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

透過區域網路建立跨平台整合之離線棘波分類系統

Cross-Platform Offline Spike Sorting System Using Local Area Network

研究生：　李　光　耀　撰

中華民國　一百零三　年　七　月

誌謝

首先，要非常感謝我的指導教授 黃文吉 教授，在碩士班求學期間對我不辭辛勞地指導，讓我有機會去學習研究的方法以及待人處世的態度，當我遇到困難時與我討論並引導我通往正確方向，使得我在這段學習的時間獲得了許多的成長與寶貴的經驗，在此獻上最誠摯的謝意。同時也感謝國立宜蘭大學電子工程學系 林秀菊 教授、財團法人資訊工業策進會 陳靜芳 經理撥冗來參加我的碩士論文口試，並給予論文上的評閱與建議。

接著，很榮幸能夠來到國立臺灣師範大學資訊工程研究所就讀，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成長。我要感謝已畢業的學長姐：建廷、清志、國璿、聖穎、任軒和翰逸，還有博士班的學長姐們，謝謝你們的教導，同時也感謝在我碩士求學生涯與我一起奮鬥的同學們：奇恩、思淮、一修、煥元、皓棠、雅姿、烝祺，另外也感謝可愛的學弟們：建旻、元品、承祐、信豪、柏佑和映綸，給予我學業與生活上協助，在此一併致謝。

最後，更要感謝家人與所有關心我的朋友，有了你們的支持、關懷與鼓勵，讓我有勇氣去面對各種的挫折與壓力，並順利地完成學業。謹將此論文成果獻給所有關心我的人，希望大家與我分享這份喜悅與榮耀。

中文摘要

近年來有關大腦活動情形的研究越來越熱門，目前也提出了許多有關腦波訊號處理的相關研究。本論文於不同平台上實現了一套跨平台的棘波分類系統，因此在這透過區域網路建立一個跨平台的棘波分類系統，將不同平台所讀取的棘波序列傳送至現場可程式邏輯閘陣列 (Field Programmable Gate Array , FPGA) 開發板中進行運算，以節省棘波分類軟體所運行的時間。在這使用 MATLAB 讀取大量的棘波序列(Spike Train)，再透過區域網路將棘波訊號傳送到晶片網路(Network-On-a-Chip , NoC)平台所開發的棘波分類系統中。

本論文所提出之架構與現有軟體做比較，從實驗數據結果可以得知其效能遠優於一般軟體的棘波分類系統，使得本系統架構在棘波分類應用上更具有優勢，因此，可以說本論文所設計的跨平台棘波分類系統是一項有實際需求且有效率的電路架構設計。

關鍵字： FPGA、棘波分類、棘波序列、區域網路、NoC

目錄

附表目錄

附圖目錄

第一章 緒論

本章節主要探討本論文的研究背景與動機、研究目的與方法，並簡單的說明各章節主要的內容。

1.1 研究背景與動機

人類腦中有一百多億個腦細胞，不同位置的腦細胞掌管了不同的生理功能，這些大量的腦細胞透過突觸相互連接形成網狀，腦波的產生主要是大腦皮質層所產生的電位變化，是由神經元細胞以電位訊號的方式來傳遞訊息，一般腦波都是平穩波動幅度小的，而棘波(Spike)是呈現振幅落差明顯的波型，是神經元接收到棘激後所發出的短暫且特殊的波型。棘波的偵測主要是透過微電極探針來擷取的，而偵測到棘波訊號的總和稱為棘波序列(Spike Train)，即為一連串神經元細胞所接收到的動作電位訊號。棘波分類(Spike Sorting)則是將不同神經元所發出的訊號加以區分。

目前棘波分類的相關研究日漸趨於成熟[1,2]，其中加州理工學院(California Institute of Technology)公開了兩套相當完整棘波分類系統：Wave_clus 和 Osort。但是在兩套系統中的棘波排序(Spike Sorting)都是由軟體運算，由於從人類腦中所擷取的真實棘波紀錄，會是一個非常大的稀疏矩陣(Sparse matrix)，因此在處理大量棘波序列(Spike Train)時，需要消耗相當多的時間。棘波分類的應用非常的廣泛，可以應用在腦機介面(Brain Machine Interface，BMI)[3]，或是在醫學及

相關產業中應用，可以提供重症病患透過腦波控制機械的四肢，而一般正常人操控四肢軀幹神經元間的訊號傳遞時間大約都在幾百個毫秒以內[4]，因此如果棘波分類過程所需的時間太長，即使想要簡單的跑步或是任何相關活動，都可能會造成四肢不協調而跌倒受傷，因此讓裝有機械義肢的行動者無法如期動作，使得減少了在棘波分類的相關應用，而要讓裝有義肢的重症病患們能夠自在的活動，必需要達到即時的控制的效果。因此在棘波分類過程中，想透過網路將棘波序列傳給硬體 FPGA[5]來做運算，以減少運算時間。

1.2 研究目的與方法

本論文的目的為透過區域網路建立一個跨平台的棘波分類系統，以有效的縮短棘波分類的時間，因此在這使用 Matlab 讀取大量的棘波序列(Spike Train)，再透過區域網路將棘波訊號傳送到 Network-On-a-Chip(NoC)平台所開發的棘波分類系統中。首先，必須分為軟體 Matlab 設計與使用 Nios II 透過 C 語言控制 FPGA 硬體兩個部分來處理。軟體部分：透過 Matlab 將棘波序列數值由 ASCII 編碼轉換成 Binary 型態，使用 TCP/IP 協定建立 Socket 連線，將轉換後的數值透過區域網路傳送至 FPGA 上使用 NoC 平台開發的系統中。透過該平台所提供的大量元件配置與豐富的的溝通管理介面，可以使整個系統開發變得更加容易且更具有彈性。硬體部分：在 NoC 平台中新增 Triple-Speed Ethernet MAC Controller 使得能透過區域網路將值放入接收與傳遞的位移暫存器中，透過

Scatter-Gather DMA 將存放值的空間位址紀錄在 Descriptor Memory 中，而 On-Chip RAM 中存放的數值即為要使用到的訓練向量。

由於 NoC 平台中的處理器本身並不參與棘波分類的計算，它只負責控制 NoC 平台中不同元件間的資料傳遞。因此使用此系統中的 DMA Controller 將訓練向量從 On-Chip RAM 傳送至客製化電路 GHA 中進行運算。最後將運算完後的權重向量結果傳回到 Matlab 進行處理，以進行棘波分類。最後實驗結果得以證明透過 FPGA 硬體進行棘波特徵擷取運算的時間，比軟體所運算的時間還要快了許多，即有效的提高其運算速度以及未來棘波相關應用大幅提升其運用性。

1.3 全文架構

本篇論文主要分為五個章節，以下為各章節內容概述：

【第一章】緒論

說明本論文的研究背景、動機、目的、方法以及全文架構。

【第二章】基礎理論與技術背景介紹

介紹本論文所使用的演算法、理論基礎、技術背景、以及棘波分類系統的運作流程。

【第三章】系統開發架構

介紹本論文所開發的系統架構、以及開發的流程，並對其進行說明。

【第四章】實驗數據與效能比較

主要包括實驗環境說明、以及相關的實驗數據分析、軟硬體效能比較以及與現有的系統做比較。

【第五章】結論

對於此系統實驗結果進行總結。

第二章 基礎理論與技術背景介紹

在本章節將介紹棘波分類流程、與現有公開的兩套棘波分類軟體 Wave_clus 與 Osort，以及本論文所使用的 NoC 架構與棘波分類相關演算法，並說明其應用。

2.1 棘波分類步驟

完整的棘波分類流程主要可以分為三個步驟：棘波偵測(Spike Detection)、特徵擷取(Feature Extraction)、以及棘波分群(Clustering)，以下將對這三個步驟做一個詳細的介紹。

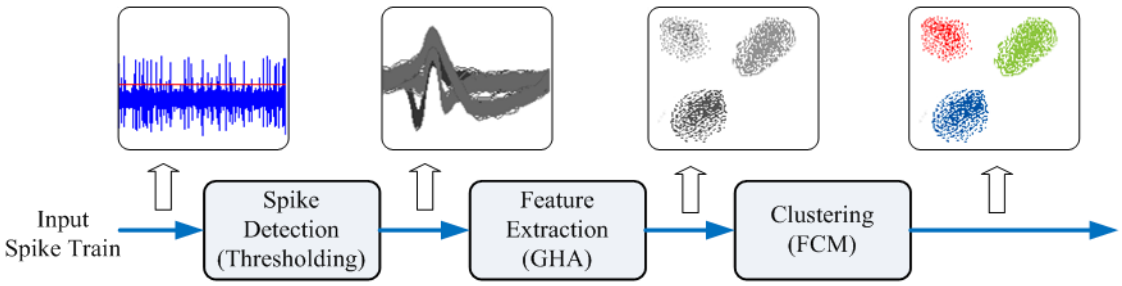


圖 2.1 棘波分類流程圖

棘波偵測的目的是從棘波序列中找出棘波發生的位置。通常會先將棘波序列經由帶通濾波器(Bandpass Filter)，使得頻率調整在限定的範圍內，經由設定閾值(Threshold)來偵測，如果超出了閾值就認定為是一個棘波。下圖 2.2 例子看出

是如何經由閥值認定而取得一個完整的棘波的，通常都是連續超過閥值後的幾個點加上特定的維度來當作一個完整棘波的末端位置。而經由所紀錄的末端位置向前取一個棘波維度的大小即將棘波完整的擷取出來，過濾掉雜訊了。

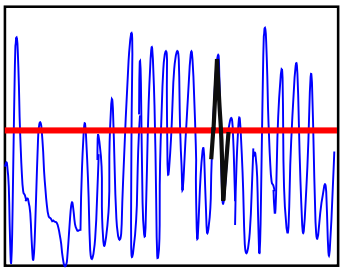


圖 2.2 棘波偵測示意圖

棘波偵測後，一般都會做特徵擷取，降低棘波序列的資料維度，以利後續的棘波分群。特徵擷取的部分在本論文是採用通用赫賓學習法則(Generalized Hebbian Algorithm , GHA)演算法，其避免了傳統主成分分析法則(Principal Component Analysis , PCA)需要計算共異數矩陣的複雜過程。棘波分群主要是利用特徵擷取後結果，即利用訓練後的特徵值進行運算，計算出每一群的質心，最後再利用質心將不同神經元所產生的棘波訊號加以分類。常見的棘波分群採用非監督式分群演算法(Fuzzy C-Means , FCM)，是經由 K-Means 演算法改良而來的，可以經由判斷設定群的數量。

2.2 公開的棘波分類軟體

Wave_clus 為加州里工學院在西元 2004 年所開的棘波分類系統[10]，作者為 Rodrigo Quian Quiroga，其開發目標為自動偵測棘波與棘波分類，在棘波偵測的部分使用自動閥值(Automatic Threshold)來進行偵測，其特徵擷取是使用小波轉換(Wavelets)，而棘波分類是使用(Superparamagnetic clustering , SPC)分群演算法則。相較於其它傳統的偵測與分類演算法，Wave_clus 有效的提高了其偵測與分類效果。且當中提供了從人類腦中所紀錄的真實棘波數據。而下圖 2.3 是 Wave_clus 的簡易操作，當中載入了真實的腦波數據並經過橢圓形濾波器後縮減了其振幅大小。

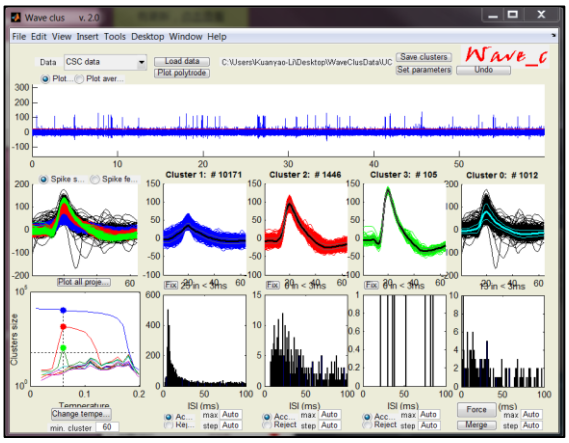


圖 2.3 Wave_clus 的簡易操作圖

OSort 為加州里工學院在西元 2005 年所開發的棘波分類系統，作者為 Ueli Rutishauser，其開發目標為做棘波分類，當中使用了一套自有的演算法來進行分類處理，以下來做一個簡單棘波分類示意圖介紹：

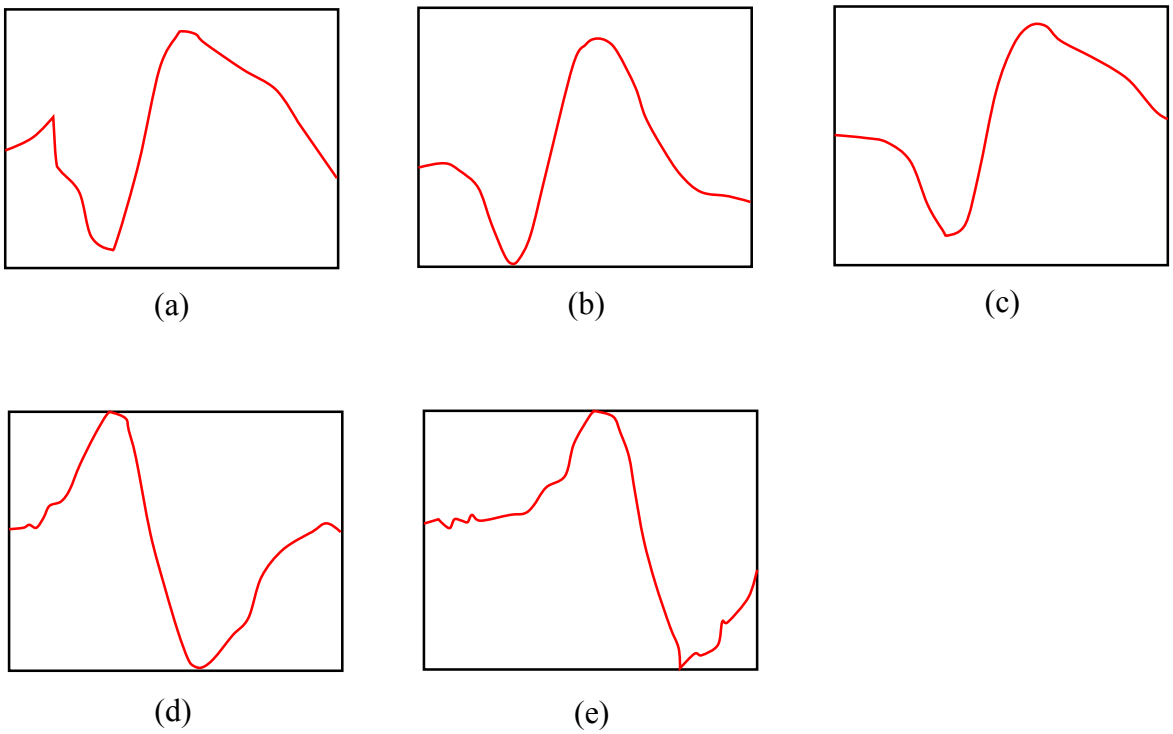


圖 2.4 棘波分類示意圖

假設示意圖(b)為第一個棘波時，第二個棘波(a)會與棘波(b)進行比對，其比對結果為相似，因此結合成了棘波(c)。當棘波(d)與棘波(c)進行比對時明顯不一樣，因此棘波(d)將變成一個新的比對的目標，即模板(Template)。假設經由多次結合後，產生了棘波(e)模板，由於棘波(d)與棘波(e)兩塊模板相似，因此 OSort 演算法運算過程會將兩塊相似模板結合，以提供有效的正確分類。

兩套棘波分類軟體都提供了一個簡易的操作介面，讓使用者能夠輕易的上手並進行分析，且當中都有使用到從人類腦中所紀錄的真實棘波數據，而兩套系統的最大不同點在於使用的棘波分類演算法，Wave_clus 使用小波轉換與 SPC 分群演算法，而 OSort 則是使用一套自有的演算法，類似於模板的比對方式，而詳細的差別可以參考下表 2.1。

	WaveClus	**OSort**
Features	Wavelet transform	Raw data points
Clustering method	Superparamagnetic clustering	Template matching
Open source	Yes	Yes
GUI available	Yes	Yes
Real Data	Yes	Yes
Sampling Frequency	32051	25000

表 2.1 兩套棘波分類軟體比較圖

2.3 通用赫賓學習法則(GHA)

全名為 Generalized Hebbian Algorithm，是棘波特徵擷取常用的法則之一，此法則之輸入向量代表了一個棘波，GHA 經由輸入向量去訓練出權重向量，訓練出權重向量後，將可輸入任意的棘波去計算出其特徵向量。

在這假設$x(n)$為輸入的棘波，令

$$x(n) = [x_1(n), \dots, x_m(n)]^T, n = 1, \dots, t \quad (1)$$

而$y(n)$代表了經由輸入向量所訓練出的特徵向量，在這令

$$y(n) = \left[y_1(n), \dots, y_p(n)\right]^T, n = 1, \dots, t \quad (2)$$

由上式(1)與(2)中探討，n 代表了第幾筆輸入向量與輸出向量，而 m 代表向量的維度，p 代表主成分個數、t 代表了輸入向量個數。而輸出向量$\mathbf{y}(n)$與輸入向量$\boldsymbol{x}(n)$的關係如下式(3)：

$$y_j(n) = \sum_{i=1}^{m} w_{ji}(n)x_i(n) \quad (3)$$

其中$w_{ji}(n)$代表第 n 次迭代過程中第 j 個神經元的第 i 筆突觸權重值。令

$$w_j(n) = \left[w_{j1}(n), \dots, w_{jm}(n)\right]^T, j = 1, \dots, p \quad (4)$$

由上式(3)與(4)中探討，j代表第幾筆突觸權重向量，每筆突觸權重向量$\boldsymbol{w}_j(n)$根據赫賓學習法則 (Hebbian learning rule) 來進行調整，為其公式(5)。其中 η 代表學習率。經過多次迭代計算調整後 $\boldsymbol{w}_j(n)$ 將趨近於輸入向量之共變異數矩陣的第 j 筆特徵值 λ_j。

$$w_{ji}(n+1) = w_{ji}(n) + \eta\left[y_j(n)x_i(n) - y_j(n)\sum_{k=1}^{j} w_{ki}(n)y_k(n)\right] \quad (5)$$

而在這為了降低計算時的複雜度，可將公式(5) 可以改寫成公式(6)：

$$w_{j,i}(n+1) = w_{j,i}(n) + \eta y_j(n)\left[x_i(n) - \sum_{k=1}^{j} w_{k,i}(n) y_k(n)\right] \quad (6)$$

由這些已訓練完成之突觸權重向量 $\mathbf{w}_j, j = 1, \ldots, p$ ，使用已訓練完之突觸權重向量與原本輸入之訓練向量做內積得到的特徵向量稱為 $\mathbf{f}_n$ ，如下式(8)(9)：

$$\mathbf{f}_n = \left[f_{n,1}, \ldots, f_{n,p}\right]^T \quad (8)$$

當

$$f_{n,j} = \sum_{i=1}^{m} w_{j,i} x_i(n) \quad (9)$$

為 $\mathbf{f}_n$ 中第 j 個元素。而獲得的特徵向量集合 $F = \{\mathbf{f}_1, \ldots, \mathbf{f}_t\}$，可用於後續之棘波分類，而更多關於 GHA 的應用[6,7]，將會在後續章節中討論。

在本論文實作中，我們設定採樣率(Sampling Rate)為 24000 採樣點/秒，由於每個棘波訊號長度為 2.67 毫秒，因此也代表著每個棘波訊號擁有 64 個採樣點(Sample)，也就是 m=64。我們同時讓主成份個數 p=2，且訓練向量總數為 t=800。

2.4 NoC 架構

NoC(Network on Chip)是一個將網路訊息傳遞應用於晶片上的一個新技術，解決了傳統的 SoC 通訊結構中有著延遲、通信效能、設計效率不佳的問題，NoC 可以說是 SoC 的升級版。為了確保每個元件間資料傳輸與訊號溝通的正確性，將網封包傳輸的概念應用於晶片上，讓不同元件間的資料能夠正確交換。

NoC 系統當中包含了 IP cores、網路介面、與路由器，下圖 2.5 即表示了 NoC 的基本架構：

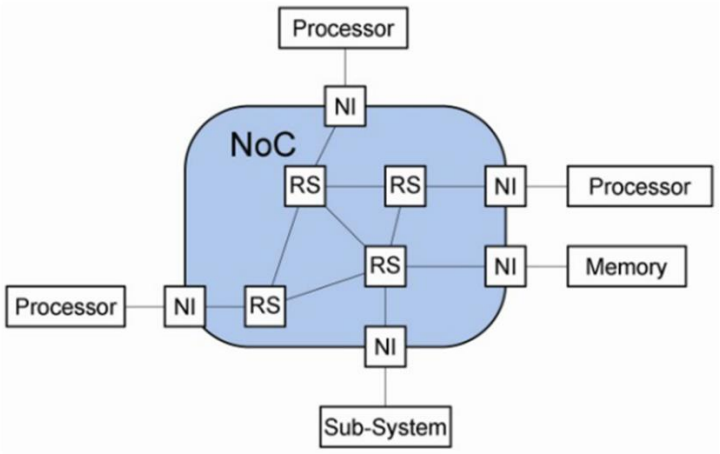


圖 2.5 NoC 的基本架構圖

主要是以封包的方式讓不同元件通訊，網路介面的功能是將資料做壓縮與解壓縮，讓各個元件間建立起連線，即取代了以往的匯流排。Router 在 NoC 中扮演著相當重要的角色，可以讓一些特定的元件來當 Router，它能讓封包化後的資料正確的傳送到目的地。採用 NoC 架構間的各個元件可以獨立運作，相較於 SoC，可以節省傳遞資料過程中所造成的資源浪費，也可以節省傳遞所花費的時間。NoC 相較於 SoC，對於實現多核心的嵌入式系統也有較好的效能。

SoC 使用了共用匯流排，同一時間只允許一個處理器進行存取，當處理器數量多的情況下，會造成了整個系統的效能變低，而 NoC 架構改善了這個缺點，因此整個效能有大幅的提升。由以下圖 2.6 與圖 2.7 中可以對於 NoC 有進一步了解。

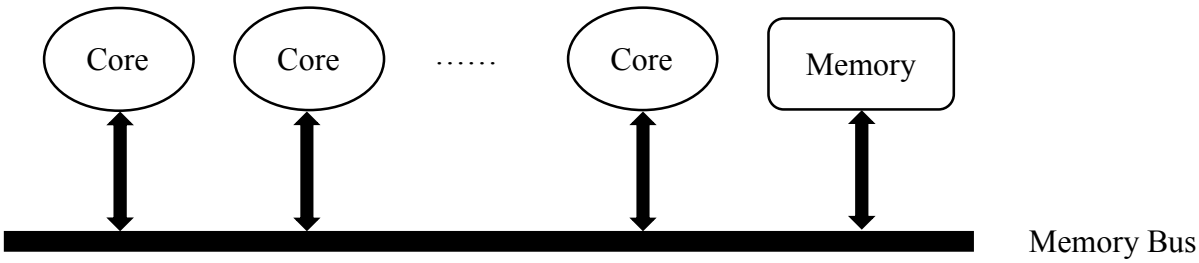


圖 2.6 傳統 SoC 使用了共用匯流排

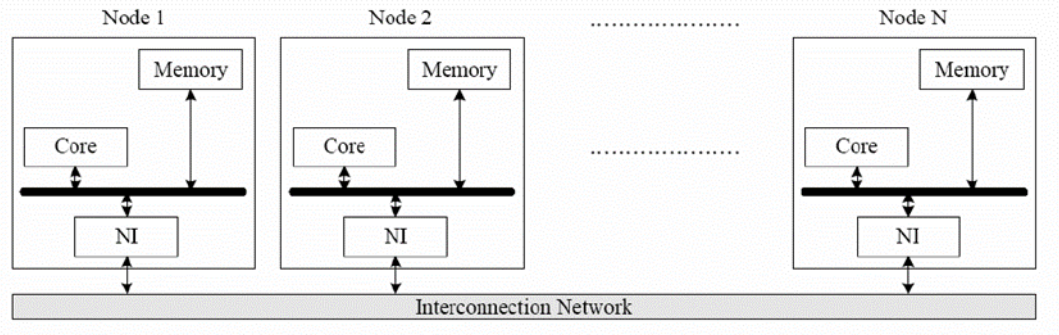


圖 2.7 NoC 架構間的各個元件可以獨立運作

第三章 系統架構

本章節會詳細介紹本論文所提出以區域網路實現跨平台之離線棘波分類系統，將說明系統架構、其運作流程，以及如何在 Network-On-a-Chip(NoC)平台中建立客製化邏輯電路之相關元件，其主要包含了 Triple-Speed Ethernet、On-Chip RAM、GHA 電路、以及 SG-DMA Controller、DMA Controller 兩個不同應用之元件，並說明其軟體設計之方法。

3.1 實現跨平台棘波分類系統之架構

本論文實現方法必須分為 MATLAB 軟體與 FPGA(Field-Programmable Gate Array)硬體兩個部分來設計，其目的為透過區域網路(Local Area Network, LAN)將棘波序列傳送至棘波分類法則中進行運算，最後將其運算結果傳回至 MATLAB 來進行處理。首先，先來介紹如何在 Altera 公司所推出的 FPGA 開發板中實現具有跨平台功能的棘波分類系統，棘波序列的來源必須透過 MATLAB 軟體來進行讀取，而在硬體部分必須能夠即時的使用其棘波序列，因此使用區域網路來進行高速傳輸，由於乙太網路的在非阻塞的情況下資料傳輸速率可高達 10Mb/s、100Mb/s、1000Mb/s，因此可以即時的將棘波序列來進行特徵擷取。由以下的系統架構圖中，可以清楚的了解到本論文透過 GHA 客製化邏輯電路來進行棘波特徵擷取，但該如何使用區域網路將棘波序列傳送至客製化邏輯電路

做運算呢？該如何將計算完後的特徵向量傳回到 MATLAB 進行處理呢？其必須根據上述軟硬體設計兩個部分來處理。軟體部分必須將讀取的棘波序列進行處理，而硬體部分必須透過 Altera 公司所推出的 Quartus II 10.1 後版本來進行開發，使用其開發系統內的工具 Qsys 來建立本論文在 NoC 平台上所需要的元件，其設計方法會在下一節做詳細介紹。硬體實現方法需分為兩個部分來處理。(1)使用 Quartus II 中的工具 Qsys 建立跨平台棘波分類系統，即在 NoC 平台所需要的元件，其主要包含函了 MAC Controller、DMA Controller 與 GHA 客製化邏輯電路。(2)在 Nios II 上使用 C 語言設計 Socket Server，將訓練向量傳送至客製化邏輯電路中進行運算。

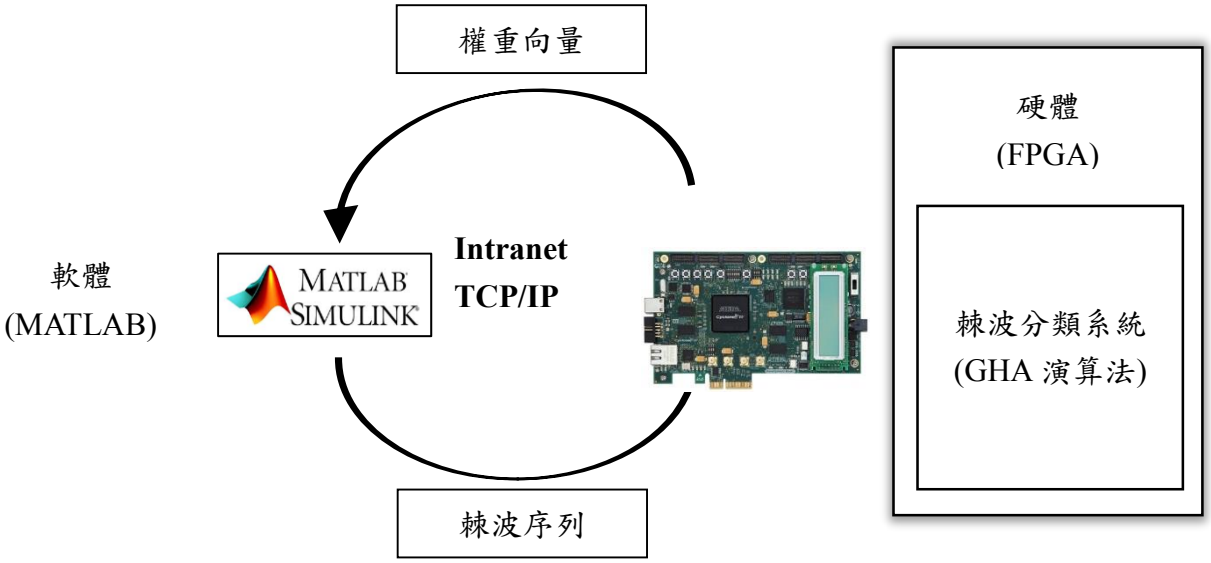


圖 3.1 透過區域網路建立跨平台棘波分類系統之架構圖

3.2 系統元件建立與其應用說明

使用 Qsys 建立 NoC 系統之程序包含建立元件以及將元件納入 Qsys 之專案中。有部分元件如 CPU 或者是 On-Chip RAM 等……可由 Qsys 取得不需要我們自行建立，但 GHA 電路需要由我們建立再納入系統中。本系統所採用的元件名稱以及其功能如表 3.1 所示，我們將在下面每一節逐一討論每一元件建立的流程。

元件名稱	功能	可否由 Qsys 取得
NIOS II	CPU	是
SSRAM Controller	儲存程式	是
On-Chip RAM	儲存 Spike 資料	是
JTAG	發展系統連接主電腦與 NOC 平台	是
Triple-Speed Ethernet	實現網路傳輸	是
SG-DMA Controller	網路與 On-Chip RAM 之資料傳送	是
DMA Controller	On-Chip RAM 與 GHA 之資料傳送	是
GHA	將棘波進行特徵擷取	否

表 3.1 本系統所採用的元件名稱以及其功能

3.2.1 建立 NIOS II 元件

首先，在這必須先打開 Quartus II 10.1 後的版本，因為是 Altera 所推出的新工具。在這點取選單 Tools 裡的 Qsys：

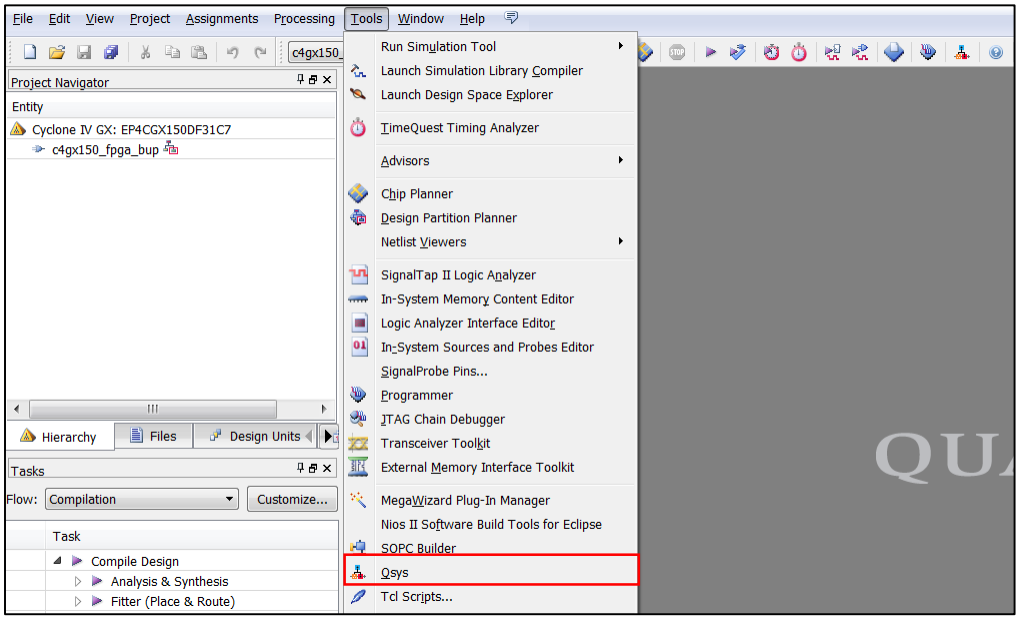


圖 3.2 Quartus II 10.1 後版本中提供了 Qsys 工具

在 Component Library 中輸入關鍵字 NIOS II，將會出現 Nios II Processor 供點選以進行參數設定。

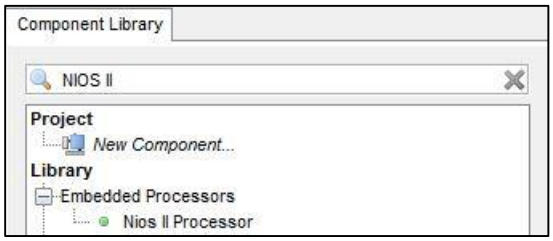


圖 3.3 透過 Library 新增 Triple-Speed Ethernet 元件

點選上方 Core Nios II 欄位，由圖 3.4 可見 Reset Vector 與 Exception Vector 欄位所設定的相關參數，選擇 ext_flash.uas 即可將程式放於 Flash 中，正常情況下由起始 0x1720000 位址開始執行，而發生例外的情形將經由所選擇之 ssram.uas 而去 SSRAM 中的 0x00000120 位址執行。

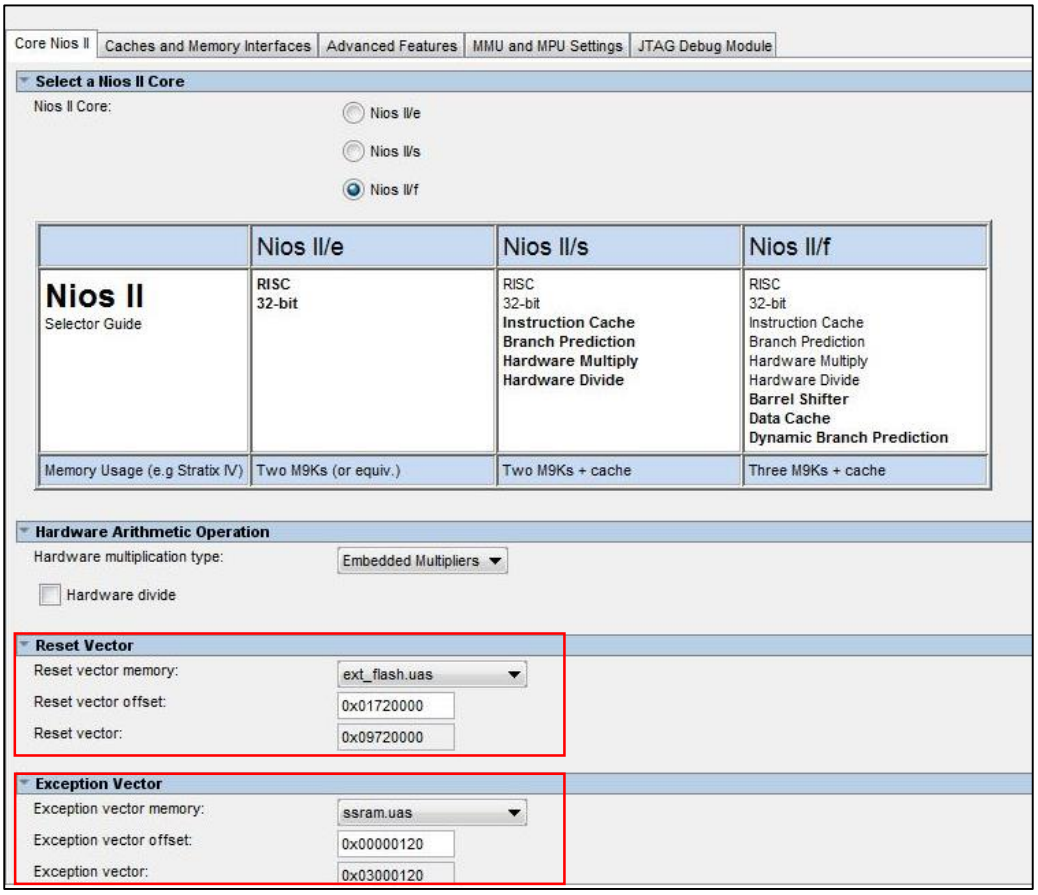


圖 3.4 Nios II Processor 之 Core Nios II 欄位相關參數設定

在 Caches and Memory Interfaces 欄位中，經由 Instruction cache 去設定為 4Kbytes，而 Data cache 設定為 2Kbytes，Data cache line size 設定為 32Bytes。

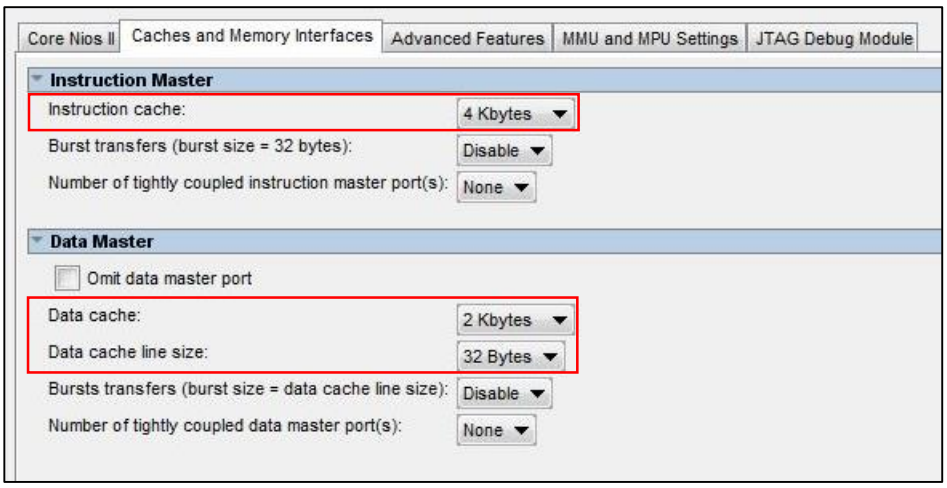


圖 3.5 Nios II Processor 之 Caches and Memory Interfaces 欄位相關參數設定

3.2.1 建立 SSRAM Controller 元件

同步靜態隨機存取記憶體(Synchronous Static Random Access Memory , SSRAM)，其功能為儲存程式，在 Component Library 中輸入關鍵字 Generic Tri-State Controller，將會出現 Generic Tri-State Controller 供點選以進行參數設定。

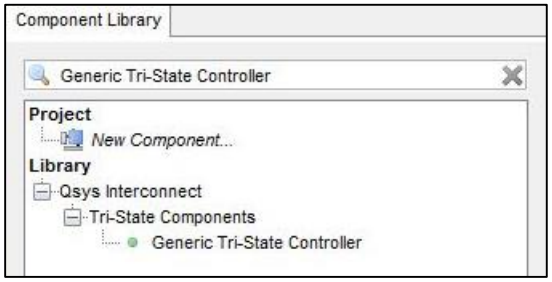


圖 3.6 透過 Library 新增 Generic Tri-State Controller 元件

在 Signal Selection 欄位中設定 Address width 為 32、Data width 為 16、Byteenable width 為 2、Bytes per word 為 2，在 Enable the following signals 中勾選設定資料可以讀寫以及一些額外的設定訊號。

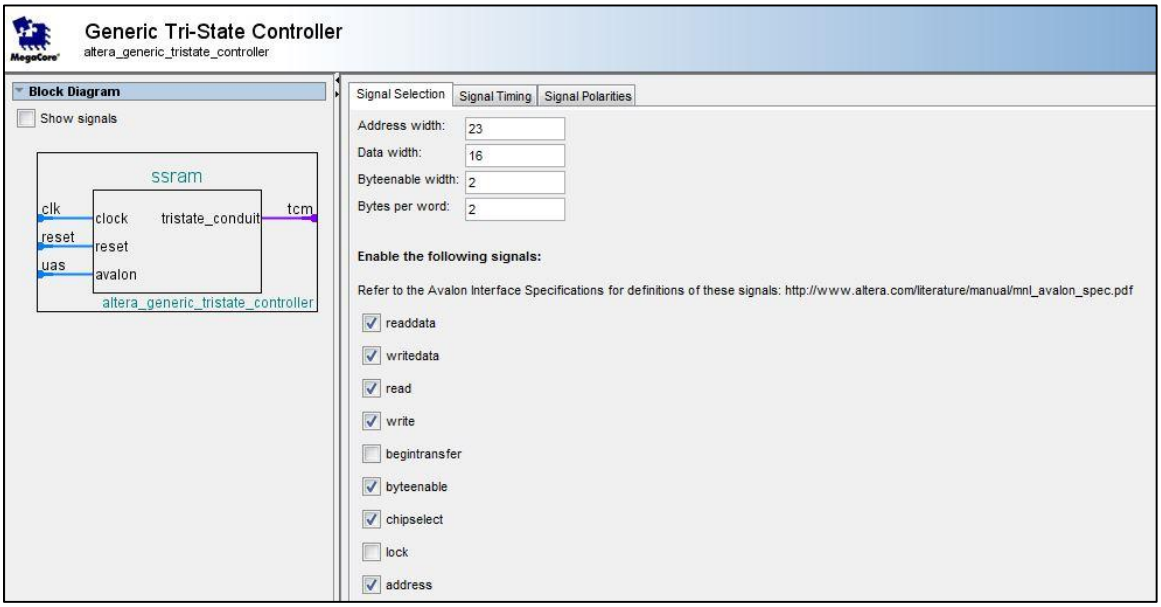


圖 3.7 Generic Tri-State Controller 中 Signal Selection 的參數設定

在 Signal Timing 中設定其 Read wait time 為 4、Write wait time 為 1，以及其相關參數設定。

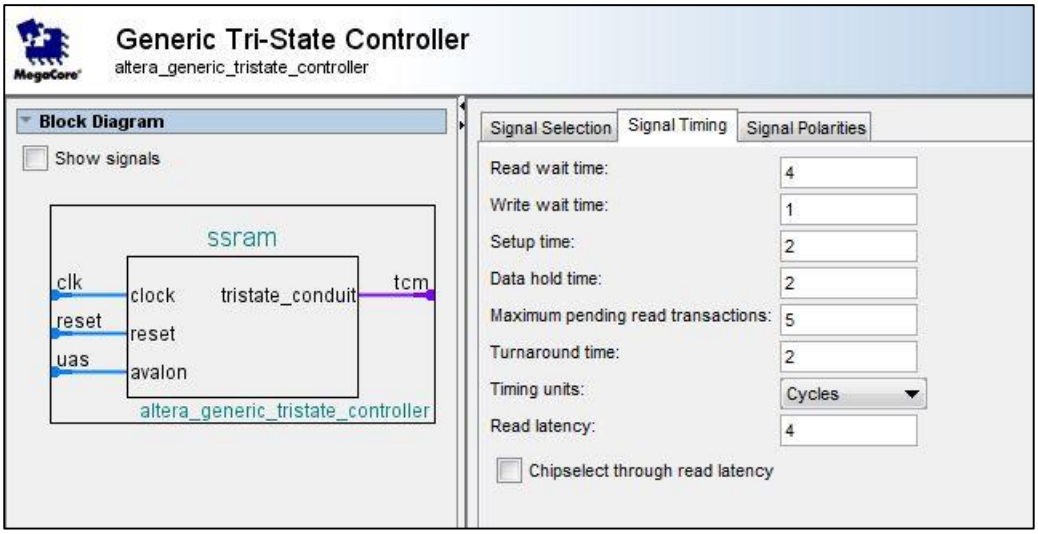


圖 3.8 Generic Tri-State Controller 中 Signal Timing 的參數設定

3.2.2 建立 On-Chip RAM 元件

在 Component Library 中輸入關鍵字 On-Chip Memory，將會出現 On-Chip Memory(RAM or ROM)供點選以進行參數設定。

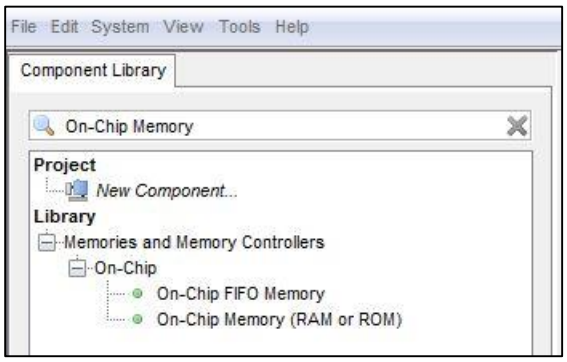


圖 3.9 透過 Library 新增 On-Chip Memory(RAM or ROM)元件

在這先新增實際所存放資料的 On-Chip RAM，在 Size 欄位的 Data with 與 Total memory size 中，可以設定接收的字串長度與設定其 Memory 總共的大小。而在 Memory initialization 欄位勾選 Initialize memory content 則可以初始化其記憶體，在每個區段位址的值為設定為 0。

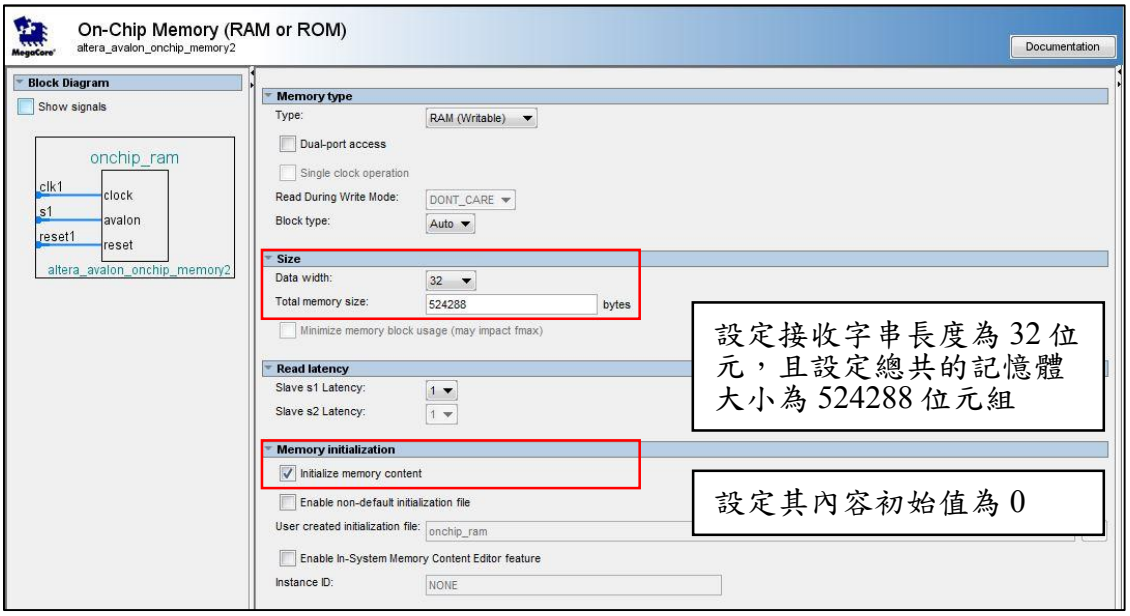


圖 3.10 設定 On-Chip Memory 參數

3.2.3 建立 JTAG 元件

其主要為發展系統連接主電腦與 NOC 平台的重要元件，首先在在 Component Library 中輸入關鍵字 JTAG，將會出現 JTAG UART 供點選以進行參數設定。

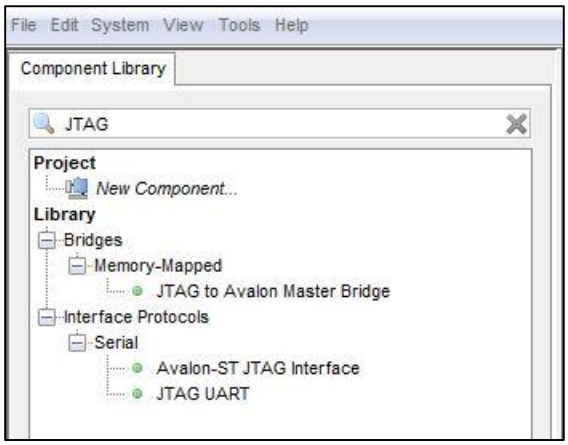


圖 3.11 透過 Library 新增 JTAG UART 元件

在 Write FIFO(Data from Avalon to JTAG)中設定 Buffer depth 為 512 位元組，而 IRQ threshold 設定為 8，在 Read FIFO(Data from JTAG to Avalon)中設定 Buffer depth 為 512 位元組，而 IRQ threshold 設定為 8。

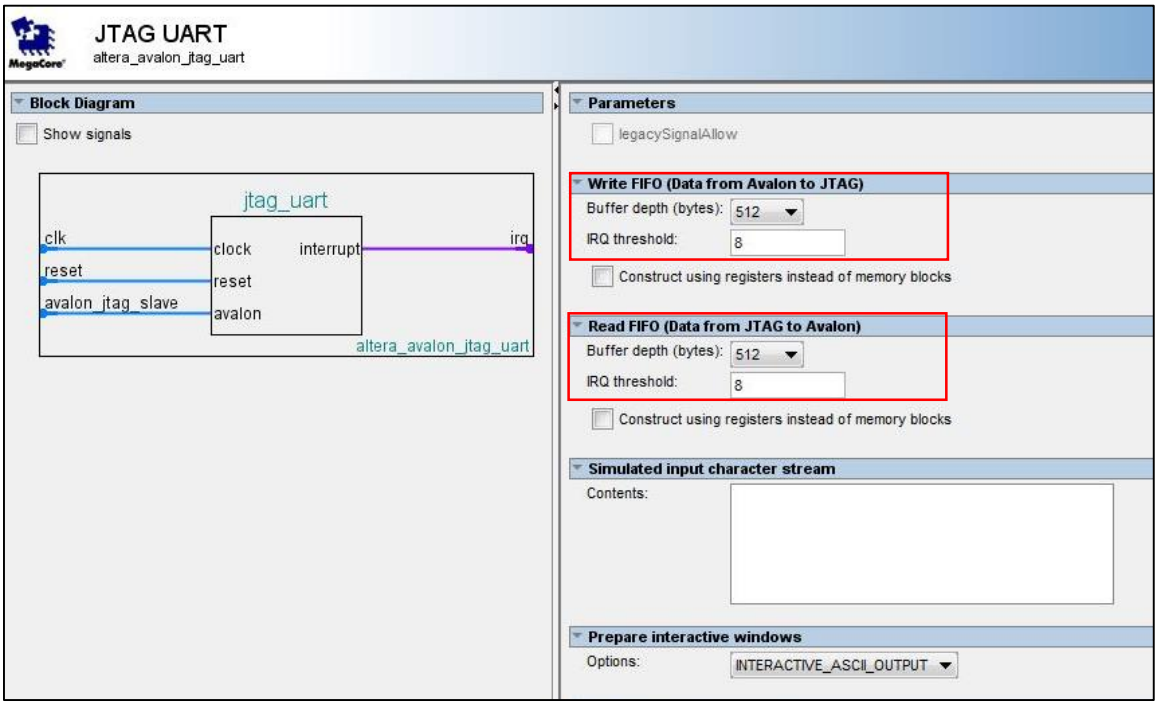


圖 3.12 設定 JTAG 元件之參數

3.2.4 建立 Triple-Speed Ethernet 元件

元件中包含了傳送和接收的 FIFO Buffer，其 Memory 的寬度在這設定為 32Bits，其 FIFO Buffer 的大小會依據 Memory 寬度呈倍數成長，所以這邊設定傳送和接收的 FIFO Buffer 大小為 1024x32Bits，即 4096 位元組。

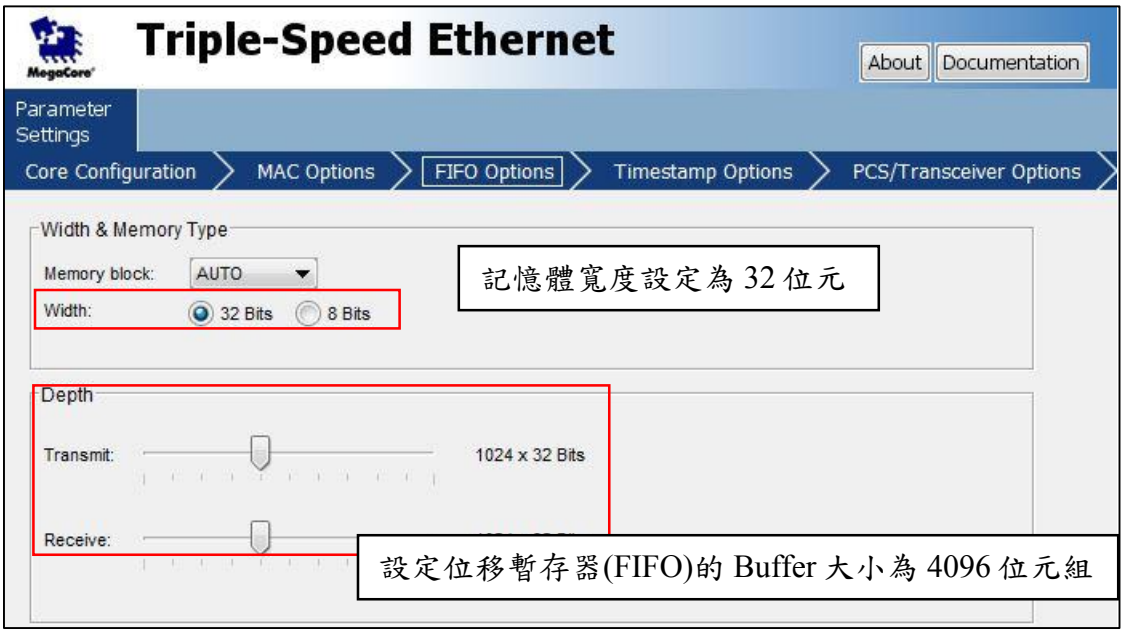


圖 3.13 設定 Triple-Speed Ethernet 元件之參數

3.2.5 建立 Scatter-Gather DMA Controller 元件

在 Library 輸入 DMA 關鍵字，選擇 Scatter-Gather DMA Controller。

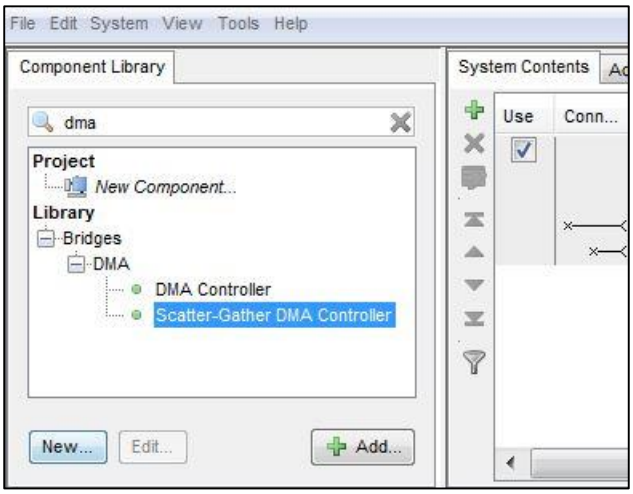


圖 3.14 透過 Library 新增 Scatter-Gather DMA Controller 元件

Transmit：設定 Memory 中的資料傳送到 MAC Controller 的 FIFO Buffer 中。

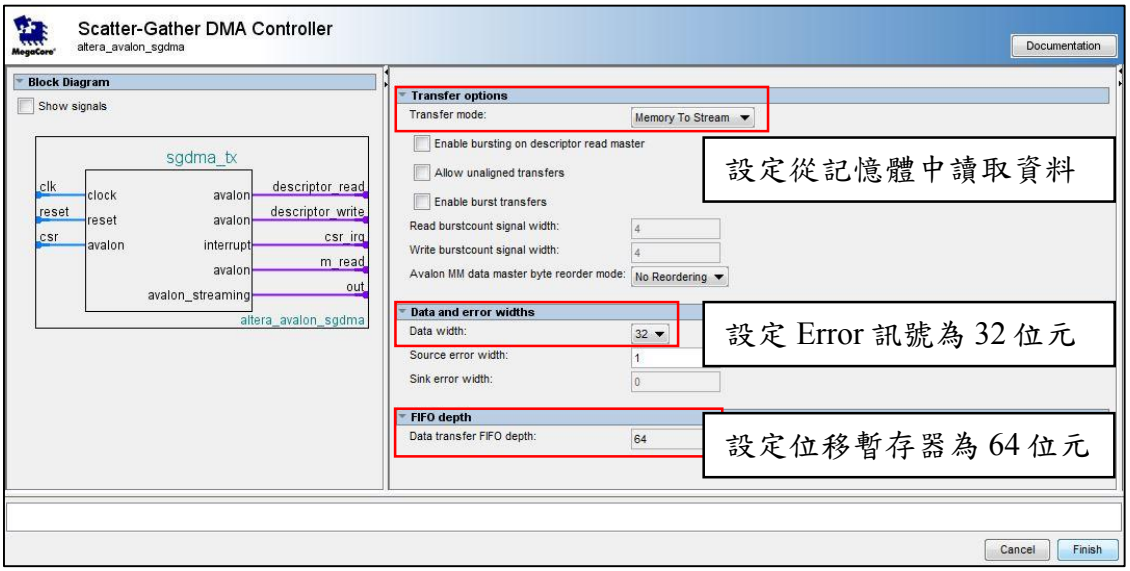


圖 3.15 設定 Scatter-Gather DMA Controller 參數

Receive：設定 MAC Controller 的 FIFO Buffer 中的資料傳送到 Memory 中。

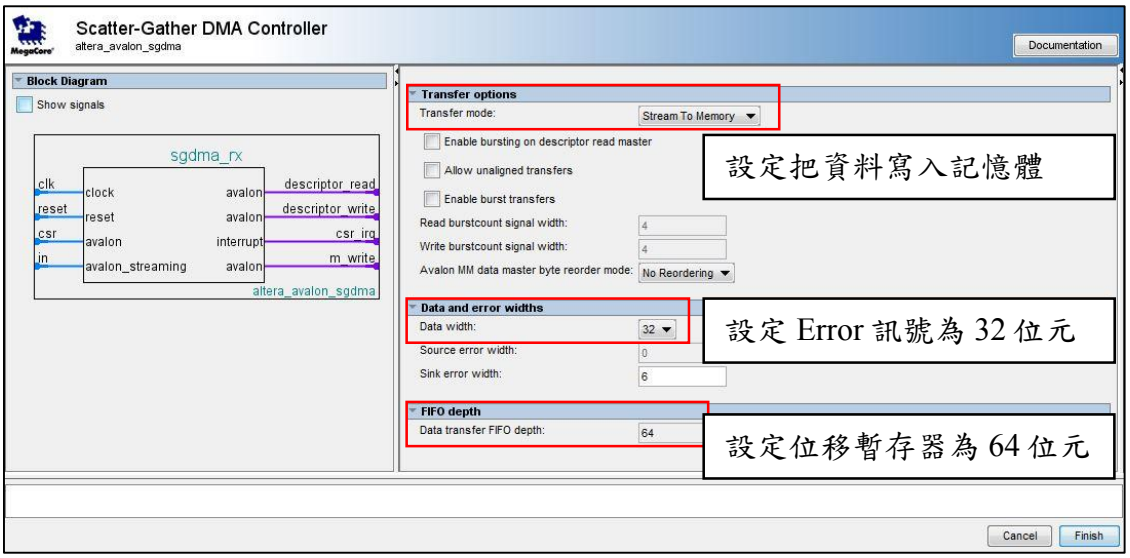


圖 3.16 設定 Scatter-Gather DMA Controller 參數

3.2.6 建立 DMA Controller 元件

在這使用 DMA Controller 來將訓練資料快速的傳給電路，而以下是其參數設定以及說明，其 Transfer size 設定為 32bits、位移暫存器大小設定為 32。而在 Advanced 中設定其傳輸的相關設定。詳情可由下圖 3.17、圖 3.18 可以得知。

圖 3.17 DMA Controller 參數設定

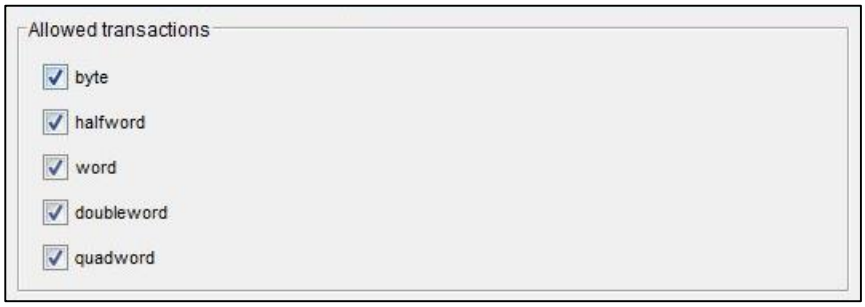


圖 3.18 DMA Controller 參數設定

3.2.7 建立 GHA 元件

在這將使用本研究室現有的客製化邏輯電路來建立 GHA 元件[9]，首先，在 Component Library 中點擊 New Component 來加入我們所需要的客製化電路 GHA，由下圖 3.19 可見，在這設定客製化電路名稱。

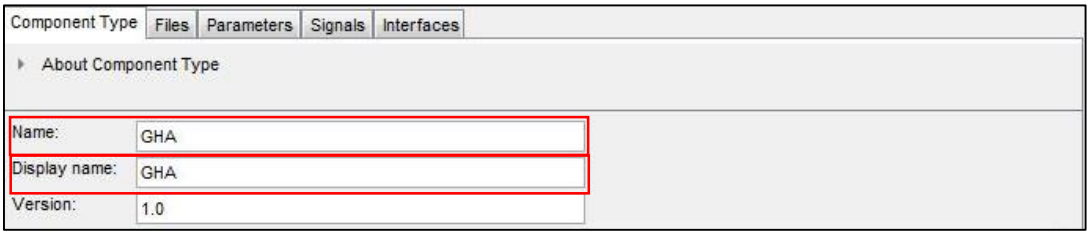


圖 3.19 設定客製化電路名稱

由下圖 3.20 可見，在這加入硬體描述語言 Verilog 所撰寫的檔案，即 GHA 的電路設計相關檔案，並在加入完後使用 Analyze Synthesis File、Copy from Synthesis Files 進行分析與複製。

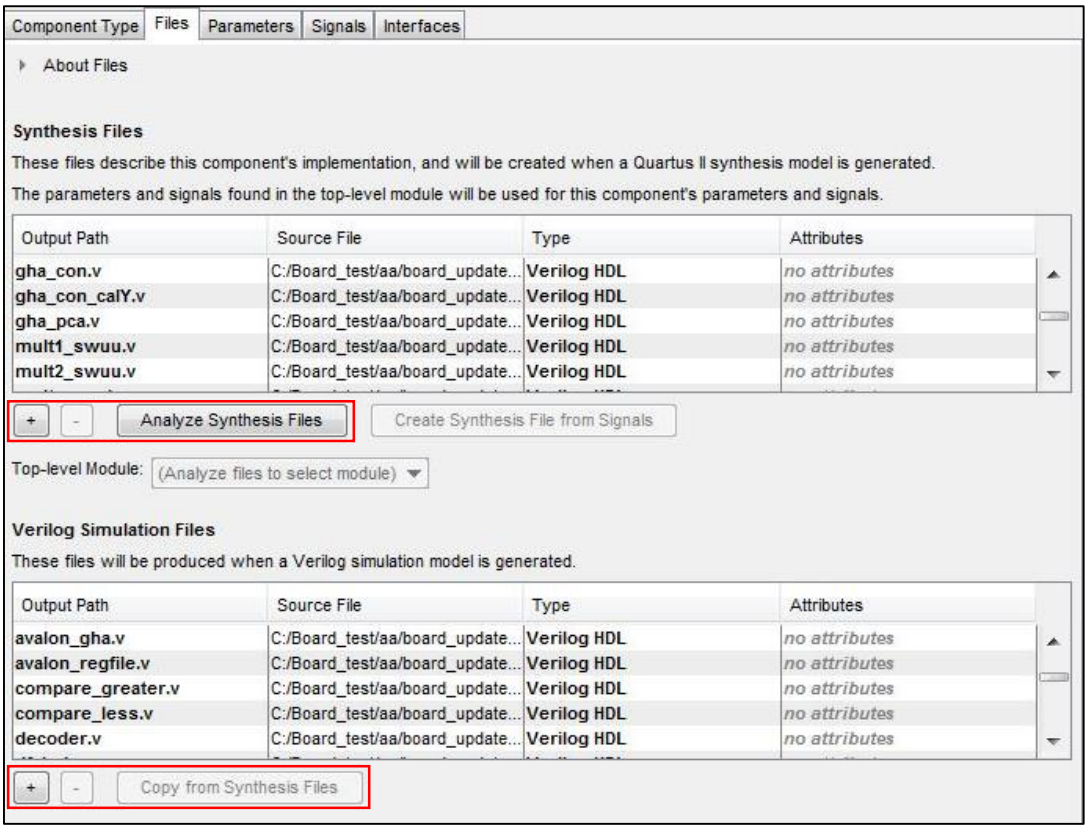


圖 3.20 加入硬體描述語言 Verilog 所撰寫的檔案

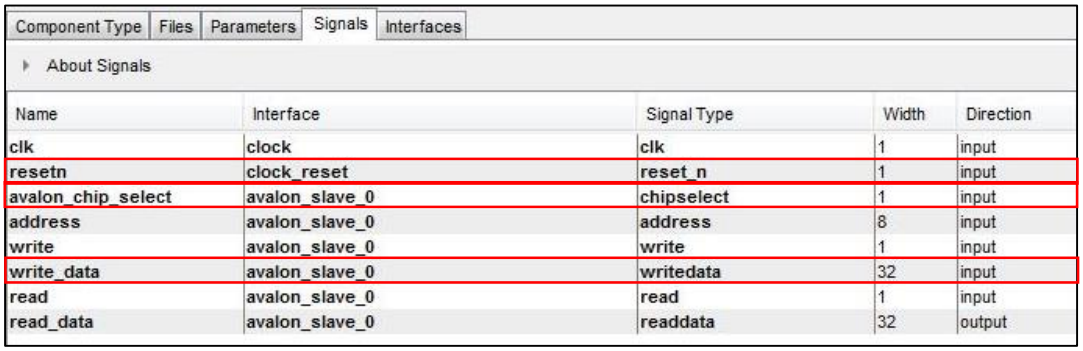
接下來在 Signal 欄位中設定 Interface 的訊號總類，由下圖 3.21、圖 3.22 清楚的可見，首先 resetn 的 Interface 由 avalon_slave_0 調整為 clock_reset，而 Signal Type 則由 beginbursttransfer_n 調整為 reset_n。而 avalon_chip_select 與 write_datad 的 Signal Type 皆各調整為 chipselect 與 writedata，即設定好訊號種類了。


Component Type | Files | Parameters | Signals | Interfaces

About Signals

Name	Interface	Signal Type	Width	Direction
clk	clock	clk	1	input
resetn	avalon_slave_0	beginbursttransfer_n	1	input
avalon_chip_select	avalon_slave_0	beginbursttransfer_n	1	input
address	avalon_slave_0	address	8	input
write	avalon_slave_0	write	1	input
write_data	avalon_slave_0	writebyteenable_n	32	input
read	avalon_slave_0	read	1	input
read_data	avalon_slave_0	readdata	32	output

圖 3.21 原本預設的訊號總類


Component Type | Files | Parameters | Signals | Interfaces

About Signals

Name	Interface	Signal Type	Width	Direction
clk	clock	clk	1	input
resetn	clock_reset	reset_n	1	input
avalon_chip_select	avalon_slave_0	chipselect	1	input
address	avalon_slave_0	address	8	input
write	avalon_slave_0	write	1	input
write_data	avalon_slave_0	writedata	32	input
read	avalon_slave_0	read	1	input
read_data	avalon_slave_0	readdata	32	output

圖 3.22 經由調整後的訊號總類

而訊號總類都設定完後，將對所有 Interface 的參數進行調整，由下圖 3.23、圖 3.24 可見，原本的 avalon_slave_0 中的 Associated Reset 欄位為 none，經由調整後設定為 clock_reset，而在 Parameters 中的 Associated reset 欄位中的值將會自動設定為 clock_reset，接下來要調整 Timming 中的 Read wait 欄位，原本的值為 1，在這將設定為 0。

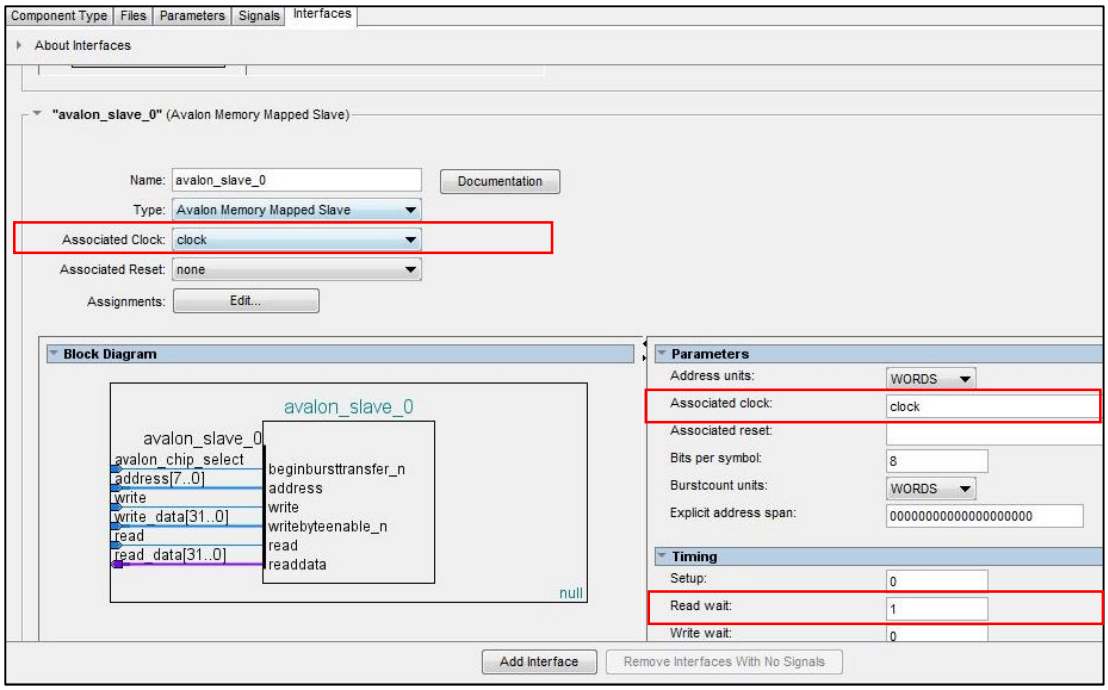


圖 3.23 原本 Interface 的參數設定

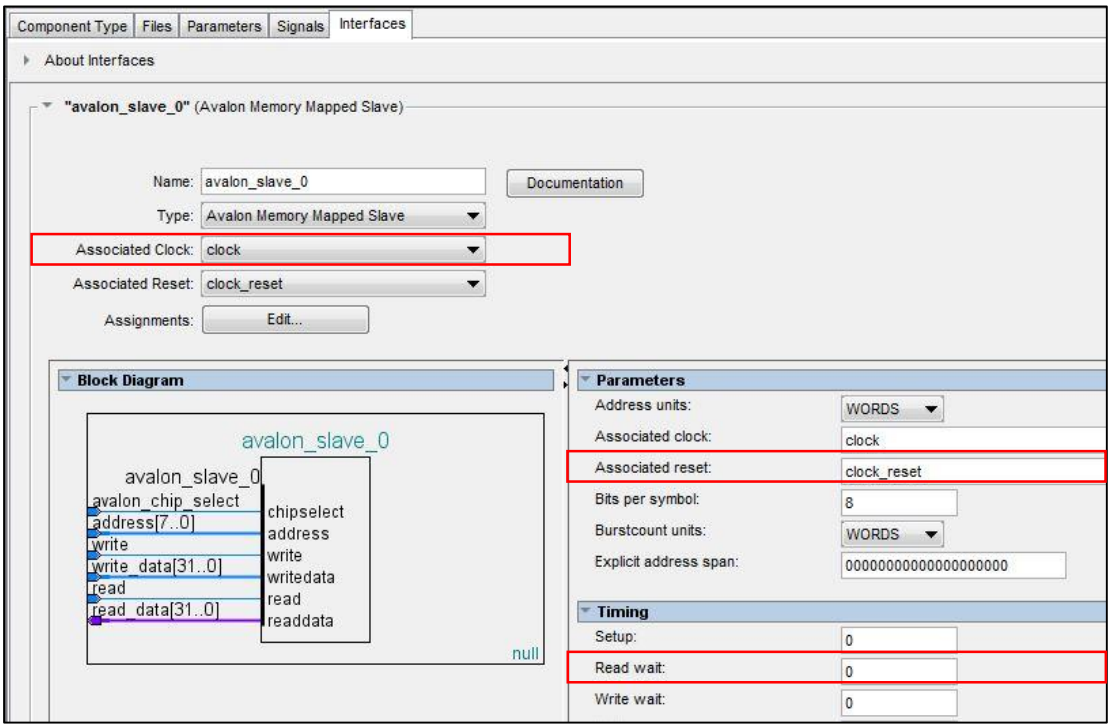


圖 3.24 經由調整後的 Interface 參數

3.3 系統整合與設計

由於系統元件都已新增完畢，在這必須將其元件整合再經由 C 語言設計後，才能將所接收到的訓練向量放入 On-Chip RAM 中，但如何將訓練資料放入 On-Chip RAM 中呢？其整個資料傳送至 GHA 電路的運作流程如下圖 3.25 可以得知，MATLAB 透過 TCP/IP 網路將資料傳送至 Ethernet Controller，接下來將所接收到的資料使用 SG-DMA Controller 放入置 On-Chip RAM 中，接著使用 DMA Controller 將資料快速傳送至 GHA 中進行運算。

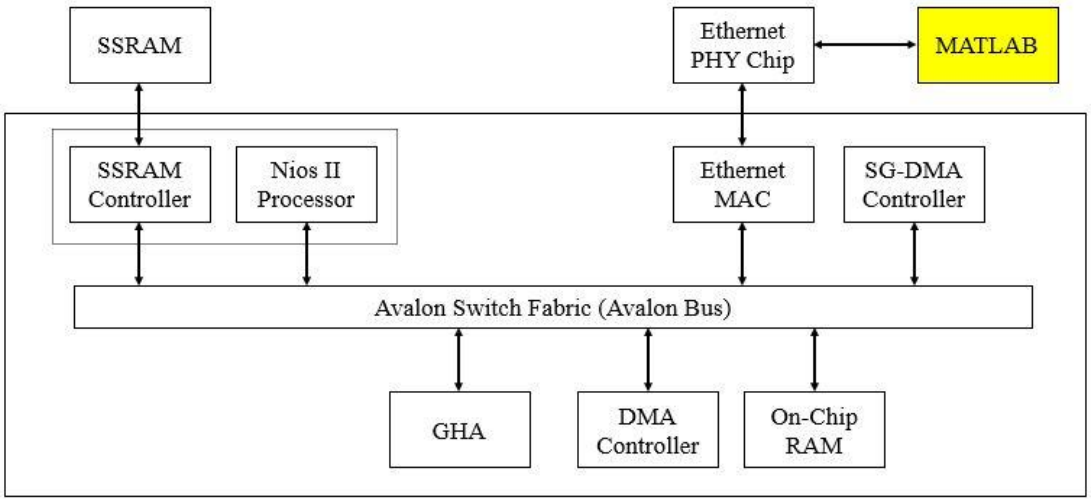


圖 3.25 資料傳送至 GHA 之運作流程

了解到了其資料傳送的運作流程，因此必須介紹整個 Triple-Speed Ethernet 的核心架構，以及介紹如何整合 On-Chip RAM 與 Triple-Speed Ethernet MAC Controller，其目的為設定 MAC Controller 之 Target Address 為 On-Chip RAM 之 Device Address。

由下圖 3.26 可以得知，在 FPGA 上使用 Triple-Speed Ethernet 核心架構中包含了兩個 FIFO Buffer，分別為 TX 和 RX FIFO，比較特別的可以看到有兩個 Memory，其中 Descriptor Memory 是用來紀錄資料的位址，而 On-Chip RAM 是用來實際儲存資料的。而兩個 Scatter-Gather DMA 是用來控制接收和傳送資料到 Memory 的，最後 CPU 會透過存放在 Descriptor Memory 的資訊來取得 On-Chip RAM 中的資料，而詳細的整合方法將在下一段做說明。

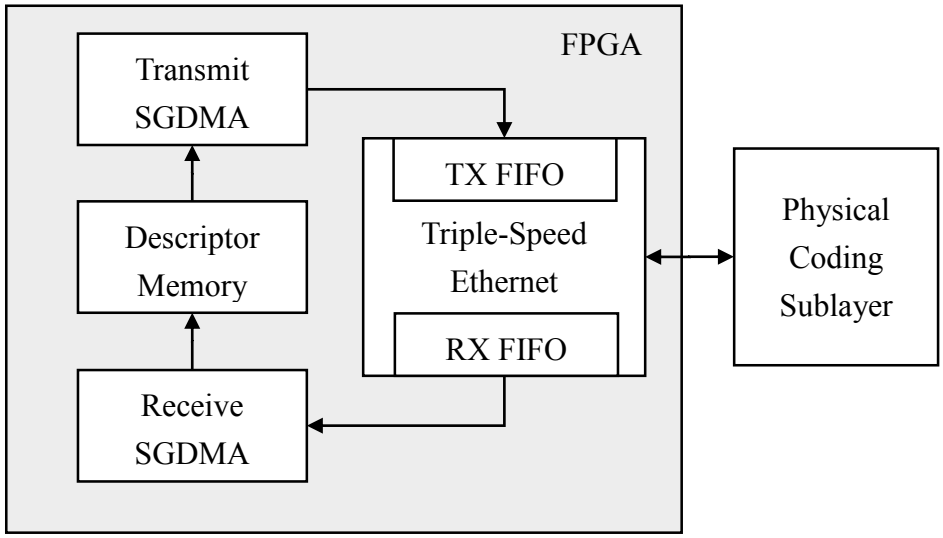


圖 3.26 使用 Triple-Speed Ethernet 核心架構圖

3.3.1 在 Qsys 中連接各元件

使用 Scatter-Gather DMA 將所接收到的資料位址紀錄在 Descriptor Memory 中，必須將 Qsys 中元件的 Port 做連接，以下將做接收與傳送資料的 Port 連接介紹。這邊先處理接收到資料的部分：(a)先將 Triple-Speed Ethernet 的通訊埠 receive，連接到 Scatter-Gather DMA Controller 的通訊埠 in。(b)再透過

Scatter-Gather DMA Controller 的通訊埠 m_write 連接到 On-Chip Memory 的通訊埠 s1，descriptor_read 和 descriptor_write 連接到 Descriptor Memory 的通訊埠 s1。發出中斷訊息後，CPU 即會發出命令，透過連接埠 data_master 向 Descriptor Memory 與 On-Chip RAM 取得資料。

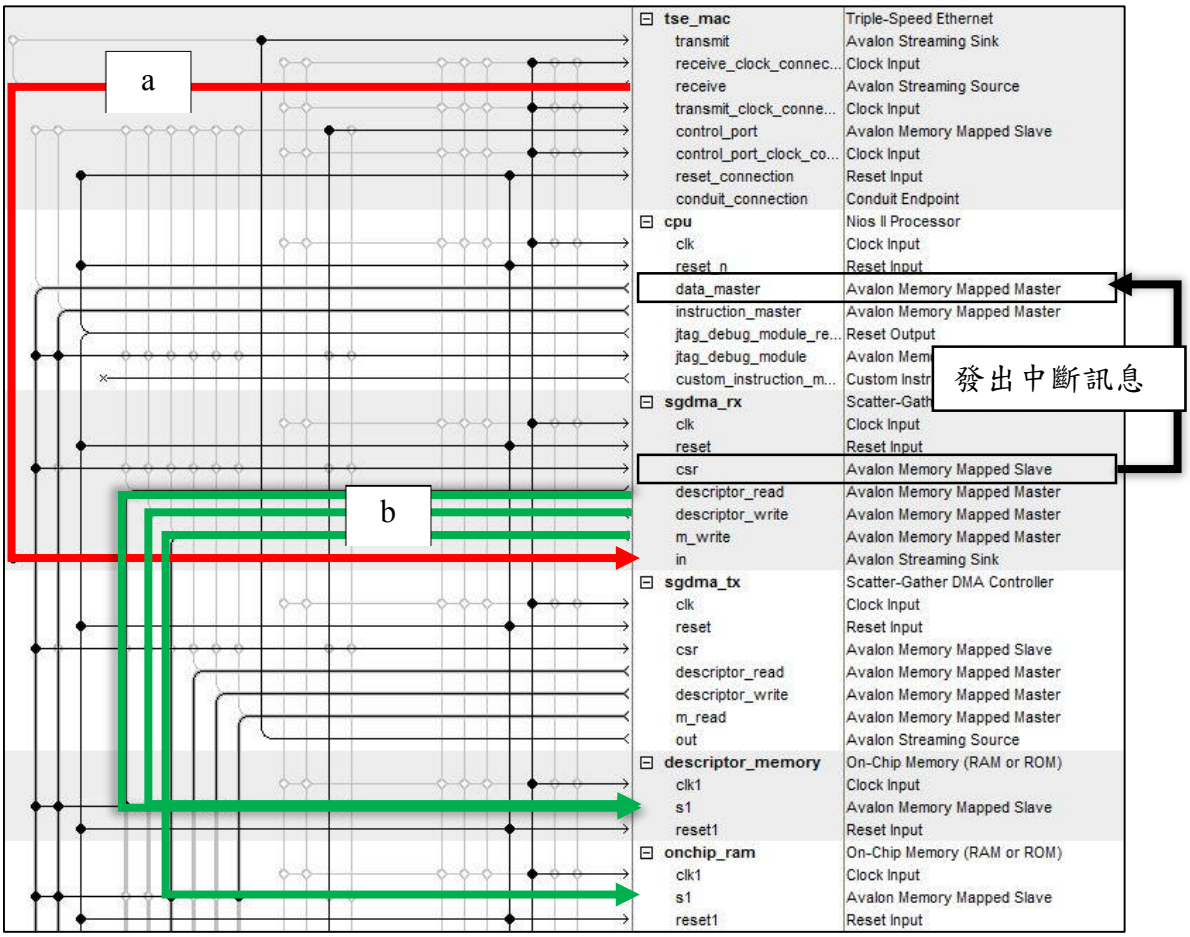


圖 3.27 Qsys 設計元件處理接收到資料的部分通訊埠連接圖

再來處理傳送資料的部分：(c)將 Triple-Speed Ethernet 的通訊埠 transmit，連接到 Scatter-Gather DMA Controller 的通訊埠 out。(d)再透過 Scatter-Gather DMA Controller 的通訊埠 m_read 連接到 On-Chip Memory 的通訊埠 s1，descriptor_read 和 descriptor_write 連接到 Descriptor Memory 的 s1。發出中斷訊息後，CPU 即會發出命令，透過連接埠 data_master 向 Descriptor Memory 與 On-Chip RAM 取得資料。

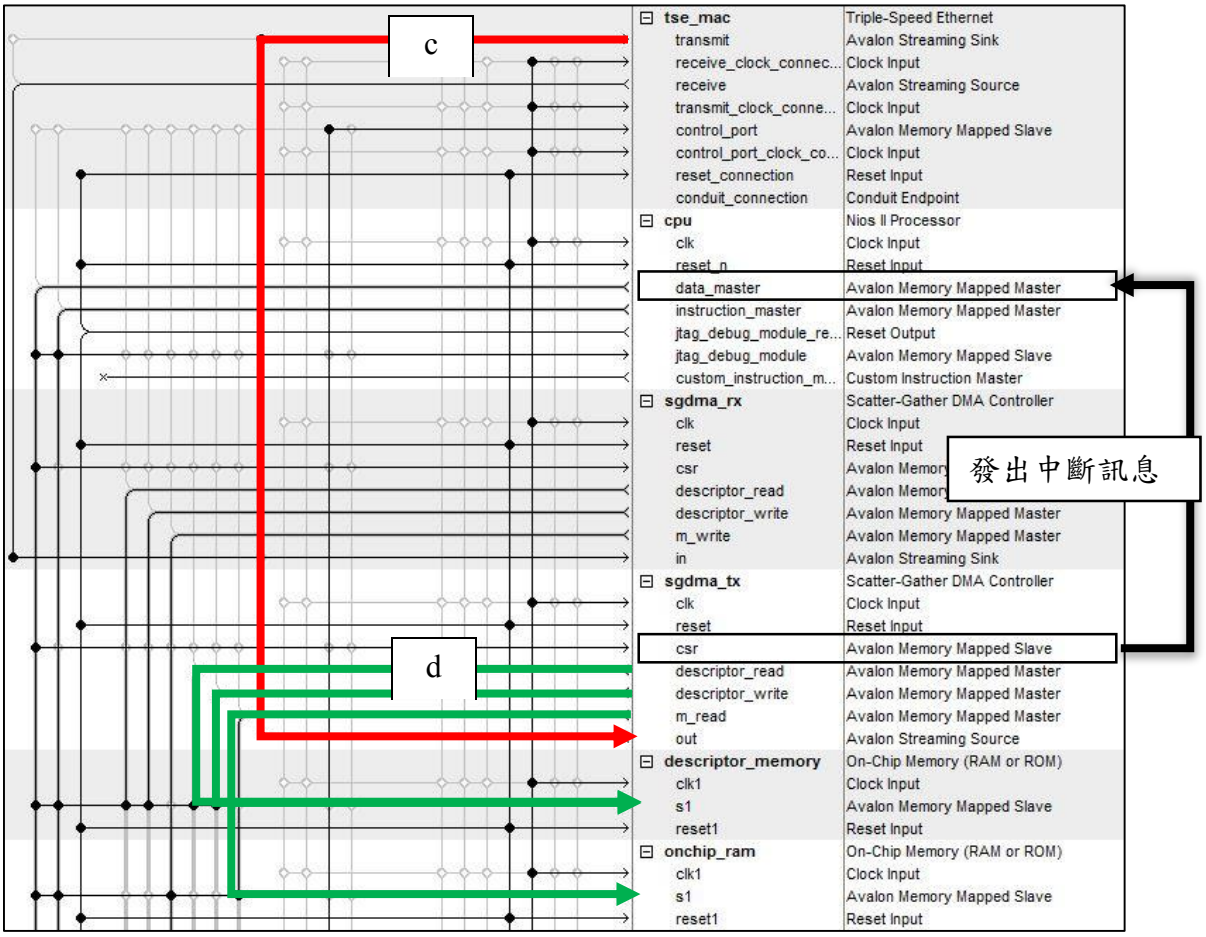


圖 3.28 Qsys 設計元件處理傳送資料的部分通訊埠連接圖

3.3.2 使用 C 語言整合應用

使用 Qsys 工具建立完 NoC 平台元件後，接下來要處理以 C 語言控制 Nios 部分，首先，必須使用 Nios II 建立一個 Simple Socket Server (RGMII)的 Template，能夠接收 MATLAB 的棘波序列與將計算完後的權重向量傳回去，那如何將所接收到的訓練向量從 MAC Controller 放入至 On-Chip RAM 中呢？在 altera_avalon_tse.c 檔的 tse_mac_sTxWrite 函式中設定其相關參數，當 MAC Controller 接收到網路所傳送的資料後，透過下圖 3.29， 了解到了 MAC Controller 在進行資料傳送前會設定初始傳送之參數 tse_mac_base、sgdma_txdev 與 sgdma_rx_dev，即初始化設定了 TSE MAC Controller 的起始位址，以及 Transmit 與 Receive 的 SG-DMA 的起始位址。而其起始位址的資訊都是由 system.h 檔取得。經由初始後即可與如圖 3.30、圖 3.31 中使用 alt_avalon_sgdma_do_async_transfer 函式來進行資料傳送。

```
/* Get the Rx and Tx SGDMA addresses */
sgdma_tx_dev = alt_avalon_sgdma_open(tse_hw->tse_sgdma_tx);

if(!sgdma_tx_dev) {
   dprintf("[triple_speed_ethernet_init] Error opening TX SGDMA\n");
   return ENP_RESOURCE;
}

sgdma_rx_dev = alt_avalon_sgdma_open(tse_hw->tse_sgdma_rx);
if(!sgdma_rx_dev) {
   dprintf("[triple_speed_ethernet_init] Error opening RX SGDMA\n");
   return ENP_RESOURCE;
}

/* Initialize mtip_mac_trans_info structure with values from <system.h>*/
tse_mac_initTransInfo2(&tse[iface].mi, (int)tse_hw->tse_mac_base,
                                  (unsigned int)sgdma_tx_dev,
                                  (unsigned int)sgdma_rx_dev,
                                  0);
```

圖 3.29 設定初始傳送之參數 tse_mac_base、sgdma_txdev、sgdma_rx_dev

```
alt_32 tse_mac_sTxWrite( tse_mac_trans_info *mi,
                         alt_sgdma_descriptor *txDesc)
{

  alt_32 timeout;
  alt_u8 result = 0;
  alt_u16 actualBytesTransferred;

  // Make sure DMA controller is not busy from a former command
  // and TX is able to accept data
  timeout = 0;
  //tse_dprintf("\nWaiting while tx SGDMA is busy..... ");
  while ( (IORD_ALTERA_AVALON_SGDMA_STATUS(mi->tx_sgdma->base) &
           ALTERA_AVALON_SGDMA_STATUS_BUSY_MSK) ) {
           if(timeout++ == ALTERA_TSE_SGDMA_BUSY_TIME_OUT_CNT) {
            tse_dprintf(4, "WARNING : TX SGDMA Timeout\n");
            return ENP_RESOURCE;  // avoid being stuck here
           }
  }

  // Set up the SGDMA
  // Clear the status and control bits of the SGDMA descriptor
  IOWR_ALTERA_AVALON_SGDMA_CONTROL (mi->tx_sgdma->base, 0);
  IOWR_ALTERA_AVALON_SGDMA_STATUS (mi->tx_sgdma->base, 0xFF);

  // Start SGDMA (blocking call)
  result = alt_avalon_sgdma_do_sync_transfer(
                mi->tx_sgdma,
                (alt_sgdma_descriptor *) &txDesc[0]);

  /* perform cache save read to obtain actual bytes transferred for current sgdma descriptor */
  actualBytesTransferred = IORD_ALTERA_TSE_SGDMA_DESC_ACTUAL_BYTES_TRANSFERRED(&txDesc[0]);

  return actualBytesTransferred;
}
```

圖 3.30 alt_avalon_sgdma_do_async_transfer 函式來進行資料接收與傳送

```
alt_32 tse_mac_aRxRead(
  tse_mac_trans_info *mi,
  alt_sgdma_descriptor *rxDesc)
{
  alt_32 timeout;

  alt_u8 result = 0;

  // Make sure SGDMA controller is not busy from a former command
  timeout = 0;
//  tse_dprintf("\nWaiting while rx SGDMA is busy.........");
  while ( (IORD_ALTERA_AVALON_SGDMA_STATUS(mi->rx_sgdma->base) &
           ALTERA_AVALON_SGDMA_STATUS_BUSY_MSK) ) {
    if(timeout++ == ALTERA_TSE_SGDMA_BUSY_TIME_OUT_CNT) {
        tse_dprintf(4, "WARNING : RX SGDMA Timeout\n");
        return ENP_RESOURCE;  // avoid being stuck here
    }
  }


  // SGDMA operation invoked for RX (non-blocking call)
  result = alt_avalon_sgdma_do_async_transfer(
                mi->rx_sgdma,
                (alt_sgdma_descriptor *) &rxDesc[0]);


  return SUCCESS;
}
```

圖 3.31 alt_avalon_sgdma_do_async_transfer 函式來進行資料接收與傳送

由 simple_socket_server.h 中可設定其硬體 FPGA 之 IP 位址、預設閘道、遮照、與 PORT 之參數，以利與 MATLAB 端建立連線。由下圖 3.32 將可清楚的知道其設定之參數。 在這設定 IP 位址，其參數為 IPADDR0=140、IPADDR1=122、IPADDR2=184、IPADDR3=1，設定閘道參數 GWADDR0=140、GWADDR1=122、GWADDR=184、GWADDR=254，設定遮罩參數 MSKADDR0=255、 MSKADDR1= 255、MSKADDR2=255、MSKADDR3=0，最後設定其連接埠參數 SSS_PORT=30，將可完成 IP 相關參數之設定。

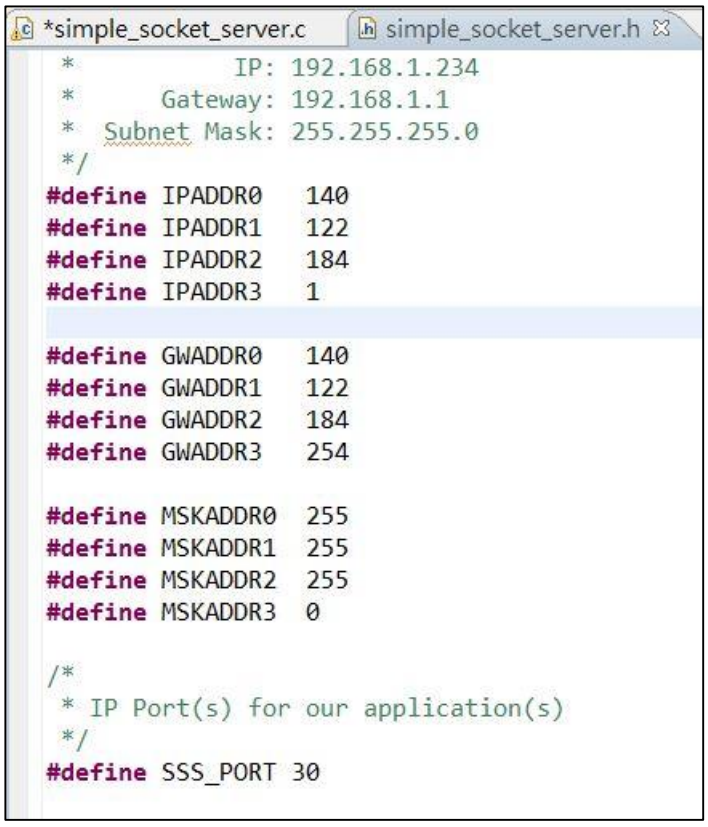


圖 3.32 IP Address 相關參數之設定

設定完後，即能在 Socket 專案中使用 recv()與 send()函式接收與傳送資料，而使用 send()函數來向 TCP 連接的另一端發送數據，總共有四個參數要設定，

第一個參數是指向目的地的指標，第二個參數代表要傳送的資料，即為我們所訓練的權重向量，第三個參數代表所傳送的資料大小，第四個參數通常設為 0，是傳送資料的起始位址。而 recv()函數則是向 TCP 連接的另一端接收數據，且總共有四個參數要設定，第一個參數是指向目的地的指標，第二個參數代表要接收的資料，即為我們所訓練向量，第三個參數代表所接收的資料大小，第四個參數通常設為 0，是接收資料的起始位址。而詳細流程與整個建立連線至接收資料的流程可以先參考下圖 3.33、圖 3.34。

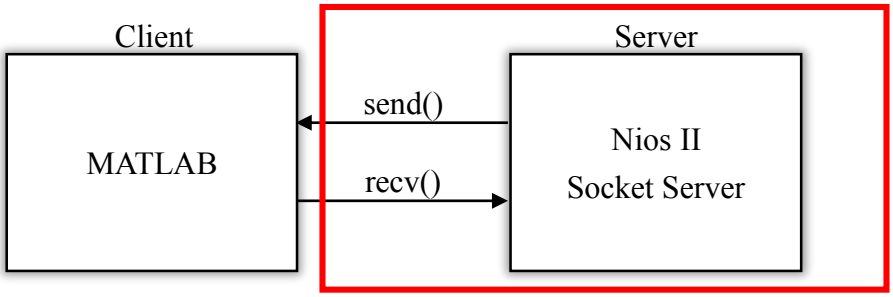


圖 3.33 Server 與 Client 之間的流程

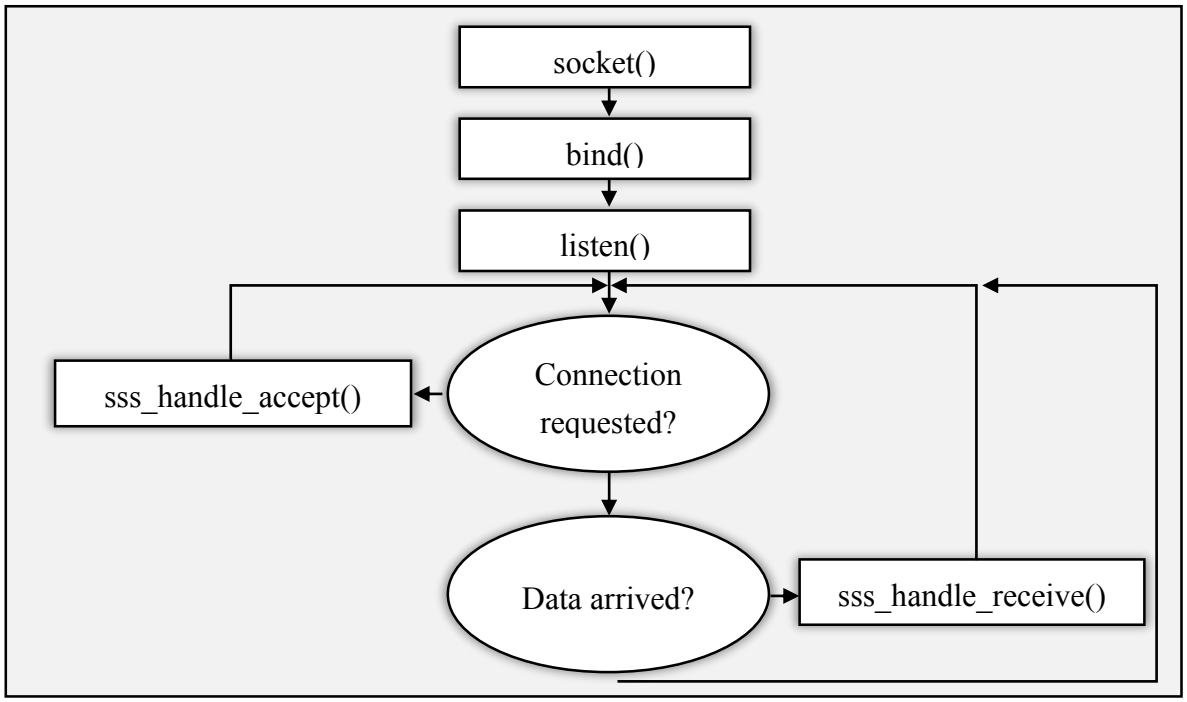


圖 3.34 Socket 建立連線至接收資料的流程

而整個 Socket 流程建立後，必須做的事情有以下幾件，將所接收的資料放入 On-Chip RAM 中，使用 DMA Controller 以遞增位址將訓練向量傳送客製化邏輯電路 GHA 的單一位址中進行運算。最後將其所訓練出的兩個權重向量傳回至 MATLAB 中，待全部傳完後將關閉 Socket 連線，而建立連線後整個設計的運作流程可以參考如下圖 3.35。

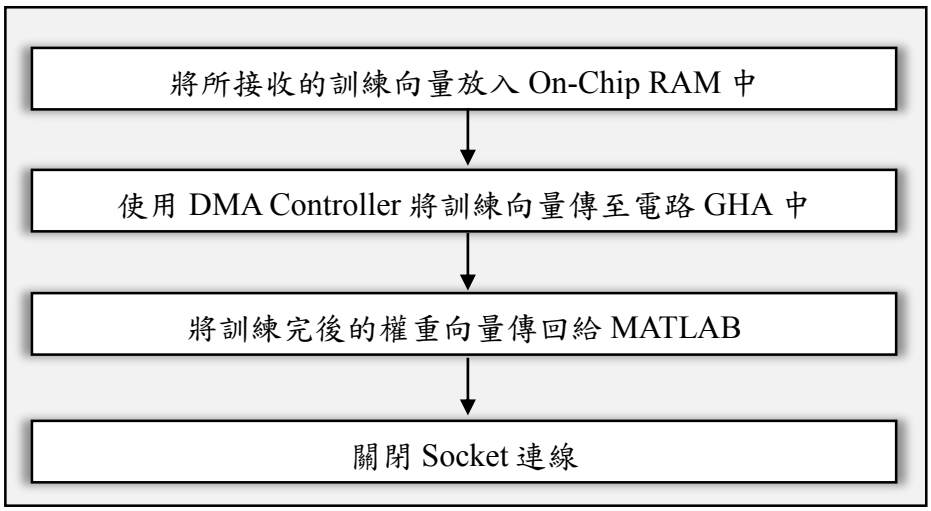


圖 3.35 建立連線後此設計的運作流程圖

接下來該討論如何設定將所接收的訓練向量放入 On-Chip RAM 中，在 Socket Server 這邊設定紀錄傳送和接收資料的位址，可以參考下圖 3.36，將紀錄接收位址的變數 rx_rd_pos、與紀錄傳送位址的變數 rx_wr_pos 分別指向到 rx_buffer，即是使用系統所提供的記憶體。

simple_socket_server.c

```
void sss_handle_receive(SSSConn* conn)
{
  int data_used = 0, rx_code = 0;
  INT8U *lf_addr;

  conn->rx_rd_pos = conn->rx_buffer;
  conn->rx_wr_pos = conn->rx_buffer;

  printf("[sss_handle_receive] processing RX data\n");

  while(conn->state != CLOSE)
  {
    /* Find the Carriage return which marks the end of the header */
    lf_addr = strchr(conn->rx_buffer, '\n');
```

圖 3.36 設定紀錄傳送和接收資料的位址

再來尋找系統所提供 On-Chip RAM 的起始位址，打開 CycloneIV_Socket_bsp 專案資料夾，在 system.h 檔裡定義了許多參數。

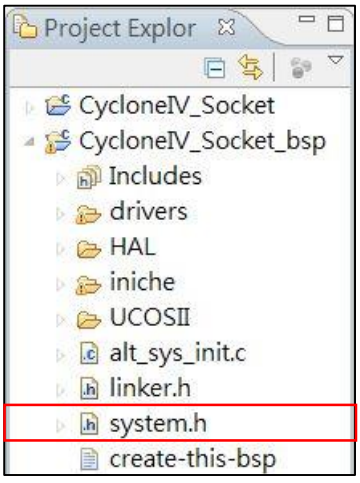


圖 3.37 在 bsp 相關資料夾內尋找 system.h 參數檔

這邊尋找 On-Chip RAM 參數設定，可以得知在這定義了 ONCHIP_RAM_BASE 起始垃址為 0x0。

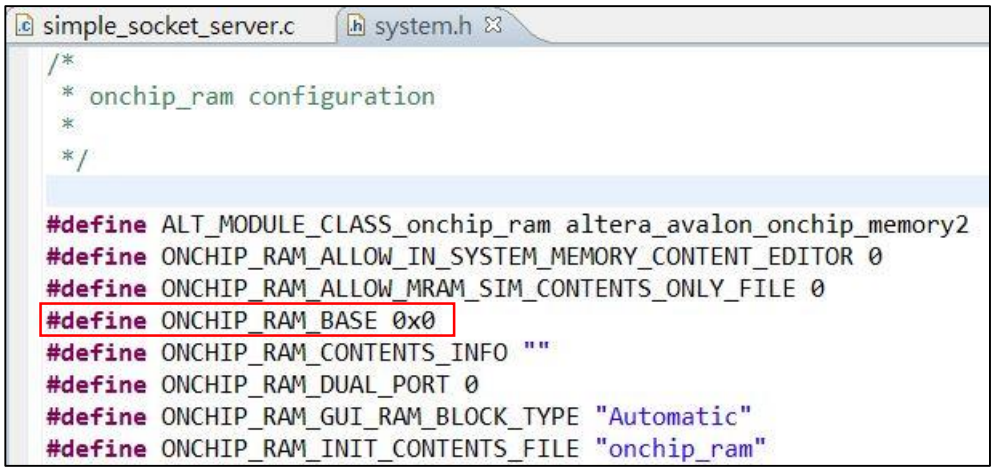


圖 3.38 取得定義的 ONCHIP_RAM_BASE 起始位址

因此把紀錄接收位址的變數 rx_rd_pos、與紀錄傳送位址的變數 rx_wr_pos 分別指向到 ONCHIP_RAM_BASE，再透過 recv 函數將接收的資料存放在 rx_wr_pos 中，即存放在 On-Chip RAM 中。

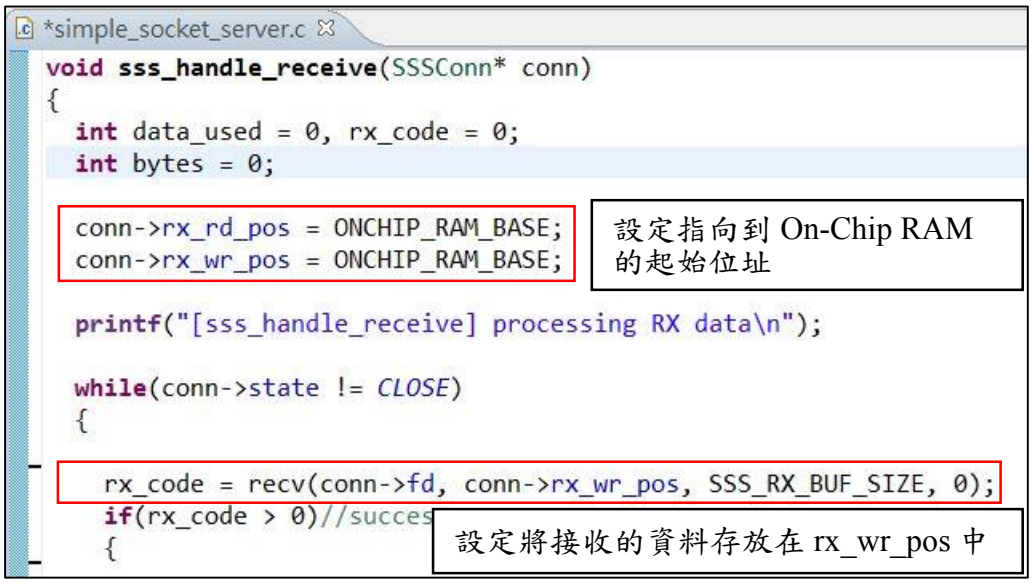


圖 3.39 使用 recv 函數將接收的資料存放在 rx_wr_pos 中

而當網路所接收的訓練向量透過 SG-DMA Controller 放入置 On-Chip RAM 後，接下來將討論如何再將 On-Chip RAM 中的訓練向量傳送至 GHA 客製化電路中進行運算，在這使用 DMA Controller，而以下圖 3.40 為 C 語言所使用函式之實例。得知所要使用的 DMA Controller 名稱為 dma_0，因此使用在 dev 目錄下的 dma_0 來開啟 DMA，並測試其是否已準備好，若未準備好則印出錯誤訊息；若已準備好，則將 On-Chip RAM 中的資料透過專用通道傳送至 GHA 電路中，並且等待 DMA Controller 回傳資料傳送完畢之訊號。

```
//Test the DMA*************************************************************
  printf("Test the DMA\n");

  alt_dma_txchan tx;
  tx=alt_dma_txchan_open("/dev/dma_0");
  if(tx==NULL)
  {
    printf("\n DMA test: DMA ERROR");
    exit(1);
  }
//Test the DMA*************************************************************

//using DMA to deliver x from onchip to gha circuit & Calculate exec. time********

  void* tx_data1=training_vector;
  rc=0;
  //tx為傳送通道、ALT_DMA_TX_ONLY_ON模式、(void*)(AVALON_GHA_FCM_0_BASE+131)為目標位址
  alt_dma_txchan_ioctl(tx,ALT_DMA_TX_ONLY_ON,(void*)(AVALON_GHA_FCM_0_BASE+131));
  alt_dma_txchan_ioctl(tx,ALT_DMA_SET_MODE_32,NULL);
```

DMA Controller 若未準備好則印出錯誤訊息

在這設定其電路目標址址

圖 3.40 設定 DMA Controller 目標位址為 GHA 電路位址

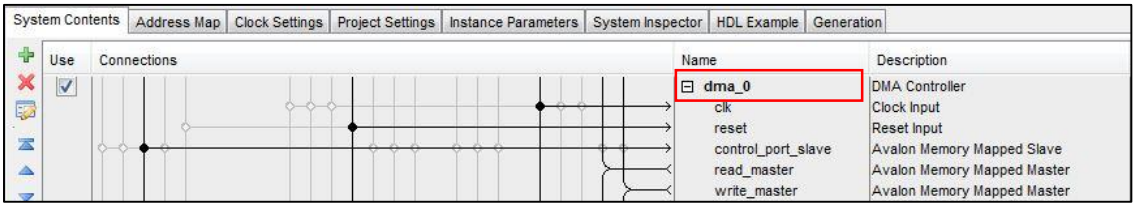


圖 3.41 透過 Qsys 工具所新增的 DMA Controller

3.4 軟體設計實現方法

本論文實現方法必須分為 MATLAB 軟體與 FPGA(Field-Programmable Gate Array)硬體兩個部分來設計，已在 NoC 平台中建立好跨平台棘波分類系統所需要的元件，而接下來必須處理 MATLAB 軟體設計的部分。而詳細流程與其之間的關系，可以參考下圖 3.42、圖 3.43，而在下段並會對此進一步說明。

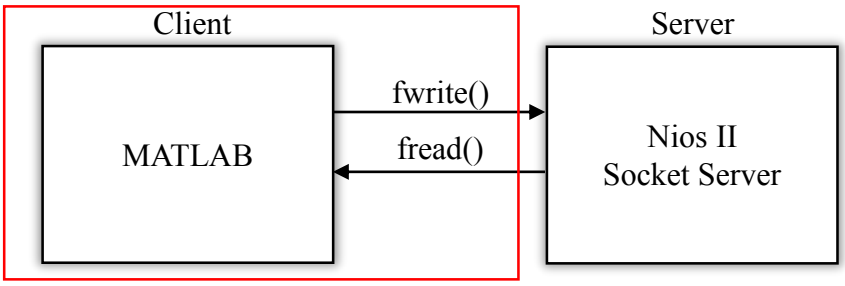


圖 3.42 Server 與 Client 之間的流程

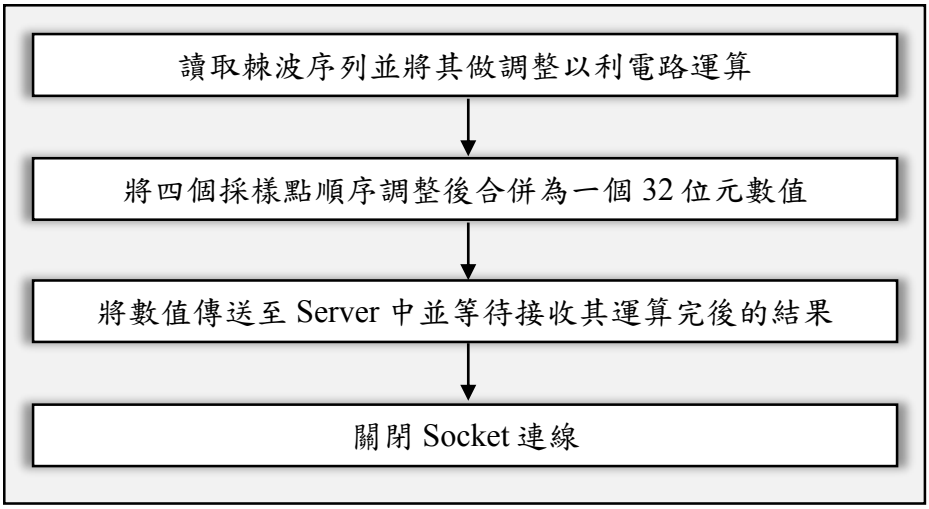


圖 3.43 建立連線後此設計的運作流程圖

3.4.1 資料格式處理

而在傳送過程前，必須先將 MATLAB 端所讀取的棘波序列進行處理，一個訓練向量(Training vector)的維度為 64 個採樣點，且每個採樣點經由調整後，其範圍在 0 至 255 之間，可以由下圖 3.44、圖 3.45 得知。

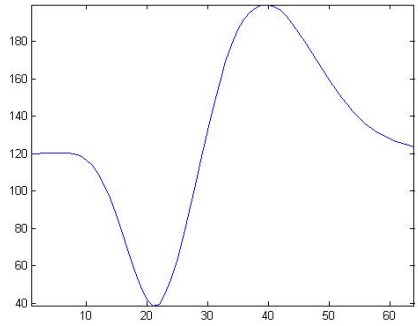


圖 3.44 經過調整垂直刻度後的棘波

Sample 1 = 120	Sample 2 = 121	Sample 2 = 123	……	Sample 64 = 126

圖 3.45 一個棘波總共有 64 個採樣點(Sample)

由於客製化邏輯電路將使用存放在 On-Chip RAM 裡的資料來做運算，在此系統中 On-Chip RAM 的寬度設定為 32 位元(Bit)，因此每個位址可以存放 4 個採樣點(Sample)。所以在 MATLAB 將值傳送前，必須先將每 4 個採樣點合併成一個 32 位元的數值後才能傳送。如下圖 3.46 所示：

Sample 4 (8bits)	Sample 3 (8bits)	Sample 2 (8bits)	Sample 1 (8bits)

圖 3.46 MATLAB 端將要傳送的值大小為 32 位元

由於在傳送過程中是以一個字節(Word)為單位，因此想要傳送的數值超過 8 位元將會產生大端模式(Big endian)的問題，由下圖 3.47 可以清楚的得知，MATLAB 端所傳送的值為 4 個 8 位元採樣點的合併，而以 C 語言控制 Nios 接收訓練向量後，將值放入 On-Chip RAM 後將會呈現如下圖 3.48。

Sample 4 (8bits)	Sample 3 (8bits)	Sample 2 (8bits)	Sample 1 (8bits)

圖 3.47 MATLAB 端將要傳送的值大小為 32 位元

Sample 1 (8bits)	Sample 2 (8bits)	Sample 3 (8bits)	Sample 4 (8bits)

圖 3.48 以 C 語言控制 Nios 所接收後放入 On-Chip RAM 的值

理想情況下高位元字節應同存放在 On-Chip RAM 的高位，而實際上並非如此，因此在 MATLAB 將訓練向量傳送前做了些調整，可以由下圖得知在這將原本訓練向量由存放在高位的字節的採樣點，改存放在低位字節的位置中，而低位字節的採樣點存入高位字節位置，經由此方法解決大端模式問題後，由圖 3.49、圖 3.50 比對將可以看出訓練向量如理想的存入 On-Chip RAM 中了。

Sample 1 (8bits)	Sample 2 (8bits)	Sample 3 (8bits)	Sample 4 (8bits)

圖 3.49 MATLAB 端將要傳送的值大小為 32 位元

Sample 4 (8bits)	Sample 3 (8bits)	Sample 2 (8bits)	Sample 1 (8bits)

圖 3.50 以 C 語言控制 Nios 所接收後放入 On-Chip RAM 的值

以上過程即可透過 MATLAB 將訓練向量成功傳送至 On-Chip RAM 中，等待客製化邏輯電路 GHA 運算完後，權重向量將會傳回至 MATLAB 中，在這使用 fread 函式來接收，透過 Binary 型態接收數據以達到高速傳輸的目的。而在這必須將所收到的權重向量來進行處理，本論文在 GHA 演算法中所使用的主成份個數為 2，因此將會有 w_1 與 w_2 兩個如下圖 3.51、圖 3.52 表示的 64 維度權重向量。

$w_{1(1)}$	$w_{1(2)}$	$w_{1(3)}$	……	$w_{1(64)}$

圖 3.51 GHA 客製化電路運算後將傳回的權重向量 w_1

$w_{2(1)}$	$w_{2(2)}$	$w_{2(3)}$	……	$w_{2(64)}$

圖 3.52 GHA 客製化電路運算後將傳回的權重向量 w_2

經由 w_1、w_2 權重向量與訓練向量做內積後將會得到特徵向量，由於本論文採用由英國學者 L. S. Smith 與 N. Mtetwa 所開發棘波訊號產生模擬器，即使用產生神經元訊號的模擬器作為測試資料來源[8]，該模擬器提供了大量的參數以用來模擬不同的棘波活動狀況，也提供了每個棘波的正確分群結果。因此本論文將使用到其正確分群結果與本系統所運算出的特徵向量做處理，得以驗證本系統的正確性。

3.4.2 建立連線與資料傳輸

首先，必須使用 MATLAB 建立 TCP/IP 連線，使用 instrfind 函式定義通訊連接口相關參數，即設定其通訊協定類型、IP 位址、以及連接埠，由圖，即可將此連接口當作文件般來進行讀寫，即達到傳送與接收目的，接著使用 tcpip 函式設定目標 IP 位址以及連接埠的參數即可開啟通訊連接口。

```
% Find a tcpip object.
obj1 = instrfind('Type', 'tcpip', 'RemoteHost', '140.122.184.1', 'RemotePort', 30, 'Tag', '');
% Create the tcpip object if it does not exist
% otherwise use the object that was found.
if isempty(obj1)
   obj1 = tcpip('140.122.184.1', 30);
else
   fclose(obj1);
   obj1 = obj1(1);
end
```

圖 3.53 定義通訊口連接參數

而在棘波傳送前，必須了解到從人類腦中所紀錄的棘波序列是非常龐大的，因此在本論文的系統中希望能將棘波序列快速的從 MATLAB 端傳送至 FPGA 開發板中，因此在這使用 fwrite 函式將棘波序列透過 Binary 型態來進行高速傳輸，而在這將介紹其詳細用法。需要傳入三個參數，第一個是指向目的地的指標，第二個是要傳入的資料，即為我們所要傳入的訓練向量，第三個是資料的型態，本論文使用一個為 32 位元的整數。而下圖 3.54 是我們所應用的實例。首先，將讀取的 Spike 進行處理，由於電路需使用正整數，且資料來源為經過濾波器之數值，因此

範圍為-127 至 127 間，因此在這將所讀取的採樣點加上 127 以調整為正整數並放入 SpikeData 變數中，接下來將每 4 個採樣點合併為一個 32 位元值放入 SpikeData32bit 變數中，再經由大端轉換後，最後即使用 fwrite 函式將數值傳送至 FPGA 上所開發的 Socket Server 中。

```
fid3 = fopen('rawSpikeData.txt', 'r');
SpikeData=fscanf(fid3,'%f');
SpikeData=round(SpikeData(:))+127;

j=1;
for i=1:4:Sample_Number;
   SpikeData32bit(j)=SpikeData(i)*1+SpikeData(i+1)*256+SpikeData(i+2)
   *65536+SpikeData(i+3)*16777216;
   SpikeData32bit(j)=z2dec( dec2bin(uint32(SpikeData32bit(j)),32));
   j=j+1;
end

for i=1:(Sample_Number/4), %12800
   pre_data=dec2bin(typecast(int32(SpikeData32bit(i)),'uint32'),32);
   aft_data(1:8)=pre_data(25:32);
   aft_data(9:16)=pre_data(17:24);
   aft_data(17:24)=pre_data(9:16);
   aft_data(25:32)=pre_data(1:8);
   %use the function to slove nagtive number
   TransferData(i)=bin2dec(aft_data);
end

for i=1:32:(Sample_Number/4),
   fwrite(obj1,TransferData(i:i+31),'uint32');
end
```

將採樣點調整為正整數

將四個採樣點合併

進行大端轉換處理

使用 fwrite 函式將值傳送至 Server 中

圖 3.54 使用 fwrite 函式之部分實例

當電路運算算完後將會回傳所訓練出的權重向量，在此使用 fread()函式來接收資料，其第一個參數為指向目的地的指標，而第二個參數則為所要接收的字數，2 個主成分即為 128 筆數字，而第三個參數則設定接收的資料型態，在這使用 UINT32 以利後續進行數值的轉換，經過大端轉換後將值分為 w1 與 w2 兩筆陣列。

```
ReceiveData=fread(obj1,128,'uint32')
```

使用 fread 函式接收電路計算完的權重向

```
for i=1:128
   pre_data=dec2bin(ReceiveData(i),32);
   aft_data(1:8)=pre_data(25:32);
   aft_data(9:16)=pre_data(17:24);
   aft_data(17:24)=pre_data(9:16);
   aft_data(25:32)=pre_data(1:8);
   %use the function to slove nagtive number
   w(i)=z2dec(aft_data);
end
```

進行大端轉換處理

```
w1=w(1:64)
w2=w(65:128)
```

將資料分別存入代表權重向量的 w1 與 w2 中

圖 3.55 使用 fread 函式之部分實例

第四章 實驗結果與數據討論

本章節主要為介紹本實驗所使用的環境以及實際數據的量測與比較

4.1 開發平台與實驗環境介紹

本論文所提出的跨平台棘波分類系統硬體部分主要是使用 Altera 公司所設計的 Cyclone IV GX EP4CGX150N FPGA 開發板，可以參考如下圖。其主要的特點為低功率消耗以及低開發成本，非常符合棘波分類的應用，同時 Cyclone IV GX FPGA 採用了 65 奈米製程，使得比以往 FPGA 功率消耗來低個約 30%，因此選用此開發板來進行本論文之研究。

圖 4.1 Cyclone IV GX EP4CGX150N FPGA 開發板

軟體實驗環境是使用 Intel® Core™ i7-2600 CPU @ 3.4GHz 處理器、16.0 GB 的記憶體的電腦，並且使用 MATLAB R2009b 版本來進行軟體開發，其提供了一個非常大優勢，即提供了強大的繪圖工具，且能夠快速的開發出一個系統，以及完整的資料格式支援。

表 4.1 為本實驗開發板所提供的相關硬體資源，由於提供了大量的硬體資源與相關元件豐富的溝通介面，使得能透過不同平台所設計的演算法與開發板中的演算法進行整合，成為一個完整的跨平台嵌入式系統。

Feature	**Cyclone IV**
Device	GX EP4CGX150N
Process	65nm
LEs	149760
Embedded Memory(Kbits)	6480
18-bit × 18-bit Multipliers	360
Clocks	50MHz

表 4.1 Altera Cyclone IV GX EP4CGX150N FPGA 開發板

本論文是使用 Altera Quartus II 12.1 版本搭配 Verilog 硬體描述語言所設計的電路來開發，由工具 Qsys 在 NoC 平台上建立客製化系統，於平台中加入 CPU、DMA Controller、On-Chip RAM、Triple-Speed Ethernet MAC Controller 等相關元件，並加入了 GHA 客製化電路。同時並透過 Quartus II 所提供的語法檢查、邏輯元件配置等多種功能，將可快速的建出此系統，接著使用 Altera 公司提供的以 Eclipse 為基礎的 NIOS II 軟體做為開發，以 C 語言撰寫使用其函式庫以及驅動程式來與 FPGA 開發板溝通，而下圖 4.2 為整個 Qsys 系統開發流程。

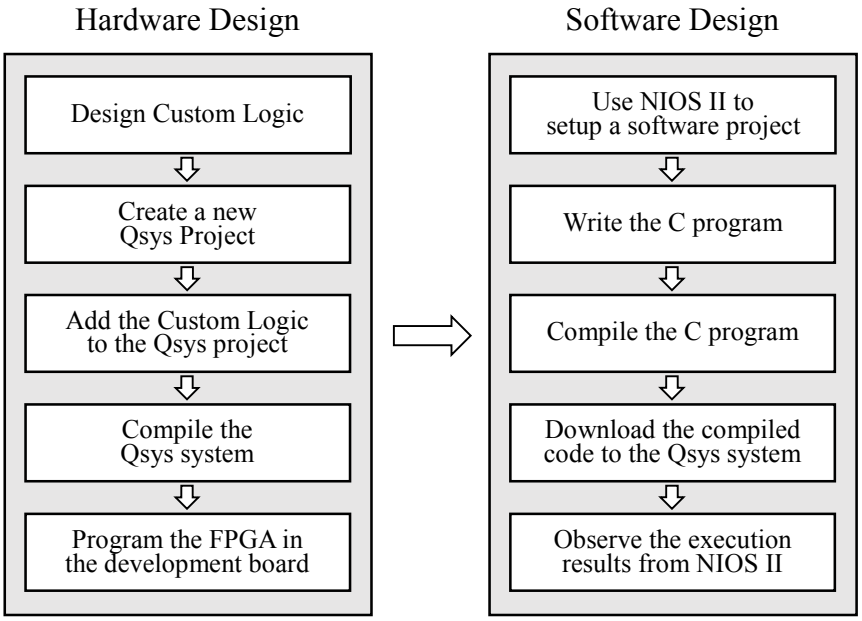


圖 4.2 Qsys 系統開發流程

另外也會以 MATLAB 軟體來實現 GHA 演算法，並運作於 Intel® Core™ i7-2600 CPU @ 3.4GHz 處理器當中，以相同的演算法則來與本論文所設計系統來做比較，實驗環境將會影響整個系統的運作效能與其開發設計，尤其在數據呈現方面，實驗環境將會影響到系統執行速度，因此在下表 4.2 中提供了硬體與軟體的實驗環境。

	硬體實驗環境	**軟體實驗環境**
Device	Altera Cyclone IV GX DK-DEV-4CGX150N	Winsows 7 作業系統
CPU	NIOS II 1GHz	Intel® Core™ i7-2600 CPU @ 3.4GHz
Memory	128MB DDR2 SDRAMs	DDR III 16GB

表 4.2 本系統的實驗環境說明

4.2 實驗數據比較

為了測量本研究所提出的跨平台棘波棘波分類系統，本論文採用由英國學者 L. S. Smith 與 N. Mtetwa 所開發棘波訊號產生模擬器，使用產生神經元訊號的模擬器作為測試資料來源，在當中設定了許多環境變數，其中使用了採樣率(Sampling Rate)設定為 24000 採樣點/秒，每個棘波訊號長度為 2.67 毫秒，因此也代表著每個棘波訊號擁有 64 個採樣點(Sample)。而在此將傳送的棘波序列資料量做個整理，總共為 800 個棘波(Spike)、即 51200 個採樣點(Sample)、即 800 個訓練向量(Training vector)。本實驗將對軟體所執行的棘波分類系統進行數據量測，由於 GHA 在運算過程中所使用的訓練次數多寡，將影響了特徵擷取以及在後續做分群的準確性，因此在這將對不同的訓練次數執行的時間進行量測，而量測的結果如下表 4.3，其量測訓練次數為 100 至 1000 次，測量時間單位為秒，而從圖 4.3 中可以更清楚的看到，當訓練次數愈多次時，其整個棘波分類所花費的時間是呈現明顯線性成長的，因此想要做的愈準確，所付出的時間代價是相當可觀的。

訓練次數	**100**	**200**	**300**	**400**	**500**	**600**	**700**	**800**	**900**	**1000**
軟體	2.61	4.20	6.16	8.07	10.07	11.80	13.87	15.69	17.80	19.72

表 4.3 軟體進行特徵擷取所花費的時間

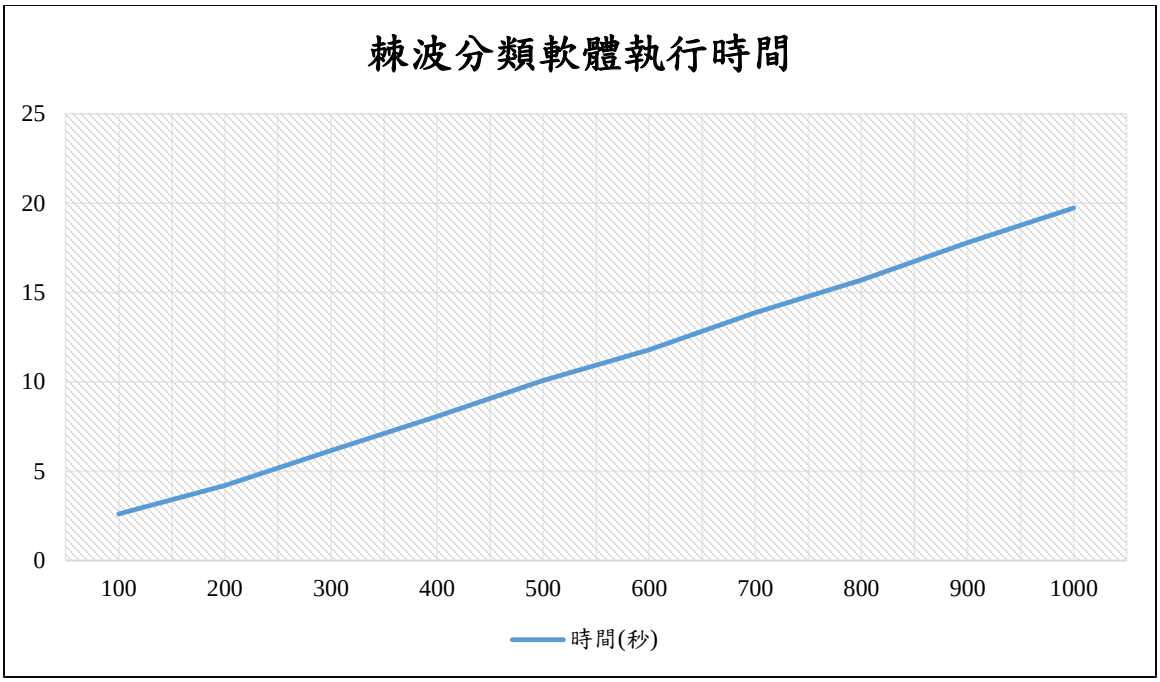


圖 4.3 棘波分類軟體所執行的時間

以下是本論文所提出的系統，經由實驗後所得出的數據結果，實驗所使用的資料來源與軟體一致，皆是使用英國學者 L. S. Smith 與 N. Mtetwa 所開發棘波訊號產生模擬器所產生的棘波序列，實驗數據結果主要分為三個部分：透過網路傳送棘波序列所花費的時間、硬體上執行特徵擷取演算法所花費的時間、將所訓練出的權重向量回傳的時間，下表 4.4 是所測量的結果，以及圖 4.4 呈現了本論文所提出系統之優點，傳輸時間約為 0.45 秒，運算的時間也小於軟體執行時間。

訓練次數	100	200	300	400	500	600	700	800	900	1000
傳送時間	0.46	0.45	0.44	0.45	0.46	0.45	0.45	0.44	0.44	0.44
運算時間	0.94	1.90	2.84	3.79	4.75	5.68	6.60	7.57	8.49	9.45
整體時間	1.40	2.35	3.29	4.24	5.21	6.13	7.05	8.02	8.94	9.89

表 4.4 本論文提出之系統經實驗所量測的時間(秒)

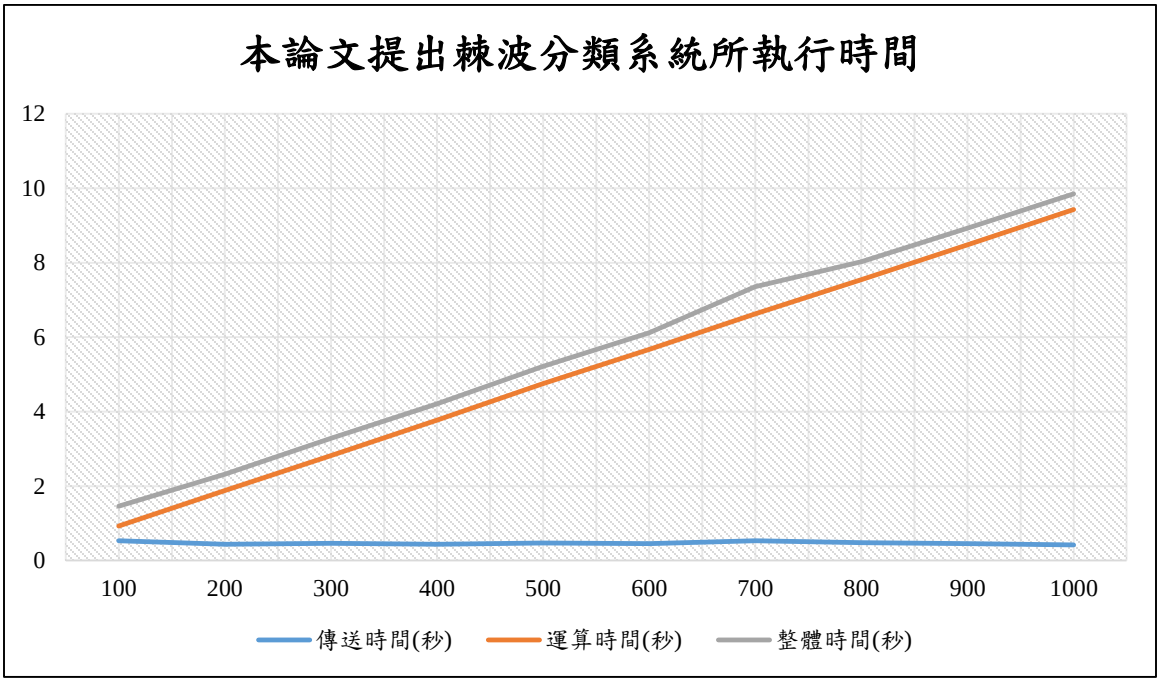


圖 4.4 本論文提出棘波分類系統所執行時間

在此對原本軟體與本系統所執行花費時間做比較，可以從表 4.5 與圖 4.5 中清楚的了解到本論文所提出的跨平台棘波分類系統，其效能遠大於原本軟體所花費的時間，相差的時間幾乎隨著訓練次數而呈線性成長，由原本訓練次數 100 時，兩者相差時間為 1.21 秒，其影響較為小，但在訓練 1000 次時，其所相差了 9.83 秒，遠遠的優於軟體所執行的時間。

訓練次數	100	200	300	400	500	600	700	800	900	1000
軟體	2.61	4.20	6.16	8.07	10.07	11.80	13.87	15.69	17.80	19.72
硬體	1.40	2.35	3.29	4.24	5.21	6.13	7.05	8.02	8.94	9.89
相差	1.21	1.85	2.87	3.83	4.86	5.67	6.82	7.67	8.86	9.83

表 4.5 軟體與本論文所提出系統之執行時間比較(秒)

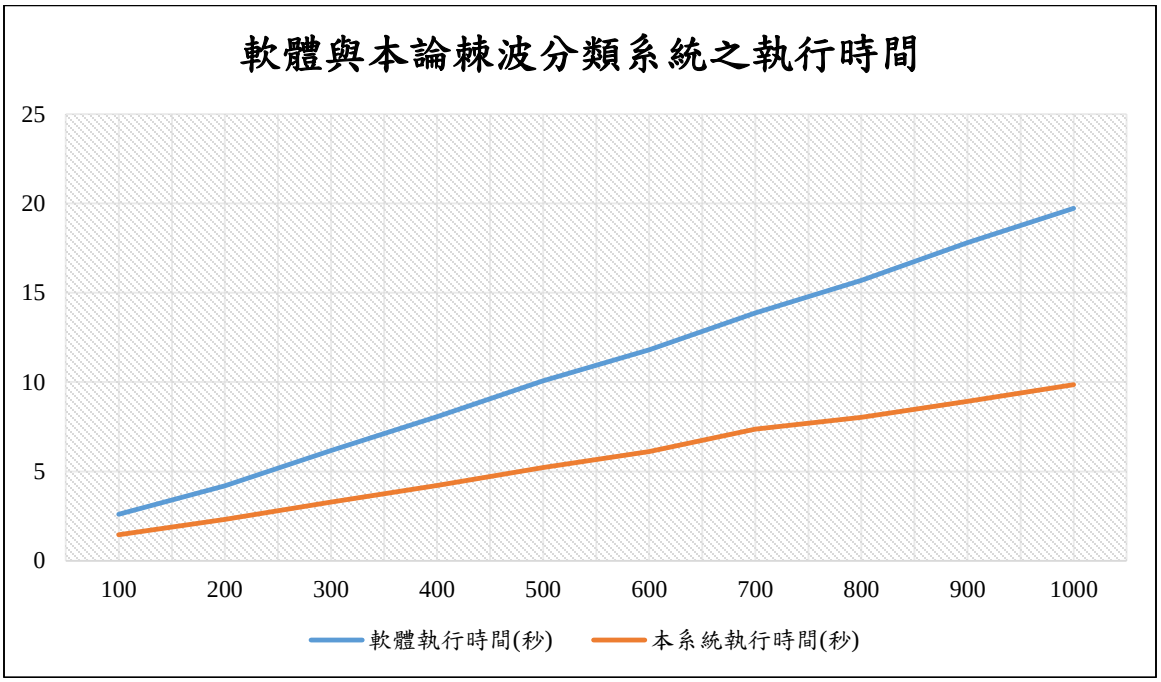


圖 4.5 軟體與本論文所提出系統之執行時間呈線性成長

第五章 結論

近年來有關大腦活動情形的研究越來越熱門，目前也提出了許多有關腦波訊號處理的相關研究。本論文於不同平台上實現了一套跨平台的棘波分類系統，透過區域網路將不同平台所讀取的棘波序列傳送至 FPGA 開發板中進行運算，以節省棘波分類軟體所運行的時間。

尤其在現今棘波分類應用非常的廣泛，可以提供醫學或是相關產業進行應用，提供了重症病患透過腦波控制機械義肢的可能性，經過本論文的研究後，將有利於未來人們能夠透過攜帶式小型裝置，將所紀錄的腦波透過網路傳送至具有高計算速度的棘波分類系統中進行運算，再將其結果加以利用。相信不管是在醫學或是相關娛樂產業中，都能夠有效的利用。

本論文所提出之架構與現有軟體做比較，從實驗數據結果可以得知其效能遠優於一般軟體的棘波分類系統，使得本系統架構在棘波分類應用上更具有優勢，因此，可以說本論文所設計的跨平台棘波分類系統是一項有實際需求且有效率的電路架構設計。

參考文獻

[1] Y. Sun, S. Huang, J. J. Oresko, and A. C. Cheng, “Programmable Neural Processing on a Smartdust for Brain-Computer Interfaces,” IEEE Trans. Biomedical Circuits and Systems, Vol. 4, pp.265-273, 2010.

[2] L. S. Smith and N. Mtetwa, “A tool for synthesizing spike trains with realistic interference,” Vol. 159, pp.170-180, Journal of Neuroscience Methods, 2007.

[3] M.A. Lebedev and M.A.L. Nicolelis, “Brainmachine interfaces: past, present and future,”Trends in Neurosciences, Vol.29, pp.536-546, 2006.

[4] E.E. Fetz, “Real-time control of a robotic arm by neuronal ensembles,”Natural Neural Science, Vol. 2, 00.583-584, 1999.

[5] S. Hauck and A. Dehon, Reconfigurable Computing:The Theory and Practice of FPGA-Based Computing, Morgan Kaufmann: San Fransisco, CA, USA, 2008.

[6] S. Haykin, Neural Networks and Learning Machines, 3rd ed.; Pearson: Upper Saddle River, NJ, USA, 2009.

[7] T.D. Sanger, “Optimal unsupervised learning in a single-layer linear feedforward neural network,” Neural Network, Vol. 12, pp.459-473, 1989.

[8] L. S. Smith and N. Mtetwa, “A tool for synthesizing spike trains with realistic interference,” Vol. 159, pp.170-180, Journal of Neuroscience Methods, 2007.

[9] 柯奇恩, “多通道棘波分類系統之低功率ASIC電路設計” 國立臺灣師範大學資訊工程研究所,2014.

[10] R. Quian Quiroga, Z. Nadasdy and Y. Ben-Shaul, “Unsupervised Spike Detection and Sorting with Wavelets and” Neural Computation 16, 1661-1687; 2004.