國立臺灣師範大學
資訊工程研究所碩士論文

指導教授：　吳 榮 根　博士
共同指導教授：　黃 文 吉　博士

摺積神經網路全連結層 FPGA 實現之研究

The FPGA Implementation of Fully-connected Layers of Convolutional Neural Networks

研究生：　紀凱文　撰

中華民國　一百零五　年　七　月

中文摘要

本論文旨在於 FPGA (Field Programmable Gate Array) [1] [2]平台設計實現全連結架構，並與摺積神經網路結合，成為高速的人工視覺辨識系統。

本論文之基礎建立於類神經網路之全連結的使用，除了將全連結硬體化之外，並與具有即時運算的能力摺積神經網路(Convolutional Neural Network)整合。現存的摺積神經網路系統大多以 GPU 實現，雖具有高速的運算，但同時也擁有高功率消耗等缺點。雖然以 FPGA 為主之設計可有效降低功率消耗，但也有許多可改善之處。首先是在運算過程中會產生許多的中繼結果，這會使記憶體增加儲存資料之負擔；其次是現有硬體實現之架構僅具焦於摺積神經網路內的摺積層架構，往往忽略了其他重要架構像是全連接層(Fully-Connected Layer)之設計，根據上述原因導致無法實現高速及高準確度之人工視覺系統。

本系統採用全連結架構做為硬體實現，此架構大致上可分為 2 個全連結層，利用將這 2 個層級硬體化，進而實現高速的全連結運算。除了實現高速運算之外，為了提高此系統的辨識率，將以此架構與摺積神經網路整合，使辨識率大幅增加。此系統通常應用於字元及人臉辨識，透過我們的實驗結果顯示此架構適合使用於需要高速運算、高準確度、高可攜度、低功率消耗等的人工視覺辨識系統之應用程式。

關鍵字：FPGA、類神經網路、全連結、摺積神經網路

誌謝

在這兩年碩士論文的研究期間當中，首先要感謝的是我的指導教授 吳榮根博士以及我的共同指導教授 黃文吉博士。在研究過程當中，教授們不斷地鼓勵我嘗試自己所提出的想法並提供我豐富的研究資源，並且在遇到問題時有耐心地與我討論、修正，提供更好的意見，進而使我找到最合適的方法。除此之外，教授們也會與我分享一些自己人生經歷過程，從教授的言談當中，我學到了不僅僅是研究方面，在一些生活及人生的經歷方面，也學習了非常的多，使我這兩年當中獲益良多，在此至上我最深的謝意。

除此之外，實驗室的學長姐們及學弟妹們也是我在這兩年期間的良師與益友，謝謝學長姐在工作奮鬥之餘，還能聽取我的問題，一起與我商討解決的方法。另外也分享許多自己在業界裡的經驗談，以及在業界一些技術性的技巧，我也從中學習了許多經驗，避免未來在業界可能發生的錯誤，期望自己透過學長姐們分享的經驗談當中能夠避免諸多的錯誤，並且在畢業後能快速且順利地與未來工作環境接軌，並盡全力地奉獻自己能力給需要的人，在此謝謝我的學長姐及學弟妹們。

最後，我要謝謝陪伴我最久的家人們，謝謝他們在我碩士的期間當中，給予我滿滿的鼓勵與關心，即使在研究方面無法給我許多意見，但家人的關心是我在研究遇到困難時最大鼓勵，也因為有這些鼓勵，使我能在研究中克服許多問題，並完成碩士論文與口試，順利地畢業，完成碩士學位，在此謝謝我最愛的家人們。

目錄

附表目錄

附圖目錄

第一章 緒論

本章節主要論述本論文的研究背景、研究動機與目的以及研究方法，並大略說明各章節的主要內容與重要特性。

第一節　研究背景

類神經網路起源於 20 世紀中期，為現今人類發展人工智慧以來重要的一個理論。而稱做為類神經網路是因為其主要是模仿人類大腦神經元的組織及運作方式。人類大腦的神經元會運算各種所遭遇到的複雜情形，使人類可即時判斷各種事件發生的應對情況，包括聽、說、讀、寫、看等各種情況；類神經網路的架構也是模仿此方式，也因為如此，我們通常將類神經網路運用在影像的處理或者是語音辨識等方面的問題，而最早使用類神經網路的神經元模型稱之為感知機。

類神經網路在現代的運用非常廣泛，通常運用在影像處理及語音辨識方面。影像處理的部分像是提款機臉部辨識、汽車駕駛者確認、年齡或是性別的辨識等，而語音辨識的像是數字辨認等，這些都是類神經網路之應用，且這些應用的辨識效果通常都優於其他的演算法，而最近在全世界關注下的世界圍棋比賽，由人工智慧 AlphaGo 對上世界冠軍李世石，其中的人工智慧 AlphaGo 也是使用類神經網路做為系統的演算法，並且透過此演算法擊敗了人類的世界冠軍，由此可見，類神經網路對於人工智慧的發展是非常重要的一環。

摺積神經網路 [3] [4]是類神經網路的其中一種架構，通常包含了數個層級，包含了摺積層、採樣層、以及全連結層，其中全連結層是本論文主要研究之層級。

全連結是類神經網路最基本的架構，屬於淺度學習。全連結通常分為二層，分別為輸入層及輸出層。輸入層也就是我們所要運算的輸入資料。而輸出層理所當然是我們最後所運算完的輸出資料，而輸出資料也是會由我們輸入層的資料經過運算之後產生。在每一個層級的運算結果都代表著一個神經元。每一個神經元產生的方式會由前一層的每一個神經元和所對應到的權重做乘法運算，再將運算出來的結果做加法運算，也就是說每一個神經元都會由前一層的所有神經元乘以不同的權重，在做相加而產生。

第二節　動機與目的

在現今類神經網路多元的發展下，除了功能的實現為重要的一環，更重要的是追求時間及功率消耗，為了達到此目的，硬體化會是個好選擇。將系統硬體化除了能追求時間之外，也可控制功率消耗的問題。

我們在上一節有討論到，全連結的運算是由多個乘法和加法所組成，訓練過程簡單。 但由於它是全連結，會將所有的輸入和輸出做乘法加法運算，所以當某一層的輸入或者是輸出的數量較多的時候，則會導致權重數過多，權重數過多進而造成計算複雜度的增加，而我們也無法簡化計算複雜度。計算複雜度的增加相對的會使運算速度變慢，除此之外，全連結的辨識率也是個問題。在先前有提到全連結屬於淺度學習，所以增加全連結的層數對於辨識率並不會有太大的影響，這對追求效能及辨識率的情況下，無疑是一大問題。在現今科技，想要運算速度快且辨識率高，可使用 GPU 達到此成就。使用 GPU 雖然可達到運算速度快且辨識率高的效果，但也會造成功率消耗過高的問題。

根據上述原因，本論文主要有兩大目的實現。其一為了隨著計算複雜度的增加，而又能使運算速度快及功率消耗低 [5] [6]，我們選擇將全連結架構做硬體化實現 [7] [8] [9] [10]，這也可以使在計算複雜度高 [11]的情況下，我們也能達到運算速度快的效果。其二是提高全連結的辨識率。在先前提到就算增加全連結層的層數也無法改善辨識率，所以我們選擇了另一個方法來達到增加辨識率的效果，那就是在全連結層的運算之前增加了摺積層以及採樣層的運算，將摺積層及採樣

層所運算的出來的結果，再去做全連結運算，這可以使辨識率達到幾乎百分之百的效果，有效地改善全連結辨識效果不佳的情形。

本論文採用 FPGA 做為實驗平台，並列舉下表比較本論文架構與對比軟體於 CPU 及 GPU 運作之優劣勢。

表 1. 1 為 FPGA 與軟體運作於 CPU 及 GPU 之特性比較

	FPGA	CPU	GPU
運算速度	快	慢	快
功率消耗	低	高	高
可攜性	高	低	低

本論文所提出之硬體架構具有下列優勢：

1. 快速運算
2. 低功率消耗
3. 高可攜性
4. 高辨識率

第三節　研究方法

本系統設計可分為二個部分，一是將輸入資料經過摺積層運算，二是將摺積層運算後之結果，再進行全連結運算，最後會得到我們所運算出的結果，再根據運算結果判斷何為辨識之人臉。而無論是輸入資料或者是所運算之結果，都是以IEEE754 單精度浮點數格式所表示。

摺積層運算首先會經由核心對輸入影像進行摺積運算，將影像的特徵提取出來，進而讓影像的重要特徵訊號增強；而核心會作用於輸入影像的不同區域，產生出對應的特徵訊號。再來將特徵訊號進行採樣，而採樣的方法分為兩種，第一種為採用最大值作為我們所要結果，第二種則是採用計算平均值作為我們所要結果；本系統採用的為第一種方法。再來就是本論文的核心架構，全連結。我們會將摺積層所運算完的結果，再進行全連結運算，但由於是全連結運算，會將所有的輸入值和權重進行乘法和加法運算，如果一次將所有的結果運算完成，資源消耗量會過大，造成 FPGA 無法正常運作。為了解決這個問題，須透過分次運算，先依序取某部分輸入值運算出某部分的輸出結果，最後再將這些部分結果做相加，進而得到最後的輸出結果。

本論文測試樣本是由美國耶魯大學資料庫取得，它是個開放性的資料庫，由28 個人根據不同的表情、不同角度及不同的光照條件建立而成，其中訓練資料的部分由上述 28 個人組成，每個人取 500 張，並且都是不同的條件之下，共有 14000 張影像做為訓練資料，而測試資料同樣由上述 28 個人，每個人取 76 張，同樣也

是在不同的條件之下，共有 2128 張影像做為測試資料，每張都是 32×32 Pixel 的圖片。做硬體測試之前，先分別將 2128 張測試資料用軟體做運算，取得運算及辨識結果，之後再將硬體運算出的結果和軟體運算結果做比對，確認硬體所運算之結果和軟體運算的結果為一致，除了結果一致之外，還需確定辨識是否正確，辨識方法為在最後的 28 筆輸出資料中，判斷第幾筆為最大值，而這第幾筆也代表此次的輸入資料為第幾個人的人臉，經由硬體運算出的結果透過辨識也發現，辨識結果也和軟體為一致，進而證明此硬體架構不但擁有和軟體一樣的功能，並且運算速度比軟體快速許多。

本論文使用 FPGA 為系統開發平台，實際執行運算及測量效能，使用可程式化系統晶片(System on Programmable Chip；SOPC)實現此電路設計。SOPC 為 Altera 公司所設計，包含 NIOS Processor、本論文硬體電路及開發板上所需之元件。透過此系統，開發者可以更便利於設計及驗證電路之正確性。

本論文電路開發使用 Altera 公司所提供的開發軟體—Altera Quartus II，使用這套軟體可以進行多項的效能量測。例如電路資源消耗、功率消耗、系統運算時間等。

第四節　全文架構

本論文分為五個章節，以下為各章節內容概要：

第一章 緒論

研究背景、研究動機與目的、研究方法及全文架構之說明。

第二章 基礎理論及技術背景

使用之理論、演算法則及技術背景之簡介。

第三章 系統架構

系統硬體電路架構及運作流程之討論。

第四章 實驗數據與效能分析

實驗環境、實驗數據分析以及軟硬體結果之比較。

第五章 結論

本論文硬體架構以及實驗結果之總結。

第二章 基礎理論及技術背景

本章節會介紹本論文所使用的基礎理論及研究技術背景的探討，其中將分為五小節分別對相關的基礎理論及技術背景做介紹。第一節介紹類神經網路；第二節則是介紹類神經網路-全連結；第三節則是多層全連結架構介紹，第四節介紹摺積神經網路(Convolutional Neural Network)LeNet-5 架構，最後一節則是本架構實現於 FPGA 之技術及系統之整合。

第一節　類神經網路

類神經網路是一種計算系統，模仿生物神經網絡的結構及功能。生物的神經系統會由神經網路所組成，並且具有學習能力；而類神經網路就是模擬生物的神經系統，建立一個與生物神經系統類似的神經網路，希望透過類神經網路，使電腦具有學習的能力，並且使用學習後的經驗來將資料做處理。

在生物的神經系統中，是由神經元所組成，而類神經網路亦是如此。類神經網路由許多節點組成，每個節點都是一個神經元，每個神經元之間的連接都代表一個通過該連接神經元的加權值，也就是所謂的權重，而這個權重會影響到後面的神經元。

類神經網路有單層神經元網路及多層神經元網路兩種。單層神經元網路為最基本的神經元網路形式，由有限的神經元組成，神經元的輸入向量都是來自同一個向量。另一種則是多層神經元網路，本論文提到的類神經網路就是由多層神經元網路所組成，每一層的神經元擁有輸入和輸出，它的輸入會是由前一層神經元

的輸出所得來，也就是說這一層神經元的輸入會是上一層神經元的輸出，而這一層神經元的輸出會是下一層神經元的輸入。

下圖 2.1 為神經元之示意圖，A_1~A_n 為輸入向量的各個分量，W_1~W_n 為各個連接神經元的加權值，Active function 為傳遞函數，通常為非線性函數，T 為最後輸出的神經元。由上圖可知，輸出神經元會由輸入向量與加權值做內積，在經過一個非線性函數而得到一個純量結果的神經元。

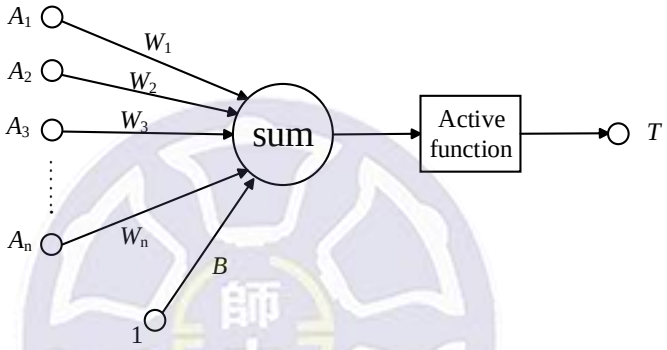

圖 2. 1 神經元示意圖

類神經網路是一種具有經驗累積的數學運算模式，會透過已知的資料庫建立輸入以及輸出資料之對應關係，也就是說類神經網路需要有輸入資料之外，也需要最後的輸出資料，且利用這些輸入和輸出之間的複雜運算關係建構運算模型。在現今類神經網路主要可協助解決特徵辨識、特徵預測及分析、辨識最佳化等問題。

為了能夠讓類神經網路能夠正確的辨識及預測分析，須透過不斷重複訓練的方式，使類神經網路能夠記憶並學習，而剛開始訓練時輸出的結果會和目標值有所差異，且差異度可能會非常大，類神經網路會透過訓練次數的增加逐漸調整權重值，使輸出結果會經由訓練次數的增加會越來越接近於目標值，當輸出值和目

標值的誤差趨近於一個最小值，則之後的訓練對於類神經網路是無效的，也代表類神經網路已經訓練完成，此時的權重值會使神經元產生最佳的輸出結果。類神經網路經過不斷的重複訓練,可以記錄下所有所需資訊,提高辨識影像的準確率。

類神經網路有兩種學習方式，第一種為監督式學習網路，第二種為非監督式學習網路。監督式學習網路是透過已知的輸入和期望的輸出所建立的類神經網路，它的特點是透過期望的輸出和實際的輸出做比較，當兩者的差異較大時，監督式學習網路會修正加權值，使實際輸出和期望輸出差異越來越小，小到一個最小值則會停止修正的動作。而在修正的過程中所使用的資料為訓練資料，由於此類網路需要不斷的訓練學習，並將加權值做修正，讓其值為最佳化，所以需要大量的訓練資料以利於學習。另外當訓練完成時則需要測試資料，而測試資料須與訓練資料完全不同，才可證明此種網路是否已完成。非監督式學習則是在無法預測資料輸出的情況下，只利用輸入資料並藉由這些資料的規則性建構而成，可自行判斷輸入資料裡的哪些特徵是重要的或者是哪些是可以忽略的，利用此特性可將資料做分群處理。而最後的輸出資料的型態完全取決於資料本身的特性與形態。非監督式網路經常會被用來做前處理動作，將資料的重要特徵擷取出來，而不重要的可直接忽略。一般來說類神經網路的分析資訊是屬於監督式學習，集群分析則是屬於非監督式學習。

第二節　類神經網路-全連結

全連結是類神經網路最經典的架構，屬於淺度訓練。全連結的架構通常包含二層，分別是輸入層及輸出層。輸入層為全連結最前面的一層，也就是我們所給的輸入資料，這全連結的所有輸入資料皆會影響到後面的輸出。而輸出層，也就是我們最後運算的輸出結果，而輸出結果的數量通常會由我們要辨識的數量所決定。全連結的運算由多個乘法和加法組成，也就是說，它的訓練運算過程簡單，只做乘法和加法，下圖 2.2 為全連結示意圖。

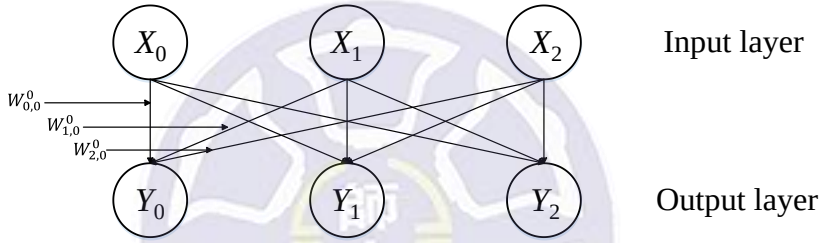

圖 2. 2 全連結示意圖

上圖 2.2 為全連結示意圖，由圖可知，因為是全連結運算，前一層每一個神經元皆會影響下一層的所有神經元，而下一層的每一個神經元又會影響到下下一層的所有神經元，以此類推。上圖的 X_0、X_1、X_2 代表輸入層，也就是輸入神經元；；Y_0、Y_1 及 Y_2 為輸出層，也就是最後的輸出神經元；而$W^0_{0,0}$、$W^0_{1,0}$、$W^0_{2,0}$分別為輸入層神經元對應於輸出層神經元之關係權重，上標數字為第幾層之權重，而下標數字則是輸入的第幾個神經元對應到輸出的第幾個神經元。從上圖看來，輸入層神經元有 3 個，而輸出層神經元有 3 個，所以從輸入層至輸出層的權重共有 9 個。而此全連結的運算方式如下列公式(2.2.1)：

$$Y_0 = X_0 \times W_{0,0}^0 + X_1 \times W_{1,0}^0 + X_2 \times W_{2,0}^0 \quad (2.2.1)$$

$$Y_1 = X_0 \times W_{0,1}^0 + X_1 \times W_{1,1}^0 + X_2 \times W_{2,1}^0 \quad (2.2.2)$$

$$Y_2 = X_0 \times W_{0,2}^0 + X_1 \times W_{1,2}^0 + X_2 \times W_{2,2}^0 \quad (2.2.3)$$

經由此運算式可知，Y_0、Y_1 及 Y_2 皆會由 X_0 、X_1 、X_2 乘以對應之權重之後相加而來， 根據上述之運算式子便可運算出最後的輸出神經元，這是最基本的只有一個全連結層的例子。

而在其他的運算中，全連結可能不只一層，也可能包含好幾層，以下圖 2.3 為例，此圖為全連結為 2 層的架構。H_0、H_1、H_2 代表第一個全連結的輸出，也就是中間的暫時結果

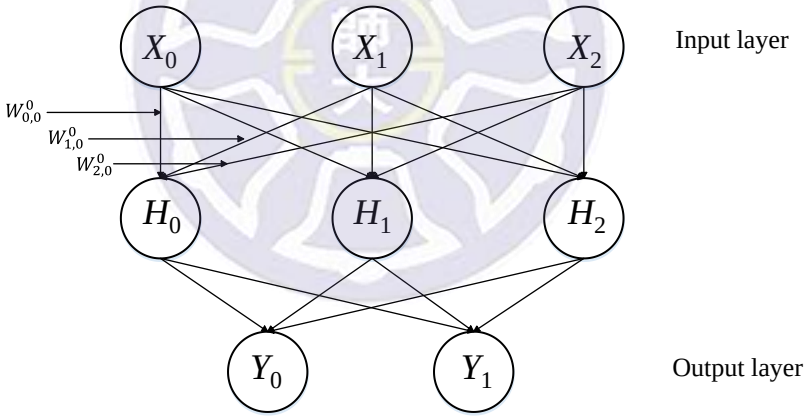

圖 2. 3 為全連結 2 層全連結之架構圖

最上層為第一層全連結層的輸入資料，最下層為第二層全連結層的輸出結果，也就是最後的輸出結果。由上圖可知第一層全連結有 3 個輸入神經元及 3 個輸出神經元，分別以 X_0、X_1、X_2 及 H_0、H_1、H_2 表示；而第二個全連結層的輸入神經元為 3 個，輸出神經元則有 2 個，分別 H_0、H_1、H_2 及 Y_0、Y_1 表示；由上圖可知，此架構有 2 層全連結層，運算式子如下：

$$H_0 = X_0 \times W_{0,0}^0 + X_1 \times W_{1,0}^0 + X_2 \times W_{2,0}^0 \quad (2.2.4)$$

$$H_1 = X_0 \times W_{0,1}^0 + X_1 \times W_{1,1}^0 + X_2 \times W_{2,1}^0 \quad (2.2.5)$$

$$H_2 = X_0 \times W_{0,2}^0 + X_1 \times W_{1,2}^0 + X_2 \times W_{2,2}^0 \quad (2.2.6)$$

$$Y_0 = H_0 \times W_{0,0}^1 + H_1 \times W_{1,0}^1 + H_2 \times W_{2,0}^1 \quad (2.2.7)$$

$$Y_1 = H_0 \times W_{0,1}^1 + H_1 \times W_{1,1}^1 + H_2 \times W_{2,1}^1 \quad (2.2.8)$$

由上述運算式可發現，2 層全連結層和只有一層全連結層之運算方式是一樣的，只是每多一層全連結層，就需要多做一次的全連結運算，而假如某一層的全連結之輸入層或者是輸出層的神經元較多，則需要計算的神經元會增加，進而會造成的是計算複雜度變高。

上述架構為全連結從輸入層至輸出層之運算過程，輸入神經元 X_i 表示之，i 為神經元個數，輸出神經元以 Y_j 表示，j 為神經元個數，圖上的$w_{i,j}^{l-1}$為連結輸入輸出之權重，透過圖上也可以發現，除了將所有輸入和權重所乘出來的結果做累加之後，運算完最後還需經過一個 function 運算，運算完之後才是我們最後所要的結果。

第三節　多層全連結架構

在上一節我們提到了只有 1 層全連結層及 2 層全連結層的架構，而當然全連結層可以更多層，我們以圖 2.4 為例，

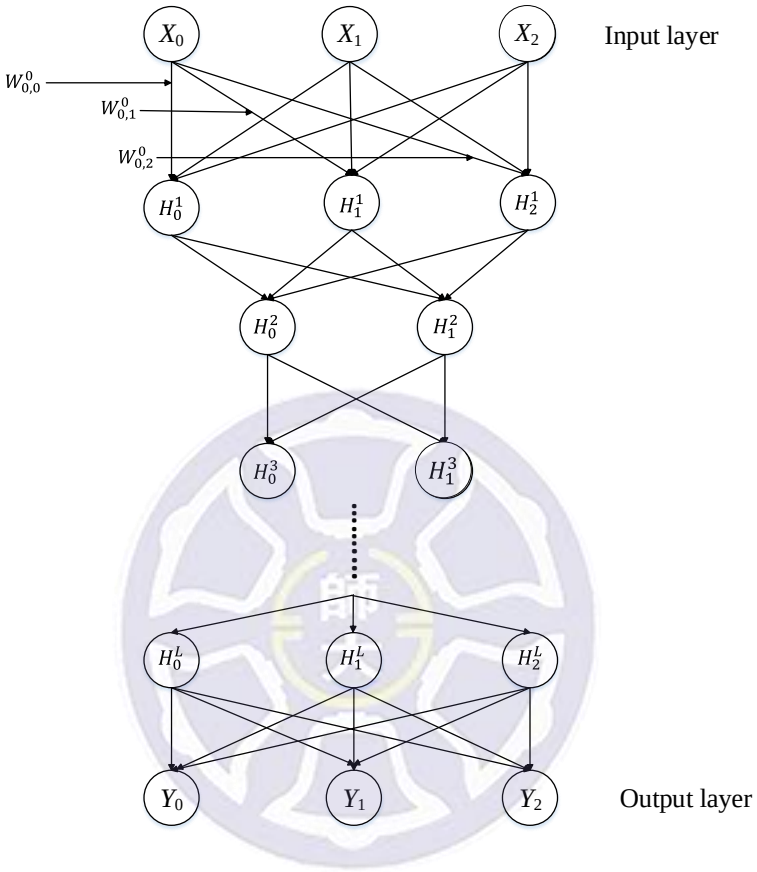

圖 2. 4 多層全連結層架構圖

由此圖可知，此架構除了有輸入層及輸出層之外，更重要的是它在輸入層和輸出層的中間夾雜了 L+1 層的全連結層，全連結層的層數越多，相對的神經元的個數越多，而神經元對應之權重也會越多，神經元跟權重越多，造成的是計算複雜度會越高，且也因為是全連結的架構，我們也無法去簡化它的計算複雜度，除此之外，在先前的第一章第二節也提到了，增加中間全連結層的層數對於最後運算出來的輸出結果，並不會有太大的影響，也就是說其實在全連結層只有 2 層或是 3

層的時候，辨識結果可能已經呈現停滯的情況了，更何況我們將中間的全連結層設定為 *L*-1 層，對最後的運算結果不會有任何的幫助，因此我們在現有的文獻裡找到了一個更好的方法，就是在只有 2 層的全連結架構的前面，增加摺積層，而摺積神經網路(Convolutional Neural Networks)就是一種摺積層的架構。藉由此方式，可有效的增加我們最後運算出來結果的準確率，並且使全連結的辨識結果也不會呈現停滯的狀況。

第四節　摺積神經網路-LeNet-5

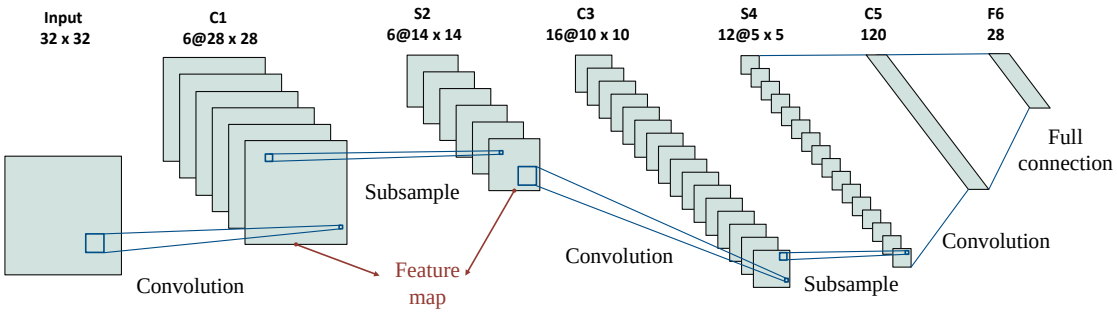

圖 2. 5 LeNet-5 架構圖

LeNet -5 是摺積神經網路的其中一種架構，圖 2.5 為 LeNet -5 之架構。一個完整的 LeNet-5 除了輸入資料之外還會有 C1、S2、C3、S4、C5 和 F6 這些層級，而這幾層由三種不同的層級重複組合而成，分別為摺積層、採樣層以及全連結層三種層級，每一個層級分別會執行不同的運算處理。例如摺積層，圖中的 C1 和 C3 為摺積層。摺積層主要是在做特徵截取，他會接受前一層的輸入特徵影像，將這些影像做為輸入資料，並把這些輸入影像和權重做摺積運算，並且一次透過多個核心來計算，而輸出的影像資料則會在繼續給下一層做運算，也就是說，摺積層的輸出資料會給採樣層當作輸入資料。那我們以下列式子表示摺積層的運算，假設 $X^{p,r}$ 和 $Y^{q,r}$ 為第 r 層的第 p 個的輸入特徵影像和第 r 層的第 q 個的輸出特徵影像，而 $W^{p,q}$ 是和輸入特徵影像做摺積運算的權重，*為摺積符號，M 和 N 為輸入和輸出的影像數量。

$$Y^{q,r} = \sum_{P=1}^{M} X^{p,r} * W^{q} \ , q = 1, \dots, N \qquad (2.3.1)$$

另外我們將核心和權重運算的尺寸假設為 $K \times K$，當 K 為奇數上述式子可改寫如下：

$$Y_{i,j}^{p,r} = \sum_{p=1}^{M} \sum_{m=-\lfloor K/2 \rfloor}^{\lfloor K/2 \rfloor} \sum_{n=-\lfloor K/2 \rfloor}^{\lfloor K/2 \rfloor} X_{(i+m),(j+n)}^{p,r} W_{m,n}^{p,q}, q = 1, \ldots, N \quad (2.3.2)$$

$X_{i,j}^{p,r}$ ，$Y_{i,j}^{q,r}$和$W_{i,j}^{p,q}$分別為 $X^{p,r}$ 、$Y^{p,r}$ 及 $W^{p,q}$ 的第(i,j)個元素，並舉例圖 2.6 假設核心運算的尺寸為 3 × 3 並有 4 個輸入特徵影像及 4 個輸出特徵影像。

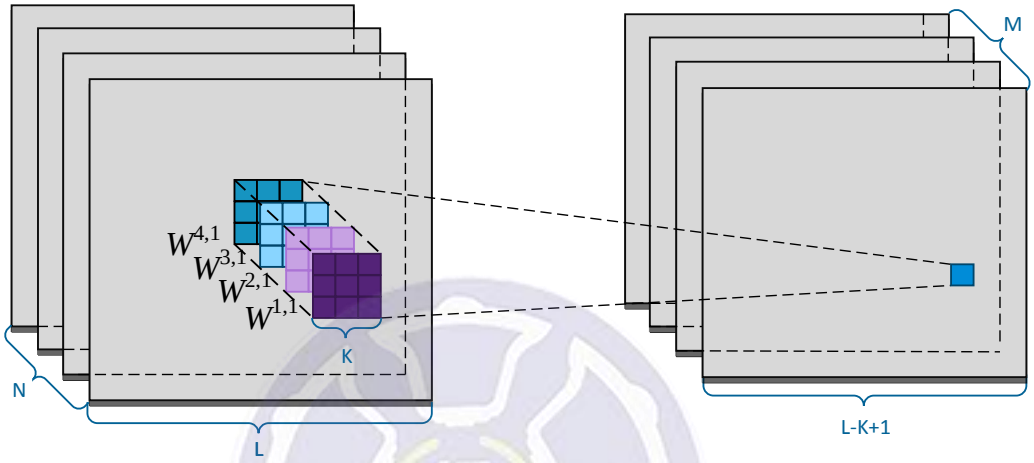

圖 2. 6 摺積運算架構圖

做完摺積運算之後還要做一個非線性的運算，就是 ReLU 的動作，而式子如下：

$$X_{i,j}^{p,r+1} = \text{ReLU}(Y_{i,j}^{p,r}) \quad (2.3.3)$$

當$Y_{i,j}^{p,r}$大於等於 0 時，$X_{i,j}^{p,r+1}$會等於$Y_{i,j}^{p,r}$ ，如果$Y_{i,j}^{p,r}$小於 0 時，則$X_{i,j}^{p,r+1}$等於 0。

再來是採樣層，圖中的 S2 和 S4 為採樣層。採樣層的作用是將縮小特徵影響的尺寸。方式為每次取一個 2 × 2 的區塊做採樣，而採樣的方法為 Max pooling，依序將所有的影像做完。最後為全連結層，經過全連結層的運算之後，輸出值就是為最後的所要輸出。而全連結層在本論文的第二章第二節有做論述，在此就不多做說明。

最後為本論文之重點全連結層，在 LeNet-5 裡全連結為最後兩層，也就是 C5 及 F6，並透過圖 2.7 對全連結層詳細解說。

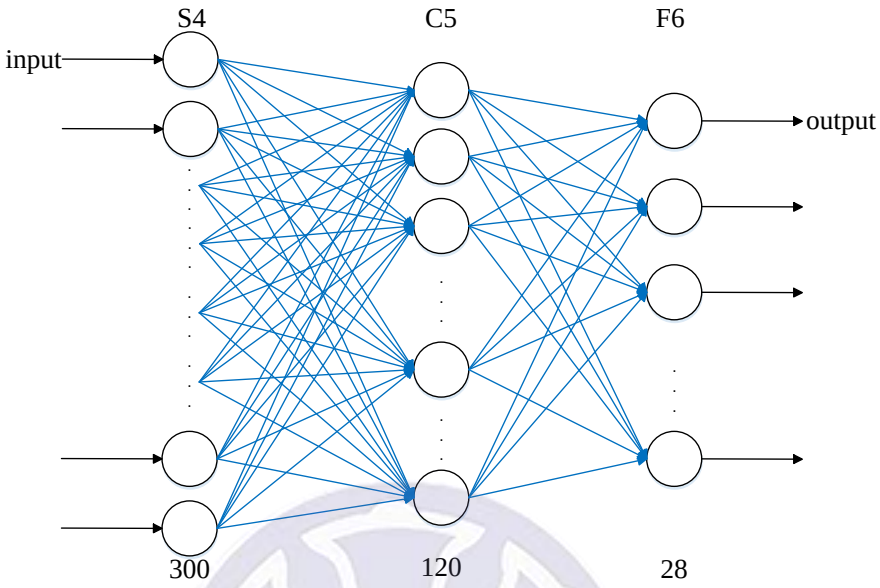

圖 2. 7 LeNet-5 之全連結層示意圖

由圖 2.7 可知，LeNet 有兩個全連結層，分別是 S4-C5 及 C5-F6，而 S4 的輸入資料為 300 筆，C5 輸出資料為 120 筆，F6 的輸出資料有 28 筆。根據圖 2.8，下列將介紹 S4 至 C5 之全連結層。

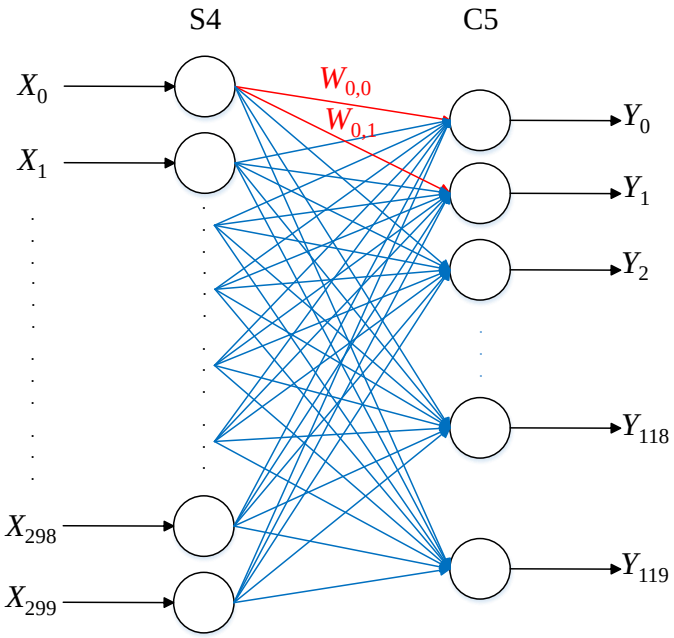

圖 2. 8 S4 至 C5 示意圖

此部分之全連結的輸入資料有 300 筆，輸入以 X_i 表示，分別為 X_0、X_1 至 X_{299}，而輸出有 120 筆，以 Y_j 表示，分別為 Y_0、Y_1 至 Y_{119}，權重以 $W_{i,j}$ 表示，i 為第幾個輸入，j 為第幾個輸出，也就是說 $W_{0,0}$ 為 X_0 對應到 Y_0 之權重，而權重數有 300×120 個，總共有 36000 個。輸出產生之方式由下列式子表示，

$$Y_j = \sum_{i=0}^{299} X_i \times W_{i,j} \tag{2.4.1}$$

透過此運算式子即可運算出 C5 層所有的輸出結果，而緊接著說明 C5 至 F6。

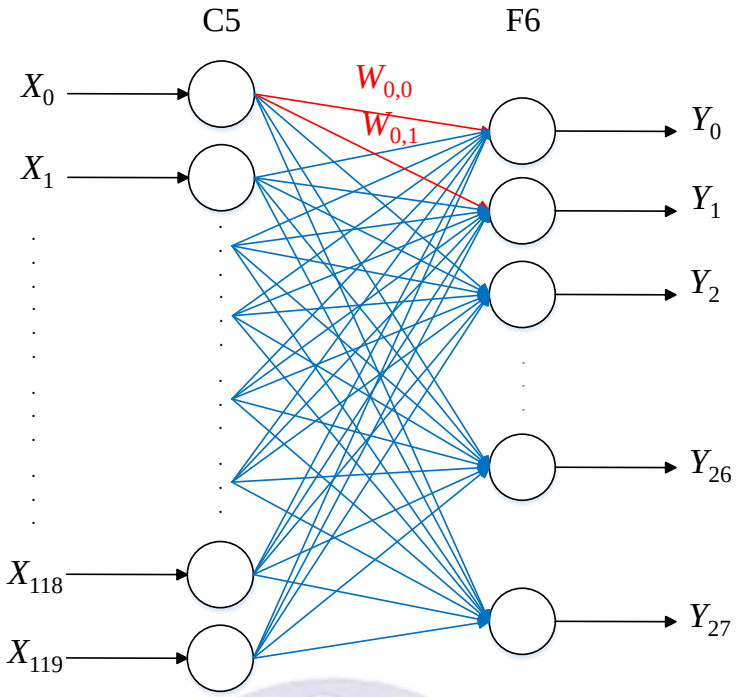

圖 2. 9 C5 至 F6 示意圖

這部分的輸入資料有 120 筆，以 X_0 至 X_{119} 表示，而輸出共有 28 筆，也就是最後所要辨識的 28 個人，以 Y_0 至 Y_{27} 表示，而運算方式和式(2.4.1)一樣，透過此運算式即可得到最後的 Y_0 至 Y_{27} 之結果，取得最後 28 筆輸出結果後，還需判斷第幾筆為最大值，透過最大值之判斷可辨識出此次輸入資料為 28 個人裡的第幾個人之臉部圖片。

第五節　FPGA 系統整合設計

在現今，積體電路之發展已經達到成熟的階段，也因為如此，積體電路的產品生產的越來越快，但也因為這個原因使產品的生命週期下降，經常有產品替換的情形發生。現在消費者對產品功能的要求越來越高，造成產品的是設計複雜度更加複雜，這也使專案的規模呈現過大的趨勢，因為如此，工程師們需要一個可以便利並且設計彈性高的環境來實現專案設計，達到可控制對成本及功率消耗等多項的指標。而 FPGA 即是符合上述需求之環境，工程師們可在此環境下設計出符合消費者所要求的產品。

本論文使用硬體描述語言(Hardware Description Language, HDL)搭配可重複程式設計的 FPGA 晶片設計電路，可在設計晶片電路時透過所提供的功能，像是繞線佈局與合成等諸多功能，反覆快速地將電路燒進 FPGA 開發板上進行實驗測試，及時達到檢驗系統電路是否正確的效果，不需要再負擔實際合成晶片電路的高成本分險，也有效地大量減少工程師開發電路的時間，提升電路的設計效率。因此,FPGA 在現在已經逐漸成為許多科技公司開發晶片電路與驗證的主要工具，也是現代 IC 設計的技術主流。

本論文使用的系統架構為 Altera 之 Qsys 提供的 SoPC(System on Programmable-Chip)架構，SoPC 是現在電腦輔助設計技術、EDA 技術和大型積體電路高度發展的產物，目的就是將嵌入式處理器系統、介面系統等各種數位系統可在單一 FPGA 中實現，達到設計更容易、快速等優點。SoPC 系統因為涉及

了底層的硬體系統和相對應的軟體設計，因此對於 SoPC 的開發者的知識範圍會有更廣的要求，除了硬體描述語言和 FPGA 元件之外，還須熟悉像是計算組織與介面、C 語言、系統晶片建構與測試、嵌入式系統開發等相關知識，這些都是必需的。

Altera SoPC 的系統可分為下列三種方式實作：

1. 於 FPGA 中嵌入硬核心的 SoPC，如在 FPGA 中預先植入嵌入式處理器如 ARM922T 等。
2. 於 FPGA 中嵌入軟核心的 SoPC，如在 FPGA 使用軟核心的處理器像是 NIOS 等。
3. 基於 HardCopy 技術的 SoPC，Altera 可將我們所在 FPGA 所完成的電路設計直接轉換成 ASIC。

由上述三種實作方式可發現，使用 Altera 所提供的 SoPC 平台來設計是相當有彈性的，可根據我們所需的不同要求配置不同的核心，而本論文使用第二種方式實作，以 SoPC 設計包括以 Nios 核心處理器作為核心的嵌入式系統的硬體配置、設計、模擬、軟體的設計及除錯等。

Altera 也提供 SoPC Builder 系統發展工具提供一個 GUI 介面讓設計者能快速在系統之間做互相整合。SoPC Builder 會經由自動程序，使設計者可將元件與 Nios 做連接。而對於每一個 Nios 系統的 SoPC 系統會自動產生一個軟體程式庫，使軟體開發更加快速、容易。

第三章 系統架構

本章節將著重於本論文類神經網路全連結層硬體架構之討論，分別會為全連結層及摺積層電路架構做介紹。以下為各節之介紹重點，第一節介紹整體系統架構，第二節介紹全連結電路架構，第三節介紹全連結電路及摺積層電路整合之架構。

第一節　整體系統架構

本節主要介紹本論文摺積神經網路全連結電路於 Qsys 系統實現在 FPGA 的整體架構，使用的硬體單元包括 Nios II Processor 做為系統電路的核心處理器，On-Chip Memory 則是用來儲存程式，另外還有儲存全連結電路所有的權重值，而輸入的影像資料則是儲存於電腦，經由寫入位址來啟動影像資料的傳輸，而傳輸方式是透過 JTAG 線直接從電腦傳入 FPGA 電路裡做運算，運算完畢再經由讀取位址的方式將運算結果同樣經由 JTAG 從 FPGA 傳回電腦。而全連結電路為本論文主要探討之電路。

第二節　全連結電路

本節主要討論本論文之重點架構全連結硬體電路，並根據第二章第二節全連結演算法做為實現電路設計之基礎理論。全連結電路可分為五個單元，一為儲存前一層運算之輸出結果的緩衝單元，二為全連結電路的主要運算單元，而這個運算單元包含了乘法及加法的運算，三為儲存要和輸入影像資料運算之權重的記憶體單元，四為儲存部分輸出的緩衝單元，而最後一個為儲存偏移值的記憶體單元，此表格會透過位址產生器，使最後輸出會和相對應之偏移值做相加。

而在先前的第二章有提到，第一層全連結所要運算的輸入層之輸入資料共有 300 筆，而輸出有 120 筆；那假設一次要將所有的輸入及所有的輸出運算完畢，因為是全連結，每個輸出都由 300 個輸入和權重相乘後再相加而得，所以需要有 300×120 個乘法器及多個加法器，在電路裡乘法器的面積其實有一定的大小，而在此電路共會有 36000 個乘法器，面積消耗會非常之大，大到無法將電路放進 FPGA 板子裡，所以此方式無法實現。圖 3.1 表示為此運算方式，

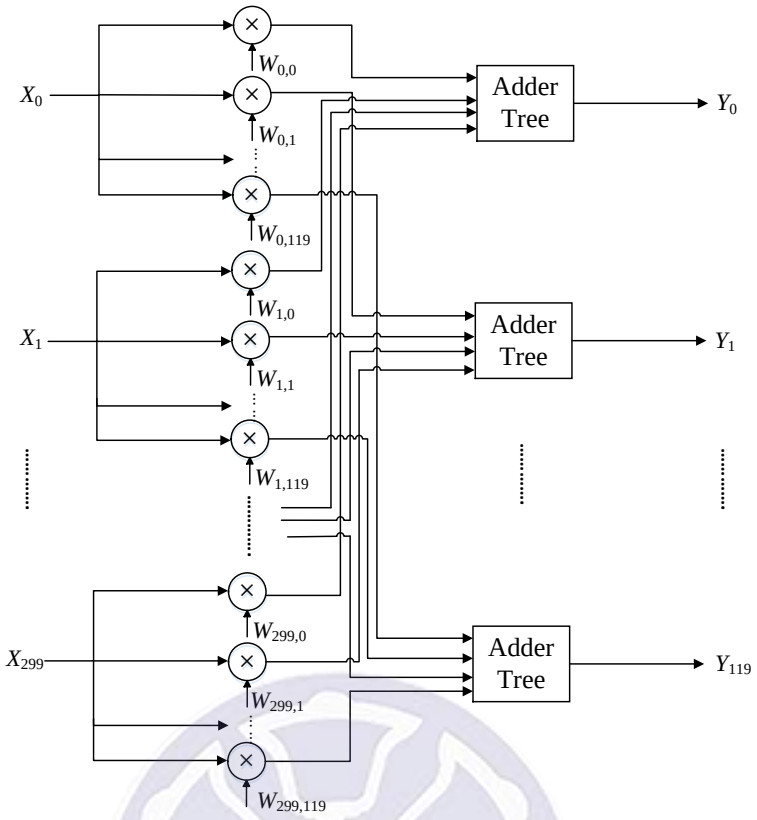

圖 3. 1 所有輸入計算所有輸出之架構圖，輸入為 300 個，輸出為 120 個

透過此圖，可看到光是前面的乘法器就有一長串，共有 36000 個，而加法器又是以樹狀圖表示，總共會有 120 個加法器樹狀圖，每一個樹狀圖的輸入會有 300 筆，持續的相加，直到最後的輸出產生才結束，也是一長串，造成此運算方式無法以電路的方式實現，所以我們需要將運算方式改變。

而那假設一次只需要運算出一個輸出之結果，這樣一次需要 300 個乘法器及 1 個加法器樹狀圖，而這 300 個乘法器也會佔據了 FPGA 大部分的面積，還是無法將此電路放進 FPGA。以圖 3.2 表示，

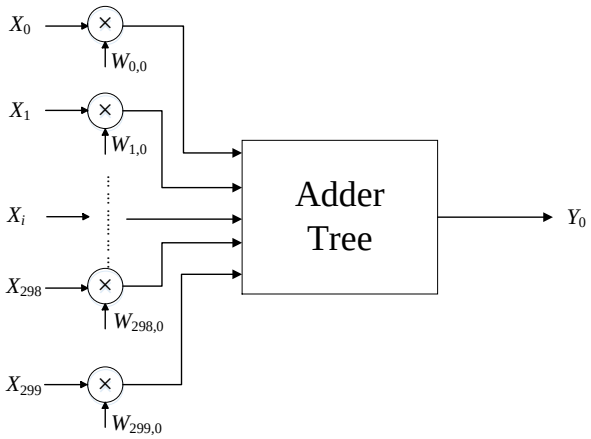

圖 3. 2 運算一個輸出之架構圖，輸入為 300 個，輸出為 1 個

以此運算方式設計電路，雖然加法器樹狀圖只有一個，但樹狀圖之輸入同樣還是 300 個，而乘法器還是有 300 個，對於面積之消耗還是過大，無法將此運算方式以電路呈現。

那如果依序一次取兩筆輸入去算出一筆的輸出，輸入資料個數以 *s* 表示，輸出資料個數以 *t* 表示，*s*=2、*t*=1，並將部分結果儲存於 SIPO 緩衝單元，等到所有輸入都已經計算完畢，再用加法器將所有部分結果相加，持續重複相加的動作，這樣即可得到完整的一筆輸出結果，但是每次依序取兩筆輸入，運算完之後再重複依序取兩筆輸入算出下一筆輸出，會造成記憶體重複存取的問題，除此之外，這個方式還是會有一個加法樹，且這個加法樹的輸入個數還是 300 個，導致資源消耗過高。圖 3.3 為以 SIPO buffer 為基礎之架構，並透過圖 3.4、3.5、3.6 敘述運作方式，並且說明此種方式之缺點。

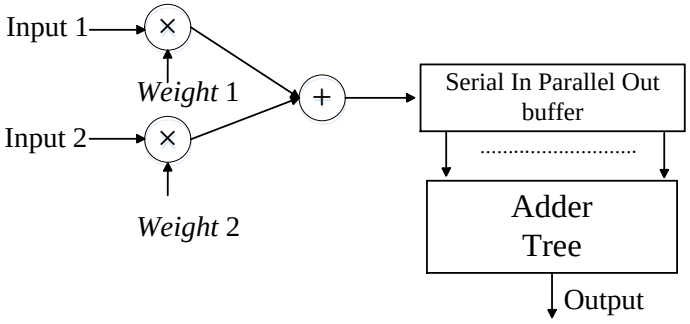

圖 3. 3 以 SIPO buffer 為基礎之架構圖，此架構 *s*=2 及 *t*=1

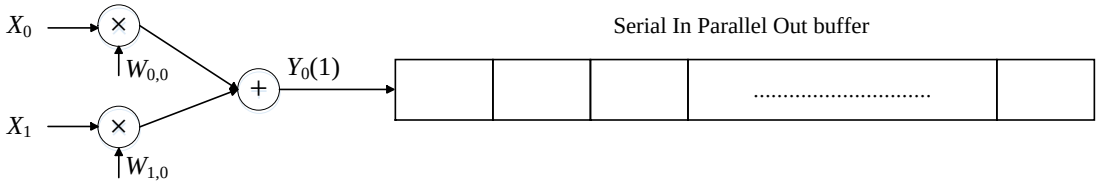

圖 3. 4 兩筆輸入運算一筆輸出部分結果電路圖

圖 3. 5 下兩筆輸入運算輸出部分結果電路圖

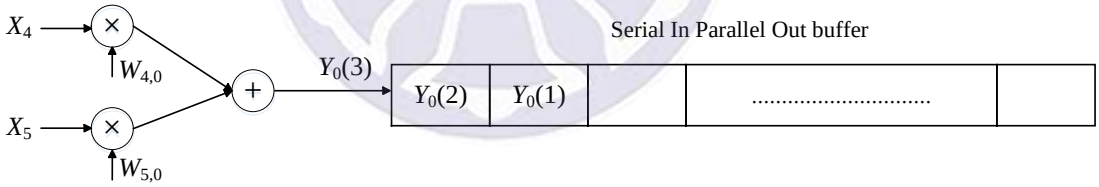

圖 3. 6 依序取兩筆輸入運算電路圖

直到運算到最後兩筆的輸入，以圖 3.7 表示，

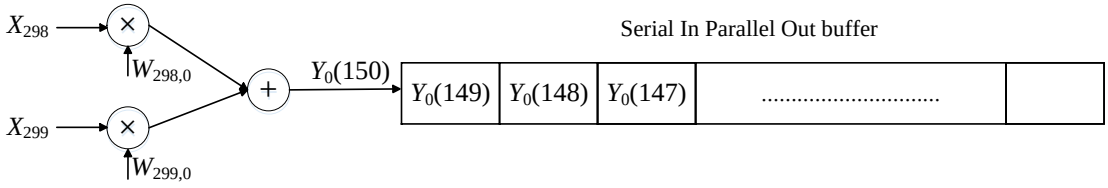

圖 3. 7 取最後兩筆輸入運算輸出部分結果電路圖

當填滿 SIPO buffer 時，buffer 的所有結果也會推進 Adder Tree 運算，並產生出第一個輸出，以圖 3.8 及 3.9 表示，

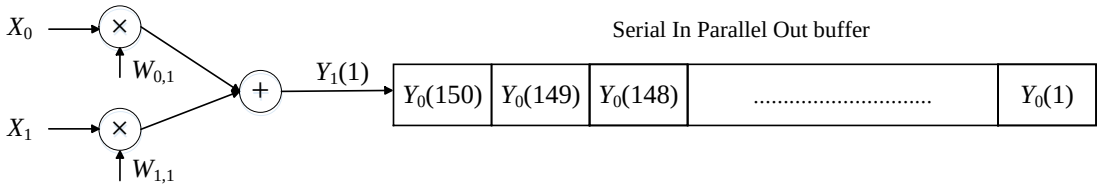

圖 3. 8 填滿 SIPO buffer 電路圖

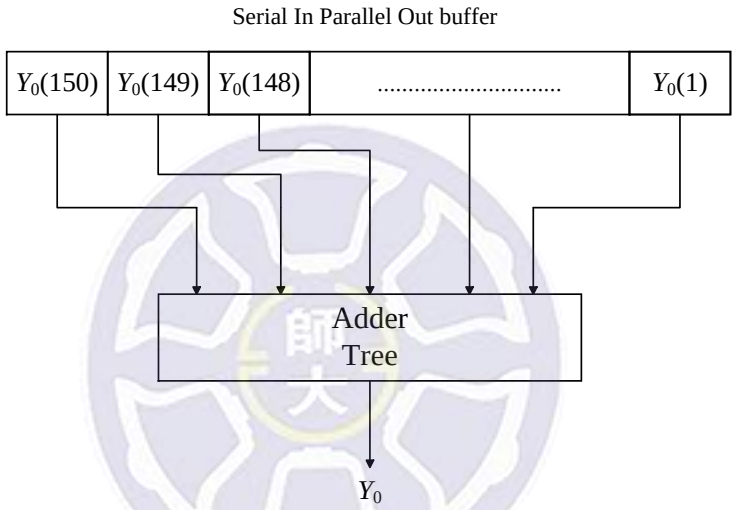

圖 3. 9 使用加法器將部分結果相加之電路圖，共 150 個部分結果相加

雖然這種方式只用到兩個乘法器及多個加法器，對面積消耗不大，但是可看到這種方式需要不斷地對記憶體重複存取 X_0 至 X_{299}，浪費許多存取的時間。

相反的假如一次取兩筆輸入，依序運算完對每一筆輸出的部分結果之後，將這些部分結果透過 SISO 緩衝單元儲存起來。此種 buffer 特性是資料會依序一筆進去一筆出來，資料進去此 buffer 之後，會持續往右位移，直至這兩筆輸入運算完畢。運算完之後再取下兩筆輸入運算，並算出此次部分結果，再將此次部分結果與先前緩衝單元儲存之部分結果做相加，重複此動作，即可得到最後之輸出結

果。這樣不僅不會造成乘法器太多的問題，也可以解決記憶體重複存取的問題。圖 3.10 為以 SISO buffer 為基礎的架構圖，紅色框線為 computation core 架構，並透過圖 3.11、3.12 及 3.13 敘述運作方式，

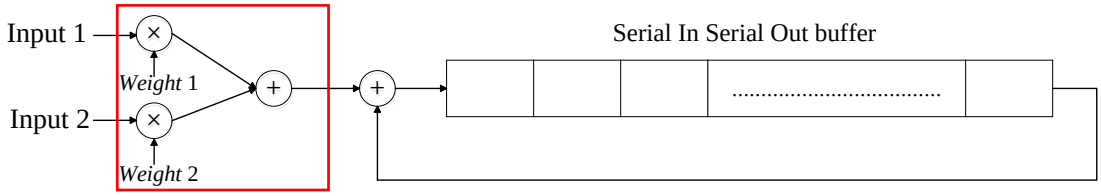

圖 3. 10 以 SISO buffer 為基礎架構圖，此架構 s=2 及 t=1

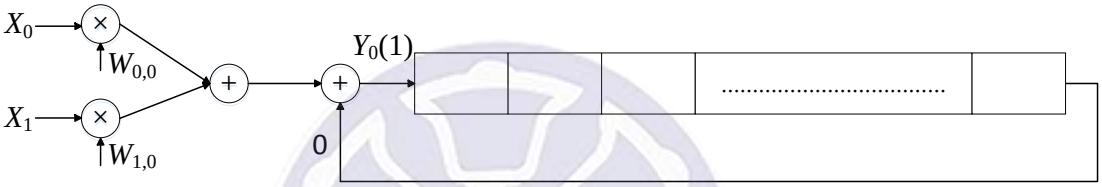

圖 3. 11 運算出第一筆輸出部分結果電路圖

圖 3. 12 運算出第二筆輸出部分結果電路圖

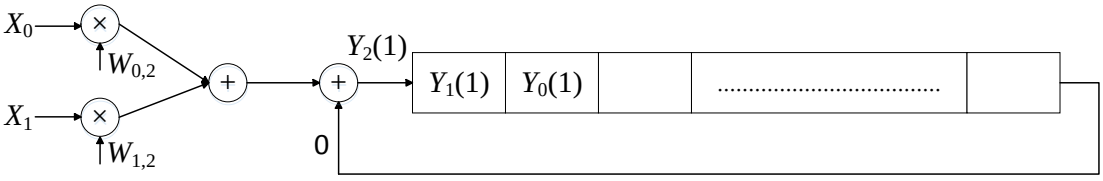

圖 3. 13 運算出第三筆輸出部分結果電路圖

依序重覆此動作，直到運算到最後一筆輸出，接著就取下兩筆輸入運算輸出部分結果，並將此次部分結果和先前運算之部分結果相加，並以圖 3.14、3.15、3.16 表示，

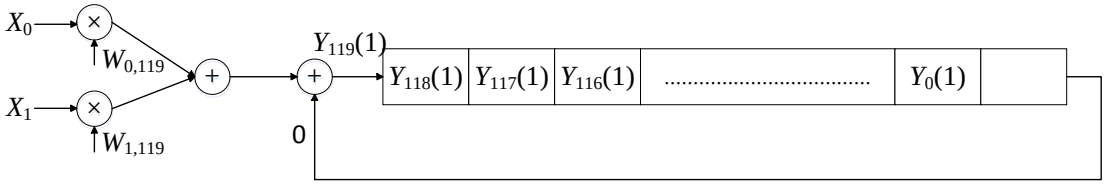

圖 3. 14 運算出最後一筆輸出部分結果電路圖

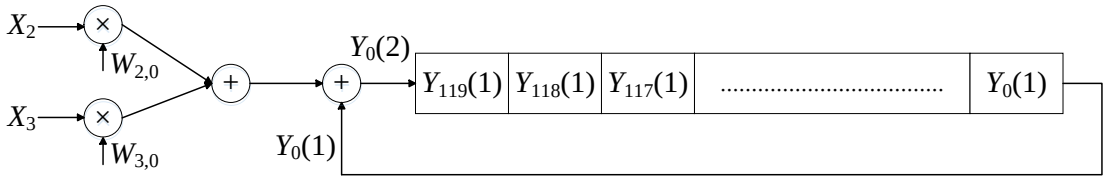

圖 3. 15 接著下筆輸入運算，並累加電路圖

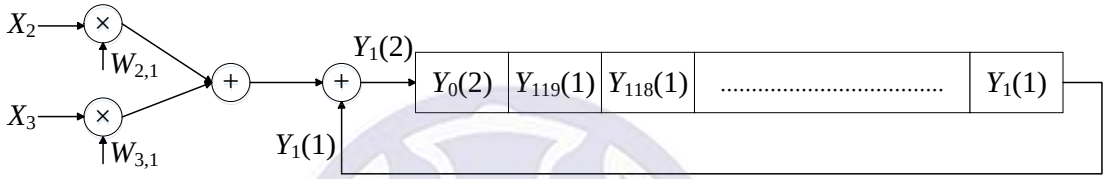

圖 3. 16 繼續運算並累加電路圖

當最後一筆輸入運算時，也代表最後一次是最後一次的累加，累加完的值為最後輸出，以圖 3.17、圖 3.18、圖 3.19 表示。

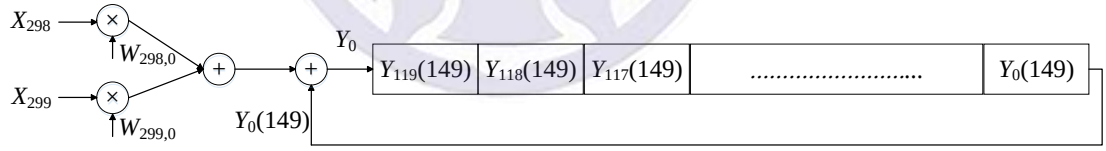

圖 3. 17 運算出最後的 Y_0 電路圖

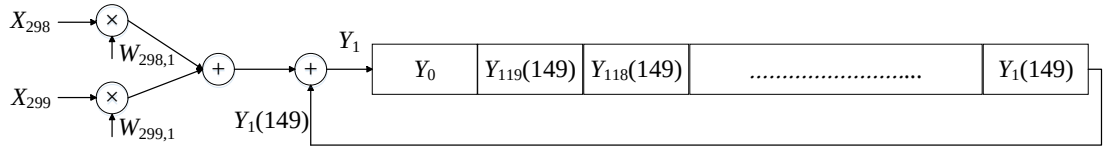

圖 3. 18 運算出最後的 Y_1 電路圖

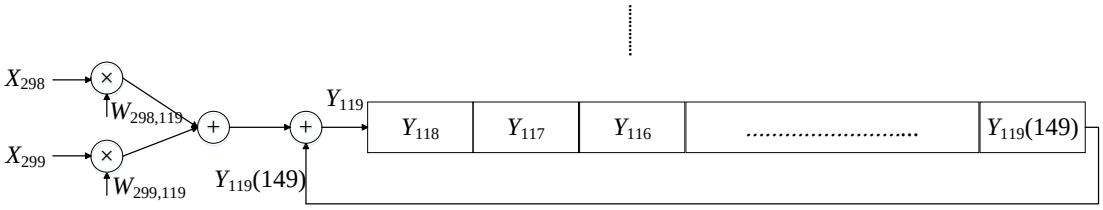

圖 3. 19 運算出所有結果電路圖

而可以每次取兩筆輸入運算出一筆輸出的部分結果，當然也可以一次取兩筆輸入運算出兩筆輸出的部分結果，只需要增加輸出緩衝單元個數。而使用 SISO buffer 為基礎的架構除了需要輸出緩衝單元之外，因為每次只取部分輸入運算，而其他未運算的輸入需要使用緩衝單元儲存起來，所以還需要儲存輸入資料的輸入緩衝單元，另外每次權重運算也是使用部分的權重運算，所以也需要將未運算的權重值儲存於記憶體單元，並利用位址產生器取得每次所要運算的權重值，藉由運用這些單元實現全連結電路架構，而最後經由綜合面積及執行時間的分析後，一次取四筆輸入並運算出四筆輸出的部分結果，這樣不僅能將電路放進 FPGA 之外，對面積消耗及執行時間也會是最佳的。下圖 3.20 為本論文全連結電路架構圖

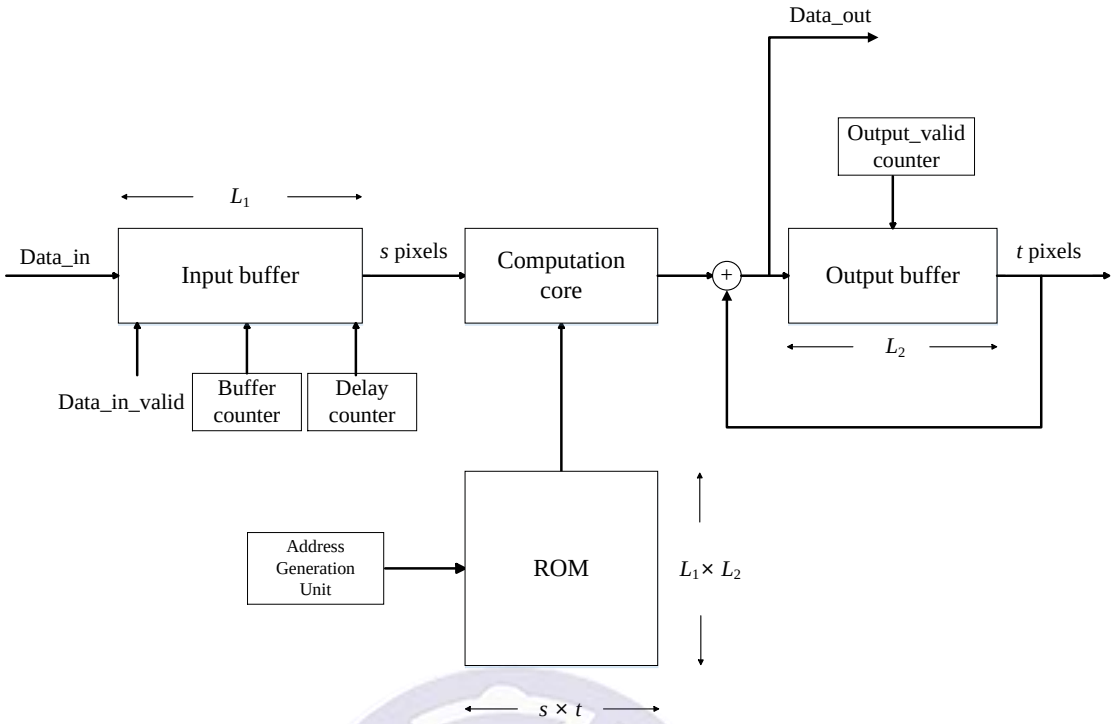

圖 3. 20 全連結電路架構圖

由圖 3.20 所示，Input buffer 為輸入緩衝單元，由位移暫存器組成，s 為運算之輸入個數，在此設計為 4 個，L_1 為輸入緩衝單元之長度，輸入層之輸入共有 300 個，一次輸入 4 個，所以輸入緩衝單元之長度為 75。Output buffer 為輸出緩衝單元，也是由位移暫存器組成，t 為輸出個數，也是設計為 4 個，L_2 為輸出緩衝單元之長度，輸出層共有 120 個輸出，一次輸出 4 個，所以輸出緩衝單元之長度為 30，下圖 3.21 為輸入及輸出緩衝單元。

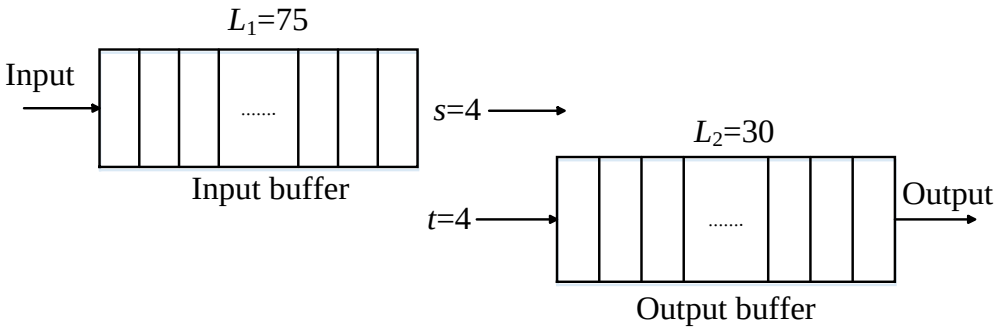

圖 3. 21 全連結電路輸入及輸出緩衝單元架構圖

ROM 為儲存運算權重之記憶體單元，此電路設計由 4 個輸入算出 4 個輸出，一次需要計算之權重為 16 個，所以儲存權重之記憶體單元需要有 16 個，而記憶體個數剛好為 $s \times t$，而權重共有 36000 個，分成 16 個記憶體單元儲存，每個記憶體單元長度為 2250，恰好為 $L_1 \times L_2$。Computation core 為運算單元，此運算單元包含了 16 個乘法器及 12 個加法器，透過乘法器及加法器運算後，進而得到 4 個輸出。圖 3.22 為 Computation core 運算單元架構

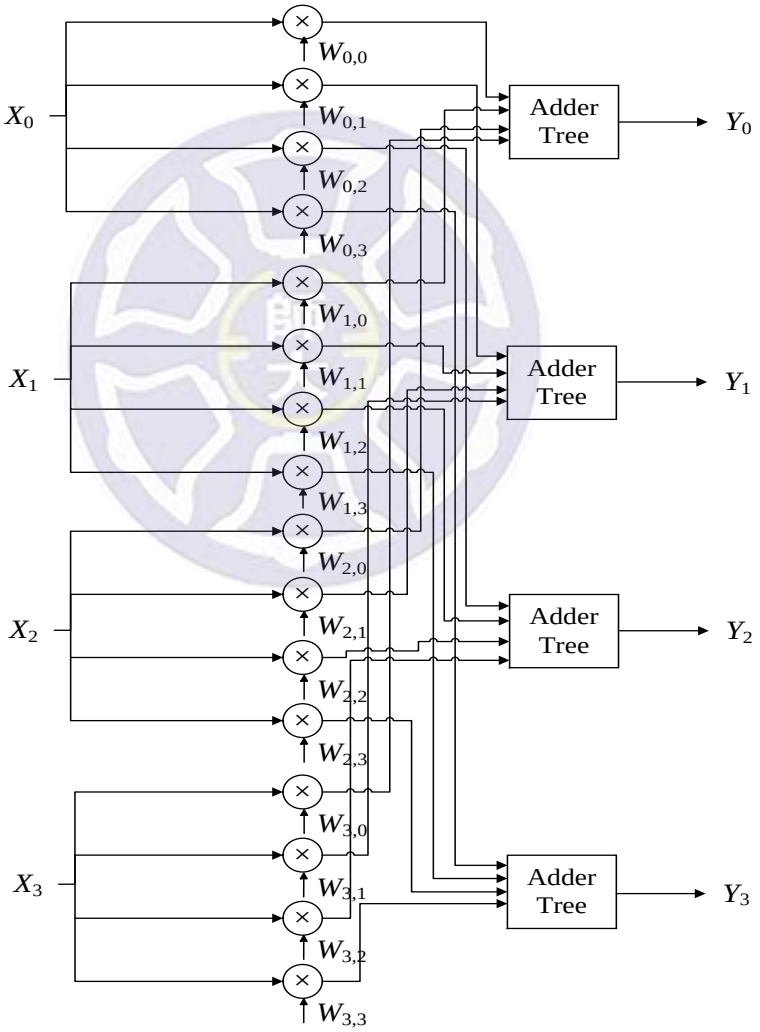

圖 3. 22 全連結電路 Computation core 架構圖

由於 4 個輸入運算 4 個輸出權重過多，論述複雜，在此舉例以 2 個輸入運算出 2 個輸出，也就是 s 及 t 各為 2，此例子共有 8 個輸入及 4 個輸出，即 S=8，T=4，權重數總共有 32 個，也就是 $S \times T$=32，分別存於 4 個記憶體單元，所以每個記憶體單元長度為 8，並透過此例子將電路運算之流程以圖的方式加以詳述。以下圖 3.23 為例，輸入共有 8 個，以 X_0 至 X_7 表示，輸出共有 4 個以 Y_0 至 Y_3 表示，權重共有 32 個，以 $W_{0,0}$ 至 $W_{7,3}$ 表示，而最左欄位記憶體位址。

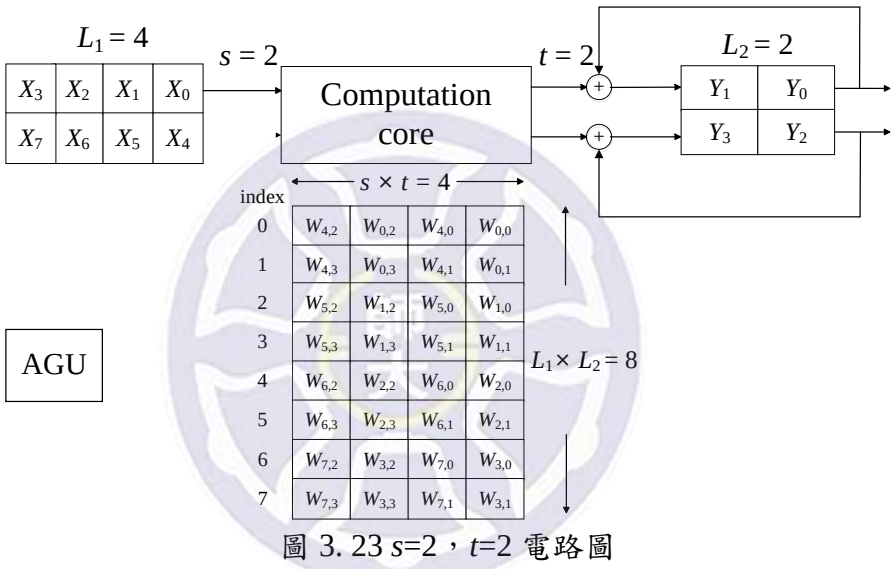

圖 3. 23 s=2，t=2 電路圖

Computation core 為運算單元，下圖 3.24 為 s=2，t=2 之 Computation core 架構圖。

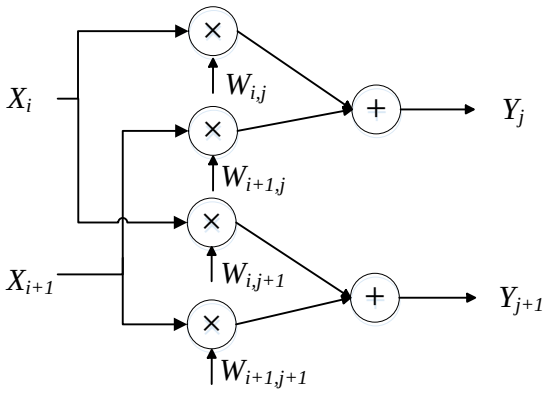

圖 3. 24 s=2，t=2 之 Computation core 架構圖

一開始輸入緩衝單元沒有任何資料，等到前一層結果存滿輸入緩衝單元後，此電路才開始運作，圖 3.25 為初始之狀態，

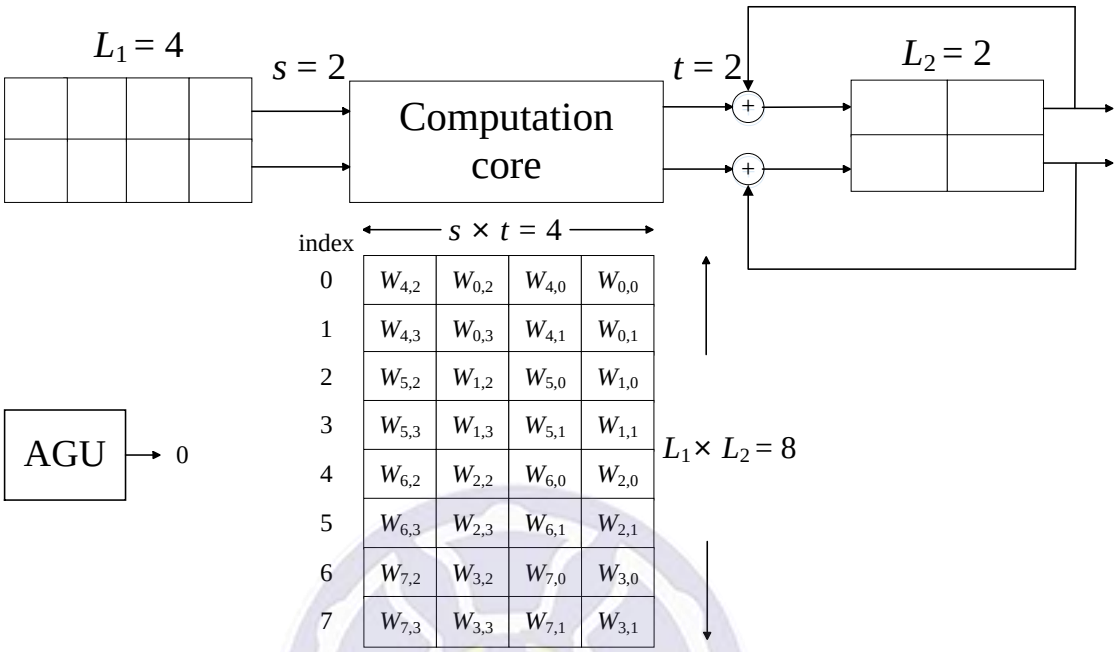

圖 3. 25 初始狀態電路圖

接著輸入開始依續存進輸入緩衝單元，並且由計數器計算，計算到填滿為止之後，發出訊號並同時啟動電路開始運作，以圖 3.26 表示，

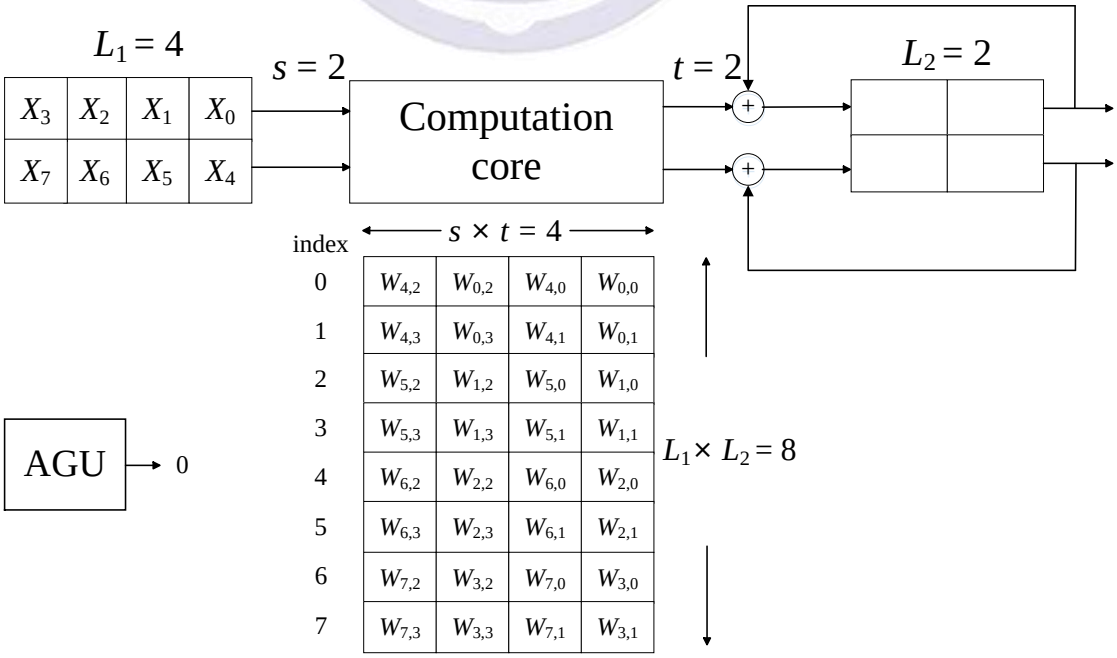

圖 3. 26 輸入緩衝單元填滿之狀態圖

啟動電路先由 X_0 及 X_4 算出 Y_0 及 Y_2 之部分輸出結果，以 $Y_0(1)$及 $Y_2(1)$表示，並將 $Y_0(1)$及 $Y_2(1)$輸出至緩衝單元，深色為此時間點運算資料，並由圖 3.27 表示。

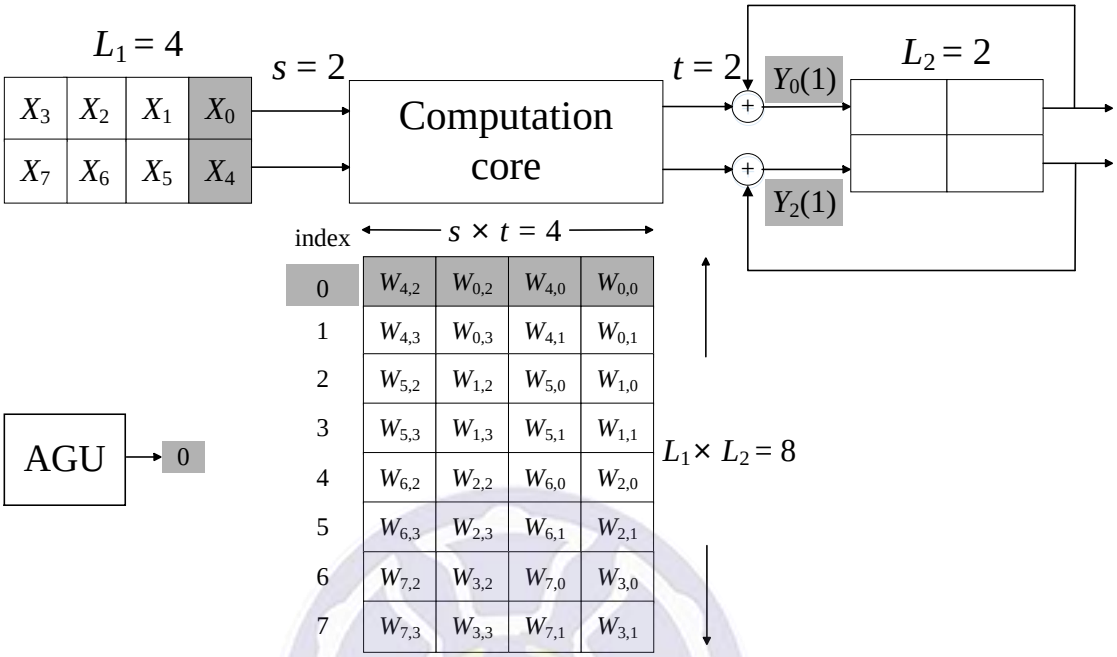

圖 3. 27 X_0 及 X_4 第一次運算電路圖

接著算出 Y_1 及 Y_3 之部分結果，得到 $Y_1(1)$及 $Y_3(1)$，並輸出至緩衝單元，以圖 3.28 表示。

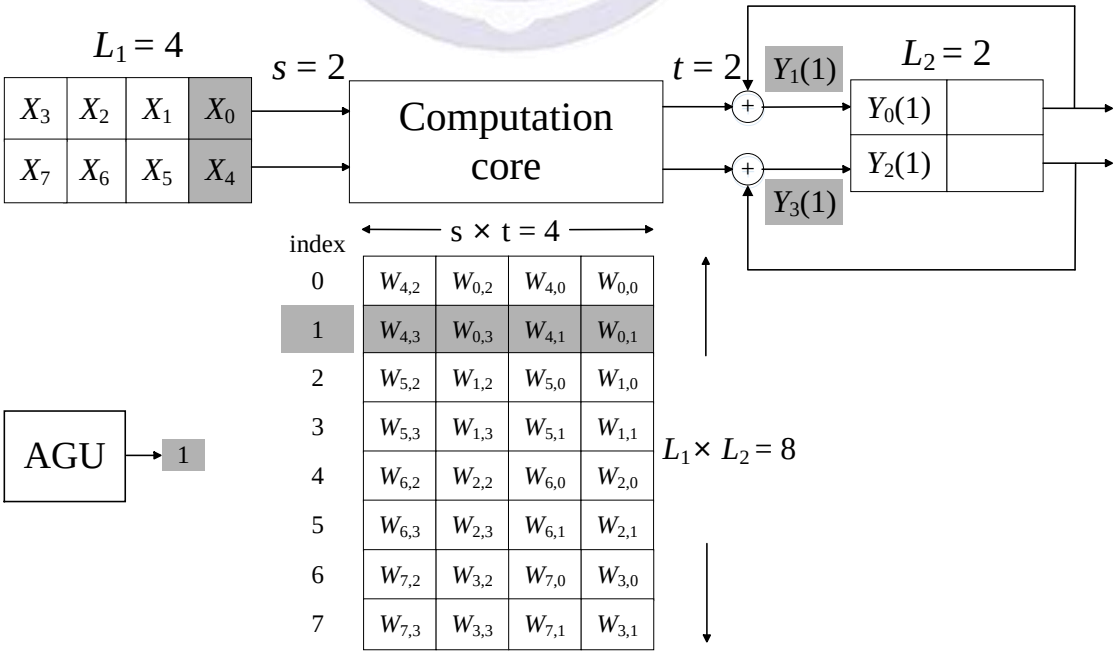

圖 3. 28 X_0 及 X_4 第二次運算電路圖

當 X_0 及 X_4 運算完之後，接著推入 X_1 及 X_5 運算，並將此次運算結果和先前算的部分結果做相加，得到 $Y_0(2)$及 $Y_2(2)$，以圖 3.29 表示。

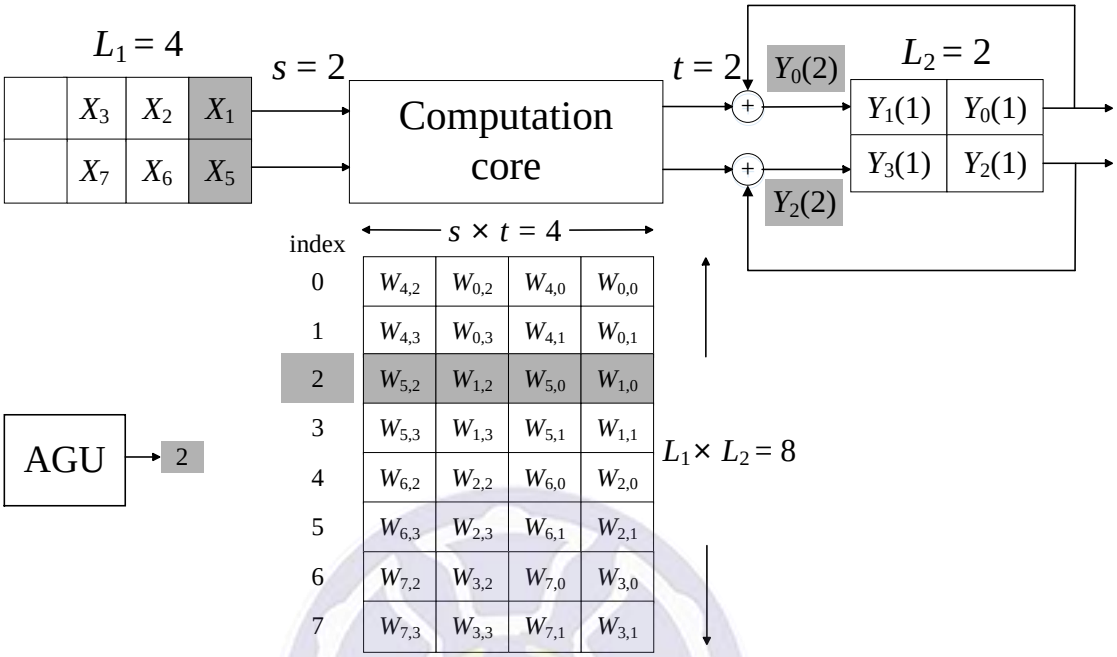

圖 3. 29 X_1 及 X_5 第一次運算電路圖

接著運算 X_1 及 X_5 對 Y_1 及 Y_3 的部分結果，並和先前的部分結果相加，並將結果輸出至緩衝單元，以圖 3.30 表示。

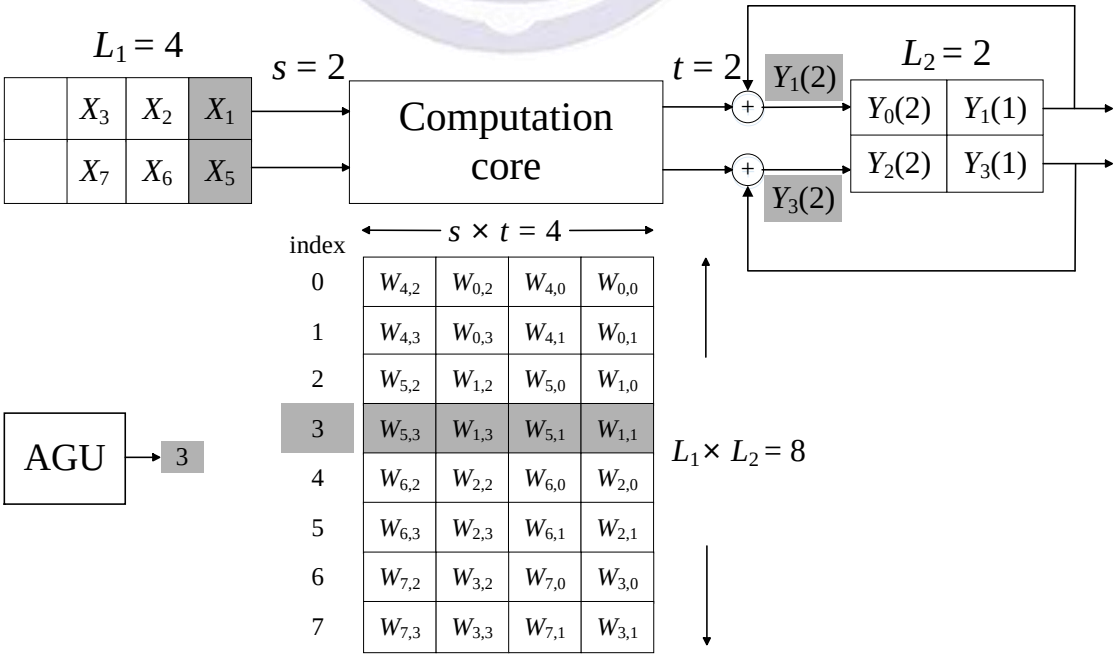

圖 3. 30 X_1 及 X_5 第二次運算電路圖

運算完接著將 X_2 及 X_6 推入至運算單元，運算結果同樣和先前結果做相加，得到 $Y_0(3)$及 $Y_2(3)$，以圖 3.31 表示。

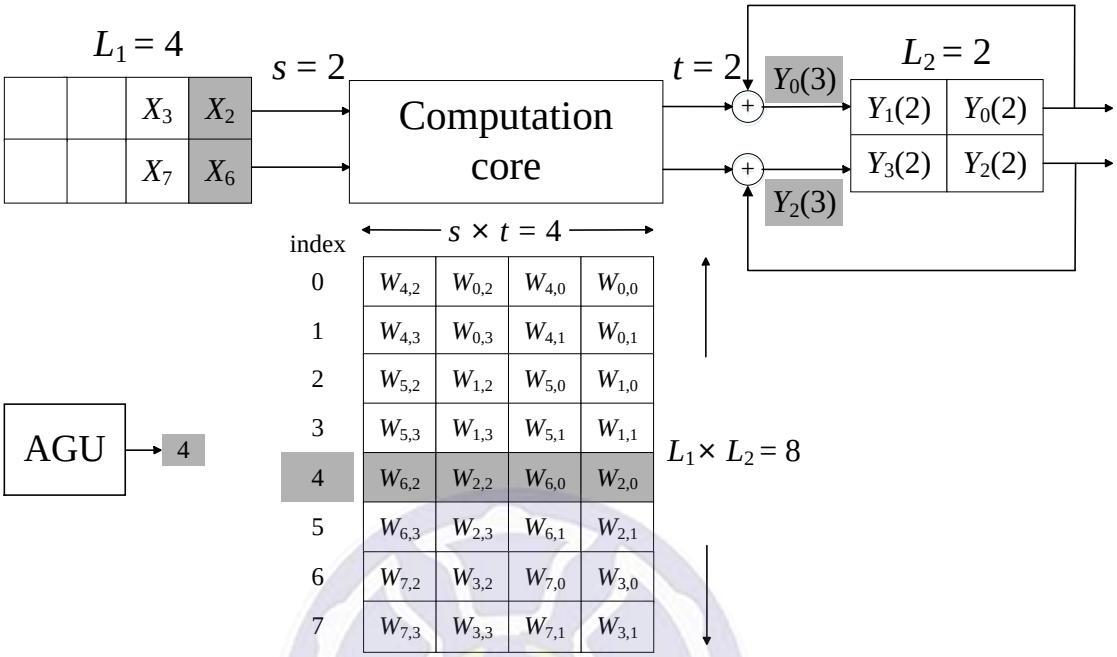

圖 3. 31 X_2 及 X_6 第一次運算電路圖

緊接著算對 Y_1 及 Y_3 之部分結果，再和先前的結果相加，得到 $Y_1(3)$及 $Y_3(3)$，以圖 3.32 表示。

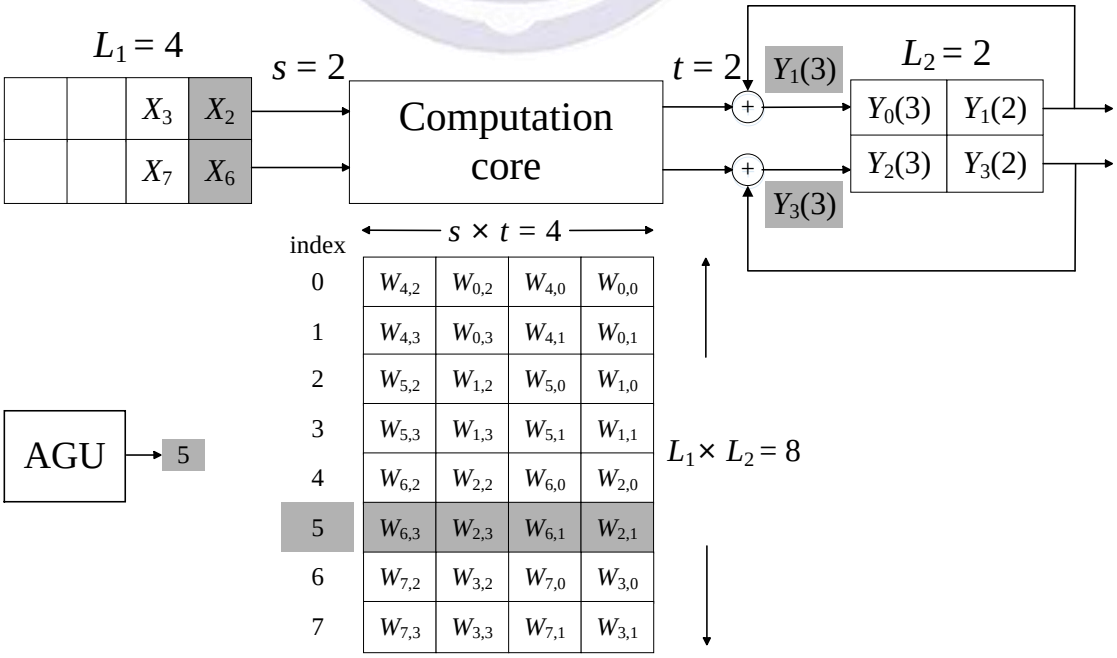

圖 3. 32 X_2 及 X_6 第二次運算電路圖

最後將 X_3 及 X_7 推入至運算單元運算，X_3 及 X_7 也是最後一筆的輸入，此次結果和先前結果的累加也是最後一次的累加，累加完之後將結果輸出至緩衝單元的同時直接將此次結果輸出。依圖 3.33 所示，

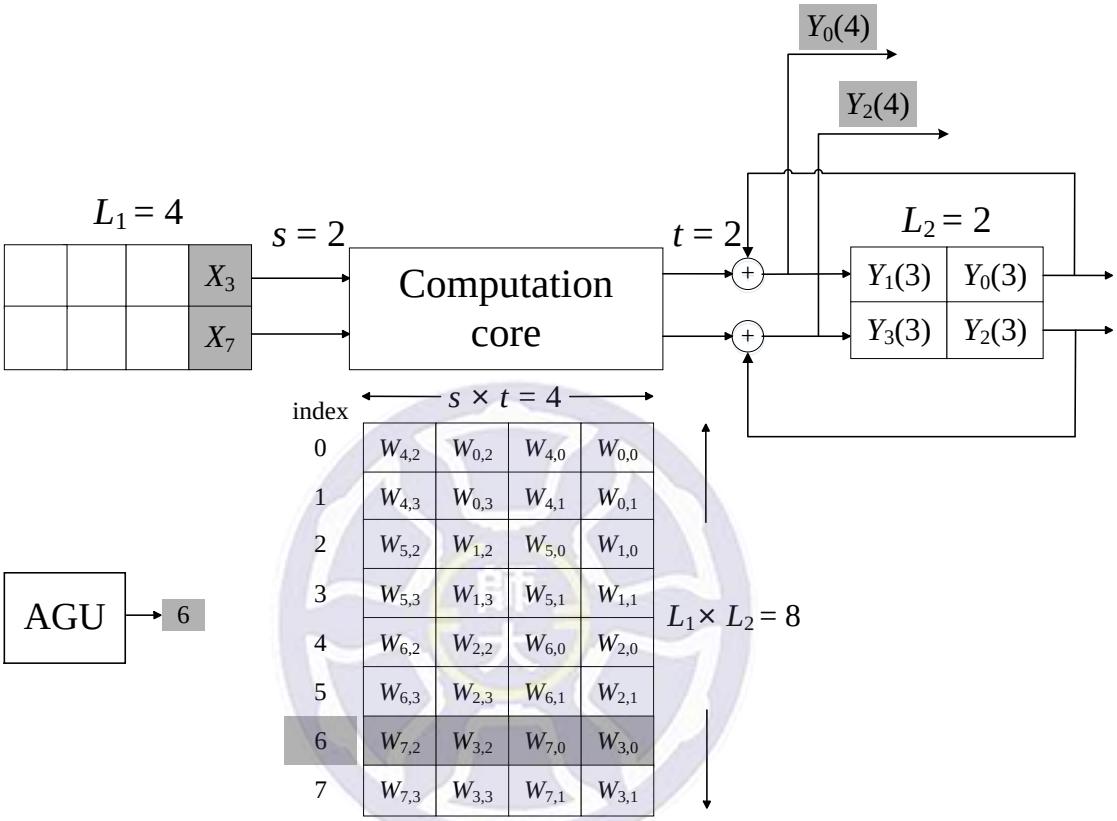

圖 3. 33 X_3 及 X_7 第一次運算電路圖，也是最後一次累加

由此圖可知，將累加結果直接輸出。當 $Y_0(4)$及 $Y_2(4)$結果產生之後，接下來運算 $Y_1(4)$及 $Y_3(4)$，以圖 3.34 所示。

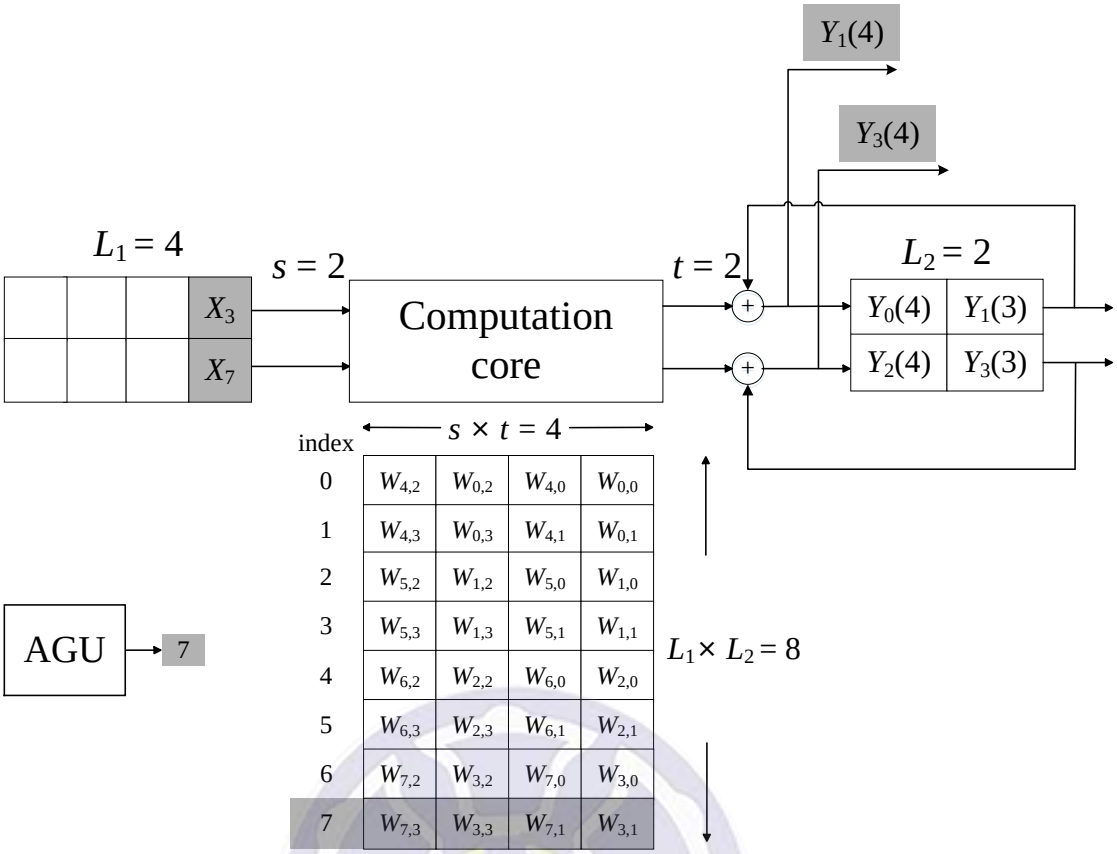

圖 3. 34 X_3 及 X_7 運算輸出電路圖

當 $Y_1(4)$及 $Y_3(4)$結果產生之後，也代表所有輸入及輸出皆已運算完成，此外還需要和偏移值相加，加完的結果才是最後的輸出結果。而偏移值我們使用表格儲存於記憶體單元，並利用位址產生器使輸出會和正確的偏移值相加，偏移值以 b_0、b_1、b_2 及 b_3 表示，相加完成之後，產生出 Y_0、Y_1、Y_2 及 Y_3，而這也是最後我們所要的輸出結果。和偏移值相加之架構請參照圖 3.35，圖 3.36 為和偏移值相加之電路圖。

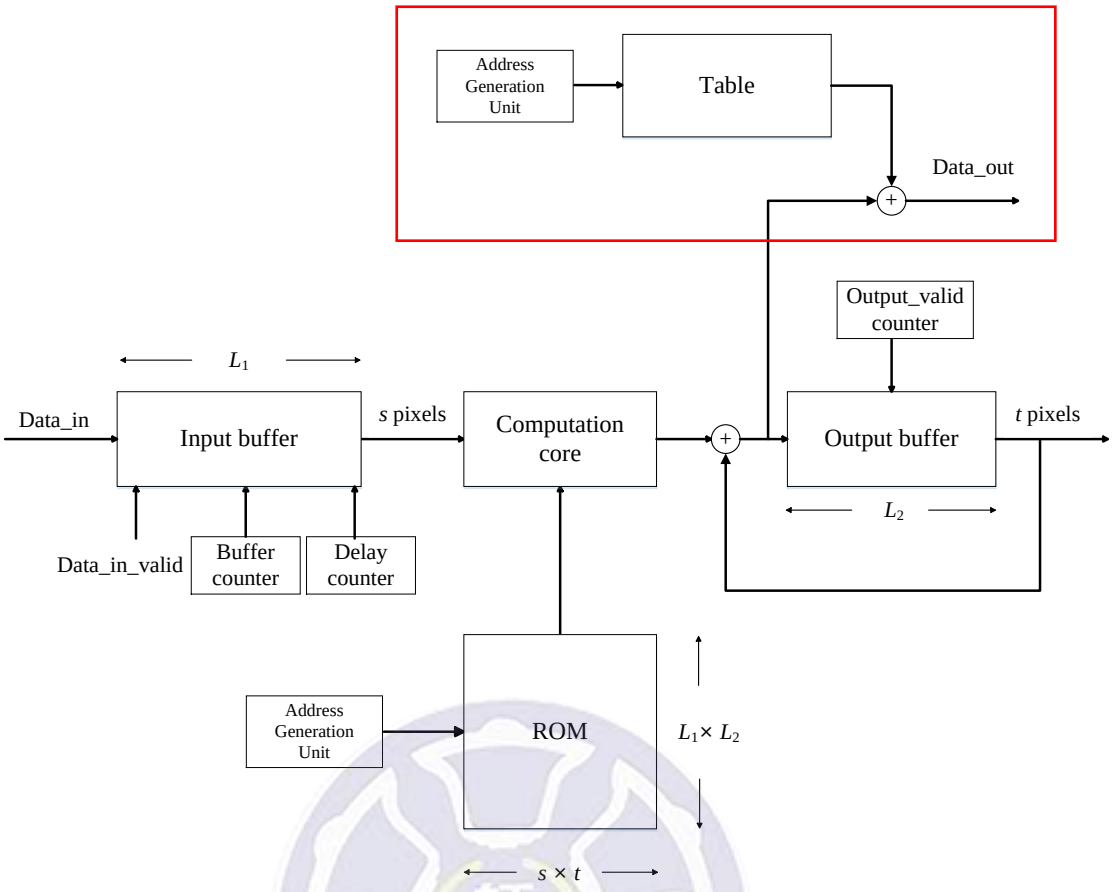

圖 3. 35 紅色框線和偏移值相加架構圖

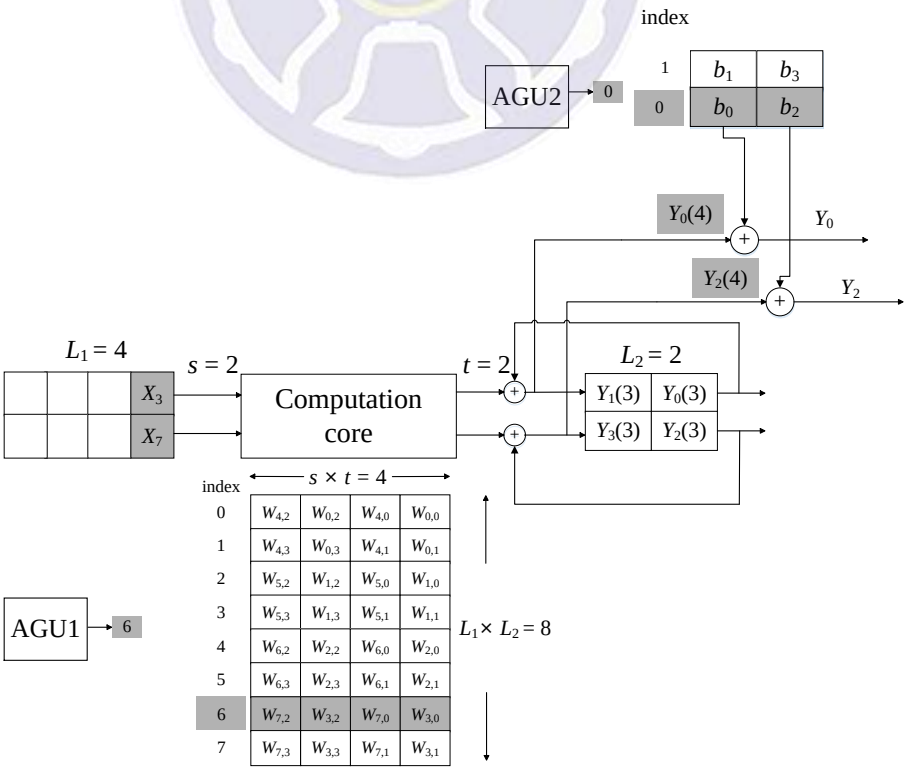

圖 3. 36 $Y_0(4)$及 $Y_2(4)$和偏移值相加電路圖

運算完 Y_0 及 Y_2 後，接著算 Y_1 及 Y_3，同樣將結果和偏移值相加，相加完之結果為 Y_1 及 Y_3 最後結果，也代表完成此全連結運算，以圖 3.37 表示

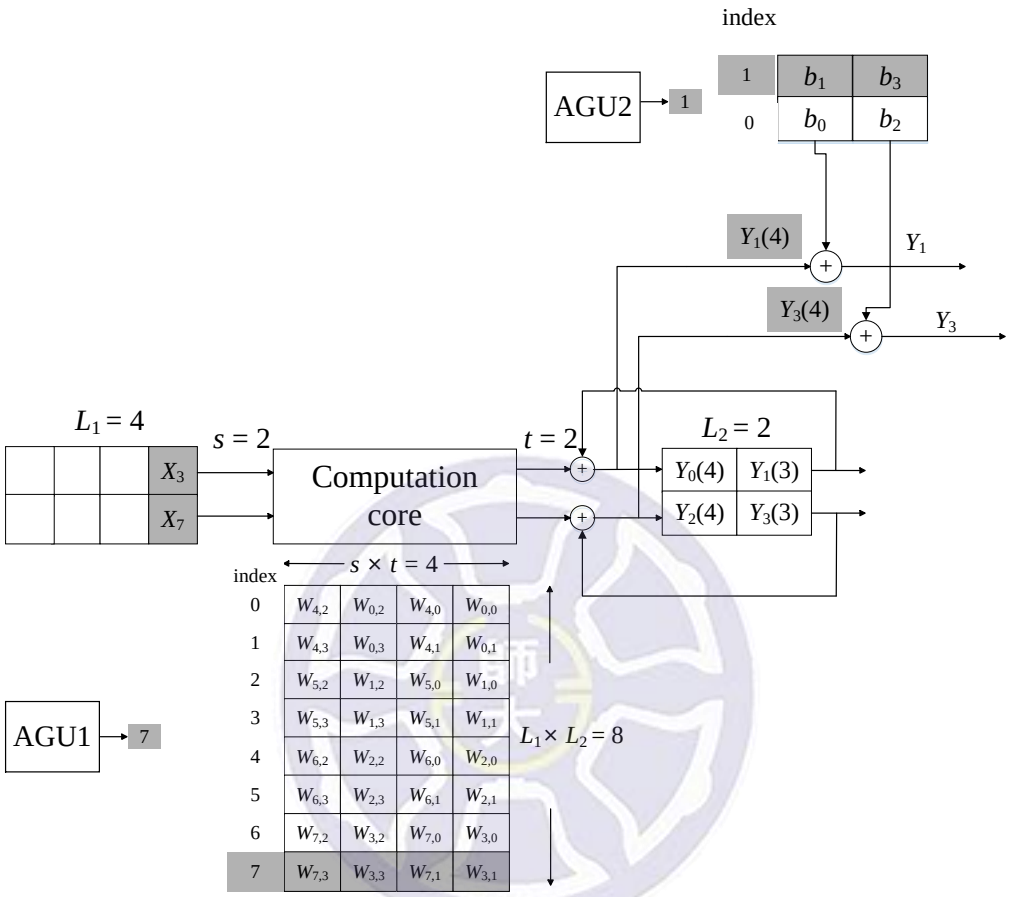

圖 3. 37 $Y_1(4)$及 $Y_3(4)$和偏移值相加之電路圖

上述之運算方法為本論文全連結架構之設計方法，有效的減少資源消耗。

第三節　應用全連結網路於 LeNet-5 架構之設計

而除了全連結架構之外，為了使辨識率更高，我們應用了摺積神經網路的架構，並將全連結電路架構和摺積神經網路整合到 LeNet-5 架構，LeNet-5 架構在本論文第二章第四節有論述，而摺積神經網路架構之應用來自於王雅慶同學之論文以 FPGA 實現摺積神經網路及應用於人臉特徵辨識之研究 [12]，藉由與摺積神經網路架構的整合，大大提高了此架構的辨識率， LeNet-5 架構圖請參照圖 2.5，圖 3.38 為 LeNet-5 電路架構圖。

圖 3. 38 LeNet-5 電路架構圖

LeNet-5 電路架構可分為 Stage1、Stage2 及 Stage3 三個部分；Stage1 包含了 C1、S2 及 Pipeline Register 1 三個部分，Stage2 包含了 C3、S4、Pipeline Register 2 三個部分，Stage3 包含了 C5 及 F6 兩個部分，而其中 Stage1 及 Stage2 的電路請參照王雅慶同學論文，而本論文全連結電路架構實現應用於 Stage3，也就是 C5 及 F6，以下會分別論述 C5 及 F6 之電路架構，下圖 3.39 為 C5 之電路架構。

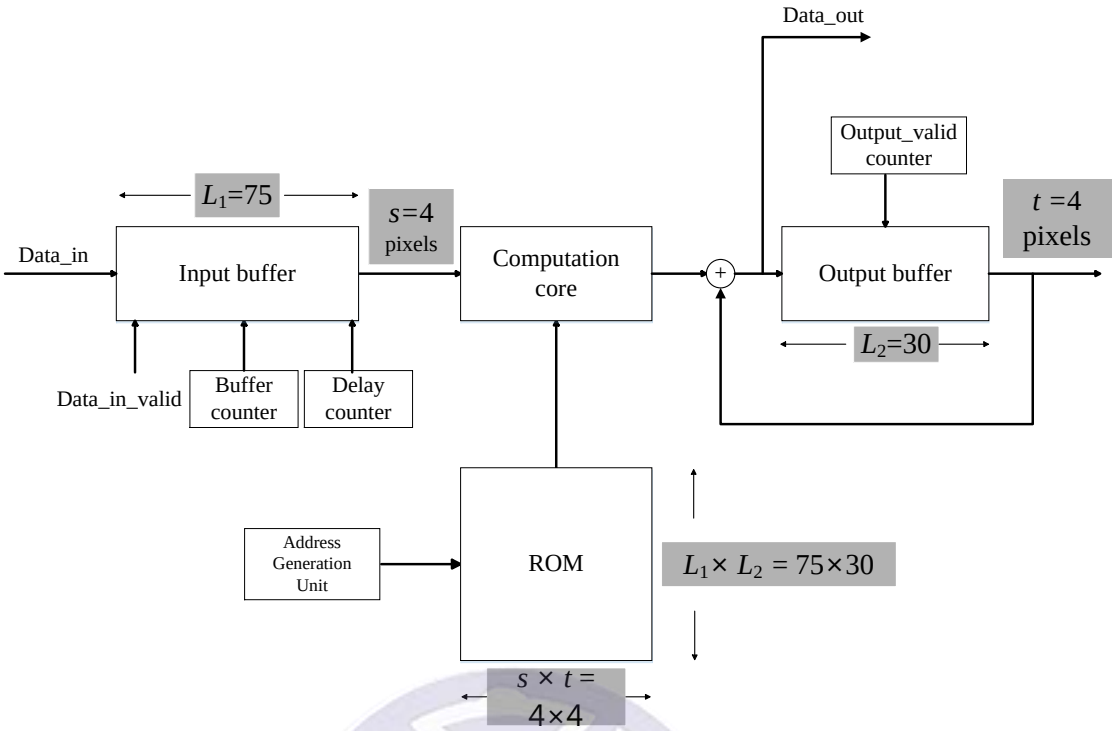

圖 3. 39 C5 電路架構

在此我們假設 S 及 T 分別為輸入及輸出總個數，但由於 S 及 T 過大時，會造成資源消耗過大，所以使用第三章第二節的電路架構，s 及 t 為每次運算的輸入及輸出個數，而 L_1 及 L_2 為緩衝單元之長度，所以每次輸入個數乘輸入緩衝單元長度等於輸入的總個數，而輸出個數乘輸出緩衝單元的長度則等於輸出總個數，也就是說 $S = s \times L_1$，$T = t \times L_2$。在 C5 這一層的輸入及輸出總數分別為 300 及 120，所以 S 及 T 分別為 300 及 120，而每次運算輸入及輸出都是 4 個，所以 s 及 t 分別都是 4，而 L_1 等於 75，L_2 等於 30。

探討完 C5，接著討論 F6，F6 架構和 C5 一樣，只是輸入總數及輸出總數的不同，所以暫存器長度也不一樣，下圖 3. 40 為 F6 之架構。

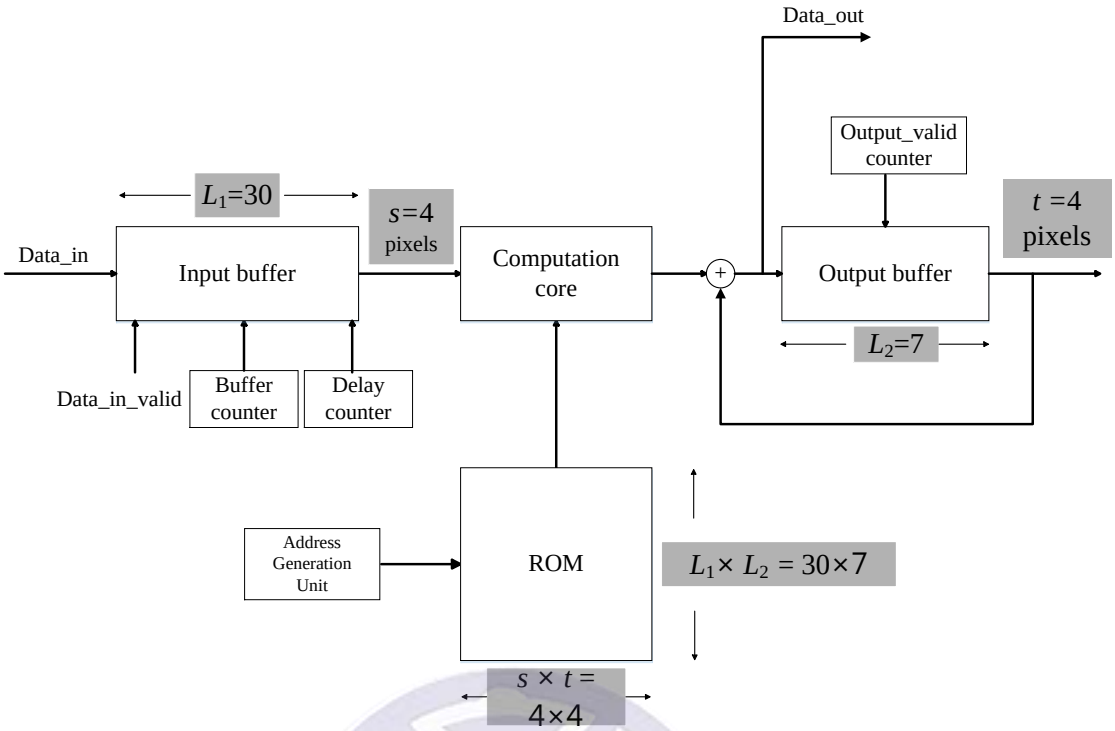

圖 3. 40 F6 電路架構

在 F6 這一層的輸入及輸出總個數分別為 120 及 28，所以 S 及 T 分別為 120 及 28，而每次運算的輸入及輸出個數和 C5 一樣，都是 4 個，所以 s 及 t 分別都是 4，而 L_1 及 L_2 分別為 30 及 7。但這裡有個不一樣的地方，就是輸出緩衝單元的長度是 7，而累加的加法器計算時間需要 7 個 clock，恰巧相同，所以我們將加法器運算的這 7 個 clock 當作為緩衝單元使用，這樣也不會消耗多於的資源。

而如何計算 C5 及 F6 的運算時間，當所有的權重運算完畢之後，也就代表此全連結運算完成，所以 C5 及 F6 的計算時間 clock 個數也就是 $L_1 \times L_2$，C5 運算所需要的 clock 個數為 2250，而 F6 為 210，完成兩個運算總共需要 2460 個 clock。

在本節我們有提到，為了使辨識率提高，除了全連結層電路之外，我們增加了摺積層電路，也就是 Stage1 及 Stage2 的電路，並且將 Stage1 及 Stage2 和 Stage3 做整合，整合到 LeNet-5 架構，同時透過控制器協調摺積層電路及全連結電路之

間的運作，另外我們為了做實際電路運作的測試，所以將 LeNet-5 架構整合於 SoPC 系統，達到實際測試電路運作的目的。而整合於 SoPC 系統則需要有一個與 Nios II processor 溝通的介面，並藉由 Qsys 系統將此系統架構實現於 FPGA。圖 3.41 為 LeNet-5 系統架構

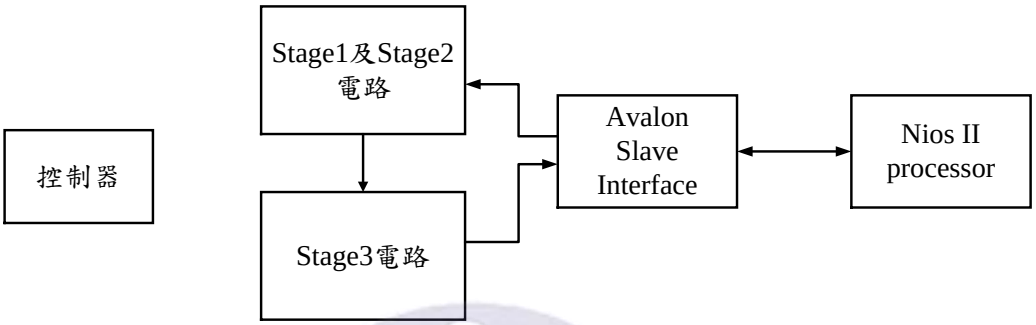

圖 3. 41 LeNet-5 系統架構

首先，電路會經由 Avalon Slave Inrerface 和 Nios II processor 溝通，並從 Nios II processor 取得輸入影像資料並傳送至摺積電路，並且開始運算，接著控制器會控制摺積層電路及全連結電路之運作，當摺積層電路運算完畢之後，透過控制器將摺積層運算之結果傳送給全連結電路繼續運算，當全連結運算完之後會將結果送回給 Avalon Slave Inrerface，經由 Avalon Slave Inrerface 將輸出結果傳遞給 Nios II processor，並且利用 Eclipse 將輸出結果寫成文字檔，有了文字檔之後即可和軟體結果依序進行比對，確認電路運作是否正確。

而由上述可知，電路和 Nios II processor 之溝通介面為 Avalon Slave Inrerface，透過 Avalon Slave Inrerface，電路才能和 Nios II processor 進行各種資料傳遞。由此可知，Avalon Slave Inrerface 在此扮演著非常重要的角色，圖 3.42 為摺積神經網路透過 Avalon Slave Inrerface 與 Network Interface 溝通架構圖。

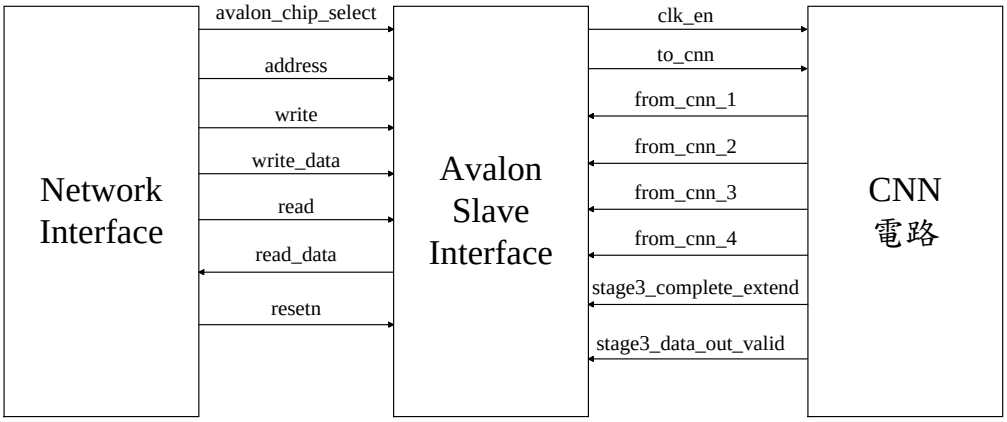

圖 3. 42 與 Network Interface 溝通架構圖

圖 3.43 可發現，Avalon Slave Interface 為 Network Interface 與 CNN 中間的橋樑。Network Interface 會使用寫入位址的方式將啟動電路指令傳送給 Avalon Slave Interface，透過 Avalon Slave Interface 的傳達，可將電路啟動並同時將輸入影像傳送給 CNN，clk_en 為啟動電路之訊號，而 to_cnn 則為輸入影像；Network Interface 會使用讀取位址的方式接收電路結束訊號及運算完之結果，當電路運算完畢，會將結束訊號及運算完之結果傳送給 Avalon slave Interface，stage3_complete_extend 及 stage3_data_out_valid 皆為結束訊號，from_cnn_1、from_cnn_2、from_cnn_3、from_cnn_4 為運算完之結果，等到 Avalon Slave Interface 接收完所有結果，會將結果傳回給 Network Interface 並寫成文字檔。

因為 Network Interface 需要寫入位址及讀取位址，而 Avalon Slave Interface 會將這兩種方式轉換成電路能夠理解的訊號。我們將透過圖 3.43 及 3.44 分別對 Avalon Slave Interface 之實現方式進行詳細的說明。

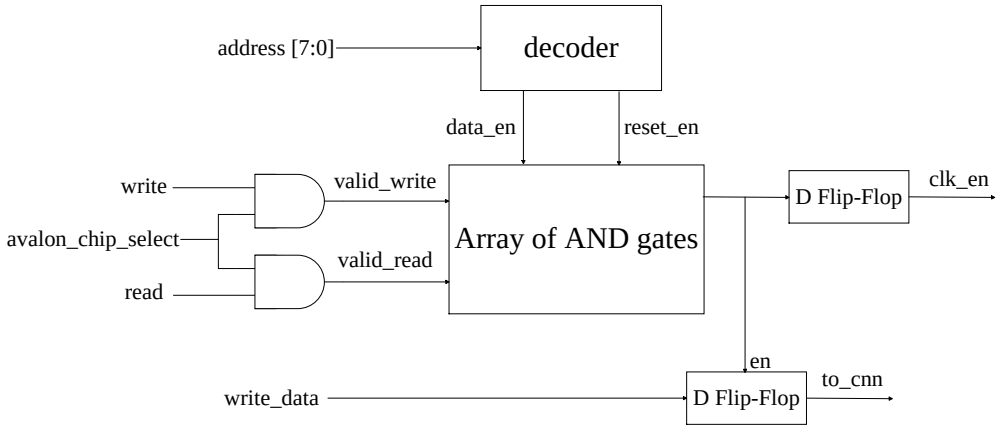

圖 3. 43 Interface Implementation I

一開始會寫入某個位址，將電路初始化，接著會寫入另外一個位址，而當寫這個位址時會啟動電路運作，並同時將輸入影像傳送給電路運算。

接著會透過讀取某個位址，等待接收電路結束訊號，當收到結束訊號時代表電路完成並已經將結果儲存於 buffer，我們透過讀取不同位址的方式分別將四個 buffer 之結果傳回給 Network Interface。

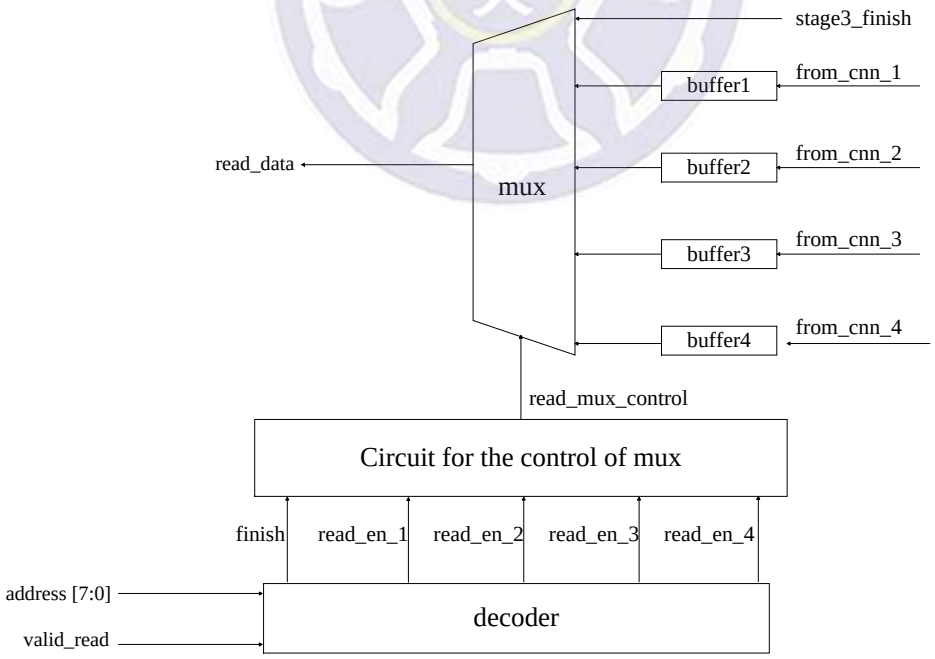

圖 3. 44 Interface Implementation II

第四章 實驗數據與效能分析

本章重點為開發平台與實驗環境設定、系統效能量測與比較及本論文系統架構整合之效果。

第一節　開發平台與實驗環境設定

本論文使用 Altera Stratix® IV GX FPGA (EP4SGX530KH40C2N) 開發板作為本論文硬體實現之平台，其外觀如下圖 4.1。並將開發板之詳細規格整理於表 4.1。

此外藉由 FPGA 所提供之 Qsys 系統，及其可程式化系統晶片(SoPC)之架構提供開發者擁有更高的設計電路之彈性，並且提供簡易的 GUI 介面供開發者操作，使開發者更易於設計實現電路。而且其可快速燒錄之特性使得 FPGA 更適合用於硬體架構之驗證，也不必負擔晶片合成的高成本風險，也因此選擇此開發板作為開發本論文架構硬體實現的平台。

圖 4. 1 Altera Stratix® IV GX FPGA (EP4SGX530KH40C2N)外觀

表 4. 1Altera Stratix® IV GX FPGA (EP4SGX530KH40C2N)詳細規格

Feature	Stratix® IV GX FPGA
Device	EP4SGX530KH40C2N
Adaptive Lgoic Modules(ALMs)	212,480
Logic Elements(LEs)	531,200
M9K Memory Blocks	1,280
M144K Memory Blocks	64
Total memory Kbits	27,376
DSP block 18-bit elements	1,024
Total PLLs	8
User I/Os	744

本論文使用之開發套件為 Altera Quartus II 14.0 軟體撰寫 Verilog 硬體描述語言做為開發之程式語言，快速建立起硬體電路之架構，並藉由 Altera Quartus II 提供之套件 Modelsim 來建立 RTL Level 與 Gate Level 模擬環境，並可藉由模擬訊號之波形圖驗證電路之正確性。待本論文之硬體電路系統架構及周邊相關電路完成後，藉由 Altera Quartus II 之 Qsys 平台將全部的相關電路系統一起掛載至 SoPC 系統上，並在最後將掛載完畢的 SoPC 系統燒錄至 FPGA DE4 開發板上，就可以觀察實際電路運作之正確及實際效能分析。SoPC 系統內包含許多元件，像是 Nios II CPU、客製化電路(全連結電路)、記憶體等等，透過 SoPC builder 介面及可配置

而成。Altera Quartus II 所使用的 PC 規格為 Intel® Core™ i7-4790 CPU@3.60GHz、32G DDR III 記憶體。

Altera 公司採用 Eclipse 為基礎的 Nios II IDE 軟體供使用者使用，讓所有於系統中之軟體開發都能在 Nios II IDE 完成。

本論文除了使用 Verilog 硬體描述語言實作硬體電路架構外，也使用 Caffe 裡的 Python 以相同演算法則與硬體電路架構執行之結果相互比較。下表為硬體及軟體實現之環境。

表 4. 2 本系統分別於硬體與軟體之實驗環境

硬體實驗環境	軟體實驗環境
Device：Altera Stratix® IV GX FPGA EP4SGX530KH40C2N CPU：NIOS II Processor 100 MHz Eclipse	CPU：Intel(R) Core(TM) i7-4790 CPU @ 3.60GHz RAM：32 GB Caffe-Python

第二節　實驗數據呈現與討論

本節將著重於實驗數據分析與討論，本架構實現全連結網路，並與摺積層結合，整合為 CNN LeNet-5 之架構，最後將硬體之結果與軟體之結果進行比較，及測試辨識率。

首先，先對本論文硬體架構的資源消耗進行討論，將全連結層分為 C5 及 F6 來討論。由於本論文電路之基礎建構元件為加法器、乘法器及暫存器，另外也探討運算時間複雜度,根據 C5 及 F6 資源消耗及時間複雜度以下表 4.3 及 4.4 表示，S_5 為 C5 之輸入個數 S，T_5 為 C5 之輸出個數 T，s_5 為 C5 之運算輸入個數 s，t_5 為 C5 之運算輸出個數 t，L_{15} 為 C5 之 L_1，L_{25} 為 C5 之 L_2，S_6 為 F6 之輸入個數 S，T_6 為 F6 之輸出個數 T， s_6 為 F6 之運算輸入個數 s，t_6 為 F6 之運算輸出個數 t，L_{16} 為 F6 之 L_1，L_{26} 為 F6 之 L_2

表 4. 3 C5 及 F6 資源消耗複雜度

	C5	F6
Adders (two-input)	$O(s_5t_5)$	$O(s_6t_6)$
Multipliers (two-input)	$O(s_5t_5)$	$O(s_6t_6)$
Registers	$O(S_5T_5)$	$O(S_6T_6)$

表 4. 4 C5 及 F6 時間複雜度

	C5	F6
latency	$O(L_{15}L_{25}) = O\left(\frac{S_5T_5}{s_5t_5}\right)$	$O(L_{16}L_{26}) = O\left(\frac{S_6T_6}{s_6t_6}\right)$

除了本論文架構的資源消耗複雜度及時間複雜度之外，我們也在現有的文獻裡發現到類似的架構，在 2009 年 C. Latino 所發表的文獻 [13]，他所使用的架構

和本論文類似，但是他是以一個輸入運算出一個輸出的方式運算，並且依續存取輸入的方式算出一筆完整的輸出之後，再重複存取輸入運算出下一筆輸出，而此架構又將輸入儲存於記憶體單元，這樣會造成 CPU 需要不斷做存取的動作，浪費了許多的時間，我們以下表 4.5 比較本論文架構及 2009 年 C. Latino 所發表的文獻做為比較，以 C5 層做為例子。

表 4. 5 本論文架構與現有文獻比較表

	本論文	C. Latino[2009]
Adders (two-input)	$O(s_5t_5)$	$O(1)$
Multipliers (two-input)	$O(s_5t_5)$	$O(1)$
Registers	$O(S_5T_5)$	$O(S_5T_5)$
latency	$O(L_{15}L_{25}) = O\left(\frac{S_5T_5}{s_5t_5}\right)$	$O(S_5T_5)$

雖然從此表看來，在加法器及成法器的複雜度方面比現有文獻高，而暫存器則是和現有文獻相同，但是在時間複雜度的方面，本論文的時間複雜度遠遠小於現有文獻，也說明了本論文架構只需要多消耗一些硬體資源，運算時間就可以比現有文獻快速許多。

本架構之設計平台為 Altera Quartus II 14.0，並使用此平台進行分析、合成、繞線佈局等動作，所有動作完成後，由 Altera Quartus II 對本論文之 LeNet-5 架構資源消耗統計，

而在第三章第三節有提到，LeNet-5 電路架構可分為 Stage1、Stage2 及 Stage3，Stage1 包含 C1、S2 及 Pipeline Register 1，Stage2 包含 C3、S4 及 Pipeline Register2、

Stage3 包含了 C5 及 F6，而電路架構圖請參照圖 3.38，下表 4.6 為各層之資源消耗

表 4. 6 各硬體架構之資源消耗

	Stage1 及 Stage2	C5	F6	Total
ALMs	120,271	14,776	16,851	151,898
Memory bits	39,040	1,152,000	107,520	1,298,560
DSP blocks	800	64	64	928

本論文全連結架構為 C5 及 F6 兩層；C5 的輸入有 300 個，輸出有 120 個，所以權重共有 36000 個，每一個權重都是 32 位元，所以 bits 數為 36000×32 個，總共 1152000 個，而這 36000 個權重會儲存於記憶體單元，所以這一層的 Memory bits 數消耗為最高，而 F6 輸入有 120 個，輸出有 28 個，權重個數共有 3360 個，bits 數為 3360×32，總共 107520 個，這 3360 個權重也都儲存於記憶體單元，所以這一層的 Memory bits 數消耗相對的也比較多。但是可發現到，DSP blocks 數目很低。DSP blocks 通常為加法器及乘法器等等的消耗，所以正常來說 C5 及 F6 這兩層所要運算的系數這麼多，消耗應該會非常大，但是在這裡的 DSP blocks 的消耗分別也才各 64 而已，比 Stage1 及 Stage2 的消耗還低，主要是因為每次只使用 4 個輸入並計算出 4 個輸出的部分結果，並將部分結果儲存下來，再做累加，所以 C5 及 F6 的乘法器個數分別都是 16 個而已，而加法器個數也都分別為 16 個而已，藉由使用此設計架構，大量減少了 DSP Blocks 數目，由此可見本架構有效的減少 DSP blocks 的資源消耗。

接著是將硬體結果和軟體結果進行比較，軟體實現方式是使用 caffe [14]，caffe 需要訓練資料及測試資料，而訓練資料及測試資料張數以表 4.7 表示。

表 4. 7 訓練資料及測試資料張數

	訓練資料	測試資料
1 人	500(張)	76(張)
共 28 人	14,000(張)	2,128(張)

並確認辨識結果之正確性，而測試圖片為 yaleB13_P00A+000E-35，此張圖為第 3 個人，代表運算結果必須是第 3 筆為最大值，下圖 4.2 為此張圖片之影像，運算結果以下表 4.8 表示，

圖 4. 2 yaleB13_P00A+000E-35 影像

表 4. 8 yaleB13_P00A+000E-35 硬體及軟體之結果

yaleB13_P00A+000E-35					
	軟體結果	硬體結果		軟體結果	硬體結果
1.	23.99806976	23.998	15.	19.42761803	19.428
2.	31.35560036	31.356	16.	4.10646152	4.106
3.	42.67484283	42.675	17.	-2.02835655	-2.028
4.	-39.60054016	-39.601	18.	24.62788773	24.628
5.	-42.34254837	-42.343	19.	22.08097076	22.081
6.	32.67694473	32.677	20.	-18.67922211	-18.679
7.	4.98132992	4.981	21.	-27.88210106	-27.882
8.	28.51237488	28.512	22.	-20.5458107	-20.546
9.	-25.82872963	-25.829	23.	-8.35191154	-8.352
10.	-15.994977	-15.995	24.	-1.33826363	-1.338
11.	11.12542057	11.125	25.	-12.08771896	-12.088
12.	5.37283421	5.373	26.	-20.08349991	-20.084
13.	-5.87290335	-5.873	27.	-14.71090698	-14.711
14.	-25.77027321	-25.770	28.	5.21999121	5.220

由於硬體之運算資料大小為 32bits，但是軟體運算為 64bits，所以在運算輸入資料及輸出資料在時候會有些許的誤差，除此之外，因為要做比較最大值的動作，但是在電路裡沒有比較的電路，所以我們會將輸出結果寫成 txt 檔之後再來做比較，而寫成 txt 檔的輸出結果都是以 16 進位表示法，所以我們要將這樣結果轉換成 10 進位以利於比較，方便找出最大值，所以在使用 32bits 跟 64bits 的 16 進位

轉換成 10 進位時也會有些許的誤差，但誤差其實只是運算到小數點後面第幾位的不同而已，像是硬體的話表示到小數點後第 3 位，而軟體則是表示到小數點後第 8 位，但是以最後的運算結果來看，大致上為一致的。除了將結果運算出來之外，還需要辨識，而此張輸入圖片為第 3 個人，上表結果第 3 筆為 28 筆結果當中之最大值，所以代表此次辨識也是成功的。而為了確保本電路架構之正確性，我們以 200 張影像做為測試，而這 200 張影像分別使用硬體及軟體進行測試，將硬體辨識率和軟體之辨識率以表 4.9 表示。

表 4. 9 測試 200 張影像結果

	硬體	軟體
Total(張)	200	200
正確	198	198
錯誤	2	2
辨識率	99%	99%

由表 4.8 發現，有時候還是會辨識失敗，為了確保是否為電路錯誤，我們將失敗圖片之硬體結果與軟體結果相互比對，確認電路是否正確，或者是此演算法無法正確辨識此張圖片。辨識錯誤圖片為圖 yaleB36_P01A+095E+00，並將影像及結果以圖 4.4 及表 4.10 表示。

圖 4.3 為 yaleB36_P01A+095E+00 影像

表 4. 10 yaleB36_P01A+095E+00 硬體與軟體結果比較

yaleB36_P01A+095E+00					
	軟體結果	硬體結果		軟體結果	硬體結果
1.	-15.11668682	-15.117	15.	-10.12393665	-10.124
2.	-8.21939087	-8.219	16.	-9.53808308	-9.538
3.	-11.29203701	-11.292	17.	-2.93057156	-2.931
4.	-3.51199055	-3.512	18.	2.65207672	2.652
5.	18.10114479	18.101	19.	-20.62793732	-20.628
6.	-15.74552536	-15.746	20.	-17.91454887	-17.915
7.	3.50301957	3.503	21.	0.09335451	0.093
8.	10.2728014	10.273	22.	2.48904824	2.489
9.	-9.7560358	-9.756	23.	14.31492329	14.315
10.	-11.01324558	-11.013	24.	20.45812225	20.458
11.	14.30819702	14.308	25.	25.1013813	25.101
12.	7.28046846	7.280	26.	25.50552559	25.506
13.	2.12293458	2.123	27.	-5.55778408	-5.558
14.	12.11584663	12.116	28.	5.21974182	5.220

從表 4.8 看來，硬體結果和軟體結果值大致上為一致，而 yaleB36_P01A+095E+00 為第 25 人之圖片，所以第 25 筆應當為最大值，但表 4.5 裡第 26 筆才為最大值，所以代表此次辨識失敗。而此張圖片已經幾乎全黑，連我們肉眼也無法辨識清楚，所以辨識失敗其實也不為過，而我們透過和軟體結果比較，本論文架構的運算結果和軟體一致。

除了運算結果和辨識率的比較之外，還做了運算時間的比較，以表 4.11 表示。

表 4. 11 運算時間比較表

	軟體	硬體
Time(ms)	0.768135	0.0131

由表可知，硬體化之優點除了辨識率和軟體差不多之外，運作時間還比軟體快了將近 60 倍左右。

第五章 結論

本論文之重點在於將全連結架構，除了將全連結硬體化之外，還透過和摺積層做整合，提高此系統的辨識率，透過第四章的實現結果，除了此系統的辨識率和軟體不分軒輊之外，並且藉由硬體化，大幅減少系統運算時間，使系統運算更快速，這也是本系統比軟體更優越的地方。

而本論文系統架構實作是以浮點數運算，有效提高精確度，和其他辨識之演算法比較占優勢的。

本論文提出之全連結硬體架構設計於FPGA實現相較於其他架構除了功率消耗較低之外，現有的硬體架構都只有支援小尺寸的全連結神經網路，對於大尺寸的全連結神經網路根本不適用，而在本論文之架構我們可以透過對運算輸入個數及輸出個數的設定，有效地控制資源消耗，也藉由運算個數的設定，除了可應付小尺寸的全連結神經網路之外，大尺寸的全連結神經網路也可支援，使電路更具有應用之彈性。

參考文獻

[1] S. Hauck and A. Dehon, Reconfigurable Computing: The Theory and Practice of FPGABased, 2008.

[2] U. Meyer-Baese, Digital Signal Processing with Field Programmable Gate Arrays, 4th Ed.,, 2014.

[3] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, Gradient-Based Learning Applied to Document Recognition. Proceedings of the IEEE, 86, pp. 2278-2324, 1998.

[4] Y. LeCun, Y. Bengio and G. Hinton, Deep Learning, Nature, 521, pp. 436-444, 2015.

[5] C. Farabet, B. Martini, B. Corda, P. Akselrod, E. Culurciello, and Y. LeCun, Neuflow: A Runtime Reconfigurable Dataflow Processor for Vision. In Proc. IEEE Workshop Embedded Comput. Vision., 2011.

[6] J. Jin, V. Gokhale, A. Dundart, B. Krishnamurthy, B. Martinit and E. Culurciello, An Efficient Implementation of Deep Convolutional Neural Networks on a Mobile Coprocessor. In Proc. IEEE Int. Midwest Symp. on Circuits and Systems, pp.133-136., 2014.

[7] S. Chakradhar, M. Sankaradas, V. Jakkula, and S. Cadambi, A Dynamically Configurable Coprocessor for Convolutional Neural Networks. In Proc. ACM Int. Symp. on Comput Architecture, pp. 247-257., 2010.

[8] C. Farabet, C. Poulet, J. Y. Han, and Y. LeCun, CNP: An FPGA-Based Processor for Convolutional Networks. In Proc. IEEE Int. Conf. Field Programmable Logic and App., pp. 32-37., 2009.

[9] M. Sankaradas, V. Jakkula, S. Cadambi, S. Chakradhar, I. Durdanovic, E. Cosatto, and H. P.Graf, A Massively Parallel Coprocessor for Convolutional Neural Networks. In Proc. IEEE Int. Conf.on Application-Specific Syst., Arch. and Proc., pp. 53-60., 2009.

[10] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong, Optimizing FPGA-Based Accelerator Design for Deep Convolutional Neural Networks. In Proc. ACM/SIGDA Int. Symp. on Field-Programmable Gate Arrays, pp.161-170., 2015.

[11] Y. Cheng, F. X. Yu, R. Feris, S. Kumar, A. Choudhary, and S.-F. Chang, An

Exploration of Parameter Redundancy in Deep Networks with Circulant Projections. In Proc. IEEE Int. Conf. Comput. Vision., 2015.

[12] 王雅慶, 以 FPGA 實現摺積神經網路及應用於人臉特徵辨識之研究,國立台灣師範大學碩士論文, 2016.

[13] Carl Latino, Marco A. Moreno-Armendariz, and Martin Hagan, Realizing General MLP Networks with Minimal FPGA Resources. In Proc. IEEE Int. Joint Conf. on Neural Net., pp. 1722-1729., 2009.

[14] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, Caffe: Convolutional Architecture for Fast Feature Embedding. In Proc. ACM Int. Conf. on Multimedia, pp. 675-678., 2014.