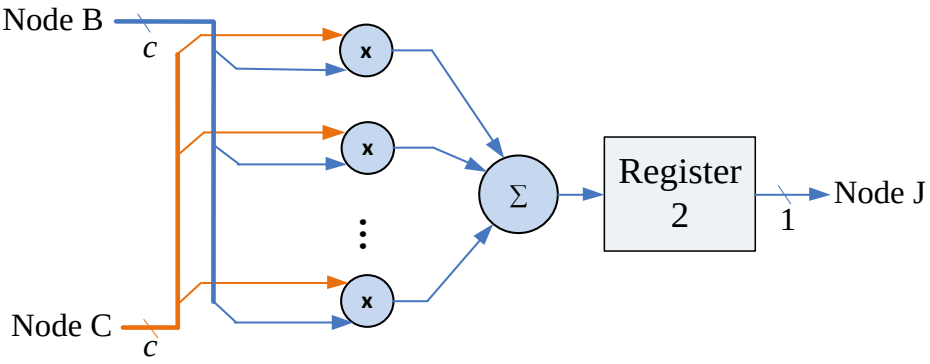

圖 3.20 Mode 2 硬體電路架構圖

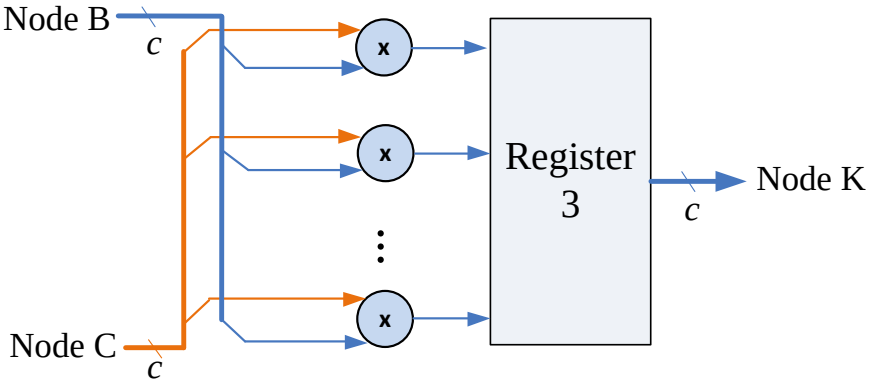

圖 3.21 Mode 3 硬體電路架構圖

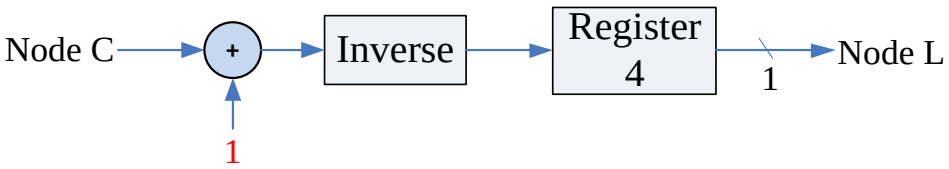

圖 3.22 Mode 4 硬體電路架構圖

3.3.4 Control Unit

Recursive LMS unit 的 control unit 協調 kernel gaussian computation unit、memory unit 和 matrix computation unit 的運作。圖 3.23 顯示 Control Unit 的狀態圖。圖 3.24 至圖 3.26 分別描繪出計算 $\mathbf{P}_k$ 和 $\mathbf{a}_k$ 狀態操作。

當 recursive LMS unit 開始執行時，會依照 control unit 的狀態開始運作。除了 Initial State 和 State 0 之外，每個 State 都會執行 matrix computation unit 的四種運作模式其中一種，即 Mode 1 到 Mode 4。此外，在計算的過程中，隨著每個狀態有不同的計算內容，會從不同的 Source Buffer 讀取資料，經過計算之後，再把計算結果寫回 Destination Buffer。以下將詳細介紹 State 1 到 State 3 的計算過程，State 4 之後之計算內容可參考表 3.1。

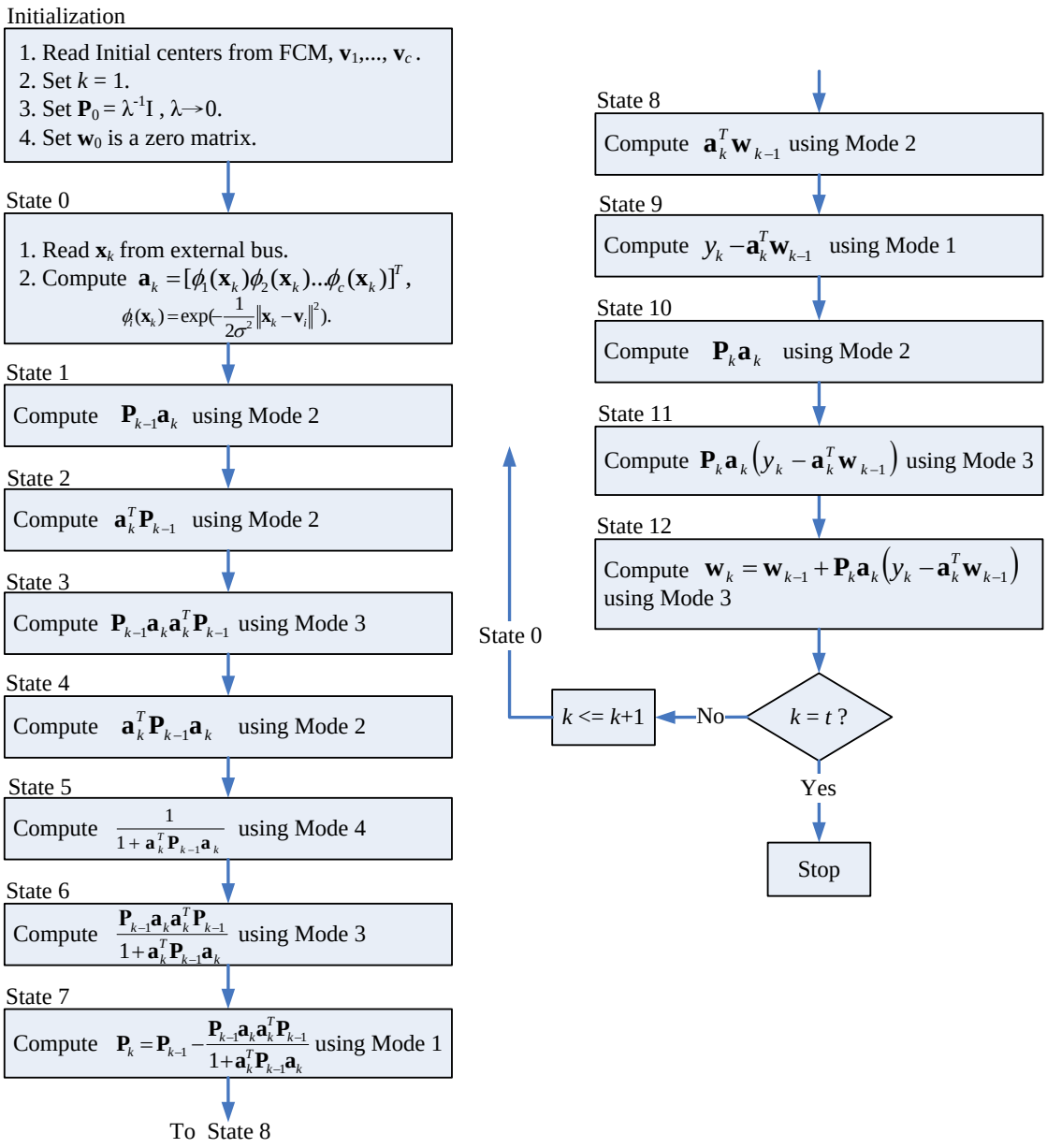

圖 3.23 Control Unit 的狀態圖

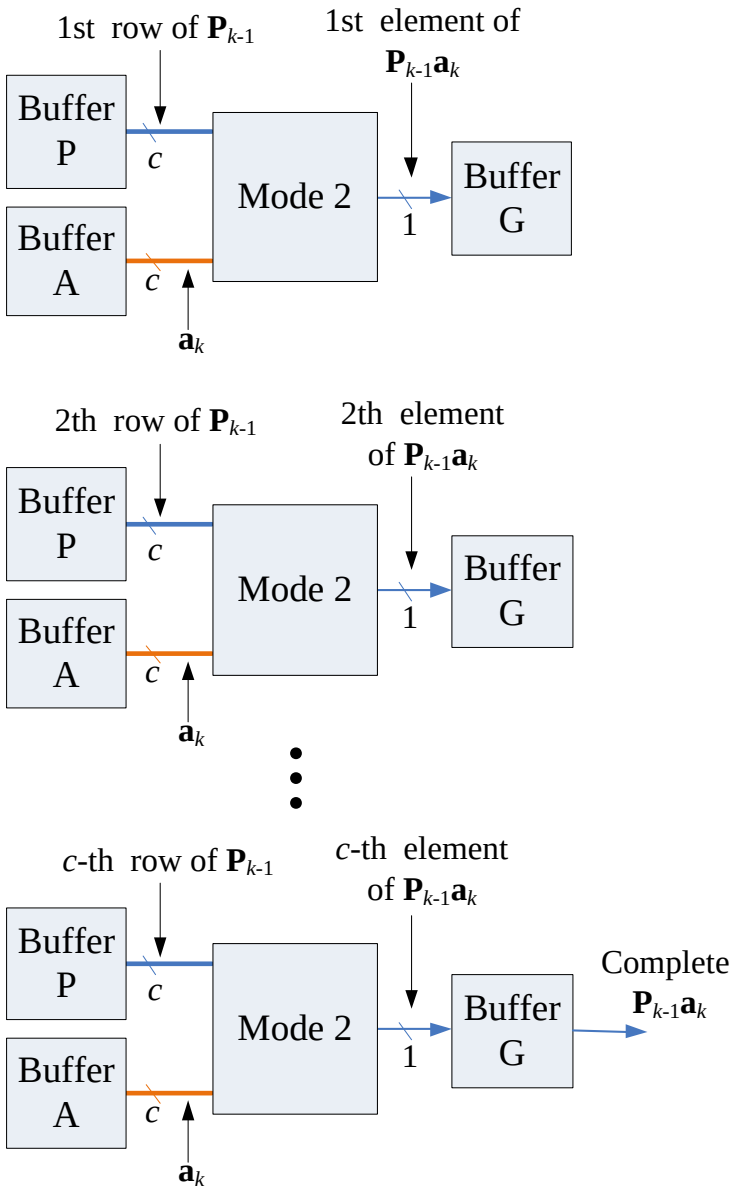

圖 3.24 計算 $\mathbf{P}_{k-1}\mathbf{a}_k$ 之過程

State 1 計算 $\mathbf{P}_{k-1}\mathbf{a}_k$，由於相當於一個維度 cxc 的矩陣乘上一個維度 cx1 的矩陣，因此選擇 Mode 2 運算模式。Matrix computation unit 從 Buffer A 讀取來自 kernel Gaussian computation unit 的計算結果，並且每個 clock 讀取 Buffer P 一列資料，執行 c 個平行乘法和加法，計算結果依序寫入 Buffer G。經過 c 個 clock，$\mathbf{P}_{k-1}\mathbf{a}_k$ 即完成計算，如圖 3.24。

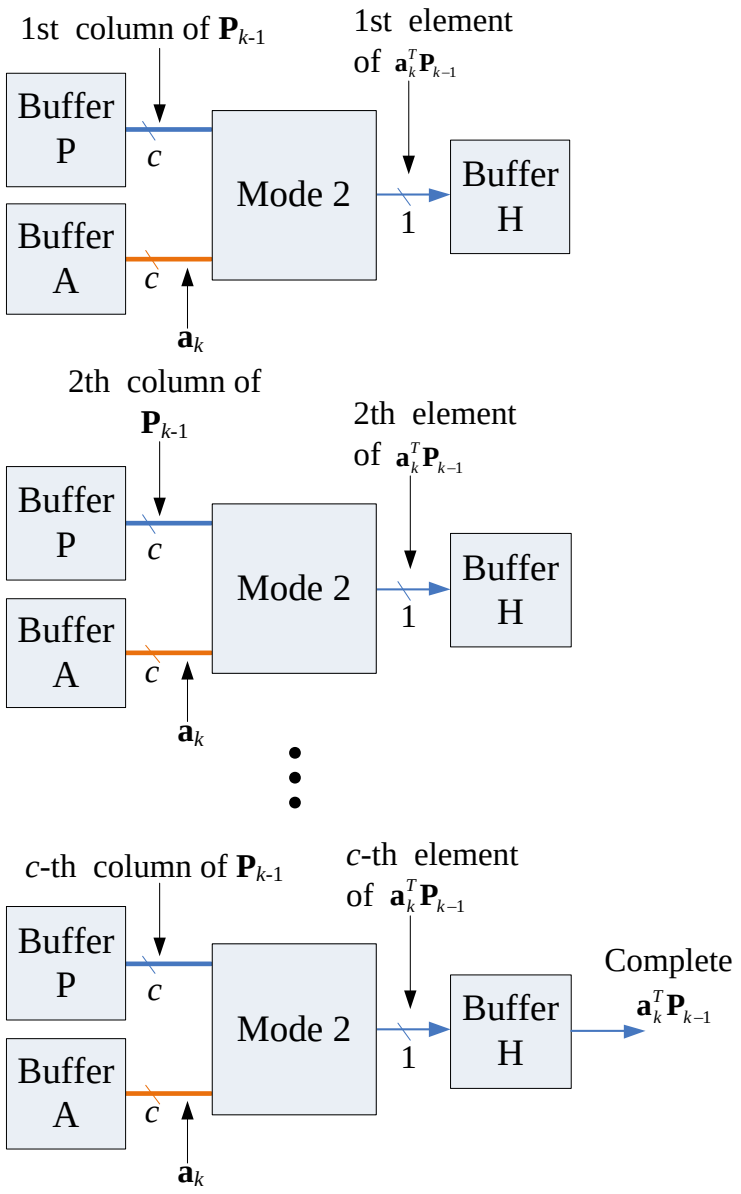

圖 3.25 計算 $\mathbf{a}_k^T\mathbf{P}_{k-1}$ 之過程

State 2 計算 $\mathbf{a}_k^T\mathbf{P}_{k-1}$，相當於一個維度 1x*c* 的矩陣乘上一個維度的矩陣 *cxc*，因此選擇 Mode 2 運算模式。Matrix computation unit 從 Buffer A 讀取來自 kernel Gaussian computation unit 的計算結果，並且每個 clock 讀取 Buffer P 一行資料，執行 *c* 個平行乘法和加法，計算結果依序寫入 Buffer H。經過 *c* 個 clock，$\mathbf{a}_k^T\mathbf{P}_{k-1}$ 即完成計算，如圖 3.25。

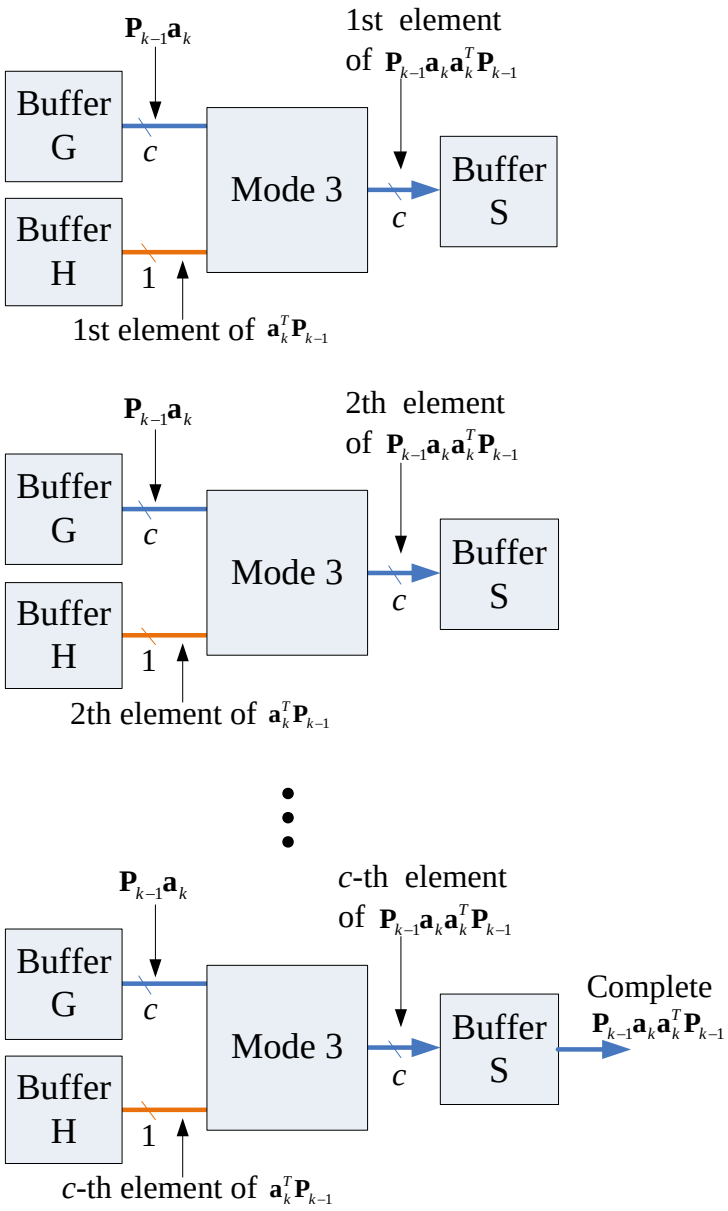

圖 3.26 計算 $\mathbf{P}_{k-1}\mathbf{a}_k\mathbf{a}_k^T\mathbf{P}_{k-1}$ 之過程

State 3 計算 $\mathbf{P}_{k-1}\mathbf{a}_k\mathbf{a}_k^T\mathbf{P}_{k-1}$，相當於一個維度 cx1 的矩陣乘上一個維度的矩陣 1xc，因此選擇 Mode 3 運算模式。Matrix computation unit 從 Buffer G 讀取 state 1 的計算結果，並且每個 clock 讀取 Buffer H 一筆資料，行執行 c 個乘法，計算結果依序寫入 Buffer S。經過 c 個 clock，$\mathbf{P}_{k-1}\mathbf{a}_k\mathbf{a}_k^T\mathbf{P}_{k-1}$ 即完成計算，如圖 3.26。

表 3.2 State 1 至 State 12 之計算內容

State	Source Buffers and their contents		Mode of Matrix Computation Unit	Destination Buffer and its contents	Number of Steps
State 1	Buffer P	Buffer A	Mode 2	Buffer G	*c*
	$\mathbf{P}_{k-1}$	$\mathbf{a}_k$		$\mathbf{P}_{k-1}\mathbf{a}_k$	
State 2	Buffer P	Buffer A	Mode 2	Buffer H	*c*
	$\mathbf{P}_{k-1}$	$\mathbf{a}_k$		$\mathbf{a}_k^T\mathbf{P}_{k-1}$	
State 3	Buffer G	Buffer H	Mode 3	Buffer S	*c*
	$\mathbf{P}_{k-1}\mathbf{a}_k$	$\mathbf{a}_k^T\mathbf{P}_{k-1}$		$\mathbf{P}_{k-1}\mathbf{a}_k\mathbf{a}_k^T\mathbf{P}_{k-1}$	
State 4	Buffer H	Buffer A	Mode 2	Buffer T	1
	$\mathbf{a}_k^T\mathbf{P}_{k-1}$	$\mathbf{a}_k$		$\mathbf{a}_k^T\mathbf{P}_{k-1}\mathbf{a}_k$	
State 5	Buffer T		Mode 4	Buffer T	1
	$\mathbf{a}_k^T\mathbf{P}_{k-1}\mathbf{a}_k$			$\frac{1}{1+\mathbf{a}_k^T\mathbf{P}_{k-1}\mathbf{a}_k}$	
State 6	Buffer T	Buffer S	Mode 3	Buffer S	*c*
	$\frac{1}{1+\mathbf{a}_k^T\mathbf{P}_{k-1}\mathbf{a}_k}$	$\mathbf{P}_{k-1}\mathbf{a}_k\mathbf{a}_k^T\mathbf{P}_{k-1}$		$\frac{\mathbf{P}_{k-1}\mathbf{a}_k\mathbf{a}_k^T\mathbf{P}_{k-1}}{1+\mathbf{a}_k^T\mathbf{P}_{k-1}\mathbf{a}_k}$	
State 7	Buffer P	Buffer S	Mode 1	Buffer P	*c*
	$\mathbf{P}_{k-1}$	$\frac{\mathbf{P}_{k-1}\mathbf{a}_k\mathbf{a}_k^T\mathbf{P}_{k-1}}{1+\mathbf{a}_k^T\mathbf{P}_{k-1}\mathbf{a}_k}$		$\mathbf{P}_k$	
State 8	Buffer A	Buffer W	Mode 2	Buffer T	1
	$\mathbf{a}_k^T$	$\mathbf{w}_{k-1}$		$\mathbf{a}_k^T\mathbf{w}_{k-1}$	
State 9	Buffer Y	Buffer T	Mode 1	Buffer T	1
	y_k	$\mathbf{a}_k^T\mathbf{w}_{k-1}$		$y_k-\mathbf{a}_k^T\mathbf{w}_{k-1}$	
State 10	Buffer P	Buffer A	Mode 2	Buffer G	*c*
	$\mathbf{P}_k$	$\mathbf{a}_k$		$\mathbf{P}_k\mathbf{a}_k$	
State 11	Buffer G	Buffer T	Mode 3	Buffer G	1
	$\mathbf{P}_k\mathbf{a}_k$	$y_k-\mathbf{a}_k^T\mathbf{w}_{k-1}$		$\mathbf{P}_k\mathbf{a}_k\left(y_k-\mathbf{a}_k^T\mathbf{w}_{k-1}\right)$	
State 12	Buffer W	Buffer G	Mode 1	Buffer W	1
	$\mathbf{w}_{k-1}$	$\mathbf{P}_k\mathbf{a}_k\left(y_k-\mathbf{a}_k^T\mathbf{w}_{k-1}\right)$		$\mathbf{w}_k$	

3.4 RBF 測試電路

在前面的章節中，我們提出 RBF 的訓練電路。在這章節中，我們亦提出 RBF 的測試電路，測試電路如圖 3.27。

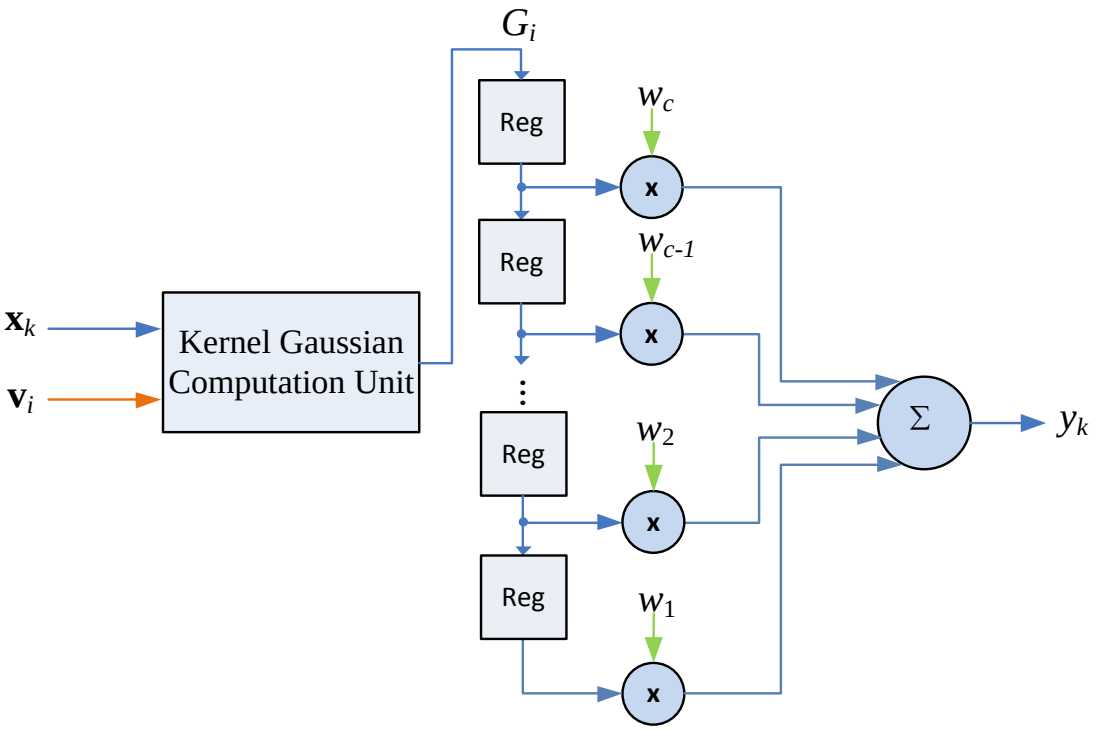

圖 3.27　RBF 測試電路圖

由於硬體資源有限，在這裡我們設計 kernel Gaussian computation unit 為管線化(Pipeline)架構，詳細的管線化架構流程如圖 3.28。依序輸入訓練向量和質量中心點($\mathbf{x}_k$, $\mathbf{v}_1$),…,($\mathbf{x}_k$, $\mathbf{v}_c$)至 kernel Gaussian computation unit，由於 kernel Gaussian computation unit 計算時間需要 64 clocks，因此經過 64 clocks 之後，會依序產生計算結果 G_1,…,G_c，儲存至位移暫存器。之後讀取位移暫存器的資料與權重係數 w_i 平行相乘後再相加，即可得到 $\mathbf{x}_k$ 相對應的 y_k 值。而 y_k 值的比較，則以軟體的方

式實行。而整個電路執行的過程為：輸入 $\mathbf{x}_k$、$\mathbf{v}_i$ 和 w_i, i=1,…c 至 RBF 測試電路，之後從電路中讀取 y_k 值，輸入至軟體做比較，判斷此訓練向量之紋理圖的類別。

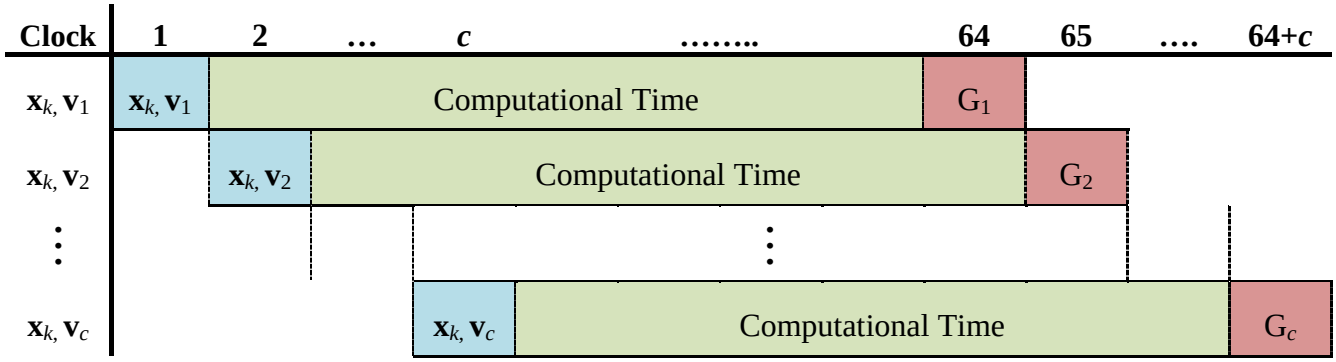

圖 3.28　Kernel Gaussian computation unit 的管線化架構圖

在傳統 RBF 紋理圖辨識系統的網路架構中，所有的紋理圖之質量中心點放在同一個 RBF 網路，因此在權重訓練的過程裡，所有的紋理圖之權重同時訓練。然而，當質量中心點個數 c 或訓練向的個數 t 變大時，訓練過程的計算複雜度變高，訓練時間變長，除此之外，這樣的訓練效果也比較不好。因此，在這裡我們提出每張圖片共享一個 RBF 訓練電路，即每張紋理圖個別訓練，訓練結束後有個別的質量中心點和權重係數，如圖 3.29。

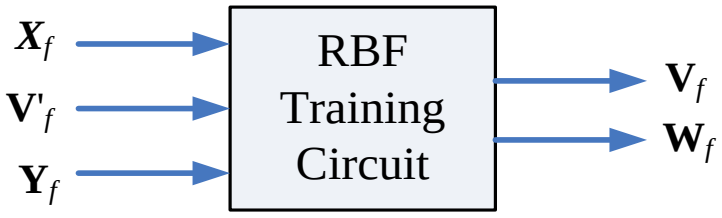

圖 3.29　RBF 訓練電路

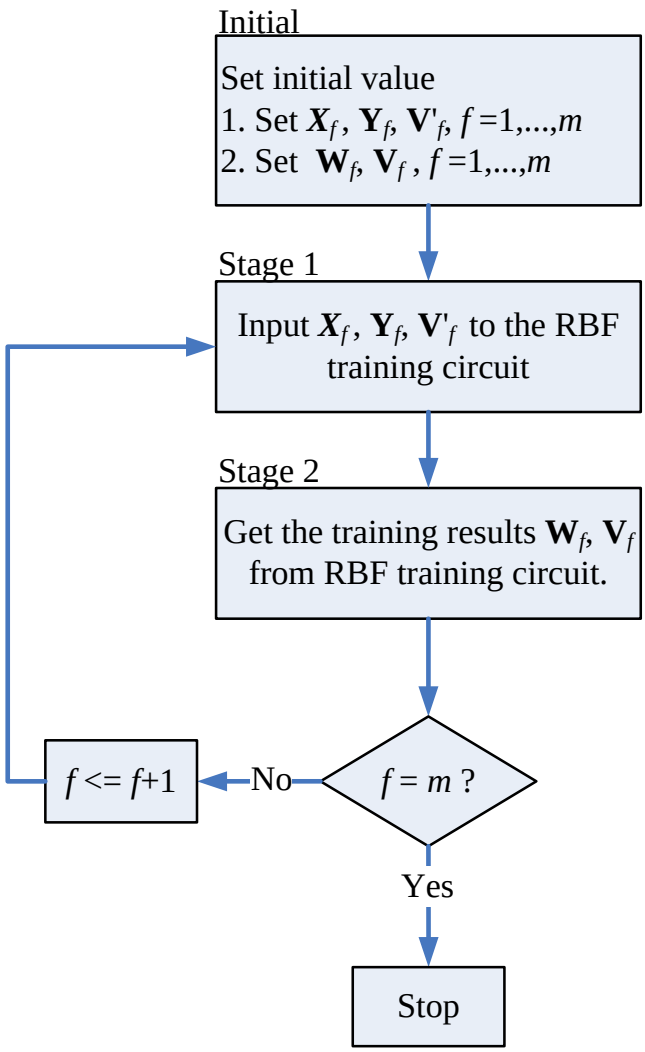

圖 3.30 RBF 訓練電路之流程圖

假設訓練 m 張紋理圖，f =1,…,m，其中 $\boldsymbol{X}_f$和 $\mathbf{V}'_f$代表第 f 張紋理圖的訓練向量集合與初始質量中心集合，$\mathbf{Y}_f$代表第 f 張紋理圖的 y_k 值集合，$\mathbf{V}_f$和 $\mathbf{W}_f$分別代表第 f 張紋理圖的訓練的結果，即更新的質量中心集合和權重係數集合。在訓練的過程中，把需要訓練的紋理圖依序輸入 $\boldsymbol{X}_f$與 $\mathbf{V}'_f$至 RBF 訓練電路中，訓練完成後，即可得到各個紋理圖之 $\mathbf{V}_f$和 $\mathbf{W}_f$，如圖 3.30 的流程圖所示，直到所有的紋理圖訓練完成。另外，所有紋理圖的 $\mathbf{Y}_f$ 中的值 y_k 都設定相同的值，因此所有的紋理圖都有相同的 y_k 值的情況下，輸入不同的紋理圖之訓練向量，每張紋理圖都會

訓練出不同的權重係數。圖 3.31 顯示 RBF 訓練電路的時間表，假設一張紋理圖訓練時間為 h，由於所有紋理圖共享一個 RBF 訓練電路，因此當所有圖片訓練完成時，則需要 mxh 的時間。

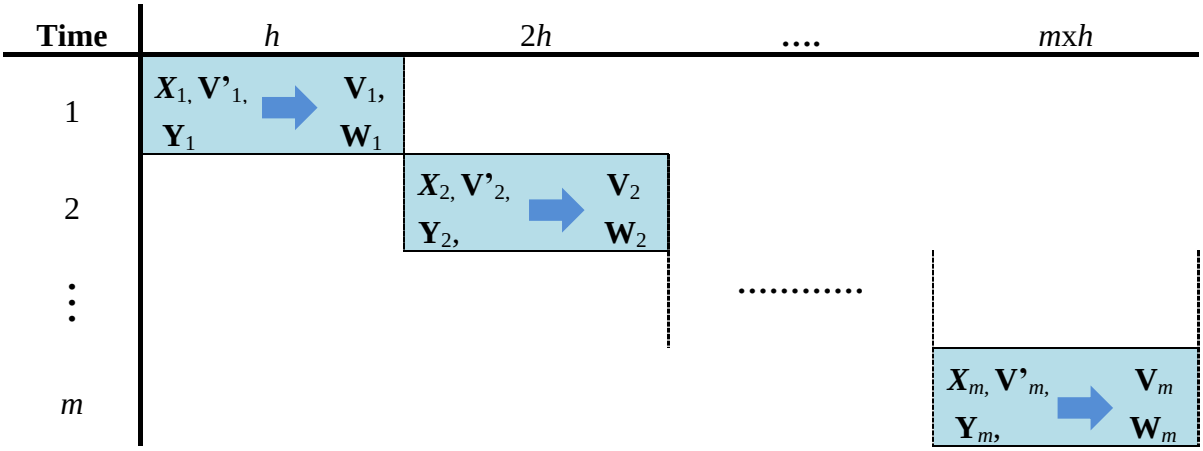

圖 3.31　RBF 訓練電路之時間表

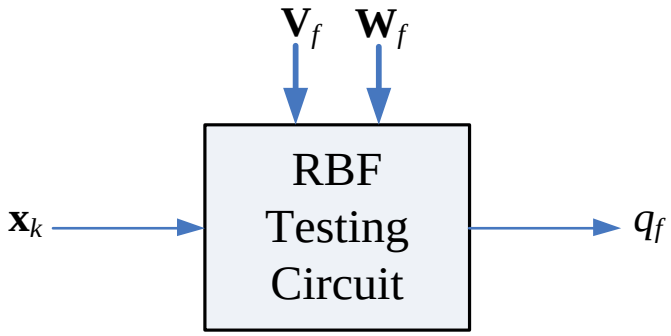

圖 3.32　RBF 測試電路

圖 3.32 顯示 RBF 測試電路圖。由於 $\mathbf{V}_f$、$\mathbf{W}_f$, f =1,…,m 代表有 m 張紋理圖，因此當測試的訓練向量 $\mathbf{x}_k$ 依序輸入到 RBF 測試電路，不同 $\mathbf{V}_f$、$\mathbf{W}_f$ 在 RBF 測試電路中會產出不同的 q_f 值，並假設不同的 q_f 值與 y_f 值會有不同的距離平方為 $\mathbf{E}_f$。我們選擇所有 $\mathbf{E}_f$, f =1,…,m 中最小者，當作訓練向量 $\mathbf{x}_k$ 歸屬的依據。例如 $\mathbf{E}_{f^*}$ 為

所有 $\mathbf{E}_f$ 中的最小值，則我們說訓練向量 $\mathbf{x}_k$ 歸屬於第 f^* 張紋理圖，如圖 3.33 所示，測試過程直到所有測試的訓練向量完成。圖 3.34 顯示 RBF 測試電路的時間表，假設一筆訓練向量 $\mathbf{x}_k$ 輸入 RBF 測試電路產生 q_f 的時間為 u，則一筆訓練向量測試試完的時間為 mxu。

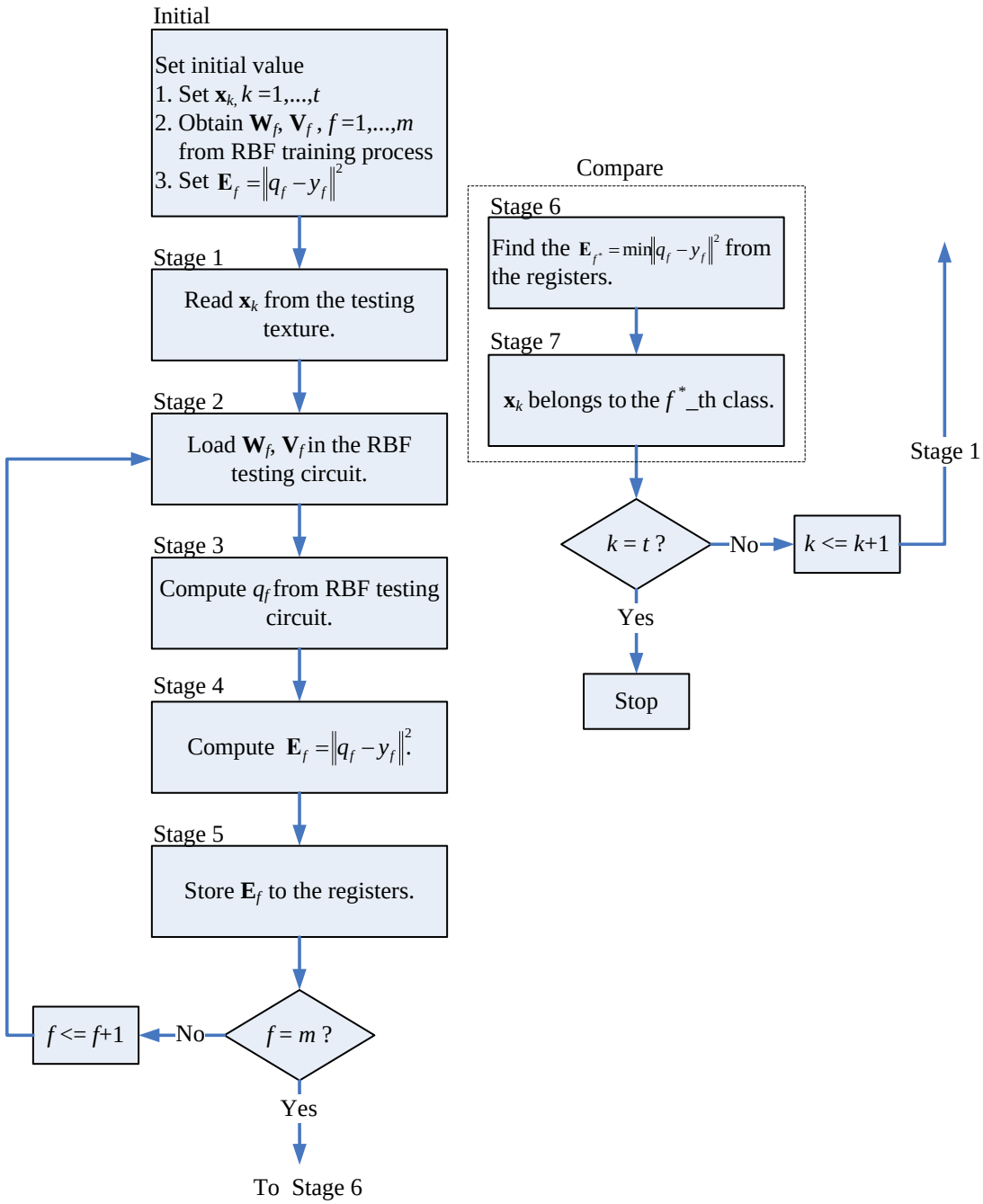

圖 3.33　RBF 測試電路之流程圖

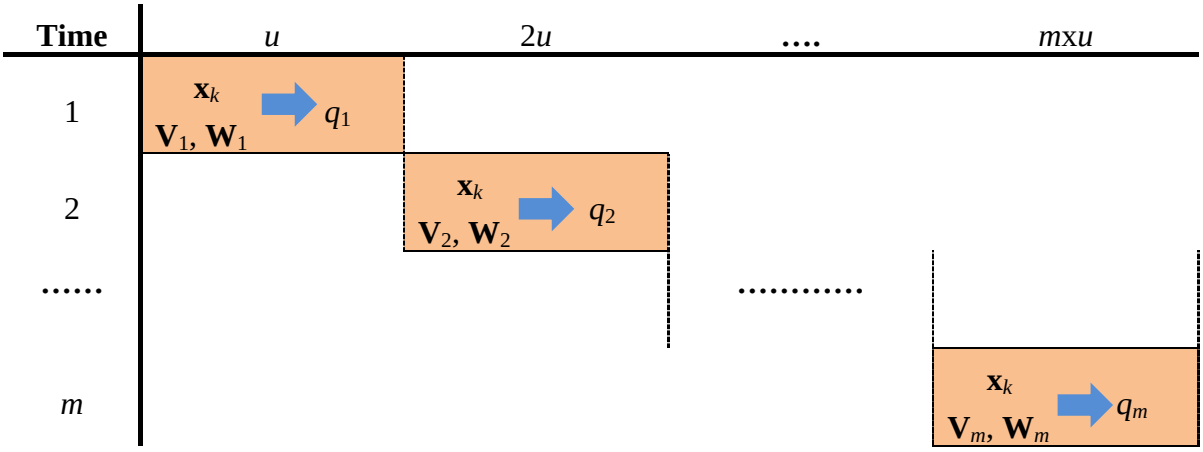

圖 3.34 RBF 測試電路之時間表

這樣的 RBF 訓練過程與 RBF 測試過程有下列優點：

1. 辨識率好：與傳統的 RBF 紋理圖辨識系統比較，擁有較好的辨識率。除此之外，本系統的辨識率可以逼近高維度 GHA 辨識系統的辨識率。
2. 延展性高：當需要辨識的紋理圖類別增多時，可以不需重複訓練已訓練過的紋理圖，只需訓練新增加的紋理圖，因此可以節省許多訓練時間。
3. 硬體資源消耗低：由於傳統的 RBF 紋理圖辨識系統將所有紋理圖之質量中心點都存在於同一個 RBF 網路，在實現 recursive LMS 硬體架構時，容易造成硬體資源消耗量大。因此，依據上述的訓練過程，每張圖片共享一個 RBF 訓練電路，如此可降低硬體資源消耗量。

3.5 FPGA-Based RBF 訓練系統

本論文提出的硬體架構為建置於 SOPC 系統中的客製化邏輯電路(custom user logic)，圖 3.35 為本系統與 SOPC 平台之關係圖。系統中包含了 NIOS CPU、DMA controller 與 on-chip RAM。利用 DMA 將所有存放於 DDRII 的訓練向量 $X = \{\mathbf{x}_1,...,\mathbf{x}_t\}$ 傳送至 on-chip memory 中，再由 NIOS CPU 將訓練向量和隨機挑選的初始質量中心經由 Avalon bus 傳送至 RBF 電路中的 FCM 電路。當 FCM 電路計算完成時，NIOS CPU 再把訓練向量傳送至 RBF 電路中的 recursive LMS 電路。在這裡 NIOS CPU 僅執行簡單的指令協調 FCM 電路與 recursive LMS 電路，而 NIOS CPU 本身並不參與 FCM 與 recursive LMS 的計算。當所有訓練向量傳送完畢後，NIOS CPU 從客製化電路中接收訓練結果 $\mathbf{v}_1,\ldots\mathbf{v}_c$ 與 $w_1,...,w_c$。詳細的訓練流程圖如圖 3.36。

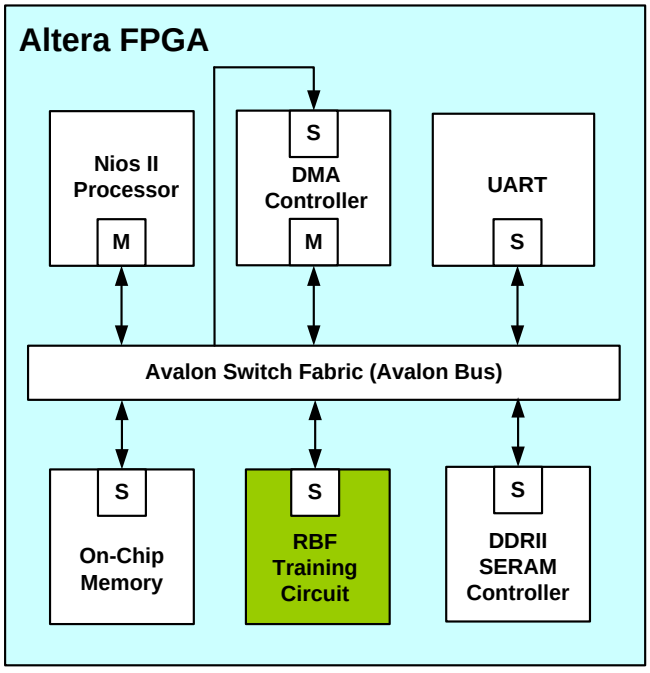

圖 3.35　SOPC 系統架構圖

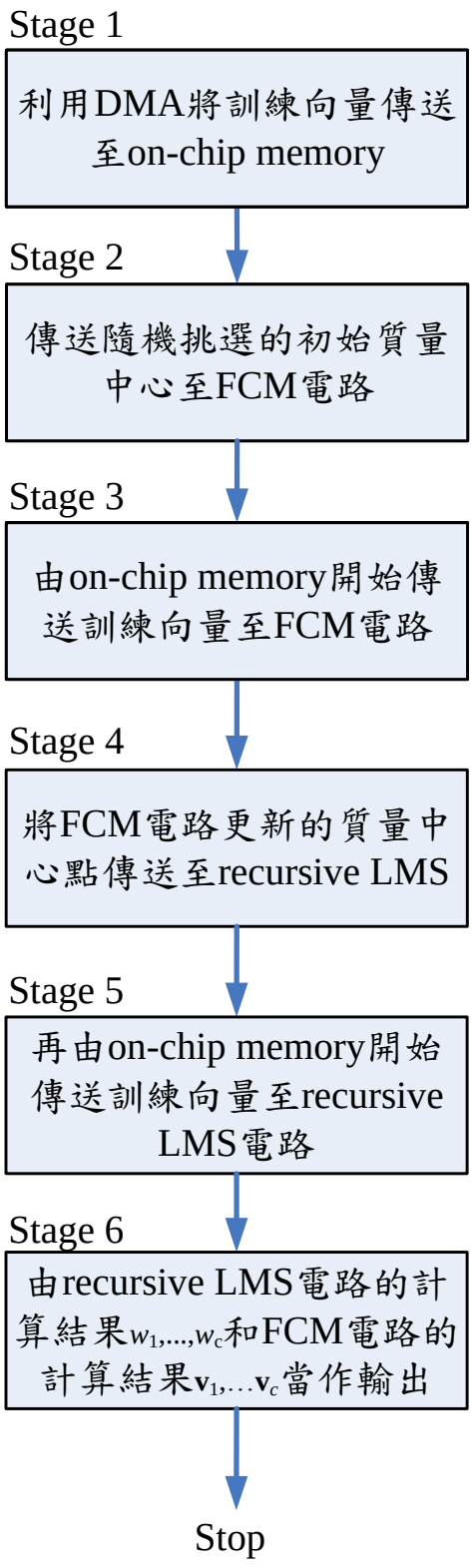

圖 3.36 SOPC 系統流程圖

第四章 實驗結果與數據探討

本章節將探討本論文所提出的硬體架構之實驗環境、實際效能測量，以及實驗結果的比較。

4.1 開發平台與實驗環境介紹

本論文提出的架構主要是以 Altera 的 Cyclone III EP3C120 FPGA 開發板為實驗平台，如圖 4.1 所示。因為可程式化系統晶片(System On a Programmable Chip, SOPC)可以將硬體設計與軟體程式整合在晶片上，達到快速的硬體開發與驗證測試，所以選擇以 FPGA 開發板來實現與驗證本論文所提出的硬體架構。表 4.1 是 Altera Cyclone III EP3C120 FPGA 開發板的詳細規格資訊。

圖 4.1 Altera Cyclone III EP3C120 FPGA 實驗開發板

表4.1 Altera Cyclone III EP3C120 FPGA開發板規格表

Feature	Cyclone III
Device	EP3C120F780C8
Total Logic Elements	119,088
Number of M9K Blocks	432
Total Memory bits	3,981,312
DSP Block 9-bit elements	576
Total PLLs	4
Total Pins	532

在設計過程中，我們使用 Altera Quartus II 10.1 版本為撰寫 Verilog 程式語言的平台，並且 Quartus II 提供語法的檢查、時序分析、元件配置、繞線佈局以及電路合成等等的便利性功能，如圖 4.2 所示。

在 Quartus II 撰寫完客製化硬體電路後，我們利用 Altera SOPC Builder 建立出自己的 SOPC 系統，包含 CPU、DMA、On-chip RAM、RBF 硬體訓練電路、記憶體等硬體組成，如圖 4.3。

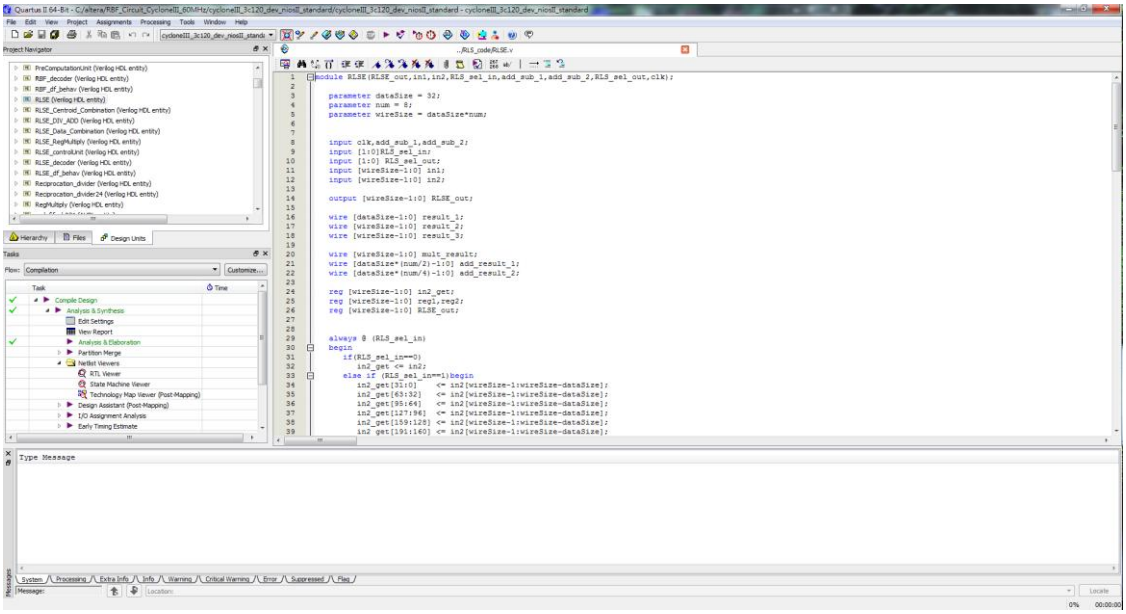

圖 4.2　Quartus II 軟體介面

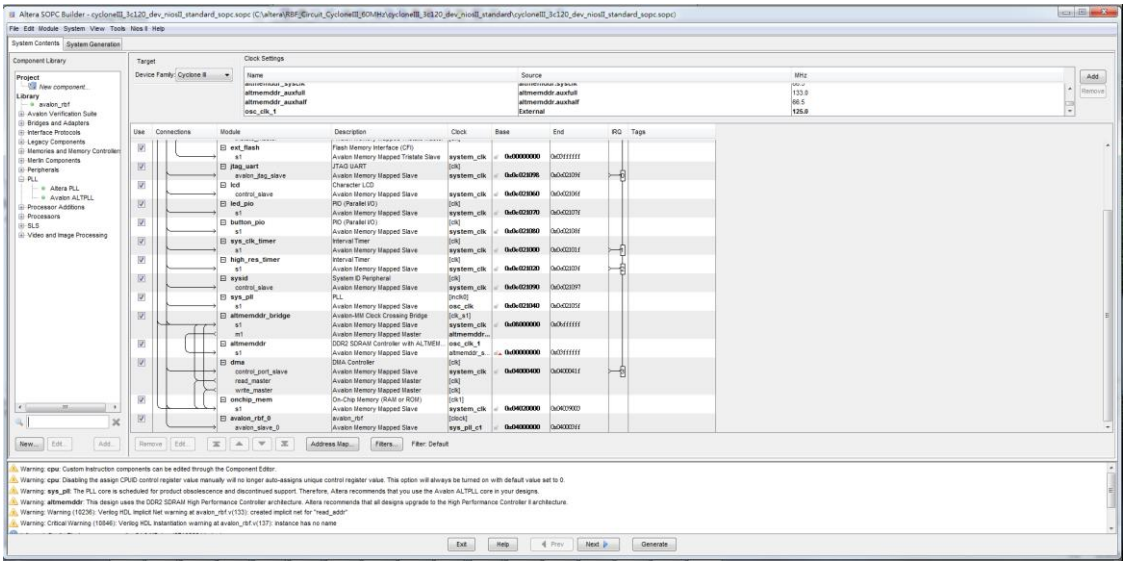

圖 4.3　Altera SOPC Builder 介面

而 Altera 公司也提供了以 Eclipse 為基礎的 NIOS II IDE 軟體供使用者使用，由於 NIOS II 系統提供設計人員在 Altera FPGA 中開發完整的嵌入式系統所需的軟體工具、應用程式、函式庫及驅動程式，方便設計人員利用此 IDE 軟體來與 FPGA 開發板溝通互動，加速 SOPC 系統的開發。在這裡我們使用 NIOS II 10.01 版本作為撰寫 C code 軟體介面，如圖 4.4 所示。

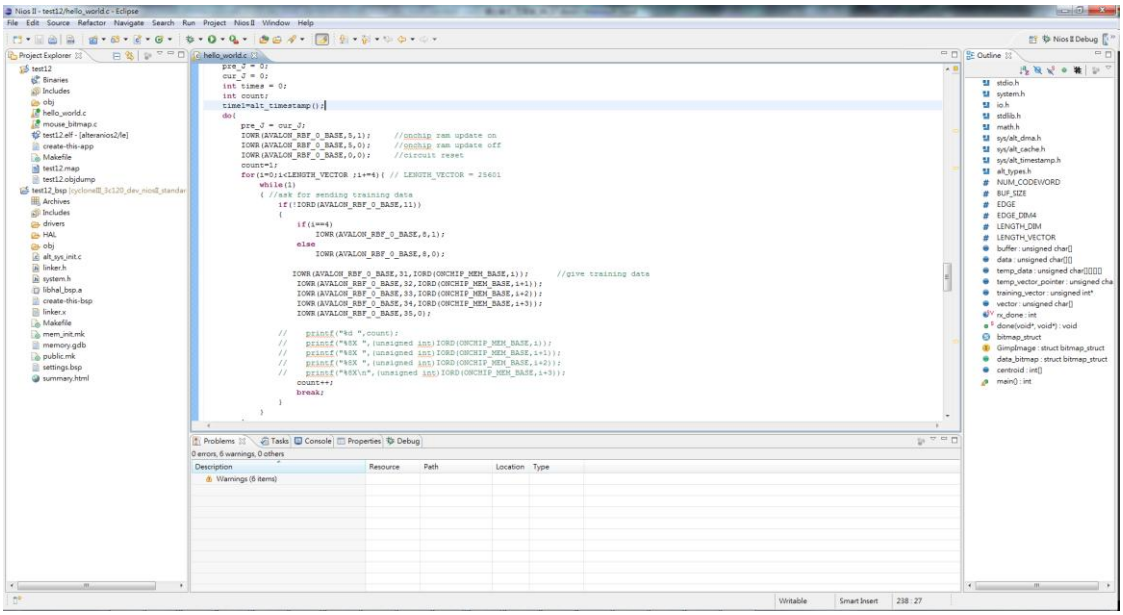

圖 4.4　NIOS II 軟體介面

表 4.2 軟硬體實現環境

軟體 實現環境	
處理器：	Intel Core i5-2467M　@ 1.6GHz
記憶體：	DDR3　4.0G
Compiler :	Dev-C++4.9.9.2
硬體 實現環境	
開發板：	Altera Cyclone III EP3C120
處理器：	NIOS II
記憶體：	256-Mbyte DDR II SDRAM
Flash Memory :	64-Mbyte flash memory

4.2 實驗數據的呈現與討論

本節主要呈現 RBF 訓練電路的硬體資源消耗、軟硬體執行時間及硬體電路實際量測的效能比較。在這裡我們採用 Altera Cyclone III EP3C120 FPGA 開發板來當作硬體架構設計的實驗平台。

首先，在這裡先設定實驗的重要參數：

n：維度，在實驗中設定訓練向量的維度為 4x4。

y_k：Buffer Y 中儲存的值，在這裡設定為 130，k =1,…, t。

σ: Gaussian kernel function 的半徑，在這裡設定為 150。

我們訓練三組不同的紋理圖來檢驗所設計的硬體架構。每一張紋理圖皆為 320×320pixels，並以維度 n=16 pixels 為單位，即 16 pixels 為一個訓練向量，將每張圖切割成 6400 個區塊(blocks)。圖 4.5、圖 4.6 及圖 4.7 分別顯示本論文所使用的 3 組訓練資料，並分別以 Training Set A、Training Set B 及 Training Set C 表示之，其中 Training Set A 有兩種紋理，Training Set B 有三種紋理，而 Training Set C 有四種紋理。

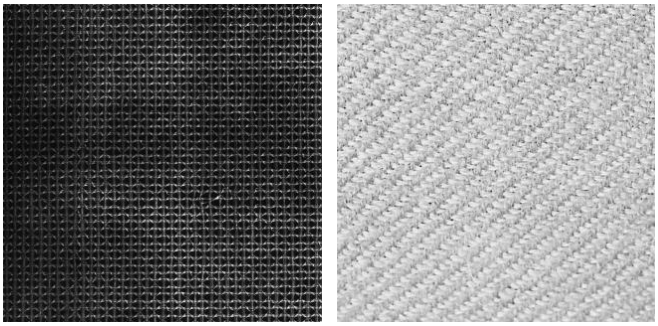

圖 4.5　Training Set A

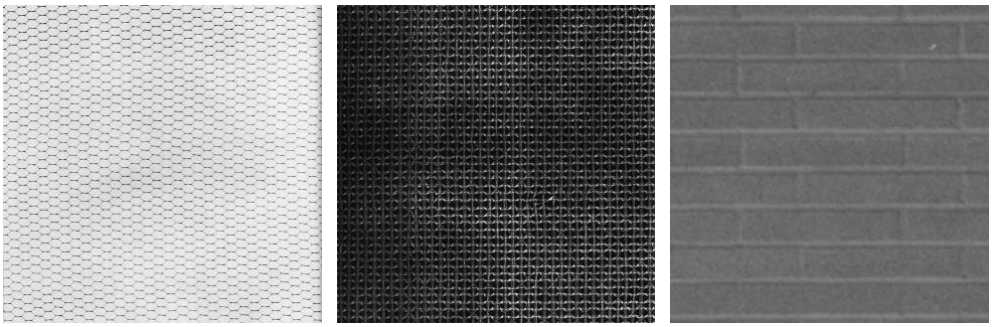

圖 4.6　Training Set B

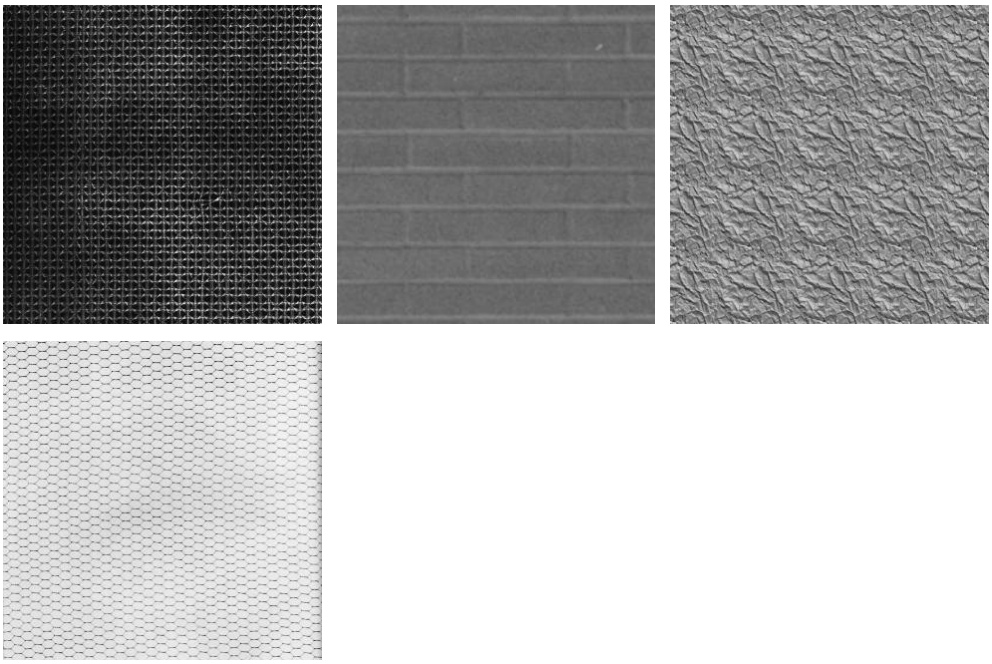

圖 4.7　Training Set C

在設計 RBF 訓練電路時，由於 recursive LMS 有複雜的數學計算以及遞迴運算，因此在實現 recursive LMS 硬體電路時，我們使用 floating point 的格式儲存資料來提升硬體計算結果的精準度。但為了可以有效地減少硬體資源消耗，我們比較 FCM 使用 fixed point 的格式和使用 floating point 的格式之分類正確率。如表 4.3 所示，當 recursive LMS 都是使用 floating point，而 FCM 使用 fixed point 和使用 floating point 時，兩者的分類正確率是非常相近的，因此在設計電路時，FCM 使用 fixed point 的格式而 recursive LMS 使用 floating point 的格式設計電路，使得電路達到低資源消耗的優點。

表 4.3 RBF 訓練電路分類正確率比較表

	FCM	Recursive LMS	Classification Correct Rate
Training Set A	Floating	Floating	100%
	8 bits	Floating	100%
Training Set B	Floating	Floating	99%
	8 bits	Floating	98%
Training Set C	Floating	Floating	97%
	8 bits	Floating	96%

表 4.4 顯示硬體資源消耗的複雜度。n 為訓練向量維度，c 為質量中心點的數目。在 FCM 電路中，加法器和乘法器使用消耗的情況呈現 O(n)的關係，原因是計算距離平方時會跟其維度有關。而 recursive LMS 加法器和乘法器使用消耗的情況呈現 O($c + n$)的關係，原因在於計算 Gaussian kernel function 時需平行計算距

離平方，因此會跟其維度以及質量中心的數目有關。在 Memory Bits 方面，由於 FCM 需要 2-port RAM 來儲存計算過程中的質量中心以及更新過後的質量中心，因此跟質量中心點的數目和維度的乘積有關，為 O(*nc*)。而 recursive LMS 則是因為 Buffer P 和 Buffer S 必須儲存反矩陣計算過程中的值，因此跟質量中心的數目平方有關，為 O(c^2)。

表 4.4 RBF 訓練電路的各種硬體資源消耗

	FCM	**Recursive LMS**	**RBF**
Adders	O(*n*)	O(*c* + *n*)	O(*c*+*n*)
Multipliers	O(*n*)	O(*c* + *n*)	O(*c*+*n*)
Memory Bits	O(*nc*)	O(c^2)	O(c^2+*nc*)
Latency	O(*ct*)	O(*ct*)	O(*ct*)

表 4.5 顯示 RBF 訓練電路的硬體資源消耗。在 FCM 電路部分，由於需要儲存計算過程中的質量中心以及更新過後的質量中心，因此 Total Memory bits 會需要較多的資源。在 recursive LMS 的部分，由於計算 Gaussian kernel function 時需要平行計算距離平方，因此會需要較多的 Total Logic Elements。而整個 SOPC 系統的總體資源消耗在 Total Logic Elements 約為 84%，Embedded Multiplier 約為 60%，Total Memory bits 約為 39%。

表 4.5 RBF 訓練電路硬體資源消耗表

	FCM	Recursive LMS	RBF	SOPC	CycloneIII 資源
Total Logic Elements	25,541 (21%)	57,987 (49%)	88,413 (74%)	99,817 (84%)	119,088
Embeded Multiplier 9-bits elements	79 (14%)	200 (35%)	342 (59%)	346 (60%)	576
Total Memory bits	132,864 (3%)	9,799 (<1%)	142,913 (4%)	1,558,177 (39%)	3,981,312

表 4.6 顯示 recursive LMS 訓練資料個數與分類正確率。由表得知當訓練向量個數介於 100 筆至 400 筆時，分類正確率持續上升。直到訓練向量個數為 500 筆時，分類正確率達到最佳的情況。訓練向量個數 600 之後，分類正確率與訓練向量個數 500 筆時是相似的，因此本實驗在執行 RBF 訓練時，recursive LMS 的訓練向量個數為 500 筆，而 FCM 的訓練向量為 3200 筆。

表 4.6 RBF 訓練電路之 recursive LMS 訓練資料個數與分類正確率 (FCM 訓練資料個數為 3200 筆)

Training data of recursive LMS	**100**	**200**	**300**	**400**	**500**	**600**	**700**
Classification Correct Rate	95%	96%	96%	97%	98%	98%	97.6%
Training data of recursive LMS	**800**	**900**	**1000**	**2000**	**3200**	**6400**	
Classification Correct Rate	98%	98%	98%	98%	97%	97%	

在執行 RBF 訓練過程方面,我們取每張紋理圖中的 3200 筆訓練向量(即 3200 個區塊)依序送入 RBF 訓練電路中的 FCM 電路計算紋理圖的質量中心，當 FCM 電路計算完成時，會將更新的質量中心和每張紋理圖中的 500 筆訓練向量(即 500 個區塊)送入 RBF 訓練電路中的 recursive LMS 電路，計算連接輸出層的的權重。等到整個 recursive LMS 計算完成，代表訓練過程結束，更換下一張紋理圖進入電路中做訓練。表 4.7 呈現出 RBF 訓練電路訓練完後的分類正確率，並同時與 GHA 電路做比較

表 4.7 RBF 訓練電路與 GHA 訓練電路分類正確率比較表

	RBF Hardware	**GHA Hardware**	**RBF Software**
Classification Correct Rate of Training Set A	100%	100%	100%
Classification Correct Rate of Training Set B	98%	99%	99%
Classification Correct Rate of Training Set C	96%	98%	97%

由表 4.7 中得知當分類的紋理圖數目變多時,分類正確率會略為下降。另外，由於軟體計算的精準度較硬體計算的精準度高,RBF 訓練硬體的分類正確率會略低於 RBF 訓練軟體的分類正確率。與 GHA 訓練電路做比較，由於 GHA 硬體訓練電路的維度較高，所以 RBF 訓練電路的分類正確率略低於 GHA 訓練電路。

表 4.8 RBF 訓練電路與 GHA 訓練電路資源消耗和執行時間比較，***K*** 類的訓練時間為此表中的訓練時間的 ***K*** 倍

	RBF Hardware		**GHA Hardware**	**RBF Software**	
CPU Time for Single Class	31.67 ms		1310.2305 ms	157.26 ms	
Number of Training Data	FCM circuit	recursive LMS circuit	400 blocks (dimension :16x16)	FCM circuit	recursive LMS circuit
	3200 blocks	500 blocks		3200 blocks	500 blocks
Platform	Cyclone III		Cyclone III	General Purpose 3.4-GHz Intel i7-2600	
Total Logic Elements	99,817		57,943	0	
Embeded Multiplier 9-bits elements	346		576	0	
Total Memory bits	1,558,177		1,443,204	0	

如表 4.8 所示，雖然 RBF 訓練硬體的辨識率略低於 RBF 訓練軟體的辨識率，但 RBF 訓練硬體的執行時間卻比 RBF 訓練軟體的執行時間快約 5 倍。除此之外，雖然 RBF 訓練電路的分類正確率略低於 GHA 訓練電路的分類正確率，但是卻以少量的時間和低維度逼近維度較高的 GHA 訓練電路之分類正確率，執行時間上比 GHA 訓練電路快速許多，大約比 GHA 訓練電路快 41 倍。(在這裡 GHA 的訓練紋理圖大小為 320x320 pixels，維度為 16x16，電路的主成分個數為 7，精準度為 8 bits，訓練次數為 1800 次)

表 4.9 顯示 RBF 訓練硬體和 RBF 訓練軟體在不同的訓練向量數目時所需的執行時間。其中 RBF 訓練軟體分別在 i7 CPU 和 i5 CPU 上執行。由表可知當訓練向量數目變多時，訓練軟體執行所需要的時間會更多，尤其在 i5 CPU 上執行更明顯。訓練硬體的執行時間雖然也會上升，但其所需要的執行時間在訓練向量數目來越大時，與訓練軟體所需要的執行時間差距會越來越遠。

表 4.9 RBF 訓練軟體和 RBF 訓練硬體之執行時間 單位：ms

Number of Training Data	**6400x1**	**6400x2**	**6400x3**	**6400x4**	**6400x5**	**6400x6**	**6400x7**	**6400x8**
Proposed Architecture	31.67	63.34	91.75	124.10	155.78	187.63	219.98	251.66
General purpose 3.4-GHz i7 CPU	157.26	314.52	471.96	660.26	818.24	979.20	1111.85	1267.73
General purpose 1.6 –GHz i5 CPU	719.81	1439.62	2074.03	2972.38	3642.17	4299.93	5079.12	5814.99

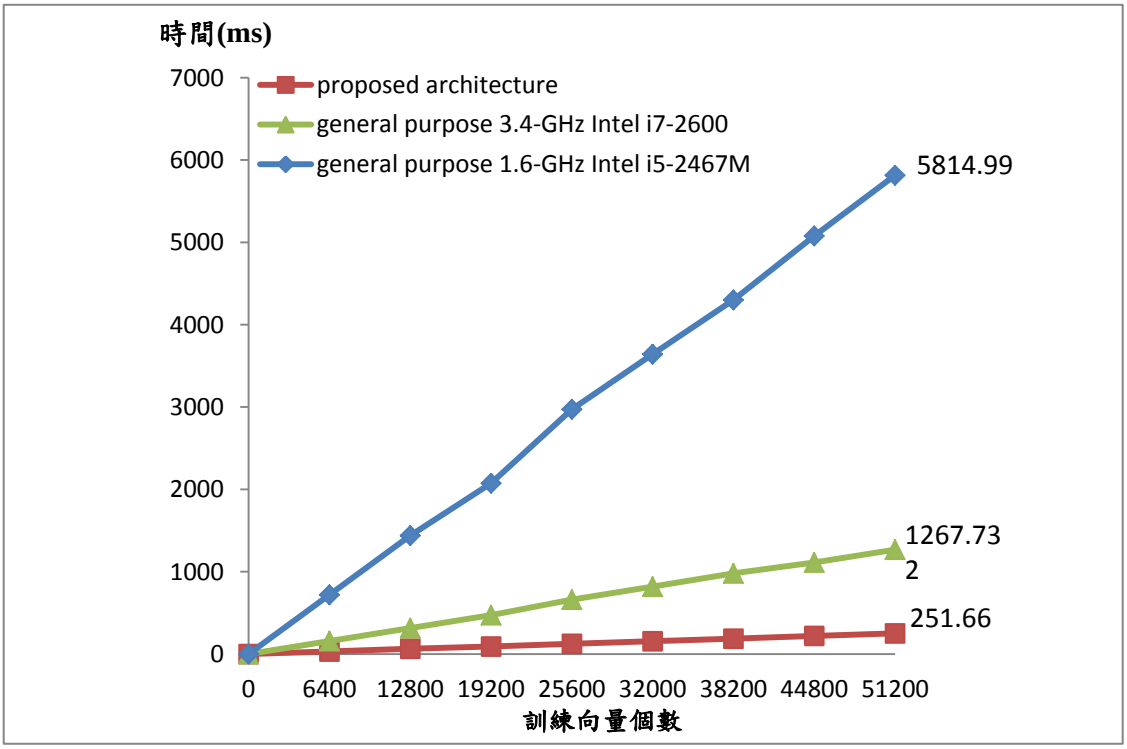

圖 4.8　RBF 訓練軟體和 RBF 訓練硬體之執行時間圖

表 4.10　RBF 訓練電路與 RBF 訓練軟體之功率消耗比較
(所有類別都試用)

	RBF Hardware	**RBF Software**
Platform	Cyclone III	General Purpose 3.4 GHz Intel I7-2600
Power Consumption	1.59 W	17.32 W

表 4.10 顯示 RBF 訓練電路與 RBF 訓練軟體功率比較，由表得知 RBF 硬體在 Cyclone III 的功率消耗為 1.59 W，而 RBF 軟體在 General Purpose 3.4 GHz Intel I7-2600 的功率消耗為 17.32 W。因此，本論文所提出的 RBF 訓練電路達到低功率消耗的優點。

表 4.11 RBF 測試電路之硬體資源消耗與執行時間，
K 類測試的時間為 ***K*** x 62.16ms

	2 Classes	**3 Classes**	**4 Classes**	**CycloneIII 資源**
CPU Time (500 blocks)	9.70 ms	14.38 ms	19.01 ms	
CPU Time (6400 blocks)	124.32 ms	183.08ms	243.09ms	
Total Logic Elements	60,508 (51%)	64,805 (54%)	69,575 (58%)	119,088
Embeded Multiplier 9-bits elements	226 (39%)	226 (39%)	226 (39%)	576
Total Memory bits	597,071 (15%)	597,071 (15%)	597,071 (15%)	3,981,312

表 4.11 顯示 RBF 測試電路之硬體資源消耗。由於高斯函數電路需要較多的乘法器和加法器，因此主要的硬體資源消耗是在實作高斯函數的部分。除此之外，由於分類數目的不同，會需要不同的硬體資源儲存質量中心和權重係數，由表得知當分類數目變多時，logic elements 的消耗也會上升。另外，執行時間也會隨著分類數目增多而上升。

第五章 結論

本論文呈現 RBF 之硬體架構,並實際測量此架構對於在不同分類數目之分類正確率。由於硬體在運算時，可以藉由暫存器儲存或讀取資料，相較於軟體運算時是藉由記憶體存取資料，因此硬體的執行速度會比軟體快速許多。另外，由於 GHA 硬體紋理圖辨識系統在訓練權重係數時需要重覆訓練才能到達收斂的效果，而 RBF 硬體紋理圖訓練系統只須執行一次即可訓練完權重係數，因此 RBF 硬體紋理圖訓練系統執行速度比 GHA 硬體紋理圖訓練系統快。根據實驗結果顯示，RBF 紋理圖辨識在分類應用上具有高達 9 成的分類正確率。除此之外，此架構與軟體在 i7 電腦上比較，大約比軟體快 5 倍，與 GHA 硬體做比較，大約比 GHA 硬體電路快 41 倍，實現出快速計算的效果，並且在分類正確率上以和低維度逼近維度較高的 GHA 電路。因此，我們所提出的硬體架構具有良好的分類結果以及高計算效能，並且提供了日後高度的延伸性與應用。

參考著作

[1] [1] Altera Corporation, NIOS II Processor Reference Handbook, 2011.

[2] S.T. Brassai, L. Bako, Gh. Pana, S. Dan,"Neural control based on RBF network implemented on FPGA," *Proc. IEEE International Conference on Optimization of Electrical and Electronic Equipment*, pp.41-46, 2008.

[3] O. Buchtala, A. Hofmann, and B. Sick, "Fast and efficient training of RBF networks," *Lecture Nots in Computer Science*, Vol. 2714, pp.43-51, 2003.

[4] B. Cao, L. Chang, and H. Li, "Implementation of the RBF Neural Network on a SOPC for maximum power point tracking," *Proc. IEEE Canadian Conference on Electrical and Computer Engineering*, pp. 981-985, 2008.

[5] I.C. Cevikbas, A.S. Ogrenci, G. Dundar, S. Balkir,"VLSI implementation of GRBF (Gaussian Radial Basis Function) networks," *Proc. IEEE International Symposium on Circuits and Systems*, Geneva, Switzerland, May 28-31, 2000.

[6] S. Haykin, Neural Networks and Learning machines, 3rd edition, Pearson: New Jersey,2009.

[7] J. Haddadniaa, K. Faeza, M. Ahmadib,"A fuzzy hybrid learning algorithm for radial basis function neural network with application in human face recognition," *Pattern Recognition*, Vol. 36, pp.1187-1202, 2003.

[8] G.-H. Lee, S.S. Kim, and S. Jung,"Hardware Implementation of a RBF Neural Network Controller with a DSP 2812 and an FPGA for Controlling Nonlinear Systems," *Proc.International Conference on Smart Manufacturing Application*, Korea, 2008.

[9] E. Gatt, J. Micallef and E. Chilton, "Hardware radial basis functions neural networks for phoneme recognition," *Proc. IEEE International Conference on Electronics, Circuits and Systems*, Vol. 2, pp.627-630, 2001.

[10] K. Okamoto, S. Ozawa, and S. Abe, "A Fast Incremental Learning Algorithm of RBF Networks with Long-Term Memory," *Proc. IEEE Joint Conference on Neural Networks*, Vol. 1, pp.102-107, 2003.

[11] W. Pedrycz,"Conditional fuzzy clustering in the design of radial basis function neural networks," *IEEE Trans. Neural Networks*, Vol. 9, pp.601-612, 1998.

[12] H. Sarimveis, A. Alexandridis, and G. Bafas,"A fast training algorithm for RBF networks based on subtractive clustering," *Neurocomputing*, Vol. 51, pp.501-505, 2003.

[13] J. K. Sing, S. Thakur, D. K. Basu, M. Nasipuri, and M. Kundu,"High-speed face recognition using self-adaptive radial basis function neural networks," *Neural Computing and Applications*, Vol. 18, pp.979-990, 2009.

[14] F. Yang and M. Paindavoine, "Implementation of an RBF Neural Network on Embedded Systems: Real-Time Face Tracking and Identity Verification," *IEEE Trans. Neural Networks*, Vol.14, pp.1162-1175, 2003.

[15] Z.-G. Yang and J.-L. Qian,"Hardware Implementation of RBF Neural Network on FPGA Coprocessor," *Communications in Computer and Information Science*, Vol. 105, Part I,pp.415-422, 2010.