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

以多核心系統架構為基礎
在可程式化系統晶片中實現島嶼式基因演算法

Multicore-based Implementation of Island Genetic Algorithm on SOPC System

研究生：　洪　鵬　傑　撰

中華民國　九十八　年　七　月

致 謝

首先感謝研究所這兩年來對我教誨不倦的老師　黃文吉教授，無論在研究上對學生的指導，以及人生處事上的訓誡，老師認真負責的態度給予學生一種莫大的正面影響亦給予學生一個正確的方向，在此致上最深的謝意。同時感謝口試委員們能夠百忙之中撥冗參加學生的口試審查，並給予學生中肯的建議。

再來是感謝一起生活兩年的同窗摯友們，嘉隆、正存、嘉儀、家璿、聖凱，在生活上以及課業上對我的幫助很大，以及可愛的學弟妹們，家祥、敦皓、政諺、嘉晏、治廣、禕璨、宗毅，在分擔一些事務上對我的幫助很大。

最後要感謝父母的默默支持，以及哥哥姊姊的教誨，我才能有前進不懈的動力，亦使我的食宿無虞讓我可以專心研究，在此特別感謝。

中文摘要

本研究為向量量化器的設計提出一個島嶼式(分散式)基因演算法的架構。研究中以多核心的系統架構為基礎，而為了獨立的基因演化過程，每一個島嶼分別都包含著一個硬體加速器以及一個 softcore 處理器。而島嶼之間的相互基因移民是採用一個共享的 on-chip RAM，而這個 on-chip RAM 被一個硬體 mutex 給控制著以避免發生資料存取誤用。這樣為硬體實現分散式基因演算法提供了一個簡單以及具有彈性的移民機制。實驗數據顯示了我們所提出的硬體架構相對於其對應的軟體模擬系統擁有較低的計算時間。

關鍵詞：分散式基因法則、系統晶片設計、多核心系統、向量量化器

ABSTRACT

This thesis presents a novel distributed genetic algorithm (GA) architecture for the design of vector quantizers. The design is based on a multi-core architecture, where each island of the GA is associated with a hardware accelerator and a softcore processor for independent genetic evolutions. An on-chip RAM with a mutex circuit is adopted for the migration of genetic strings among different islands. This allows a simple and flexible migration for the implementation of hardware distributed GA. Experimental results shows that the proposed architecture has significantly lower computational time as compared with its software counterparts for GA-based optimization.

Keywords：Distributed GA、SOPC、Multi-core System、Vector Quantizers

目 錄

圖表目錄

第一章 緒論

本章節主要探討本論文的研究背景、動機目的與方法，並大略說明各章節的主要內容及重要特性。

1.1 研究背景

基因演算法(Genetic Algorithms，GAs)[1,2,3]是一種經由模擬自然界中生物族群的自然天擇(Natural Selection)以及遺傳演化的一般用途搜尋演算法，也是最佳化工程計算中常用的一種演算工具，於 1975 年由密西根大學 John Holland 教授及他的學生首先提出之演算法則。這種演算法已經被發現可以用來有效地解決工程上、科學上以及商業上的問題。然而，當這樣的演算法使用在複雜的問題上，它的計算複雜度將變得更高。

基因演算法在初始包含了一組基因序列群，經由犧牲其他已經存在的舊有基因序列後，再生(Regeneration)得到較適合生存的基因序列，除此之外還利用交配(Crossover)、突變(Mutation)的手段來得到更好的基因序列。突變的方式是改變基因序列中的個別元素；而交配方式則是在兩個不同的基因序列之間找尋交配點做切割之後再進行接合。將上述的再生、交配、突變的過程做結合，在此稱之為一個世代

衍生(Generation)。這些基因序列的演化也許會經過好幾個世代交替，而最終得到一個最佳的基因序列也就是接近全域最佳解(Near Global Optimal Solution)。因為基因演算法具備這樣找尋最佳解的特性，近年來已經普遍地被應用於許多找尋最佳解的問題，例如：類神經網路(Neural Network)、螞蟻演算法(Ant Algorithm)、影像處理(Image Processing)、模糊邏輯(Fuzzy Logic)、VLSI 繞線佈局(Place & Route)、灰色理論(Grey Theory)等。

1.2 研究動機及目的

由於基因演算法常常應用於許多複雜的最佳化搜尋問題上，所以最讓人詬病的缺點便是它的計算複雜度太高。如果使用軟體來實現基因演算法，則可以預見的結果是整體的運算效能將會有較差的表現，而且一旦基因演算法實驗的族群數量變多時，對於某些基因演算法的應用，這些運算時間將會變得更高。然而，如果我們降低了實驗的族群數量卻又是非常不合適的，因為小的族群數量將使得基因演算法落入較差的局部最佳解(Local Optimal)。

為了解決這個問題，其中一種降低計算時間的方法就是採用島嶼式基因演算法(Island GAs)亦可稱為分散式基因演算法(Distributed GAs)。在這樣的島嶼式基因演算法中有多個族群，每個族群大部分時間是各自獨立演化，不同的族群之間偶爾會互

相交換基因序列。由於基因演算法在族群大小為較小的情況下可能會比較快收斂，但為了找到近似最佳解，較大的族群仍然是必須的，這仍會造成過多計算時間的問題，因此我們採用島嶼式基因演算法將大的族群分化成許多小的族群獨立演化；此外，若以軟體實現這樣計算複雜度高的演算法必然需要較長的執行時間，因此本論文以硬體實現之。

基因演算法第二個缺點就是再生程序主要是基於適應函數上。因此，從族群中選擇出母代序列於硬體的實現便需要大量的晶片面積(Chip Area)，而且如果想要同時平行處理當下的基因序列在突變(Mutation)和交配(Crossover)的程序，會導致大量的面積成本的消耗。

本論文主要是為島嶼式基因演算法提出一個 VLSI 的架構，此架構可以加快基因演算法的計算時間即使是較大的族群。本論文考慮於向量量化器的應用(Vector Quantizer , VQ) [4]，其中的原因在於，當基因演算法應用於向量量化器中碼字(Codeword)的訓練時，基因演算法依舊需要大量的儲存空間和漫長的訓練時間[5,6]。因此向量量化器的設計可以為驗證基因演算法架構之有效性作為一個好的範例。

1.3 研究方法

在島嶼式基因演算法的硬體架構下，為了島嶼之間各自獨立的基因演化，我們為每一個的島嶼配置了一個硬體加速器和一個處理器。另外，為了基因序列在不同的島嶼之間相互移民，我們採用一個共享的 On-Chip 記憶體來存放即將移民的基因序列，以及一個 Mutex 電路來確保此 On-Chip 記憶體同一時間內只有一島嶼正在使用。這樣的方法為島嶼式基因演算法的硬體實現上提供了一個較簡單以及較彈性的移民機制。

雖然現有的某些硬體架構在各個島嶼的基因演化可以被拿來使用在硬體加速器上，這些架構都有著下列這些缺點。首先，要儲存基因序列的時候需要大量的儲存空間；對於再生這一個程序中，通常是使用兩個族群集合的記憶體量。其中一個記憶體儲存母代的序列，另一個記憶體則是儲存再生後的子代序列。除此之外，發生於新的世代衍生開端，基因序列要從當下儲存資料的記憶體位置中挑選出初始族群到所使用的記憶體位置時，會多出額外的時間耗費。

本論文所提出的硬體加速器可以消弭上述這些缺點。此加速器採用 steady-state GA[7]使得再生、突變和交配程序更為簡單。此加速器其中包含族群記憶體單元(Population Memory Unit)、交配突變單元(Crossover & Mutation Unit)、適應值計算單元(Fitness Evaluation Unit)、生存測試更新單元(Survival Test & Update Unit)。為了降

低面積成本(Area Cost)，我們只採用一個族群記憶體。而交配和突變這兩個程序是可以同時處理的。此外，適應值計算單元是一個管線化(Pipeline)的架構，搭配使用直接記憶體存取(Direct Memory Access, DMA)使得電路的效能大幅提升。生存測試單元則是由一個硬體排序電路組成。

本論文提出的硬體架構實現於 FPGA(Field Programmable Gate Array)[8]，這樣的方式使得本架構中的處理器可以搭配採用軟核處理器(softcore CPUs)[9]。因為使用可重複編譯的硬體裝置，我們便能夠在可程式化系統晶片(SOPC)為基因向量量化器建立一個完整的系統。我們拿本硬體系統架構與它相仿的軟體系統作比較，實驗數據顯示出我們所提的硬體基因向量量化器架構有著較高的效能以及大幅降低向量量化器的碼字訓練時間。經由這些事實便能夠說明我們架構設計的有效性。

1.4 全文架構

本篇論文共分為五個章節，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、動機及目的、方法及本文架構。

【第二章】基礎理論及技術背景介紹

簡介基因演算法及本論文使用的理論基礎以及技術背景。

【第三章】島嶼式基因演算法之硬體系統架構實現

介紹所提出的島嶼式基因演算法硬體系統架構，其中也會做相關的討論與說明。

【第四章】實驗數據與效能比較

包含了系統環境的說明、相關實驗數據分析以及軟硬體效能比較。

【第五章】結論及未來展望

對於我們實驗的結果做一個總結，及未來發展方向。

第二章 基礎理論及技術背景介紹

本章節主要目的是介紹 Steady-State GA 應用在向量量化器的基本原理，並且討論了基因演算法的名詞定義、基本運作流程，包含了再生、交配以及突變的步驟。之後會對 Island GA 做一個介紹，緊接著簡介 FPGA 技術，方便讀者對本論文能夠有初步的了解認識。

2.1 Steady-State GA 應用於向量量化器之基本原理

向量量化器在資料分群(Data Clustering)中的主要目的是為了分割一個大量的資料集合 $X=\{x_1,\ldots,x_t\}$ 到 N 個不重疊的群 $C_1,\ldots,C_N$ 之中，其中 $N << t$；而分割(Partitioning)這一個步驟主要是根據一群碼字(Codewords)所構成的集合 $\{y_1,\ldots,y_N\}$ 而定，在這裡，這些碼字的維度(Dimension)和集合 X 中向量的維度都是 w。

給予某個向量 $x \in X$，當滿足式子(1)時，此向量 x 將會被指配到某個群 C_i 中。

$$i = \alpha(x) = \arg\min_{1 \le j \le N} d(x, y_j) \tag{1}$$

其中，$d(u,v)$表示兩個向量 u 和 v 的距離測量(Distance Measure)。在本論文中，

平方距離(Squared Distance)特別適合拿來做為距離測量。當應用於資料簡化(Data Reduction)的技術，例如資料壓縮時，在滿足$i=\alpha(x)$時，資料集合中的向量 x 將會被碼字 y_i 取代掉，進而達到壓縮的功能。

其中一個表示資料簡化的成本函數(Cost Function)是利用 y_i 取代 x 時，所得到的像素平均失真值(Average Distortion)。公式表式如下：

$$D=\frac{1}{wt}\sum_{i=1}^{t}d(x_i,y_{\alpha(x_i)}) \tag{2}$$

給予一個資料集合 X，向量量化器設計的目的就是為了找出在方程式(2)中，一群碼字集合$\{y_1,\ldots,y_N\}$最小化的 D。Steady-State GA 應用於向量量化器的設計時，會有 P 個基因序列來進行基因演化。每一個基因序列 r 表示的是一群碼字所構成的集合 $N=\{y_1,\ldots,y_N\}_r$。特別要注意的是，這些基因序列是向量所構成的字串，而不是二位元數字(Binary Number)所組成的字串。

2.2 基因演算法基本名詞及運算程序

2.2.1 前言

在基因演算法中，令 $S(k)$和 $D(k)$分別表示，經過 k 次基因演化後所得到的 P 個基因序列集合、以及目前最小化失真值 D。而且，令 S^*表示經過一連串基因演化過程後所得到目前最佳的基因序列。在初始的步驟中令 $D(0)=\infty$，還有令 S^*為空字串 (Null)，除此之外，我們也可以隨機地從訓練向量中選擇某些向量來當 $S(0)$中的碼字。

2.2.2 基因演算法基本名詞定義

1. 遺傳因子(Gene)：

一般所使用的基因演算法，採用的遺傳因子有兩種，第一種是採用二進制也就是 0 和 1 所組成，另一種則是採用實數做為因子，本論文的基因演算法是後者，因為應用於向量量化器，所以遺傳因子可能的值為黑白像素值從 0 到 255 之間所構成的 w 維的向量，相當於向量量化器中的碼字。

2. 染色體(Chromosome)：

由多個遺傳因子所組成的字串，便是所謂的染色體，對於不同應用要解決的

問題，有不同的編碼方式去代表染色體，於本論文中相當於向量量化器中的碼簿。

3. 族群(Population)：

由多個染色體聚集在一起的一個群，便稱之為族群，其所代表的意義即是針對問題的解找出一個最佳的群，在基因演算法中無法保證每個染色體的解是最好的，我們所找到的解只能說是接近於最佳解的近似值。

4. 世代(Generations)：

完成一次的基因演算法，稱之為一個世代，而世代次數代表著搜尋的次數，世代的次數的多寡對於解的好壞品質會有影響。

2.2.3 基因演算法運算程序

基因演算法能夠選擇物種中具有較佳適應性的母代，並且能夠隨機地交換彼此的基因資訊，再加上隨機性的突變偶爾能夠帶來更好的基因，所以能夠產生較上一母代更適合生存的子代，按照這樣的程序重複下去以期能有最佳的基因出現。一般來說，基因演算法有下列程序：初始族群產生(Population Initialization)、適應值計算(Fitness Calculation)、再生(Regeneration)、交配(Crossover)和突變(Mutation)，後面三者步驟為基因演算法主要的程序，以下再詳細的介紹。

假設第 k-1 次的基因演化已經完成，而且第 k 次($k \geqq 1$)的執行也已經準備開始。接下來執行再生、交配、突變的基因演化步驟，並且可以從 $S(k\text{-}1)$中的基因序列集合得到局部最佳化值(Local Refinement)。

再生(Regeneration)：因為在 $S(k\text{-}1)$中經過基因演化步驟的基因序列，事實上就是向量量化器中的碼簿(Codebook)，所以我們可以使用公式(2)來計算出每一個基因序列所對應的失真值 D。我們利用 D 的反函數來當作每一個基因字串的適應函數。接下來便可以利用輪盤法(Roulette-Wheel Technique)來實現再生的步驟。也就是說，後代的再生是利用旋轉一個模擬的輪盤來產生，輪盤的上面會有個別的基因序列所對應不同的機率值；在選擇的過程是採用隨機的方式去選取，所以所佔的盤面比例愈大者，被選到的機率也就愈大。例如圖 2-1 所示：

個體	適應值	佔有比率%
1	53.5	21.40%
2	12.4	4.96%
3	28.7	11.48%
4	114.3	45.72%
5	41.1	16.44%
總合	250	100%

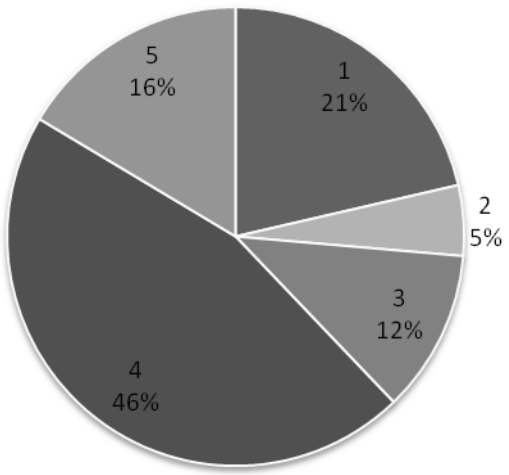

圖 2-1. 再生機制-輪盤法

一次再生，只會有一個基因序列被選擇，而被選擇到的基因序列將會被複製出一份一模一樣的基因序列，這個新產生出來的基因序列將會進行交配和突變的程序。基因演算法中，經過再生的步驟，會有 P 個新的再生基因序列被創造出來。

交配(Crossover)：在每一個再生的基因序列 r，$\{y_1,\ldots,y_N\}_r$ 上，利用一個機率值 P_c 判斷是否作單點交配。然後，隨機地從全部的基因族群中選擇出一個母代基因序列 r'，$\{z_1,\ldots,z_N\}_{r'}$。接下來產生一個在 1 到 N 之間的亂數 n；這兩個序列同時在位置 n 的地方被切成兩段，之後，將新產生的字串$\{y_{n+1},\ldots,y_N\}$和字串$\{z_{n+1},\ldots,z_N\}$彼此互換位置擺放，於是完成交配程序，如圖 2-2 所示。

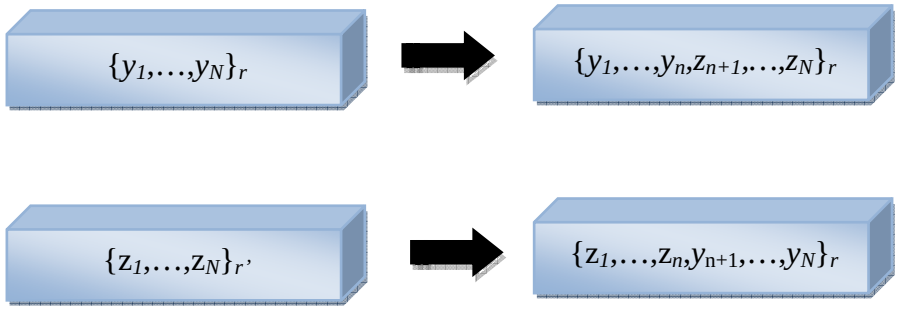

圖 2-2. 單點交配

突變(Mutation)：在交配程序中，是針對基因序列動作，也就是針對碼簿動作；而在突變步驟裡，卻是根據一個機率值 P_m，來針對每一個基因序列中的碼字做動作。假設目前有一個即將進行突變步驟的基因序列 $r=\{y_1,\ldots,y_N\}_r$，接下來會隨機地從 N 個碼字中選擇一個 y 出來，而且在碼字 y 中的 w 個元素(Element)，也會被隨機地選擇出來；接著，產生一個亂數，這個亂數即將被應用於剛被選出的某數上，加上這個亂數或減掉這個亂數，也就是+b 或-b 來達到突變的目的，如圖 2-3 所示。

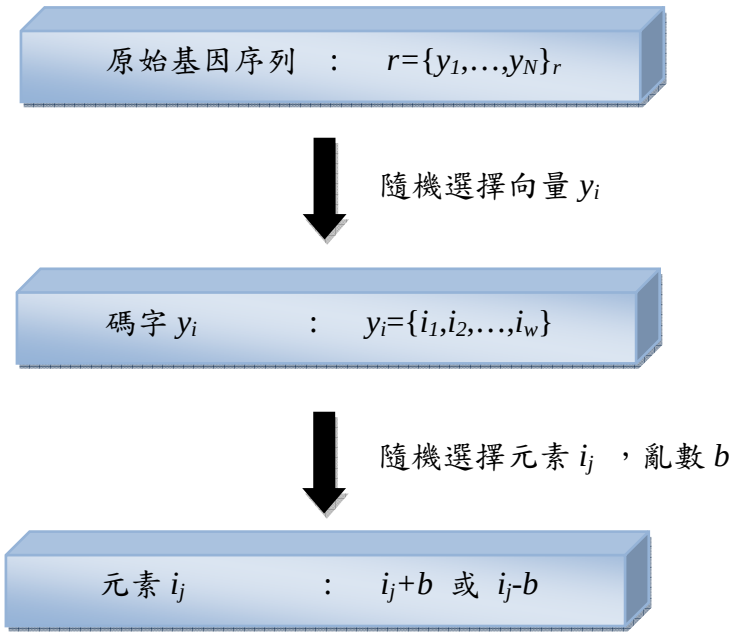

圖 2-3. 突變運算

2.2.4 基因演算法收斂條件

在基因演算法中，完成上述三個過程：再生、交配、突變，稱為一個世代(Generation)；世代會不斷的演化直到 $D(k)$數列收斂達成。我們會持續觀察，假設已經滿足連續的 L 次世代演化所產生的個別 $D(k)$皆相同，換言之，就是達成以下條件：$D(k)= D(k\text{-}1)=\ldots= D(k\text{-}L+1)$，則基因演算法將會停止，經過基因演算法後所到的目前最佳的序列 S^*，即是最佳碼簿。

2.2.5 基因演算法流程

圖 2-4 表示的是應用於向量量化器的 Steady-State GA 流程圖(本論文採用的 GA 為 Steady-State)，在 Steady-State GAs 並不會有世代的觀念，取而代之的是親子代共存的現象。在此我們將族群 S 當成 P 個基因序列的集合，亦稱為母代序列。而族群 S 中 P 個基因序列初始為隨機產生，接著會挑選族群 S 中兩個基因序列(在此表示為 r_1,r_2)來做交配及突變以得到新的子代基因序列(在此表示為 c)。而子代基因序列的適應值經過計算之後會拿來與族群 S 中所有母代基因序列的適應值做比較，如果此新的子代基因序列比所有母代基因序列還要差，那麼所有母代基因序列將不會被取代，反而言之，子代基因序列將會取代最差適應值的母代基因序列。

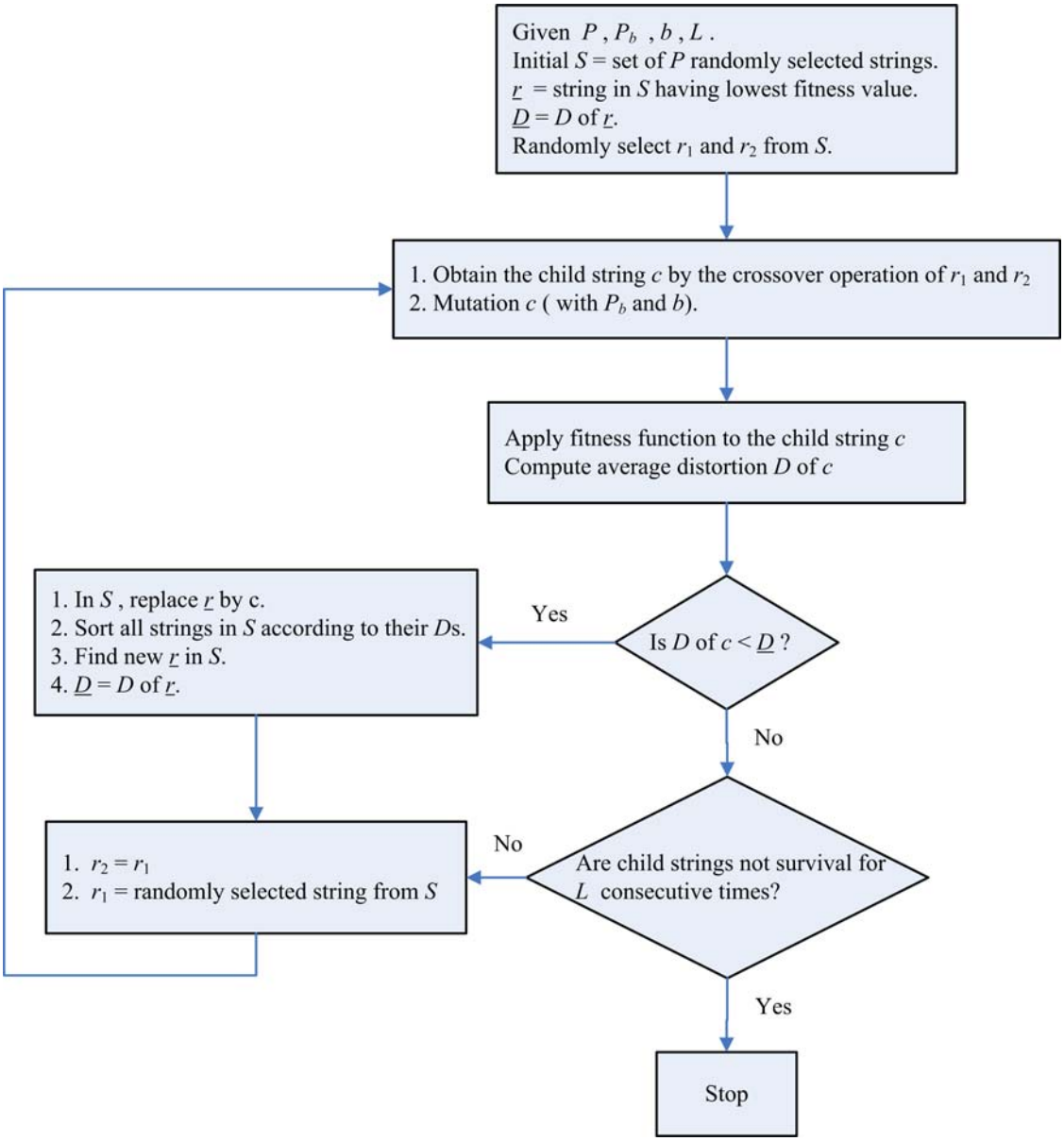

圖 2-4. Steady-State GA 流程圖

當選擇(Selection)、交配(Crossover)、突變(Mutation)、生存(Survival)和取代(Replacement)等步驟不斷地持續時，族群內每一個基因序列的適應性將會一直增加，而且愈晚產生的新子代基因序列其存活率(Survival Rate)將會愈來愈低，一直到某個時刻，新子代存活率將降低到零，此時，演化將會停止，而 Steady-State GA 演算法也會終止。

在基因向量量化器的設計裡，每一個基因序列即代表一個碼簿，對於基因序列的檢索(String Retrieval)所要花費的記憶體存取時間將會非常漫長，同理地，對於 r_1 和 r_2 的檢索程序也會造成某種程度的時間耗損(Time-consuming)。為了降低記憶體存取時間，我們設計了一個替代方案，將先前挑選出的 r_1 變成新的 r_2，然後再從族群 S 中隨機選出新的 r_1，因此相較於原本的方法約可以節省一半的記憶體存取時間。因此在我們的設計中只需要使用到一塊族群記憶體，除此之外，交配和突變程序只針對 r_1 和 r_2 序列作用，而非對族群記憶體上的所有基因序列。再者，應用於再生程序中的輪盤法也會成為硬體實現的瓶頸，我們簡化了再生程序，以致能更簡易實現於硬體中，所以在硬體電路設計上是比較有效益的。

2.3 島嶼式基因演算法

在島嶼式基因演算法中，我們假設有 M 個島嶼，每一個島嶼 i 將有各自的族群 S_i，而每個在族群 S_i 中的基因序列將根據 Steady-State GA 演算法各自獨立演化，直到新子代的存活率降為零為止。緊接著，島嶼 i 將從島嶼 j ($j \neq i$)移入 K 個基因序列，同時亦移出 K 個基因序列到島嶼 k，在此 j 與 k 為隨機選擇。在完成移民程序之後，新的演化程序將重新開始啟動。而島嶼式基因演算法一次的迴圈就包含了一次的移民程序以及一次 Steady-State GA 的演化，在每一次迴圈結束之後，有著最佳適應性的基因序列將會被紀錄下來，當連續 L 次此最佳的基因序

列皆未被改變時，那麼此島嶼的演化將會終止，而當所有島嶼都停止演化時，整個島嶼式基因演算法就完成。整個島嶼式演算法流程圖 2-5 如下所示。

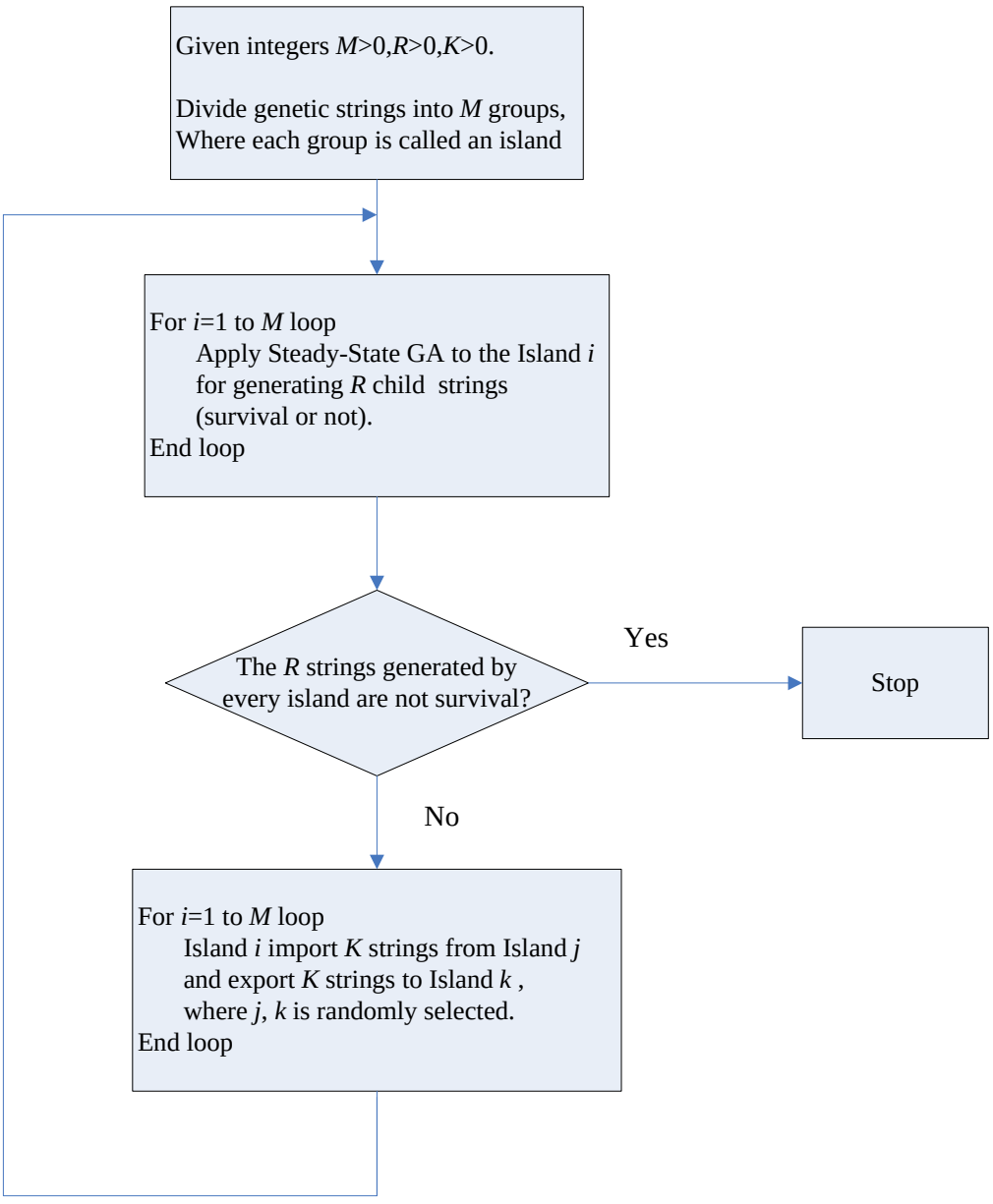

圖 2-5. Island GA 流程圖

2.4 FPGA 系統設計

最近幾年來，積體電路(IC)產品的發展是愈來愈快，然而卻造成其生命週期不斷日益下降，而且所要求的功能複雜度也提高許多，於是工業界為了在如此競爭的壓力下生存，勢必要採取更有效率的設計方法；為了因應這樣的困境，採用硬體描述語言(Hardware description language, HDL) 設計數位電路可縮短研發時間，而且更有彈性，經過簡單的合成繞線佈局，可快速重覆地燒錄至 FPGA 上進行測試，是現代 IC 設計的技術主流。目前 FPGA 系統的開發，允許軟體與硬體共同設計，來達成系統化的 IC 設計，具有開發時間短及系統可修改的優點。

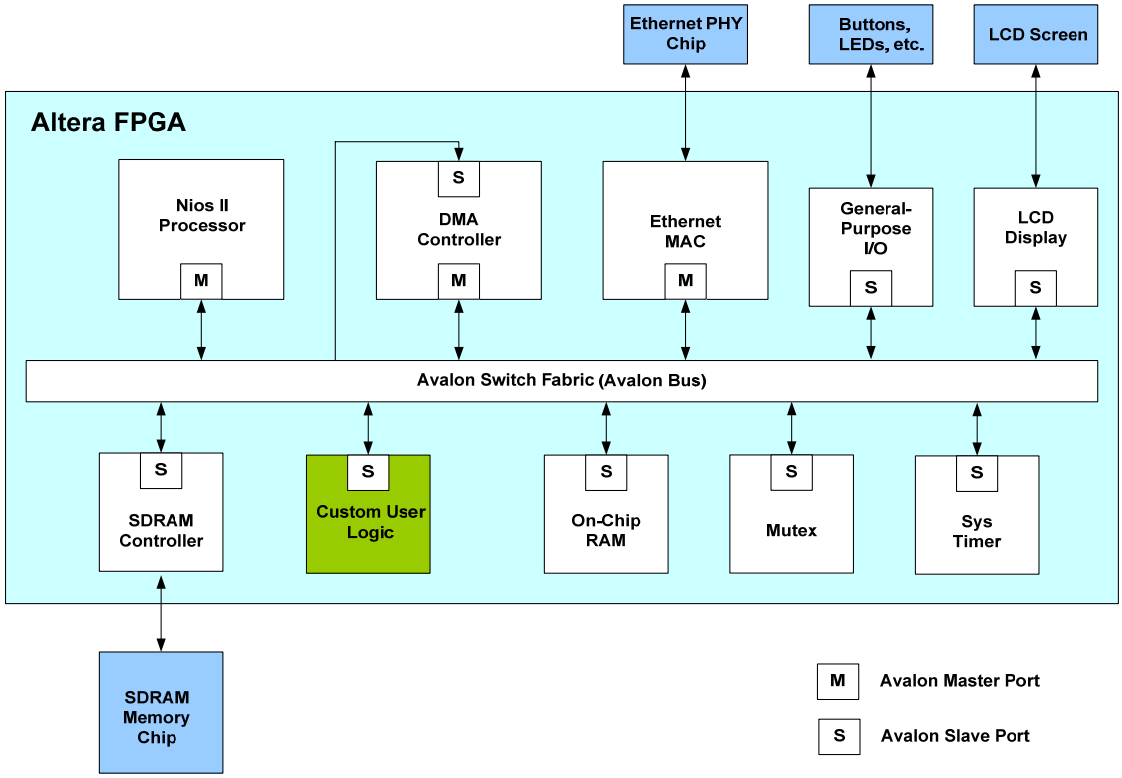

圖 2-6. FPGA 設計硬體架構

Altera 公司根據不同使用者的需求，開發出許多不同系列的 FPGA 開發板，本研究是在 NIOS development kit 中的 Stratix EP2S60 系統開發板上來實現我們島嶼式基因演算法。在 NIOS 系統中提供了一套專門給 NIOS 處理器使用的匯流排(Bus)，稱作 Avalon 匯流排(Avalon Bus)，而使用者設計出的電路(在本研究為 Steady-State GA 電路)稱之為使用者自訂邏輯(Custom User Logic)，如圖 2-6 所示。透過此匯流排上的各個訊號線，電路可以和整個系統溝通。整個系統的設計流程，下圖 2-7 所示。

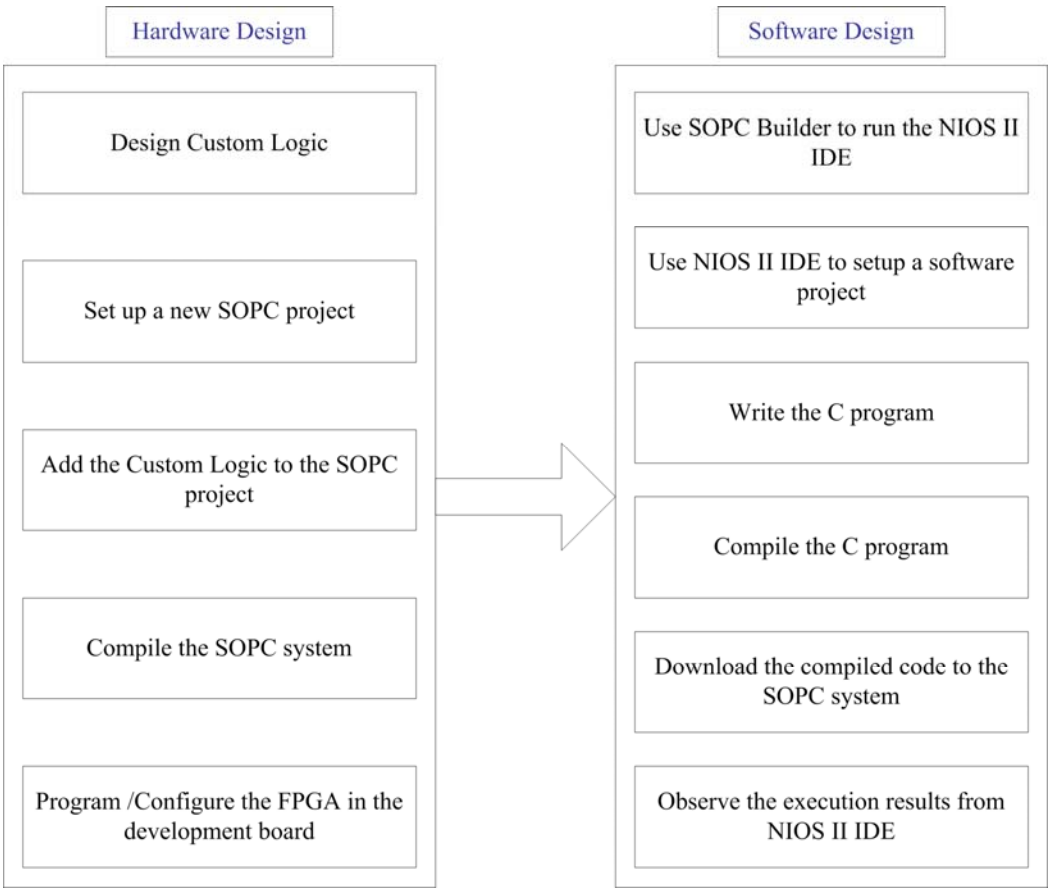

圖 2-7. FPGA 設計流程圖

NIOS 系統擁有下列優點：

1. Altera 公司所提供的 NIOSⅡ IDE 是一個視窗介面的開發工具，設計者可於其上編輯程式碼，更可編譯及除錯及觀察程式執行結果。此程式除了包含所有 C 語言的函式庫之外，還有 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 API(Application Program Interface)，如圖 2-8 所示，設計者可用 C 語言撰寫基本程式並透過 HAL API 呼叫來讓特定的裝置運作。

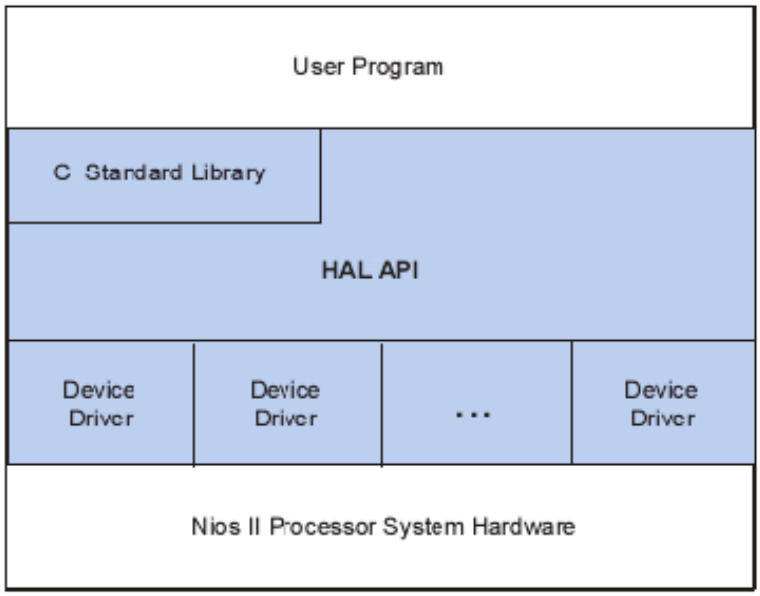

圖 2-8. HAL API

2. 提供了 Compact Flash 卡(以下簡稱 CF 卡)存取功能，讓設計者可以將資料置於 CF 卡中，並透過 HAL API 讓設計者進行讀與寫的動作。

3. 提供 DMA(direct memory access)機制讓設計者可加速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。
4. lwIP 提供基本的網路功能，除了符合嵌入式系統最基本的簡單、快速、不佔據過多系統資源之外，也讓設計者更容易修改或增刪想要的功能。
5. 設計者可依自己的需求增加或刪除想要的電路功能，需要增刪裝置時非常簡便，重新建置系統並編譯成硬體檔之後燒至板子即可。

第三章 系統架構

在本章節中，我們將會討論如何從基本 Steady-State GA 的電路架構，然後擴展到一個島嶼模組的系統架構，再來討論到整個多核心島嶼式基因演算法的系統架構，希望透過本章節的介紹，讓讀者能對本論文所提的架構更加瞭解。

3.1 Steady-State GA 之硬體電路架構

首先我們可以看的 Steady-State GA 的整體架構如圖 3-1 所示，其中包含了族群記憶體單元(Population Memory Unit)、交配突變單元(Crossover & Mutation

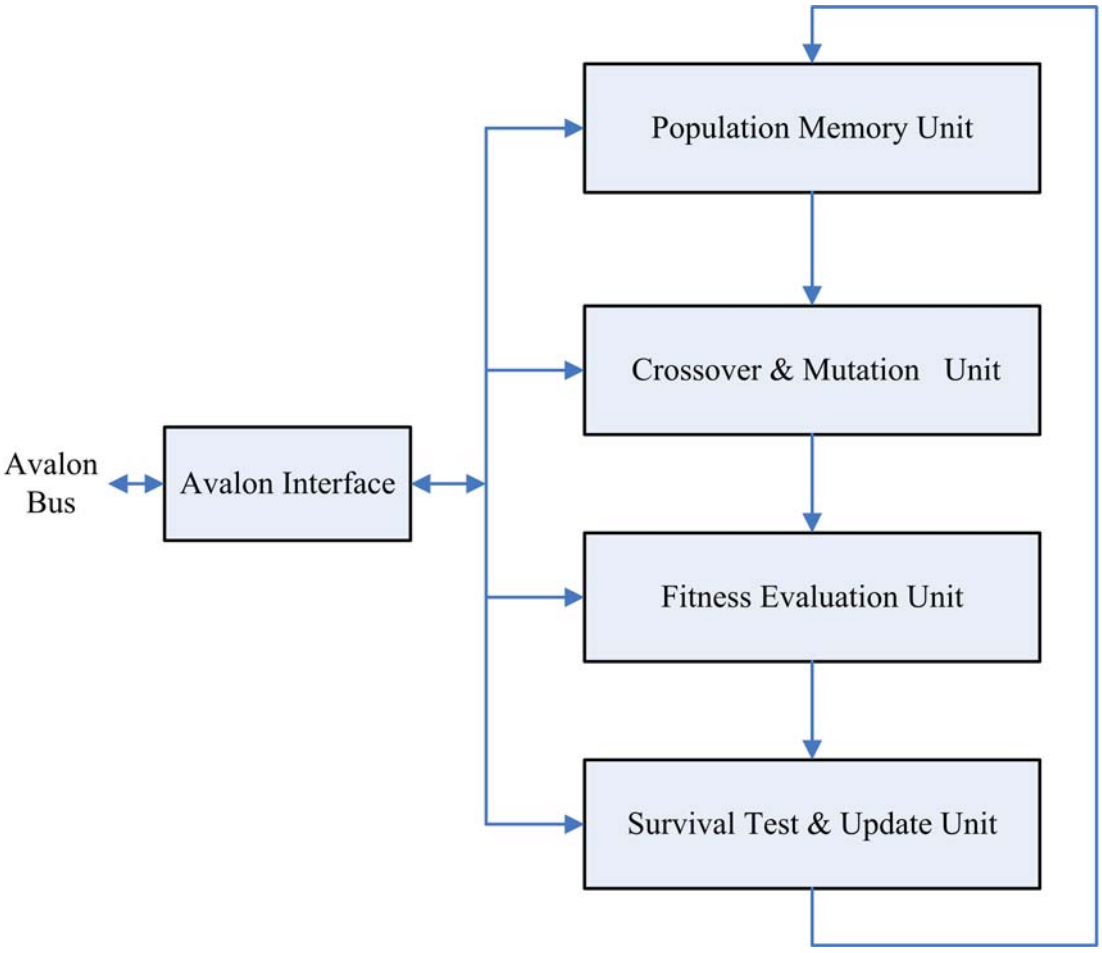

圖 3-1. Steady-State GA 之硬體電路架構

Unit)、適應值計算單元(Fitness Evaluation Unit)、生存測試更新單元(Survival Test & Update Unit)以及 Avalon 介面單元(Avalon Interface Unit)。而在族群記憶體單元和交配突變單元中會包含亂數產生器(Random Number Generator , RNG)。

在本架構中，族群記憶體單元主要是用來儲存基因序列，這些基因序列將會被隨機選擇出來，作為交配突變單元使用；這裡的隨機選擇主要是根據族群記憶體單元中的亂數產生器(RNG)產生的亂數來決定，除此之外，交配突變單元的功能是為了產生一個新的子代基因序列 c，而且在本單元中所有的交配和突變的程序都是同時處理的，接著這個新產生的子代基因序列便送到適應值計算單元來計算出其適應值。得到子代的適應值後，生存測試更新單元的目的便是決定這個新生子代是否能存活，如果能存活，則在族群記憶體中有著最差適應值的母代基因序列將會被這個新生的子代基因序列取代。在本架構中的 Avalon 介面單元主要是用來做為本電路和 NIOS-based SOPC Avalon Bus 的溝通介面，負責傳送資料和訊號。在圖 3-1 中的每個電路單元將會在接下來的四小節中詳細討論。

3.1.1 族群記憶體單元(Population Memory Unit)

圖 3-2 表示的是族群記憶體單元的基本結構，包含了一個族群記憶體和亂數產生器單元，族群記憶體是一個 2-port 的 RAM，主要用來儲存、檢索和更新族群 S 內這 P 個基因序列，在我們的設計中，這個 RAM 的實現是採用 FPGA 硬體

實驗板內建所提供的內嵌式記憶體。而亂數產生器則是用來隨機選擇出母代序列 r_1 做為交配突變單元使用，對於 RNG 的設計是採用 Cellular Automata (CA)[10]，因為 CA 在設計上擁有簡易性和規律性，所以非常適合用來實現亂數產生器於 VLSI 電路設計上。

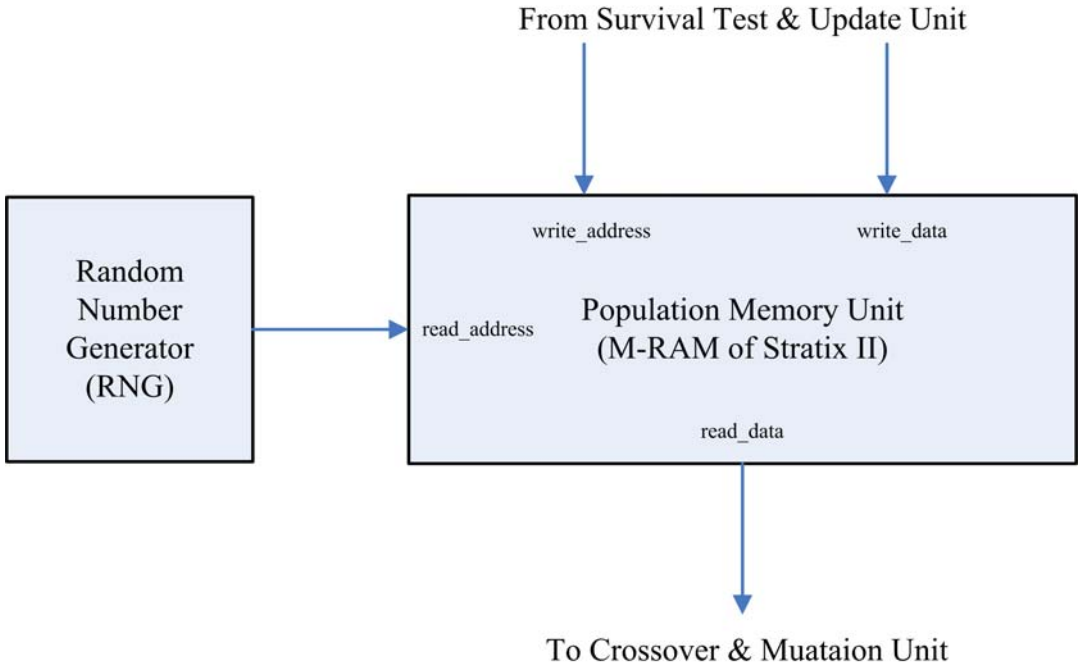

圖 3-2. 族群記憶體單元之硬體架構

3.1.2 交配突變單元(Crossover & Mutation Unit)

圖 3-3 為交配突變單元的基本結構，其中包含了三個位移暫存器(Shift Register)，分別儲存母代基因序列 r_1、母代基因序列 r_2 和子代基因序列 c，以及一些亂數產生器、比較器、多工器以及計數器來實現交配突變單元。此架構的最主要優點就是交配和突變程序可以同時執行，並且擁有低 Area Cost 的優勢。

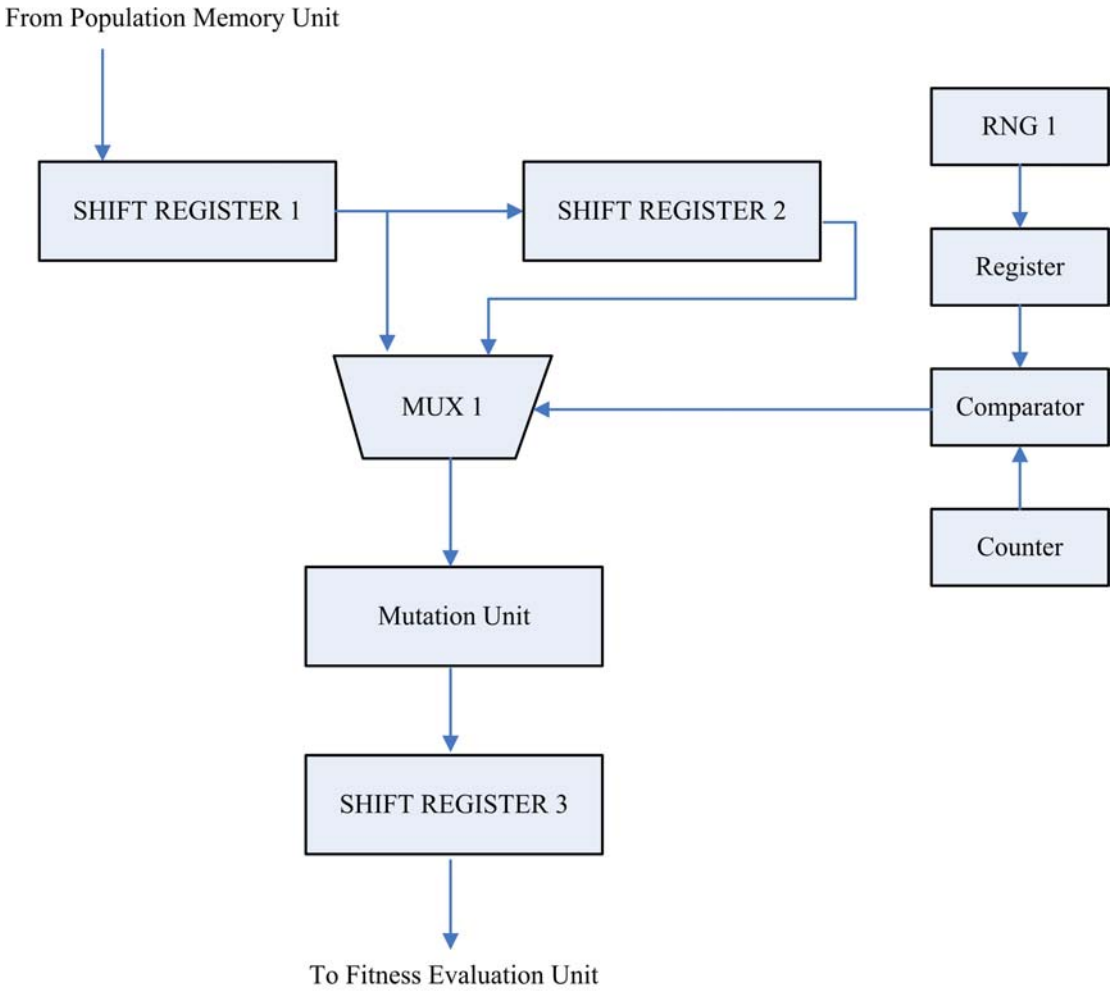

圖 3-3. 交配突變單元之硬體架構

如圖 3-3 所示，SHIFT REGISTER 1 和 SHIFT REGISTER 2 分別儲存了基因序列 r_1 和基因序列 r_2，回顧前面所提到的，在我們的設計裡並不是從族群記憶體中隨機地選擇新的 r_1 和 r_2，而是從族群記憶體中只選出一個基因序列，也就是新的 r_1，而新的 r_2，則是先前一回所挑選過的 r_1，經由這樣的改良設計，可以大大的降低記憶體存取時間和 VLSI 繞線佈局的負擔。所以，根據這樣的設計，SHIFT REGISTER 1 會儲存從族群記憶體來的基因序列 r_1，而 SHIFT REGISTER 2 則是

儲存從 SHIFT REGISTER 1 來的基因序列 r_2。

同時位移在 SHIFT REGISTER 1 和 SHIFT REGISTER 2 的基因序列到 MUX1 時，即可實現交配的程序，而且每一個位移暫存器同一個時間只會位移一個碼字，如圖 3-3 所示，經由 MUX1 來選擇從 r_1 或 r_2 的碼字進而達到交配的作用，緊接著將通過 MUX1 選擇而得的碼字送到突變單元(Mutation Unit)，經過突變單元作用後傳送到 SHIFT REGISTER 3，此位移暫存器所儲存的即是新的子代序列 c。

在圖 3-3 中的比較器(Comparator)，其作用是拿來比較亂數產生器(RNG1)產生的亂數和計數器(Counter)的值，而比較器的結果，會被連接到 MUX1 的控制線上，用來判斷選擇 r_1 或 r_2 的碼字；而計數器記錄著位移暫存器位移的數目，此外，由亂數產生器產生的亂數則是作為一個門檻值，當計數器的值比這個門檻值還要小的情況，MUX1 會選擇在 SHIFT REGISTER 1 的碼字，也就是 r_1 的碼字，再送往 SHIFT REGISTER 3，反之，則 MUX1 會選擇 r_2 的碼字。因此，我們可以將亂數產生器產生的亂數當做“交配點”(Crossover Point)，而這個交配點的值必定要比位移動作更早一步隨機產生。

如圖 3-3 所示，從 MUX1 所得到的碼字要輸出到 SHIFT REGISTER 3 儲存時，會先經過突變單元作用，而圖 3-4 即是突變單元硬體電路架構，在圖 3-4 中，所有輸出碼字中的 w 個元素將會同時地突變；為了每個元素所設計的突變單元包

含兩個亂數產生器(稱之為 RNG *ia* 和 RNG *ib*)、一個比較器(稱之為 comparator *i*)以及一個多工器(稱之為 mux *i*)。

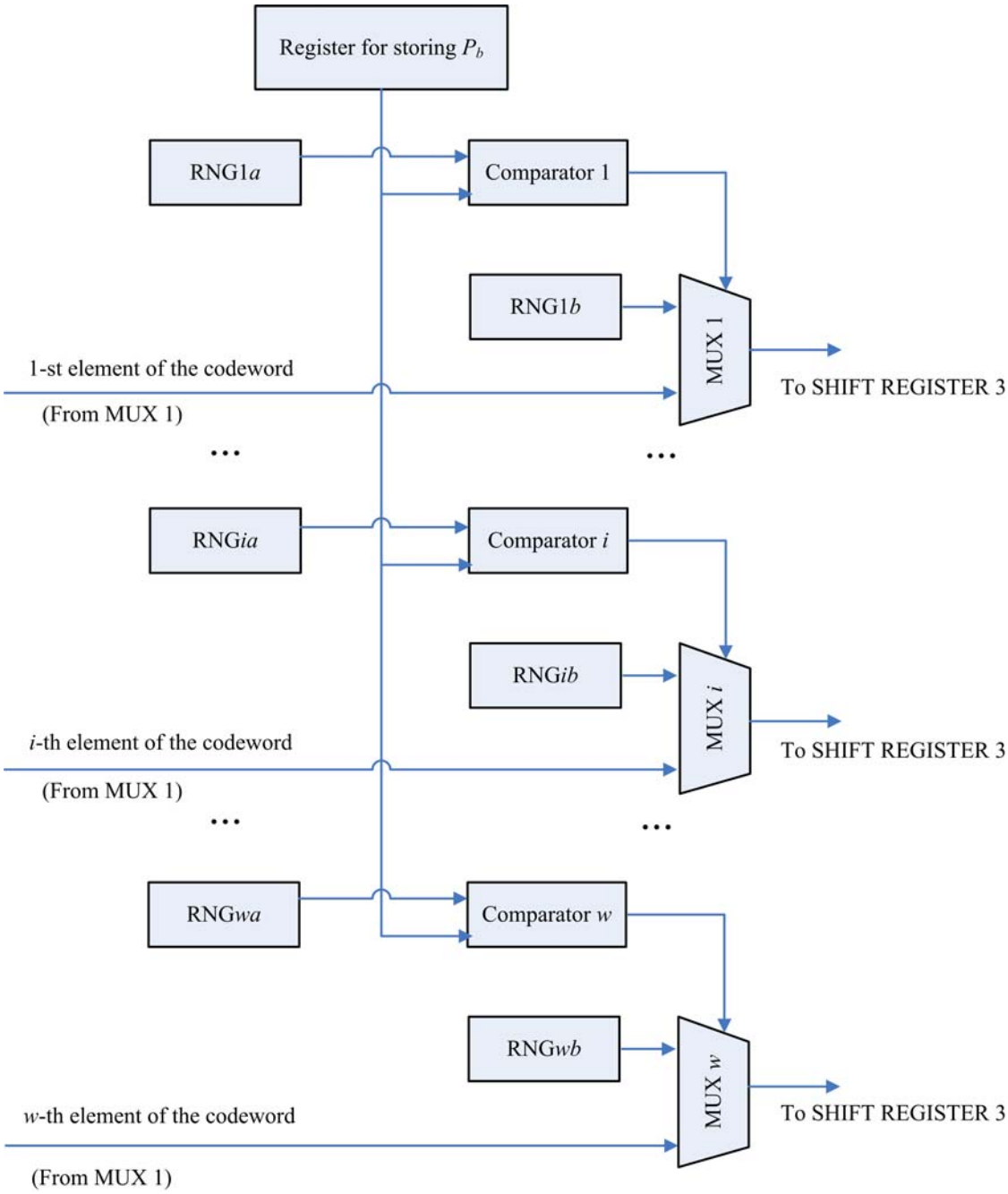

圖 3-4. 突變單元之硬體架構

突變的機率值 P_b 是儲存在一個獨立的暫存器中，而且這個機率值會被廣播到所有的突變電路上，在每個元素(Element)的突變電路中，從 RNG *ia* 產生的值會先和 P_b 做比較，假如 RNG *ia* 產生的值比 P_b 小，則會發生突變，而突變的值則是由 RNG *ib* 產生的值決定。

3.1.3 適應值計算單元(Fitness Evaluation Unit)

適應值計算單元主要的目的是，使用公式(2)來計算出經過交配突變程序後，儲存於 SHIFT REGISTER 3 的子代序列之平均失真值(Average Distortion)。如圖 3-5 所示即是適應值計算單元硬體電路架構，而此電路本身是 *N*-Stage Pipeline 的結構，其中 *N* 為碼字的數目。此 Pipeline 架構將會於每一個時脈(Clock)週期，取用一個訓練向量 $x \in X$ ，如圖 3-5 所示這些位於集合 X 中的訓練向量是儲存在 SDRAM 中，然後經由 DMA Controller 在 Avalon Bus 上傳送到適應值計算單元電路。

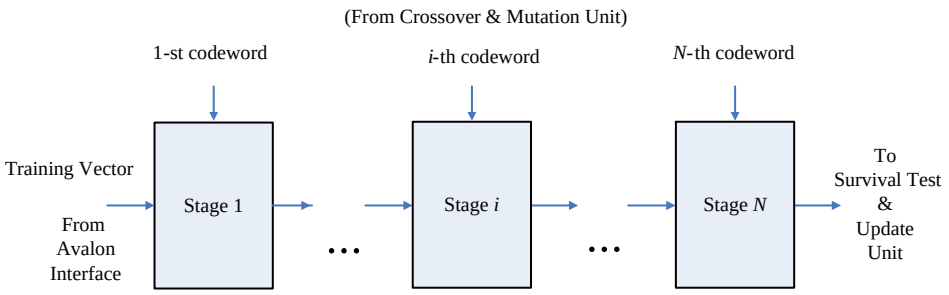

圖 3-5. 適應值計算單元之硬體架構

如圖 3-5 所示，在此 Pipeline 架構中，第 *i*- th Stage(*i* = 1,…,*N*)會計算此訓練向量和碼簿中第 *i* - th 碼字間的距離平方值，其中碼簿是經交配突變程序作用後儲存於 SHIFT REGISTER 3 的子代基因序列(即為新產生的碼簿)，得到距離平方值後，接下來再和第 *i*- th Stage 中先前計算所得的最小距離值做比較，如果算出的距離平方值比當下的最小距離值還要小，則第 *i* - th 碼字將會變成目前最佳的碼字，且其對應的距離平方值也會變成目前新的最小距離值。在完成第 *N*-th Stage 計算之後，當下所求得的最佳碼字和最小距離即是實際上此訓練向量的最佳碼字(即是在 *N* 個碼字中，距離此訓練向量最近的碼字)和其相對應的最小距離值。

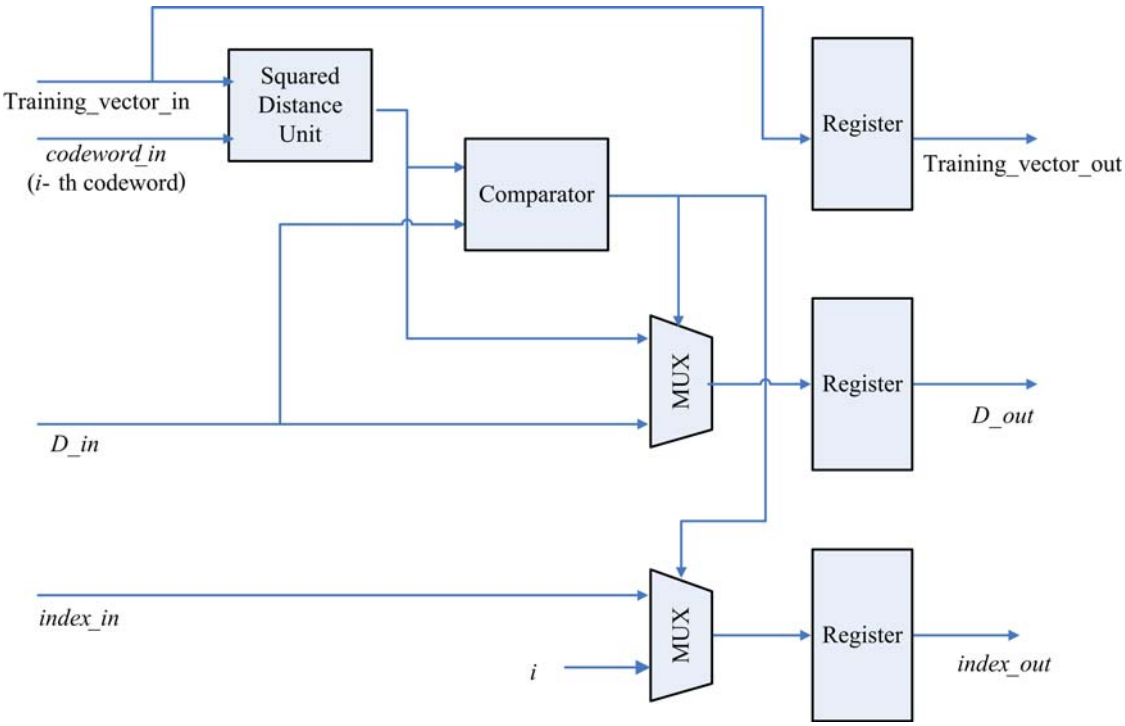

圖 3-6. 適應值計算單元 Stage *i* 之硬體架構

圖 3-6 所示即是 Pipeline Stage *i* (*i* = 1,…,*N*)硬體電路架構，我們可以觀察到此電路的輸入埠和輸出埠包含了:

(1) input ports：a. training_vector_in ：輸入的訓練向量

b. codeword_in：儲存於位移暫存器 3 中的第 *i* - th 碼字

c. D_in：Stage (*i* -1)訓練向量與最近碼字的最小距離值

d. index_in： Stage (*i* -1)的最佳碼字索引

(2) output ports：a. training_vector_out：輸出的訓練向量

b. D_out：Stage *i* 訓練向量與最近碼字的最小距離值

c. index_out：Stage *i* 的最佳碼字索引

如圖 3-6 所示，輸入的訓練向量和第 *i* - th 碼字間的距離平方值(標記為 Distortion_*i*)會先被計算出來，再和 D_in 作比較，當距離平方值比 D_in 大時，則將 index_out ← index_in 且 D_out ← D_in；反之，則將 index_out ← *i* 並且將 D_out ← Distortion_*i*。

而在第 *i*- th Stage 的輸出埠 training_vector_out、D_out 和 index_out 則分別連接到下一個 Stage(即 *i*+1-th Stage)的輸入埠 training_vector_ in、D_ in 和 index_ in；而在目前的時脈週期中，第 *i*- th Stage 所計算出的結果將會在下一個時脈週期傳送到第 *i*+1-th Stage 作運算;當訓練向量到達第 *N*-th Stage 時,最後輸出的 index_out 即是此訓練向量的最佳碼字索引，而 D_out 則是其對應的最小距離值；如圖 3-7

所示，最後此訓練向量和最佳碼字的最小距離平方值將會被傳送到最後第 N +1-th Stage 中做適應值累加計算。

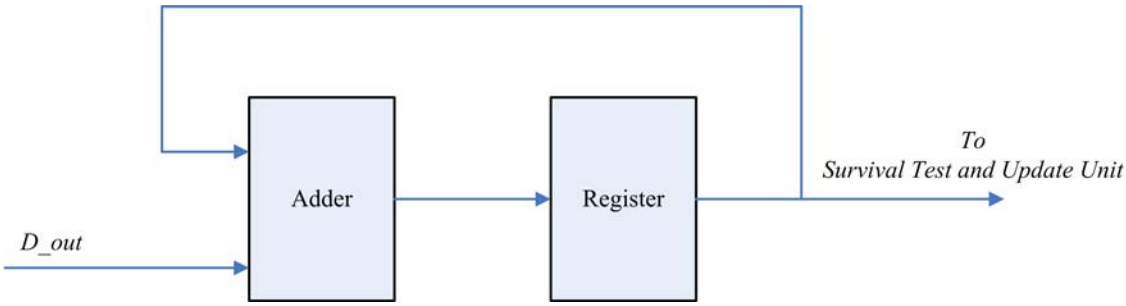

圖 3-7. 適應值計算單元 Stage *N*+1 之硬體架構

3.1.4 生存測試更新單元(Survival Test & Update Unit)

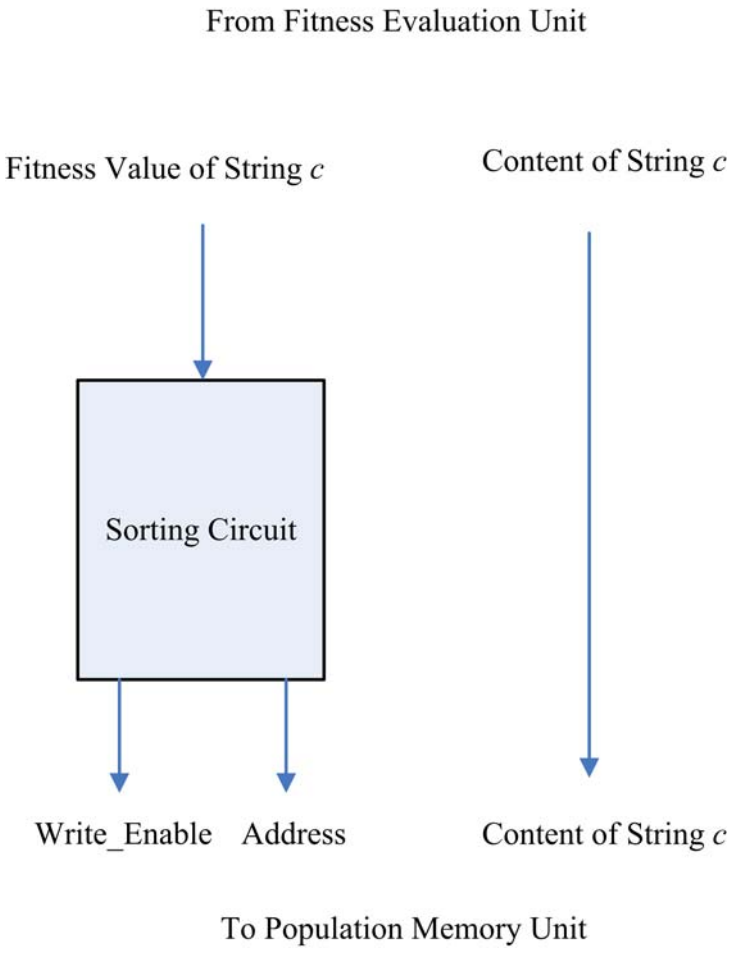

圖 3-8. 生存測試更新單元之硬體架構

如圖 3-8 所示，此生存測試更新單元包含了一個硬體排序電路(Sort Unit)[11]，此電路可以根據適應值的大小，由小到大排序 *N* 個母代序列，而圖 3-8 描述了此硬體排序電路架構，包含了 *N* 個比較器(Comparator)、多工器(Multiplexer)和暫存器(Buffer)；而 buffer *i* 儲存的內容為適應值排名 *i*-th 母代序列的失真值(標記為 *Di*)和索引值(標記為 *fi*)，在此要注意的是，每個索引值實際上表示的便是基因序列相對應於族群中的記憶體位址(Memory Address)。

在適應值計算單元完成後，可以得到新生子代基因序列的適應值，此適應值將被當做排序電路的輸入，廣播到所有的比較器。當子代的失真值(Distortion)比族群中有著最差適應值的母代還大時，換言之就是失真值比 D_N 大，此新生子代序列則無法存活，也就不需要做任何更新動作；相反地，失真值為 D_N 的母代基因序列將會被此新生基因序列取代，而其索引值將會變成此新生序列的索引值。

在排序電路中的子代基因序列位置是經由比較器的輸出所決定，而第 *i*-th 比較器的輸出則標記為 M_i，如圖 3-9 所示，比較器的輸出會連接到多工器，而每個多工器的真值表(Truth Table)於表 3-10 中表示之。而因為此特製的多工器，所以使這個排序電路能夠於一個時脈(One Clock)週期中完成任務，並且擁有 Low Area Cost 和 Routing Cost 的優點。

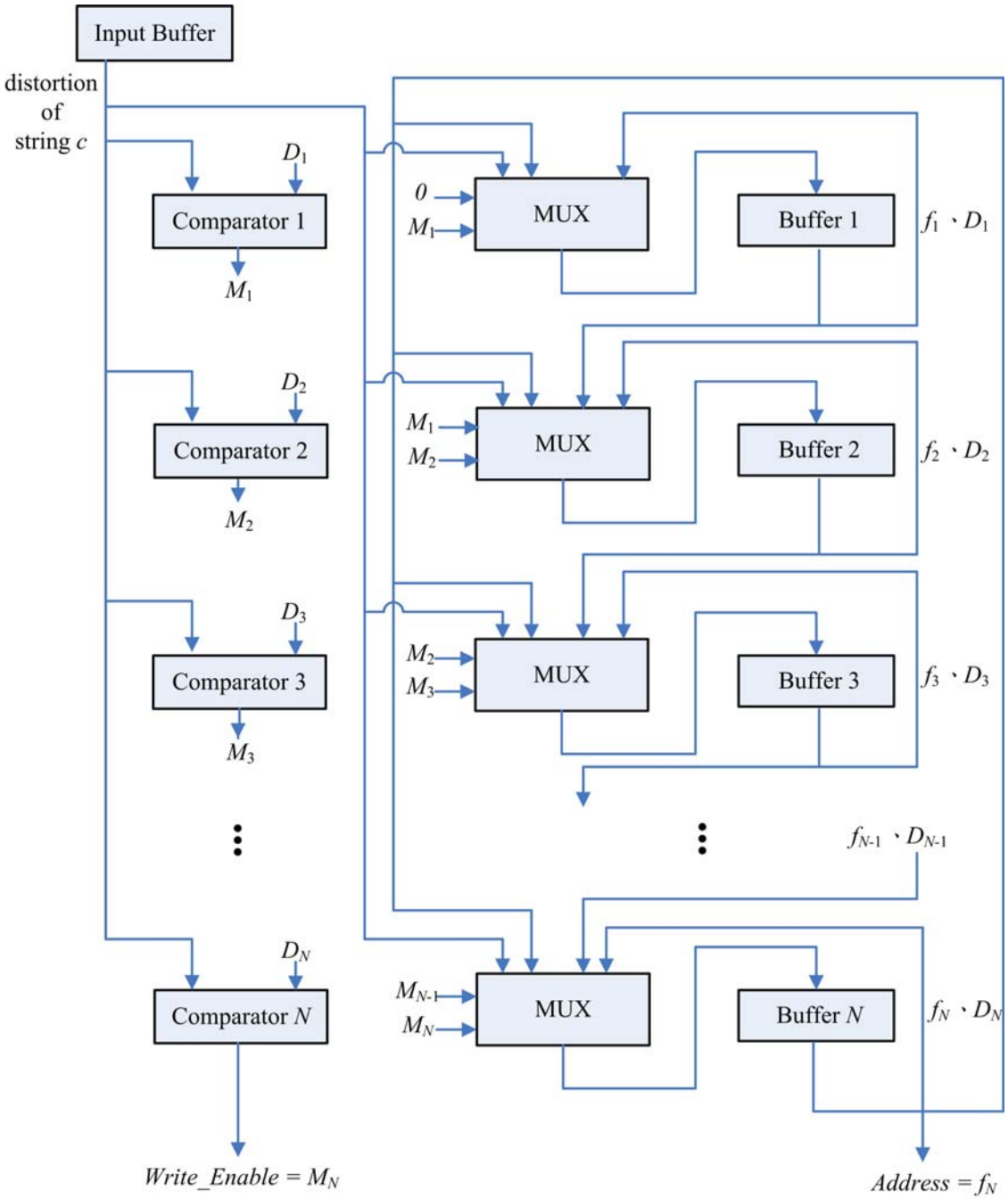

圖 3-9. 排序(Sort)電路硬體架構

M_{i-1}	M_i	Output
x	0	f_i 、D_i
0	1	f_N 、*distortion of string c*
1	1	f_{i-1}、D_{i-1}

表 3-10. 多工器 *i* 的真值表

3.2 Island GA 架構中每一個島嶼模組架構

本論文所提出的島嶼式基因演算法架構中每一個島嶼的基本硬體架構，如圖 3-11 所示，每一個島嶼都包含一個硬體加速器(即 Steady-State GA 硬體電路)、一個 DMA 控制器(Direct Memory Access)以及一個 softcore NIOS II 處理器。此硬體加速器用來加速各自島嶼內 Steady-State GA 的程序，而此加速器的硬體架構如上節的圖 3-1 所示。

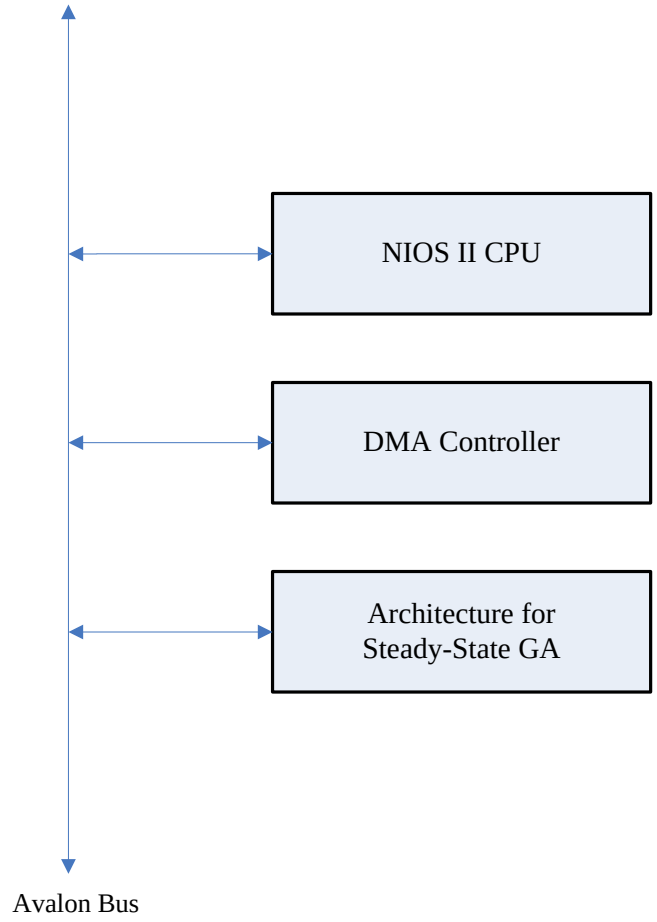

圖 3-11. 在 Island GA 架構中每一個島嶼模組架構

特別注意的是，當基因序列在計算適應值的時候會使用到訓練向量，而這些訓練向量起先儲存於主記憶體中，圖 3-11 中的 DMA 控制器會傳送這些訓練向量從主記憶體到圖 3-5 中的適應值計算單元的輸入，這樣方式可以大幅地降低存取主記憶體的時間。

而圖 3-11 中的 softcore 處理器扮演著島嶼式基因演算法程序中的協調者，這些 softcore 處理器在不同島嶼之間獨立運作，各個 softcore 處理器在不論是再生、交配突變、適應值計算或者生存測試更新這些程序，驅動著各自島嶼內的硬體加速器和 DMA 控制器。處理器之後也為子代基因序列的存活率作檢查，當存活率降至為零的時候便紀錄族群中最佳的基因序列，並且開始啟動移民基因序列的程序；而這移民程序也就是將 *K* 個基因序列從島嶼內的族群記憶體到島嶼外 on-chip RAM 內的 *K* 個基因序列作交換，在移民過後，硬體加速器以及 DMA 控制器將會再次啟動來進行島嶼內族群的演化，而 softcore 處理器也會再次判斷是否停止演化。下頁圖 3-12 則為此 softcore 處理器的流程圖。

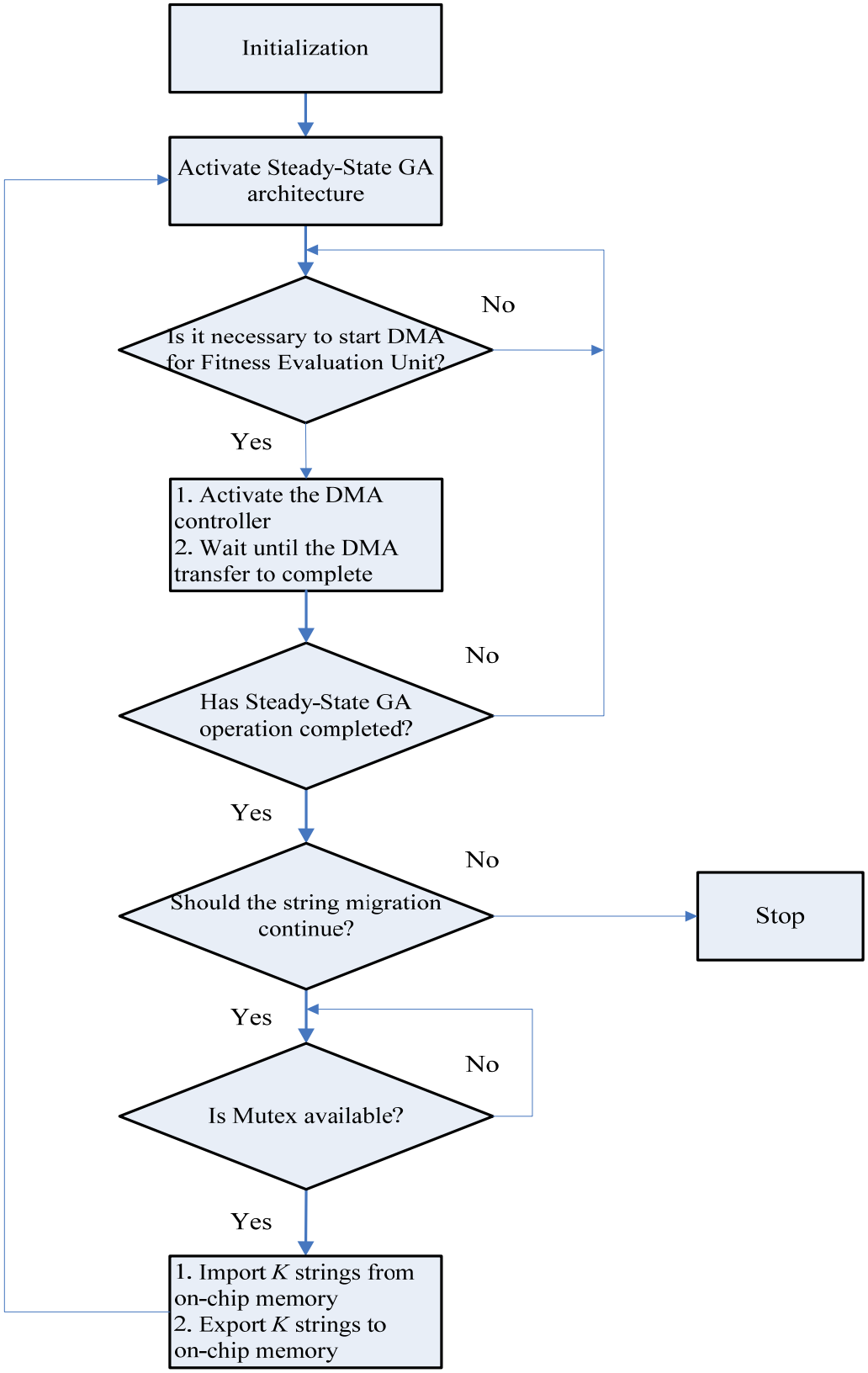

圖 3-12. NIOS II CPU 執行程式之流程圖

3.3 島嶼式基因演算法之系統架構

下頁圖 3-13 所表示的是整個島嶼式基因演算法的系統架構，其中包含了 *M* 個模組，而每一個模組皆有它各自的電路架構，也就是說，有 *M* 個模組負責各自的基因演化過程，而每一個模組則代表著一個島嶼，此一模組的架構如上節的圖 3-11。如圖 3-13 所示，由於各個模組都有一個 softcore 處理器，這整個架構可以視為一個多核心的系統架構，而每一個模組都共同分享著主記憶體。

對於島嶼式基因演算法的硬體實現，其中的一個難題就是基因序列的移民。由於每個島嶼可能得隨機挑選另一個島嶼進行基因序列的交換，而當島嶼數量很多的時候，這樣的移民會使得整個電路變的相當複雜甚至難以設計。為了簡單化基因序列移民的程序，除了將這 *M* 個島嶼模組化之外，我們還增加一個 on-chip 的記憶體(String Migration Memory)，而這 *M* 個模組將共享著此一 on-chip 記憶體。

基於我們使用了負責基因序列移民的快取(on-chip 記憶體)，我們便不需要為每一個島嶼額外再設計一個電路來選擇要移民的目標地。為了確保自己是唯一使用這塊 on-chip 記憶體，每一個島嶼將會相互競爭，而贏家將會與 on-chip 記憶體內的 *K* 個基因序列做交換，當交換完之後，贏家將會釋放此 on-chip 記憶體的主控權，同時之間，其他島嶼也會開始重新競爭之。

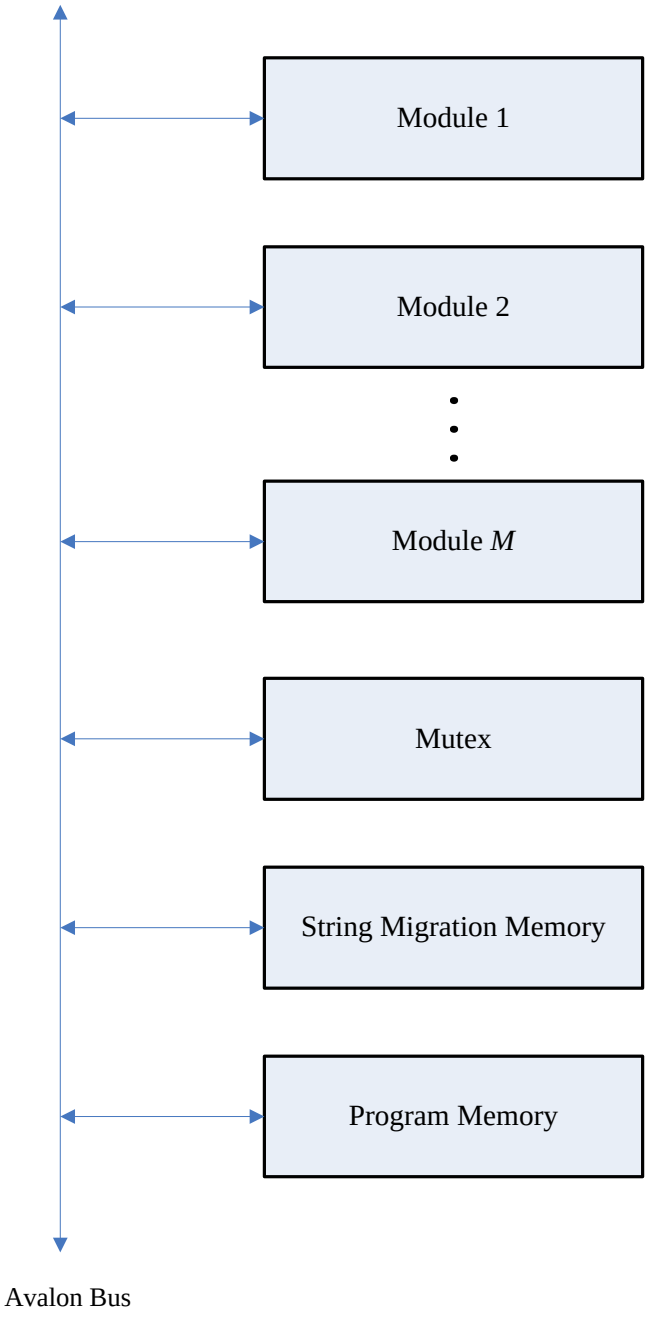

圖 3-13. 島嶼式基因演算法之系統架構

我們所謂的移民程序是基於負責基因序列移民的快取(on-chip 記憶體)，而非在兩個島嶼之間做交換的動作。事實上，一個島嶼贏家從 on-chip 記憶體所獲得的新基因序列是上一次島嶼贏家所交換過來的基因序列，而現在的島嶼贏家並不會把其基因序列移民給上一次的島嶼贏家。因此，在我們的架構中沒有所謂的基因序列互換這個動作，取而代之的是，將基因序列移民到下一個島嶼贏家的動作，而第一個島嶼贏家所獲得的基因序列則是先從族群中隨機挑選出來。這樣的機制與互換基因序列的機制會有著相似的效益，但卻可以大大降低硬體設計的複雜度。

當多個島嶼同時想要使用這塊 on-chip 記憶體時，我們必須保護這個基因序列移民記憶體免於資料存取誤用的窘境。如圖 3-13 所示，我們可以使用一個硬體 mutex 來解決這樣的難題。此一硬體 mutex 可以當作這些島嶼的協調者，它確保同時之間只能一個島嶼來存取這個基因序列移民記憶體。

特別注意的是，如果我們不是使用此硬體 mutex 這種不可分割的程序而採用其他方法，通常保護資料誤用的函式包含兩個不同的指令「測試」(Test)以及「設定」(Set)，而當某一個模組的處理器測試成功(succeed)並且是可用的(availability)，同一時間另一個模組的處理器也許也會測試成功，這樣的情形會造成兩個處理器都認為它們各自唯一可以使用這個基因序列移民記憶體。因此在我們的架構裡對於基因序列的移民機制，硬體 mutex 是相當重要的一環。

第四章 實驗數據與效能比較

本章節將呈現本論文所提出的島嶼式基因演算法系統架構之正確性、實際效能測量與效能比較，以及實驗環境的介紹。

4.1 開發平台與實驗環境介紹

本論文以 Altera 的 StratixII EP2S60 FPGA 開發板為平台，如圖 4-1 所示；而選擇以 FPGA 來實現與驗證硬體電路，是因為可程式化系統晶片設計 (System On a Programmable Chip，SOPC) 可以快速的將硬體設計實現與驗證，並且具備快速上市與系統再修改等多重優點，所以 FPGA 非常適合實現本論文提出的系統架構。如表格 4-2 是 Altera Stratix II EP2S60F672C5ES FPGA 開發板的詳細規格資訊。

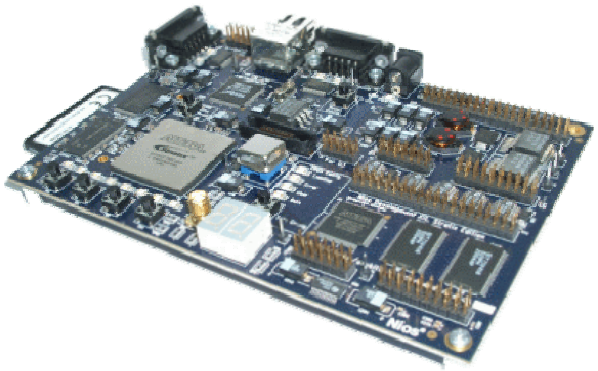

圖 4-1. Altera Stratix II 開發板外觀[12]

Feature	Stratix II
Device	EP2S60F672C5ES
Adaptive Logic Modules (ALMs)	24,176
Adaptive look-up tables (ALUTs) (1)	48,352
Equivalent Logic Elements (LEs) (2)	60,440
M512 RAM Blocks (512 bits + Parity)	329
M4K RAM Blocks (4 Kbits + Parity)	255
M-RAM Blocks (512 Kbits + Parity)	2
Total RAM bits	2,544,192
DSP block 9-bit elements	288
Total PLLs	6
Total DLLs	2
Total Pins	493

表格 4-2. Altera Stratix II EP2S60F672C5ES FPGA 開發板的規格[12]

Notes :

(1) One ALM contains two ALUTs. The ALUT is the cell used in the Quartus® II software for logic synthesis.

(2) This is the equivalent number of LEs in a Stratix device (four-input LUT-based architecture).

軟體部分使用 Altera Quartus II 7.2 當作撰寫 VHDL 硬體描述語言的平台，我們使用來設計與驗證我們的 Steady-State GA 電路，Quartus II 可提供語法檢查、時序分析、邏輯元件的配置、產生規劃檔案、電路合成以及繞線佈局等等強大的功能，如圖 4-3 所示；而 SOPC system 內許多元件包含 CPU、基因演算法電路、記憶體等等組成一個系統則是透過 Altera SOPC Builder 這套介面配置而成，如圖 4-4 所示。Altera Quartus II 7.2 所使用的 PC 規格為 Intel® Core™2 Duo CPU E8400 3.0GHz、2G DDRII 記憶體。

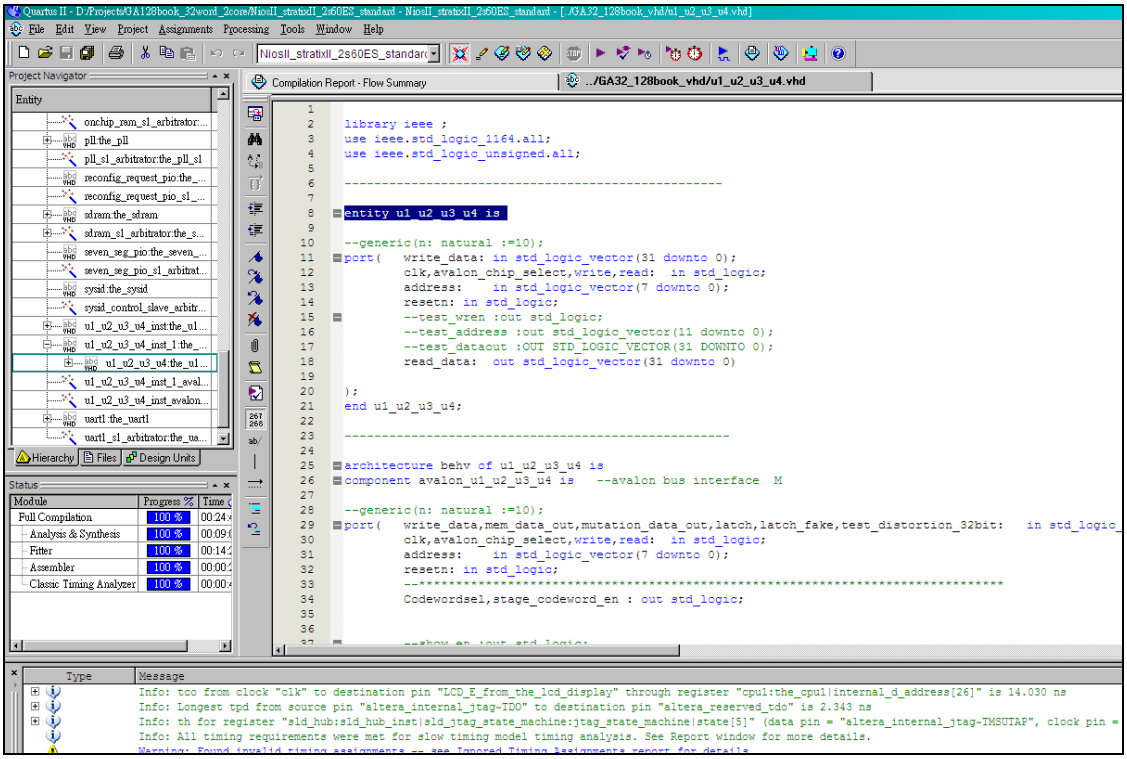

圖 4-3. Altera Quartus II 軟體介面

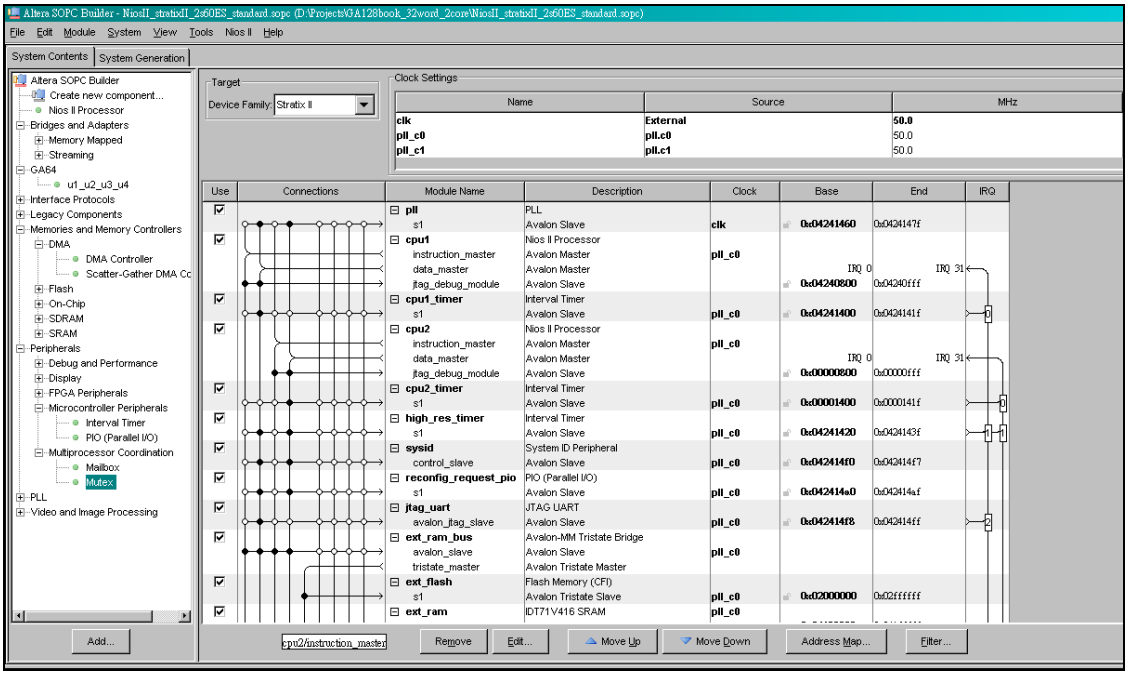

圖 4-4. Altera SOPC Builder 介面

本實驗也使用了 Linux GCC(GNU Compiler Collection)來當作我們實現相對應的軟體模擬硬體系統架構之環境，GCC 為目前 Linux 下最常用的 C 語言編譯器，實驗中以多執行緒(multithreading)的方法撰寫島嶼式基因演算法軟體模擬程式。我們利用讀取「Lena」512x512 bmp 圖檔做訓練向量的來源，如圖 4-5 所示，每個訓練向量為 2x2 的 4 維陣列(i.e., $w=4$)，所以會有 65536 個訓練向量。本實驗是在相同的訓練向量數目、碼字數目、碼簿數目，隨機從訓練向量中挑選的初始碼字和碼簿設定下，對軟體與硬體做失真值與效能的比較。

※ 硬體實現環境

Device：Altera Stratix EP2S60

CPU：NIOS II softcore

Memory：16-Mbyte SDRAM

Flash Memory：16-Mbyte

CF Card：16-Mbyte

※ 於 Linux 下軟體實現環境

Server：HP ML570

CPU：Intel(R) Xeon(TM) CPU 2.60GHz , 8 Cores

Memory：DDRII 2.0 G

Compiler：GCC (GNU Compiler Collection)

圖 4-5. Lena 測試影像

4.2 實驗數據的呈現與討論

本章節將探討以FPGA實現之實際效能的數據量測。我們採用Altera Stratix II 2S60ES [12]來當作我們硬體設計的FPGA開發實驗板，在本實驗中，adaptive logic modules(ALMs)數量總合為24176。我們採用Altera Quartus II以及 SOPC builder 來當作硬體設計的平台。實驗中碼字的維度為w = 2x2，且在VQ中會有32個碼字 (i.e., N = 32)，而突變機率為P_b = 0.03125。

	Steady-State GA Circuit	SOPC with only one module (M = 1)	SOPC with three module (M = 3)
Logic Utilization	17%	28%	94%
Estimated ALMs	3096(13%)	7162 (30%)	24139 (99%)
Block Memory Bit	16384 (3%)	613120 (24%)	643968 (25%)
DSP Block 9-bit Elements	128 (44%)	136 (47%)	288 (100%)
Total Pins	77 (16%)	183 (37%)	183 (37%)
Combinational ALUTs	3606(7%)	11955 (25%)	41019 (85%)
Dedicated Logic Registers	6250(13%)	9539(20%)	27209 (56%)

表格4-6. 於SOPC系統中Steady-State GA與Island GA的硬體資源消耗比較

表格4-6將呈現Steady-State GA架構下所消耗的Area Cost以及Island GA架構下所消耗的Area Cost，其中每一個島嶼即是一個Steady-State GA的架構，而Island

GA的島嶼數量設定為3 (i.e., $M = 3$)，每一個島嶼的基因序列數目則設定為16(i.e., $P = 16$)。如表格4-6所示，Steady-State GA電路只花費了3096個ALMs [13]。由於族群記憶體的電路是使用FPGA的內嵌式記憶體(Block Memory Bit)，在本實驗中族群記憶體儲存了16個基因序列，每一個基因序列為32碼字長度，每一個碼字由2×2維的pixel值組成，因此在Steady-State GA電路的族群記憶體總共消耗了16×32×4×8 = 16384bits。另外，Steady-State GA電路也使用了FPGA內的128個digital signal processing(DSP) block 9-bit elements來實現適應值計算單元內的距離平方計算功能。

除此之外，NIOS softcore CPU [14]和 DMA (Direct Memory Access)[15]控制器也會消耗一些硬體的資源，因此 Island GA 系統架構的資源消耗會比 Steady-State GA 系統架構的資源消耗來的高一些。然而，Island GA 系統架構下使用了 24139 個 ALMs，這樣的比例在 FPGA 開發板所提供的 ALMs 總數佔了 99%。Island GA 系統架構包含了三個島嶼架構以及一個 mutex 電路和 SDRAM 控制器，因此，整個 Island GA 系統架構所消耗的 ALMs 大約為每一個島嶼系統架構所消耗的 ALMs 三倍。

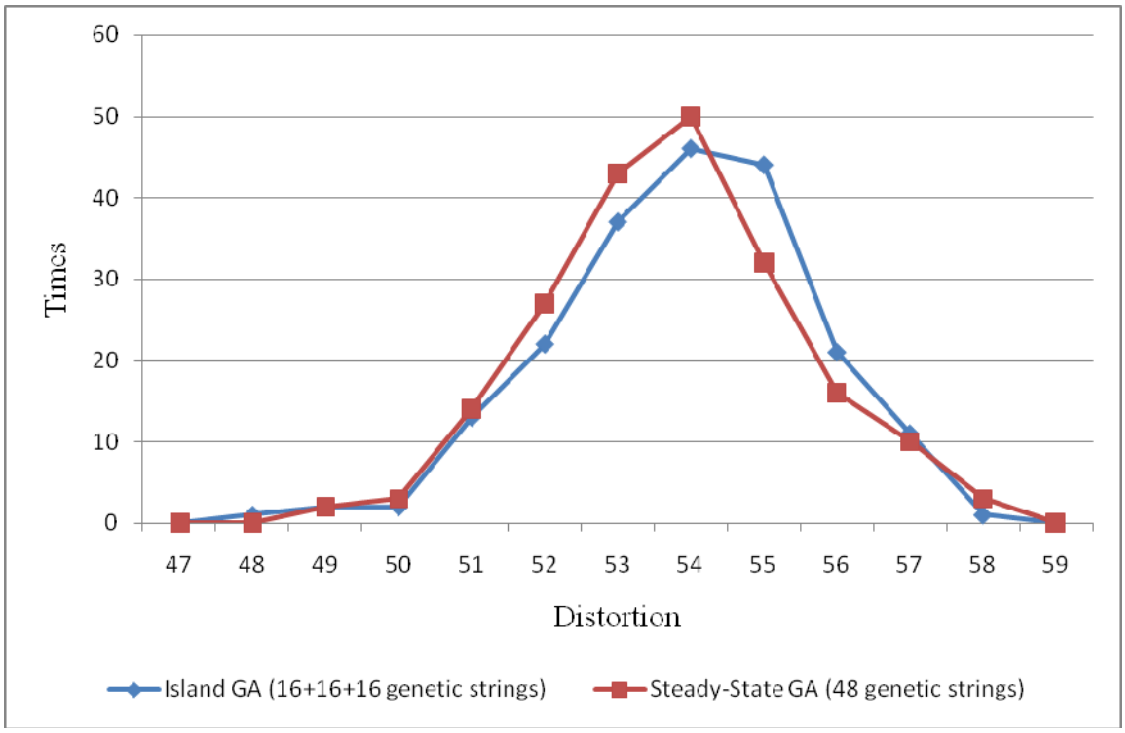

圖 4-7. 相同總數基因序列下 Steady-State GA 與 Island GA 的失真值分布圖

圖 4-7 說明了島嶼式基因演算法在島嶼數量為 3 時(i.e., $M = 3$) 平均失真值分佈的情況。實驗中設定為各別獨立執行 200 次的島嶼式基因演算法，每一個島嶼的基因序列數量為 16 個(i.e., $P = 16$)，因為在本實驗中設定為三個島嶼，因此基因序列總數為 48 個。另外，實驗中利用讀取「Lena」512x512 圖檔做為訓練向量的來源，由於每個訓練向量為 2x2 = 4 維，因此我們可取(512/2)x(512/2)=65536 個訓練向量。此外，我們亦拿 Steady-State GA 系統架構(i.e., $M = 1$) 的失真值分佈來做比較，此 Steady-State GA 系統架構同樣包含了總數為 48 個的基因序列。而圖 4-7 是兩種不同的基因演算法但卻基於在同樣的基因序列總數上所做的比較，圖 4-7 所呈現的是兩條相似的曲線分佈，因此這兩種基因演算法的失真值分布情

況是相當類似的。這樣的結果說明了一件事情，也就是在 VQ 設計上並不會因為採用這種島嶼式基因演算法而降低了原本的效益(失真值的表現)。

	Average Distortion	Average Execution Time
Hardware Island GA (*M* = 3)	54.11	1293.8 ms
Hardware Steady-State GA (*M* = 1)	54.03	1424.6ms

表格 4-8. 相同總數基因序列下 Steady-State GA 與 Island GA 的執行時間比較

表格 4-8 是根據圖 4-7 延續的實驗，由於單一 48 個基因序列收斂過程會比單一 16 個基因序列收斂過程慢，Island GA 利用這種特性，將一個數目為 48 的基因序列分成三個島嶼各有一個數目 16 的基因序列去演化。表格 4-8 說明了 Island GA 系統架構與 Steady-State GA 系統架構在平均失真值相近(如圖 4-7 的失真值分佈)的情況下執行時間的差異，而數據顯示 Island GA 執行時間為 1293.8ms 而 Steady-State GA 執行時間為 1424.6ms，因此，Island GA 相對於 Steady-State GA 的 Speedup 約為 1.10。根據圖 4-7 以及表格 4-8，我們可以歸納出本實驗提出的系統架構能夠加速基因演化程序之餘亦不犧牲它的效益(失真值的表現)。

	Average Distortion	Average Execution Time
Software Island GA (single-thread)	54.06	85208.7 ms
Software Island GA (multi-thread)	54.24	38555.9 ms
Hardware Island GA	54.11	1293.8 ms

表格 4-9. 軟硬體 Island GA 的平均執行時間比較

表格 4-9 比較了各種不同方法(包括軟體與硬體)實現島嶼式基因演算法的執行時間。在相同的實驗條件下分別執行兩百次的軟硬體島嶼式基因演算法：島嶼數目 $M = 3$，每一個島嶼的基因序列數目 $P = 16$，碼字數目 $N = 32$，向量維度 $w = 2\times2$，訓練向量數目為 65536 筆。軟體的實現環境是使用八核心工作站電腦(每個核心處理器為 Intel(R) Xeon(TM) CPU 2.60GHz)，除了一般單執行緒的方法，我們亦採用多執行緒(multi-threading)的方法來實現之，每一個島嶼的演化程序使用各自獨立的執行緒，不同的執行緒因為分配到不同的核心去處理，因此，所有島嶼的演化程序可以平行處理，使得這樣多執行緒的方法相對於單執行緒的方法所獲得的 Speedup 約可達 2.21 (理想值為 3) ；而硬體的實現環境是使用 NIOS II 處理器速度為 50MHz，如表格 4-9 所示，我們可以明顯地觀察出本實驗的島嶼式基因演算法硬體系統架構的執行時間大幅度低於其對應的軟體模擬系統的執行時間，例如島嶼式基因演算法硬體系統架構執行時間只要 1293.8ms，而其對應的單

執行緒軟體模擬系統執行時間卻高達 85208.7ms，事實上，本實驗的島嶼式基因演算法硬體系統架構相對於其對應的單執行緒軟體模擬系統之 Speedup 為 65.9 倍，而島嶼式基因演算法硬體系統架構相對於其對應的多執行緒軟體模擬系統之 Speedup 為 30 倍，在此 Speedup 的定義為軟體執行時間(2.60GHz Intel(R) Xeon(TM) CPU)除以硬體 SOPC 系統執行時間(50MHz NIOS-based softcore CPU)。

P	Entire System Embedded Memory Bits	Entire System ALMs	Average Distortion	Average Execution Time
4	619392 (23%)	22145 (92%)	58.24	869.5 ms
8	627584 (24%)	22829 (94%)	57.41	991.2 ms
12	635776 (24%)	23493(97%)	55.53	1153.7 ms
16	643968 (25%)	24139 (99%)	54.11	1293.8 ms

表格 4-10. 不同基因序列個數 *P* 下硬體資源消耗、平均失真值以及平均執行時間比較

從表格 4-10 可以觀察出在本論文所提出的 SOPC 系統架構下，每一個島嶼在不同的基因序列大小 *P*，其所消耗的硬體資源、平均失真值以及執行時間的差異。本實驗所獲得的平均失真值以及執行時間亦為平均執行兩百次的結果。從表格 4-10 可以得知，當基因序列數目 *P* 增加的時候，我們提出的 SOPC 系統架構所消耗的硬體資源只有微幅增加，而平均執行時間的增加量約莫與基因序列數目 *P* 呈線性增加。此外，當基因序列數目 *P* 變的越大，平均失真值的也會跟著降低。

T	Software Execution Time	Hardware Execution Time	Speedup
1024	1012.4 ms	83.7 ms	12.1
2048	1911.8 ms	127.5 ms	15.0
4096	3519.7 ms	187.2 ms	18.8
8192	5863.1 ms	291.7 ms	20.1
16384	10123.5 ms	445.9 ms	22.7
32768	19202.8 ms	732.9 ms	26.2
65536	38555.9 ms	1293.8 ms	30.0

表格 4-11. 不同訓練向量筆數 *T* 下硬體資源消耗、平均失真值以及平均執行時間比較

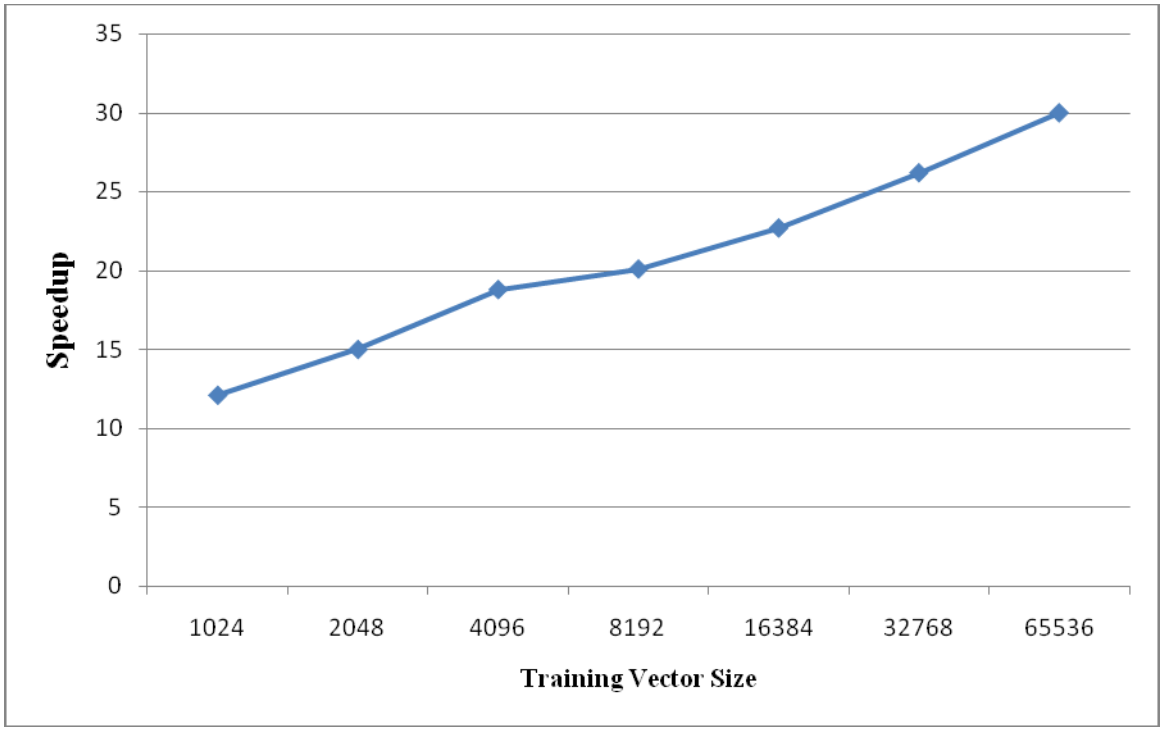

圖 4-12. 不同訓練向量筆數 *T* 下硬體相對於軟體的 Speedup

為了更進一步證明本論文所提出架構的有效性，表格 4-11 與圖 4-12 說明了在不同訓練向量筆數下多執行緒軟體模擬系統與硬體架構系統花費時間及其效能比。本實驗的條件一樣為：島嶼數目 $M = 3$，每一個島嶼的基因序列數目 $P = 16$，碼字數目 $N = 32$，向量維度 $w = 2x2$，訓練向量則為變數。從表格 4-11 可以觀察到隨著訓練向量集合的變大，硬體相對於軟體所獲得的 Speedup 亦會跟著增加。主要的原因是訓練向量集合傳送到適應值計算單元的方式不同；在軟體中訓練向量集合是儲存於記憶體中，因此在傳送訓練向量時將會耗費大量的記憶體存取時間，除此之外，適應函數是使用公式(2)，所以軟體的計算複雜度會很高。

在我們所提出的架構中，適應值計算單元中是採用 DMA (Direct Memory Access) 和 Pipeline 技術來幫助系統降低記憶體存取時間，以及適應值計算單元的計算時間；因此，當訓練向量的數目愈來愈大時，本論文提出的系統架構比上軟體的執行時間可以擁有高效能比，例如：當訓練向量個數為 65536 時，硬體 SOPC 系統架構的執行時間為 1293.8 ms，而其對應的多執行緒軟體模擬系統的執行時間則是 38555.9 ms，效能比為 30 倍。因此本實驗所測量的這些實驗數據可以更加地說明我們所提出的硬體系統架構是具備有效性。

第五章 結論與未來展望

本論文提出的架構可以有效地在硬體實現島嶼式基因演算法。由於每一個島嶼演化方式採用的是 Steady-State GA，以至於能夠降低此硬體加速器的複雜度。除此之外，對於基因的演化程序，我們為每一個島嶼配置一個獨立的處理器核心以及一個硬體加速器，這樣的好處是可以加速基因最佳化的過程而不犧牲它整體的效益。我們亦可以增加族群的數量來降低平均失真值而不花費過多的硬體資源。而隨著訓練集合數量增加，本論文提出的硬體系統架構相對於它的軟體模擬系統加速的程度也會變的更高。因此，本架構在不犧牲原有的效益表現之下，對於一些需要即時運算的基因最佳化應用是一個不錯的選擇。

參考著作

[1] Mitchell, M., An introduction to Genetic Algorithm, *MIT press*, 1996.

[2] Eiben, A. E., and Smith, J. D., Introduction to Evolutionary Computing, *Springer*, 2003.

[3] Fogel, L. J., Owens, A. J. and Walsh, M. J., Artificial Intelligence Through Simulated Evolution, New York: *Wiley*, 1996.

[4] Gersho, A., and Gray, R. M., Vector Quantization and Signal Compression, *Kluwer, Norwood, Massachusetts*, 1992.

[5] Scheunders, S., "A genetic c-means clustering algorithm applied to color image quantization," *Pattern Recognition*, Vol. 30, 6, pp. 859-866, 1997.

[6] Hwang, W. j., and Hong, S. L., "Genetic entropy-constrained vector quantization," *Optical Engineering*, Vol. 38, pp.233-239, 1999.

[7] Rasheed, K., and Davisson, B.D., "Effect of global parallelism on the behave of a steady state genetic algorithm for design optimization," In Proceedings of the Congress on Evolutionary Computation, Washington, DC, 1999.

[8] Hauck, S., and Dehon, A., Reconfigurable Computing, Morgan Kaufmann, 2008.

[9] NIOS II Processor Reference Handbook, 2008, Altera Corporation. http://www.altera.com/ literature/ lit-nio2.jsp.

[10] Barry, S., Motoo, T., Richard, J.C., Greg, S., “FPGA Implementation of Neighborhood of Four Cellular Automata Random Number Generators ” ,HP *Laboratories Palo Alto*.HPL-2001-290,2001

[11] Hwang, W.J., Li, H.Y., Yeh, Y.J. and Chan, K.F., “FPGA Implementation of Competitive Learning with Partial Distance Search in theWavelet Domain ” Chap.8 of the Book Progress in Neurocomputing Research, Edited by G. B. Kang, pp.203-221, NOVA Science Publisher, 2008.

[12] Stratix II Device Handbook, 2008, Altera Corporation. http:// www.altera.com/ literature/ lit-nio2.jsp.

[13] Hutton, M., et al, “Improving FPGA Performance and Area Using an Adaptive Logic Module," Lecture Notes in Computer Science, vol. 3203, FPL 2004, pp.135-144, 2004.

[14] NIOS II Processor Reference Handbook, 2008, Altera Corporation. http://www.altera.com/ literature/ lit-nio2.jsp.

[15] Embedded Peripherals Reference Handbook, DMA controller core, 2008, Altera Corporation. http://www.altera.com/ literature/ lit-nio2.jsp.