

國立交通大學

電機資訊學院 電子與光電學程

碩士論文

快速傅立葉轉換在軟硬體共同設計之研究



On Software/Hardware Co-Design of FFT

研究生：林煌翔

指導教授：周景揚 教授

中華民國九十四年六月

快速傅立葉轉換在軟體硬體共同設計之研究

On Software/Hardware Co-Design of FFT

研究生：林煌翔

Student：Huang-Cang Lin

指導教授：周景揚

Advisor：Dr. Jing-Yang Jou

國立交通大學

電機資訊學院 電子與光電學程



A Thesis

Submitted to Degree Program of Electrical Engineering Computer
Science

College of Electrical Engineering and Computer Science

National Chiao Tung University

in Partial Fulfillment of the Requirements

for the Degree of

Master of Science

in

Electronics and Electro-Optical Engineering

June 2005

Hsinchu, Taiwan, Republic of China

中華民國九十四年六月

快速傅立葉轉換在軟硬體共同設計之研究

研究生：林煌翔 指導教授：周景揚博士

國立交通大學
電機資訊學院 電子與光電學程 (研究所) 碩士班

摘 要

本論文針對快速傅立葉轉換，建立了一種方法去做軟體與硬體間的模擬以及共同設計。我們將使用 SID 硬體模擬器，以 ARM 微處理為核心去架構軟體快速傅立葉轉換(FFT)及硬體加速器，並針對不同的硬體架構做硬體效率,硬體花費和速度上的比較分析。最後我們提供設計者方便的模擬環境，利用系統設計者所提供的快速傅利葉轉換的點數和硬體面積的限制，我們可以將 ARM 處理器以及硬體快速傅利葉轉換做面積或者是運算速度最佳化。

On Software/Hardware Co-Design of FFT

Student: Huang-Cang Lin Advisor: Dr. Jing-Yang Jou

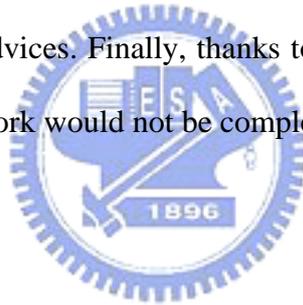
Degree Program of Electrical Engineering Computer Science
National Chiao Tung University

ABSTRACT

In this thesis, we propose a new platform for software/hardware co-design of FFT based on the SID hardware simulation software with ARM processor simulation core. With this platform, we compare the different hardware structures and analyze their efficiency, cost and speed improvements. Experiments show that this platform provides a very good simulation environment for system designers. The area and timing optimization for the hardware FFT can be easily achieved.

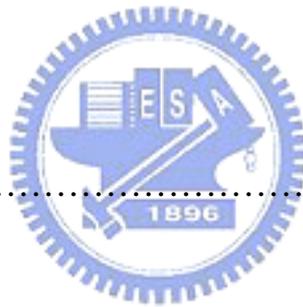
Acknowledgements

I would like to express my sincere gratitude to my advisor, Dr. Jing-Yang Jou, for his insightful suggestion and patient guidance throughout the course of this work. Besides, I have to thank all the members in the EDA group. I deeply appreciate Cheng-Yeh Wang for his constructive suggestions on this project. I also would like to thank Liang-Yu Lin for his advices. Finally, thanks to my wonderful family, without their patience and love, this work would not be completed.



目錄

| | |
|-------------------------------------------|-----|
| 摘要 | i |
| ABSTRACT | ii |
| 誌謝 | iii |
| 目錄 | iv |
| 表目錄 | vi |
| 圖目錄 | vii |
| 第一章、Introduction..... | 1 |
| 第二章、Preliminaries..... | 5 |
| 2.1 FFT Algorithm Overview..... | 5 |
| 2.2 ARM History..... | 8 |
| 2.3 Real Time Operating System: eCos..... | 13 |
| 2.4 SID Platform..... | 15 |
| 第三章、Proposed Approach..... | 17 |
| 3.1 Our Approach..... | 17 |
| 3.2 Hardware Modeling SID Platform..... | 21 |
| 3.3 SW/HW Interface Modeling..... | 27 |



| | |
|-----------------------------------------|----|
| 3.4 FFT Software Pseudo Code..... | 30 |
| 3.5 FFT Accelerator Circuit Design..... | 32 |
| 第四章、Experimental Results..... | 38 |
| 第五章、Conclusions..... | 46 |
| 参考文献..... | 48 |
| 自傳..... | 50 |



List of Tables

| | | |
|-----------|------------------------------------------------|----|
| Table 1.1 | Comparison of ASIC Implementations..... | 4 |
| Table 3.1 | FFT Accelerator Computation time (N=8192)..... | 37 |
| Table 4.1 | SW/HW FFT Computation Time..... | 43 |



List of Figures

| | | |
|-------------|--------------------------------------------------------|----|
| Figure 1.1 | Performance vs. Flexibility between Asics and ARM..... | 3 |
| Figure 2.1 | Butterfly Graph of Radix-2 DIF FFT..... | 8 |
| Figure 2.2 | SFG of 16-Point Radix-2 DIF FFT..... | 8 |
| Figure 2.3 | Von Neumann Machine..... | 11 |
| Figure 2.4 | ARM7T Architectures..... | 11 |
| Figure 2.5 | Embedded Software System of eCos Packages Layering. | 14 |
| Figure 3.1 | System Framework..... | 20 |
| Figure 3.2 | FFT Platform..... | 23 |
| Figure 3.3 | CDK Design Flow..... | 24 |
| Figure 3.4 | SID System Mointor..... | 26 |
| Figure 3.5 | Trace Program..... | 26 |
| Figure 3.6 | Simulation Tool..... | 27 |
| Figure 3.7 | FFT Sw/Hw Model..... | 29 |
| Figure 3.8 | FFT Register/State Machine Model..... | 30 |
| Figure 3.9 | R2SDF Component..... | 34 |
| Figure 3.10 | The Simulation Waveform..... | 34 |
| Figure 3.11 | The FFT Accelerator I/O Interface..... | 35 |
| Figure 3.12 | R2SDF Architecture..... | 36 |
| Figure 4.1 | Dataflow for 64-Point FFT..... | 39 |
| Figure 4.2 | Simulation Structure of FFT Implementation..... | 39 |

Figure 4.3a FFT 8192-point Computation Time vs. Accelerator Point 44
Figure 4.3b FFT 4096-point Computation Time vs. Accelerator Point 44
Figure 4.3c FFT 1024-point Computation Time vs. Accelerator Point 45
Figure 4.3d Computation cycle vs Traffic cycle 45



Chapter 1

Introduction

步入二十一世紀所謂的後 PC 時代裡，個人電腦市場已逐漸趨於飽和，而整合多種功能的消費性電子及資訊家電產品，則成為各廠商主要積極開拓與創造商機的標地。同樣以行動電話為例，過去只需要一個基頻數位訊號處理器和一個一般用途微處理器，用來接收無線射頻模組傳過來的訊號，再加上一個處理 MMI (man machine interface) 的單晶片，來處理人機介面接收鍵盤與顯示訊息用途，就可以完成一個行動電話的核心，但在望眼往後的未來，全世界的行動電話市場攤開來看，純文字介面的機種，無論在利潤上或是消費者的評比上，都比不上圖形介面在操作上的方便，而具有和絃鈴聲動態下載與彩色顯示器的機種又略勝一籌，更具吸引力。具備 FM 收音機功能、MP3 撥放功能、或具備數位相機，功能者更是市場上的最愛。然而，隨之帶來的卻是必須面對著即時大量資料流、高速運算和超低耗電量的應用系統規格。因此如何符合這些市場上的需求，並且降低體積、降低硬體

成本、擴展人類的視野，而真正改變人類的生活將是我們這些 IC 設計者的首要工作。

在現今數位影音、數據通信發展一日千里，快速傅立葉轉換 (FFT) 效能的提高將扮演一個重要的角色。數位 IC 設計者如何在短時間內設計一個符合系統需求的 FFT，則成為相當值得改進與研究的課題。本論文之主要目標為設計一方便的模擬環境，可以於設計 FFT 的同時做軟硬體間的共同設計以及驗證。首先我們利用硬體模擬器 SID 去架構我們的 ARM 微處理器的平台，讓我們的數據更符合實際的情況。這個 ARM 微處理器擁有許多優點，例如較高的系統性能、較低的耗電量，較高的設計彈性，比較快速商品化的時效性...等等。

然而現今系統的需求，不足以擁有單一 ARM 微處理器或者是單一傳統應用導向積體電路 (ASIC) 即能滿足系統所有的需求。因此，我們設計了一個新的元件，它匯集 ARM 微處理器及 ASIC 所有的優點，稱為可規劃的快速傅立葉轉換 (configurable FFT)。如圖 1.1 所示，ARM 微處理器比 ASIC 擁有更高的設計彈性，我們不需要重新設計 IC，僅需要短時間的修正韌體即可符合我們的需求，另一方面 ASIC 提供了一個效能上的需求，它可以比 ARM 微處理器擁有更快的運算效能，總而言之，如何結合兩者的優勢，去做兩者之間的取捨，將是

未來我們的設計方向。如表格 1.1 所示，我們根據 ARM 微處理器及 ASIC 在運算效能上，成本(面積)、耗電、及設計彈性上，繪出下面的表格，表格上面我們可以看出 ASIC 擁有運算效能、成本及耗電上的優勢，但在面臨市場上使用使用者即時性的需求，ASIC 商品量產化的時程上就比不上 ARM 微處理器上的快速，因此，我們從上面得到一些靈感；我們可以結合 ARM 微處理器及一個小點數的 FFT ASIC，並提供設計者一個高階的模擬環境，讓它很容易可以根據系統效能，計算 FFT ASIC 點數的大小，以達到系統之效能規格並且使用面積最小的 ASIC 為目的。

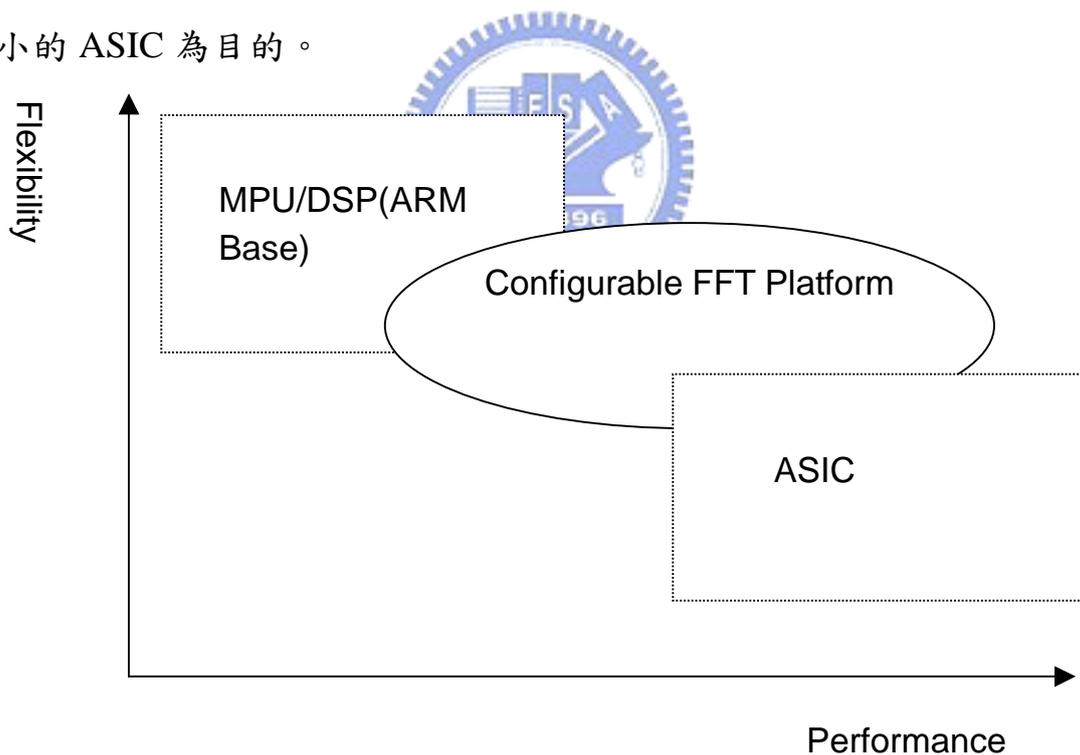
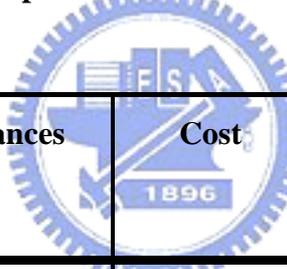


Figure 1.1 Performance vs. Flexibility between ASICs and ARM

本論文基本上分成五章：在第一章，簡要說明研究動機及方法以及論文架構與內容。在第二章，我將會對快速傅立葉轉換 (FFT)、ARM 的處理器核心架構、SID 硬體模擬器之應用做一個簡單的介紹、並加以分析。第三章，則是對快速傅立葉轉換軟體中的各個元件逐一做個介紹與分析，並提出一個新的模擬方法，加以討論。第四章，則將我們所設計的方法，提出確實的實驗數據加以討論及驗證，第五章則是本論文的討論、應用的方向與結語。

Table 1.1 Comparisons of ASIC Implementations



| Devices | Performances | Cost | Power Consumption | Flexibility |
|----------------|---------------------|-------------|--------------------------|--------------------|
| ASIC | HIGH | HIGH | LOW | LOW |
| ARM BASE | MED | MED | MED | HIGH |

Chapter 2

Preliminaries

由於超大型積體電路 (VLSI) 技術大幅改善，許多類型 FFT 處理器已經在這幾年內蓬勃發展，在這些 FFT 處理器中大多以記憶體為基礎 (memory-based) FFT 和管線化作業 (pipelined) FFT 架構為主，由於他們的結構是規則的，而且控制電路也比較容易設計，所以最廣泛的被使用。在這個章節裡，我們將介紹 FFT 的基本原理及架構，接著我們將介紹 ARM 微處理器的基本架構，最後我們也將 SID 硬體模擬器平台做詳盡介紹與討論。

2.1 FFT Algorithm Overview

離散傅立葉轉換 (DFT) 在信號分析上是非常重要的計算工具，目前傅立葉轉換的技術也廣泛應用於各個領域，為數位信號處理技術中一個重要的部分，這些應用通常要在很短的時間內完成傅立葉轉換計算，在資料眾多的情況下，一般計算機運算無法滿足速度的需求；

因此，快速傅立葉轉換應運而生。

FFT 係於 1965 年由 Cooley 與 Tukey [1] [2] 兩位教授所發明的，它的功用是改善 DFT 的運算量，以便增快運算速度；FFT 使 N 點的離散傅立葉轉的運算次數由 N^2 降為 $N \log_2 N$ ，因此我們就將焦點瞄準快速傅立葉轉換分析。根據他們的演算法中的推論，其中 N 必須是二個或多個整數的乘積，也由於這個論文的發表，使得離散傅立葉轉換在信號處理上的應用蓬勃發展，且讓一些高效率計算演算法被發現，這些演算法統稱為快速傅立葉轉換(FFT)。FFT 演算法的基本原理是將長度為 N 的序列的離散傅立葉轉換的計算分解成許多小的離散傅立葉轉換，這個原理的實現讓我們產生了各種不同的演算法，且它們在計算速度上的增進或者是面積的降低是有目共睹的，因此我們就將焦點瞄準快速傅立葉轉換架構上分析。

根據 Cooley 與 Tukey [1] [2] 兩位教授所提的演算法，若 $X(n)$ 為 N 點的序列， $n=0,1,2,\dots,N-1$ ，這個 N- point DFT 定義如下所示：

$$X(k) = \sum_{n=0}^{N-1} x(n)W_N^{nk}, \quad k = 0,1,\dots,N-1 \quad (2.1)$$

其中 $x(n)$ 為時域函數， $X(k)$ 為經過轉換後的頻域函數，N 為取樣數量，

$W_N^{nk} = e^{\frac{-j2\pi nk}{N}} = \cos\left(\frac{2\pi nk}{N}\right) - j \sin\left(\frac{2\pi nk}{N}\right)$ 稱為 twiddle factor。FFT 運算的方法有兩個方式，一為在時間上消去法，稱為 Decimation-in-time，另一個

運算方式為在頻率上消去法，所以我們稱之為 Decimation-in-frequency。

部分用來增進 DFT 計算效率的方法均利用 W_N^{nk} 的對稱性和週期性，使用這個性質及正弦和餘弦函數的對稱性，我們可根據式子(2.2)進行化簡整理，在這種方式，乘法的數目大約降低 2 倍，因此計算量可明顯的大大降低。

$$A \times W_N^{nk} + B \times W_N^{nk+N/2} = (A + B \times W_N^{N/2}) \times W_N^{nk} = (A - B) \times W_N^{nk} \quad (2.2)$$

若以 2 為基數(Radix-2) [6]來考慮，將(2.1)式中的 $x(n)$ 依 n 的奇數項和偶數項分為兩組，即 $n=2r$ 和 $n=2r+1$ 分為兩組，則

$$\begin{aligned} X(2r) &= \sum_{n=0}^{N/2-1} [x(n) + x(n + N/2)] W_{N/2}^{nr} \\ X(2r+1) &= \sum_{n=0}^{N/2-1} [x(n) - x(n + N/2)] W_N^n W_{N/2}^{nr}, \quad r = 0, 1, \dots, (N/2) - 1 \end{aligned} \quad (2.3)$$

根據式子(2.3)所示，當 2 的基數演算法每做一次蝴蝶運算後，FFT 的運算單元數能夠除以 2，所以只要 N 步，就可以算到只剩下一個運算單元。比起 DFT 所需要的乘法次數更是少了很多，以公式表示其運算量為 $O(N \log_2 N)$ ，比(2.1)式少了很多，尤其當 N 很大時更是明顯。以上就是 FFT 的精髓，也是能夠快速運算的原因。

如圖 2.1 所示，類似一隻隻的蝴蝶而稱之蝴蝶運算法， N 與執行蝴蝶運算的步驟數與乘法數以及旋轉因子有關。經過位元反置後的 N 個輸入序列，第一步產生 $N/2$ 隻蝴蝶，第二步產生 $N/4$ 隻蝴蝶，依此類推，直到只剩下一隻蝴蝶即停止運算，運算完成的 DFT 結果為依

序 0 至 N-1 的 $X(k)$ 。

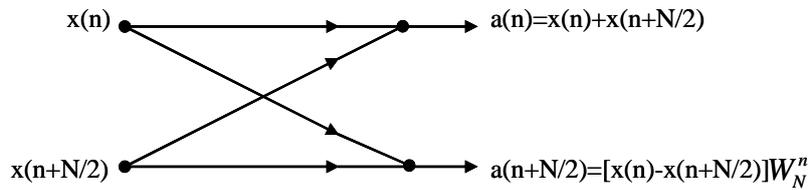


Figure 2.1 Butterfly Graph of Radix-2 DIF FFT

如果切割數 N 為 16，如圖 2.2，根據上述我們可以分為兩個 $N/2$ 位元蝴蝶運算，經過 $\log_2 N$ 次運算後就可得到我們完整的數值，其運算如下圖所示的訊號流程圖 Signal Flow Graph (SFG)[4-8]。

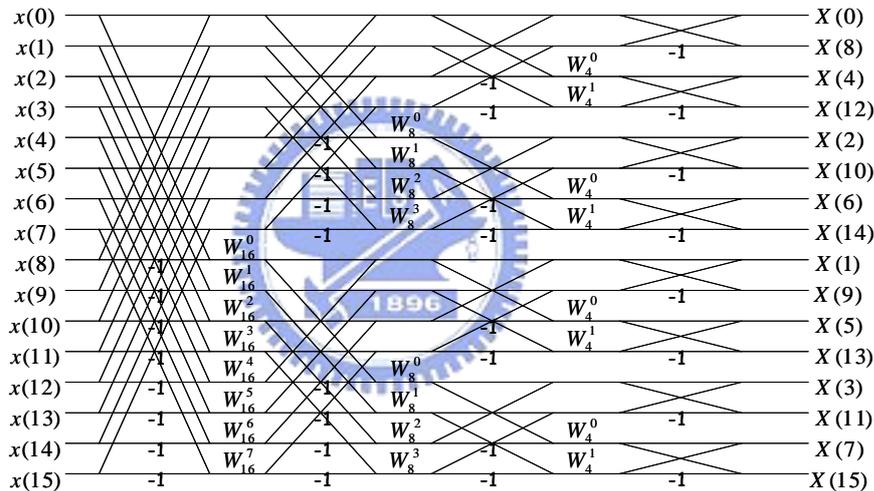


Figure 2.2 SFG of 16-Point Radix-2 DIF FFT

2.2 ARM Overview

1980 年 Patterson 和 Ditzel 兩位學者提出精簡指令集架構的概念，隨之在柏克萊大學的研究生依此理論基礎，設計出第一顆精簡指令集處理器 RISC。這個處理器遠比當時已經商業化的 CISC 處理器，來的簡單許多，在設計過程所耗費的功夫更是降低許多，從此處理器設

計方向便朝向這兩個大方向前進。RISC 與 CISC 架構不同的地方有幾點：首先是指令集設計上，RISC 架構的指令格式和長度通常是固定的（例如 32 位元指令），其指令的參數與定址模式只提供少數模式；CISC 架構下的指令集長度通常是可變的，而且可能會有多種定址與參數設定的方式。由於指令集的多寡與複雜度上的差異，讓 RISC 的處理器可以利用簡單的硬體電路設計出指令解碼功能，相對的透過 CISC 則會透過唯讀記憶體裡的程式碼來對指令做解碼。

其次，RISC 架構在設計上是一個載入/存回(Load/Store)的架構，資料運算通常針對暫存器，記憶體資料存取則透過其他的指令。為了加速程式運算，RISC 會設定多組的暫存器，而且指定特殊用途的暫存器。CISC 架構則允許資料處理的指令具備記憶體存取的功能，對指定暫存器的特殊用途需求上，相對的並不是那麼高。RISC 在指令集上，每個指令功能變的單純化，通常在簡短的一兩個週期可以執行完，因此可以允許管線作業(pipeline)，而 CISC 則因為指令功能與指令參數變化較大，執行 Pipeline 有較多的限制。當然，以上的介紹並非代表著誰是誰非，基本上要針對應用來衡量架構。同樣的應用軟體，CISC 的程式碼會比 RISC 來的小，而程式編譯器上面，RISC 也相對的必須提供較聰明的技術。另一方面 RISC 在實現上面，很容易可以縮小晶片的體積與降低功率消耗。

ARM 是一個典型的精簡指令集架構，在架構上承襲著柏克萊大學的 RISC 架構設計 (berkeley RISC)，主要有三個特徵，即載入/存回 (Load/Store) 架構、固定的 32 位元指令、與 3-address 指令格式。處理器設計上，指令通常至少會歸類成四個基本類別，即數學邏輯運算單元 (ALU : Arithmetic Logic Unit) 相關指令、載入存回指令、程式控制和跳躍指令、與其他指令。其中數學邏輯運算在一般應用裡大概佔了 30%，載入存回佔了 45%，程式控制和跳躍指令佔了 23%，剩下的 2% 屬於其他指令。通常在設計 RISC 處理器時，ALU 指令會盡量被要求設計在一個系統週期內執行運算完畢。但若是透過系統匯流排去讀取記憶體，則至少會耗費兩個以上的系統週期時間，對管線作業也會造成影響。因此，ALU 指令若不允許記憶體參考的運算子或參數，則比較容易達到這個要求。而這類運算架構，即所謂的載入/存回架構。ARM7TDMI 採用載入/存回架構，其第一個特點就是指令可以簡單化，32 位元固定的指令長度，含有運算子和運算元，而運算元最多可以達到三個稱為 3-address 指令格式。除了指令集特色外，ARM7TDMI 基本上還是屬於大家所熟知的范紐曼機器 (von neumann machine)。

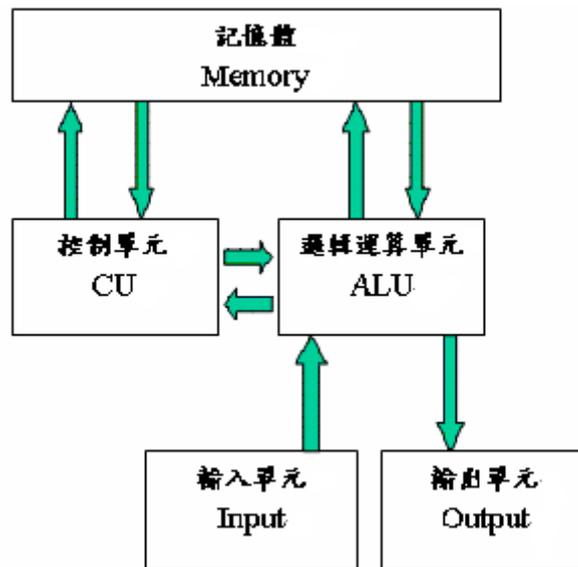


Figure 2.3 Von Neumann Machine

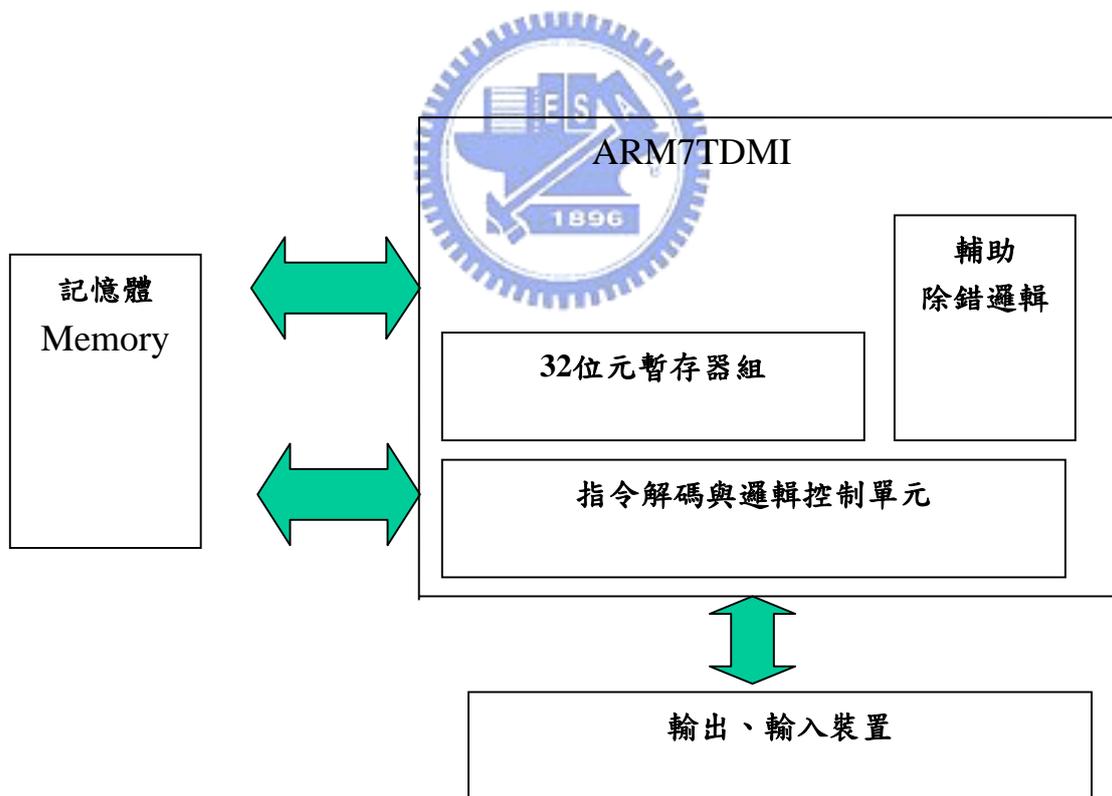


Figure 2.4 ARM7T Architectures

范紐曼機器如圖 2.3 所示，有五個構成組織，即記憶體、邏輯運算單

元、控制單元 (control unit) 、輸出與輸入設備。採取這種架構的處理器除了 ARM7TDMI 外還有 80x86，8051，6800，68000 等。通常 CPU 一個匯流排對某個記憶體區塊做資料存取需求，而整個系統記憶體在同一個時間內只能回應一個需求，因此造成了著名的范紐曼瓶頸 (von Neumann bottleneck) 。較先進的技術則會採取更高階的架構，來解決這個瓶頸。ARM7TDMI 處理器核心基本上也屬於范紐曼架構，圖 2.4 為 ARM7TDMI 所構成的一個嵌入式系統。在 ARM7TDMI 內部，主要由四大塊功能邏輯所組成：

- 32 位元的邏輯運算單元包含有基本的 32 位元加法器。
- 32*8 的乘法器、Barrel 位移器 (shifter) 、位址累加器等。
- 32 位元暫存器組則包括位址暫存器和資料暫存器、可程式的 31 個 32 位元暫存器與 6 個狀態暫存器。
- 輔助除錯單元、測試存取埠控制器與 Boundary Scan 暫存器。

另一方面，ARM7TDMI 只有一套位址匯流排與一套資料匯流排，對記憶體讀出或寫入的效率會有較大限制，尤其在管線作業下，通常一個指令的執行會被切割成幾個步驟，ARM7TDMI 為例，一個完整的指令週期分成三個獨立的步驟：

- 指令擷取 (fetch) ，即從記憶體讀入指令。
- 指令解碼 (decode) ，對指令解碼。

■ 指令執行 (execute) ，從暫存器送出資料、執行位移運動和邏輯運算、最後結果寫回目標暫存器。

另外必須注意的是，ARM 處理器利用一個程式計數器 (PC：Program Counter) 來記錄指令位址，其所記錄的位址就是目前被擷取的指令所在的記憶體位址，當執行 32 位元的指令，程式計數器每加一次要累加 4 個 Byte。

2.3 Real Time Operating System: eCos

Cygnus Solutions 於 1989 年成立，創始人為 Michael Tiemann, David Henkel-Wallace，和 John Gilmore。他們改進了 GNU 的 gcc、gdb 並把他們集合成在一起銷售。通過努力，其成果就是今天的 GNUPro 開發工具；也就是 eCos[13] 的開發工具。eCos 一開始就考慮到了嵌入式系統中內儲存資源的限制，以及嵌入式硬體平台的多樣性。它提供了硬體平台的抽象機制，並且具有高度的可配置能力，這使得它能適用於各種硬件以及不同級別的硬體配置。eCos 強大的配置工具可以顯著縮短產品開發週期。另外關鍵一點在於它是開放程式碼(Open source) 的開發工具和操作系統，這樣可以降低系統成本，這是嵌入式系統中一個非常重要的因素。用戶可以自由使用 eCos，但是對 eCos 的修改應該公布出來，這是為了提高或促進 eCos 發展的一種措施。

在所有處理器嵌入操作系統中，eCos 作業系統加上 ARM 處理器系列應用最廣。如圖 2.5 所示，eCos 嵌入操作系統包含一些標準的功能，例如中斷和例外處理(exception)，程序同步機制，和驅動程式等，這些也構成了 eCos 的核心組件，具體包括為：

- 硬體抽象層(HAL)：用來向上層提供統一的硬體結構，並統一各種不同硬體的差異。
- 實際核心：包括例外處理或者是中斷處理、程序同步機制、調度器、定時器、計數器。
- ANSI ISO C 和數學庫：標準 C 函數庫。
- 設備驅動程序：串列/並列，乙太網卡，Flash ROM，記憶體管理等。
- GDB 支持：使得目標可以和主機通訊進行逐步測試。

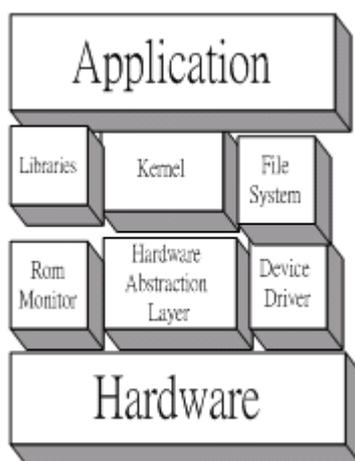


Figure 2.5 Embedded Software System of eCos

eCos 支持很多種平台，包括(ARM、Hitachi H8300、 Intel x86、 MIPS)等等。更換不同的硬體架構或者是平台，只需要移植 eCos 的 HAL 層，上層應用程式基本上可以不必修正或稍加更動即可。

2.4 硬體模擬平台

SID[14-16]它能夠以軟體來模擬硬體的機能，也就是說在軟體平台上去模擬硬體系統的行為。例如在 x86 linux 下模擬各種不同的平台微處理機，包括(ARM、Hitachi H8300、 Intel x86、 MIPS)..等等。當然 SID 的架構上的定義是使用軟體去模擬一個電腦系統，它模擬系統的能力，大到一個多處理器複雜系統，小到一個中央處理器的指令，SID 皆可以滿足使用者的需要。其核心組件包含有硬體模組、軟體控制和外部的介面。它能提供有力的工具發展軟體。

其軟體開發工具包含開發軟體過程所需的程式編譯器、程式除錯器、指令集模擬器，以及系統整合、驅動程式撰寫的時候所需要的硬體輔助工具，也就是用軟體在模擬下面硬體的行為，如 In Circuit Emulator (ICE) 和程式追蹤 (tracer) …等。我們最常見的硬體輔助工具就是 ICE，其連接在主機和目標平台之間，連接主機的介面有序列埠、並列埠、USB 或是乙太網路，連接目標平台的介面則通常是 JTAG，或 CPU 自訂的介面。不僅如此，此系統可以讓程式開發者可以設定中斷點以及單步執行，在執行時，如計數器的狀態、工作切換

狀態、記憶體狀態、暫存器狀態、變數內容等，同時透過功能強大的分析狀態、效率、及時序等。這些軟體開發工具過去通常會分開使用而且並不相容，但在許多介面標準定義下來後，SID 上的軟體開發工具已經可以做到非常不錯的整合，對軟體開發上頗有助益。

Linux [17-21]是一個以 Intel X86 系列 CPU 為平台，完全免費的與 UNIX 兼容的系統。Linux 是真正 32 位元多用戶多任務的操作系統。Linux 的起源最早是在 1991 年 10 月 5 日由芬蘭的赫爾辛基大學生 Linus Torvalds 撰寫了 Linux 核心程序 0.02 版開始的，但 Linux 的後續發展幾乎是靠網際網路上的溝通交流所完成的，所以其中沒有任何有版權問題的程式碼。本論文系統環境採用 eCos 系統及 SID 硬體模擬軟體，這個模擬的平台是在 Intel 2.53GHZ 微處理器及 1.0G Byte 記憶體 Rat Hat linux 作業系統下。

Chapter 3

Proposed Approach

在這個章節，我們建立一個新的架構，結合 ARM 微處理器及一個小點數的 FFT ASIC，並提供設計者一個高階的模擬環境，讓它很容易可以根據系統效能，估出 FFT ASIC 點數的大小。我們將在 SID 平台上介紹我們所設計的方式及流程，然後根據所訂定的流程，去做軟體及硬體加速器的快速傅立葉轉換 (FFT) 相互模擬及效率分析。

3.1 Our Approach

早期在設計一系統時，軟體開發者總是要等到硬體設計者將所設計之 ASIC 完成後，軟體開發者才能開始進行軟體的開發，這是由於系統設計者沒有一好的設計流程與方法來給予軟體開發者一明確的規格，或是軟體開發者沒有一可讓他們進行較完整的驗證與測試的平台，以致於軟體開發不知從何做起，而且也不太清楚硬體規格與動作情形。在這樣的結果下，若沒有市場上的求新求變的要求下，且沒成

本考量時，或許可以接受，但是以目前的清況來看，是無法接受的事實。當一個系統以處理器搭配程式碼作為實現的方式我們稱它為軟體設計，若是針對該系統的功能而去開發ASIC 則稱之為硬體設計。系統以軟體實現可以縮短研發時間，降低成本，可修改度高，與硬體設計相較之下顯得性能(performance)較差。反之，系統以硬體實現則需較長的研發時間，高成本，可修改度差，但性能與軟體設計相較之下顯得較高。當欲設計的系統對速度的需求較低時，整個系統以軟體設計較佳，如以軟體設計不能滿足系統的速度，那麼則需使用硬體設計。在滿足系統速度的條件之下，如何設計一個低成本系統，是我們追求的目標。我們以軟硬體共同設計(Hardware/Software co-design)的方式試著去達成目的。透過軟硬體共同設計(co-design)，我們可以得到一個折衷的方式，在設計成本以及性能(performance)之間取得一個我們所想要的平衡點。為了降低設計複雜度，硬體/軟體共同設計與驗證(Hardware/Software co-design)便成為了目前最受注目的解決方法之一。如圖3.1所示，為FFT軟硬體模擬器的組織及架構圖。我們結合ARM微處理器及一個小點數的FFT ASIC，再將功能逐一抽出改由硬體的型式來執行，以降低設計成本到最低又可達到系統的規格。此實驗的背景是採用SID硬體模擬器架構，以ARM微處理為核心去架構軟體快速傅立葉轉換(FFT)及硬體加速器，並根據使用者提供不同的限制，例

如，面積大小或者是運算時間的需求去做最佳的分析，提供設計人員一個設計時的參考。因此我們能提供數位IC設計者於快速傅立葉轉換(FFT)設計時，針對傅立葉轉換點數最佳化的選擇，提供一個方便的模擬環境，並根據軟體的建議，找到一個面積最佳化或者是運算時間最佳化結果，以達到系統之效能規格並且使用面積最小的ASIC為目的。



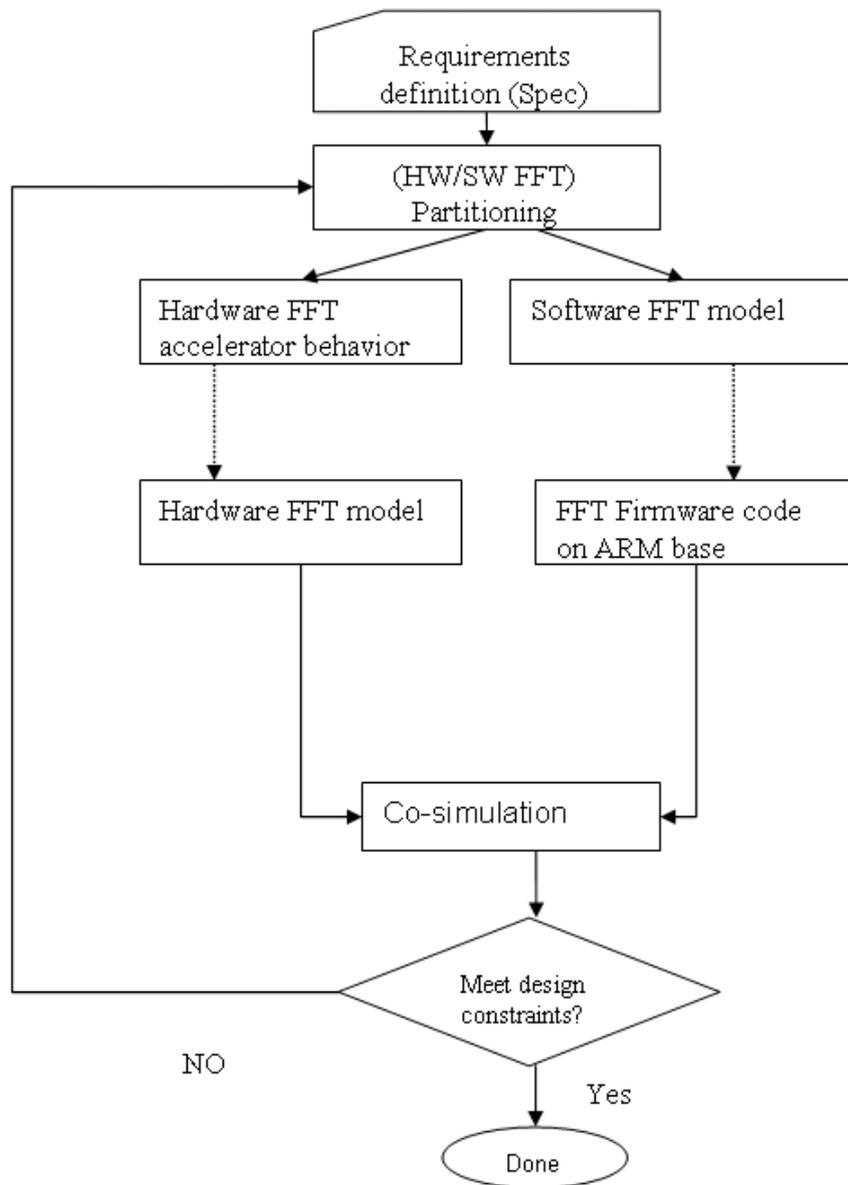


Figure 3.1 System Framework

一個簡易的 FFT 硬體最佳點數模擬器的執行步驟簡述如下，當我們輸入 FFT 的點數需求和基本的限制，例如面積的限制或者是運算時間的限制，然後去執行我們的模擬程式，最後將自動產生一個最佳化的 FFT 硬體加速器點數大小報告。

3.2 Hardware Modeling in SID Platform

我們選擇ARM為實際平台，最主要的原因是目前的ARM微處理器，所提供的性能與低耗電的特性相當適合於可攜式產品的應用及設計，並由於FFT在運算及資料量龐大上的需求所以我們建立以SID為平台，ARM為核心的模擬系統。如圖3-2所示，一個典型SID配置可以包含一個或多個微處理器 (ARM Processor)，並且內部還包含記憶體模組，N個串列或並列的輸出入埠及鍵盤、時鐘訊號、滑鼠、硬體加速器。其系統運作所需程式及資料儲存於記憶體中，CPU根據記憶體內程式動作，產生結果並輸出至輸出單元。至於硬體加速器通常用來專門處理資料路徑(data path)，類似共同處理器(co-processor)的角色。硬體加速器在應用上都是用來實現時間(timing)上很重要的函式(function)或區塊(block)，也就是這些函式(function)或區塊(block)，若以處理器來實現可能無法達到即時(real-time)的要求。它是一個以硬體設計為導向的方法(hardware dominated)，其硬體加速器在此平台上的實現方法，是由CDK(component develop kit)語法將整個系統全部描述出來，於下一段我們有詳細的介紹。在進行硬體/軟體分割時，先假設全部以硬體電路來實現，然後評估是否有符合系統的規格，若有符合且是所謂的over specification，設計者可以選擇一個適當的區塊(Block)分割給軟體執行，然後進行效能分析，分析是否還是符合系統所要求的規格，若還

是 over specification，則再分割一區塊到處理器以軟體來實現，這樣一直重覆的進行到剛好符合系統規格但又不 over specification 為止。在分割後，軟體和硬體的溝通則是選擇最容易實現且成本最低的通訊界面，然而必須保持系統規格中時間限制(timing constraints)。

其硬體加速器在應用上的操作模式是藉由位址分佈的方式，去區分每一個元件的使用位址。在匯流排部份，我們藉由使用者於 SID 元件設定檔的方式去區分每個元件的特性如 Master 元件或是 Slave 元件，因此在多個 Maser 情況下，這個微處理器元件為一個 Master，週邊的 RAM 則為 Slave，FFT 也可定義為一個 Master，它可以獨立於 ARM 處理器下去做記憶體和其他的資料搬移行為。所以當主要匯流排在做讀寫的動作時，這個仲裁者將針對各個元件的優先權進行讀寫順序的分配，並藉由位址的解碼而去讀各個特定的元件。

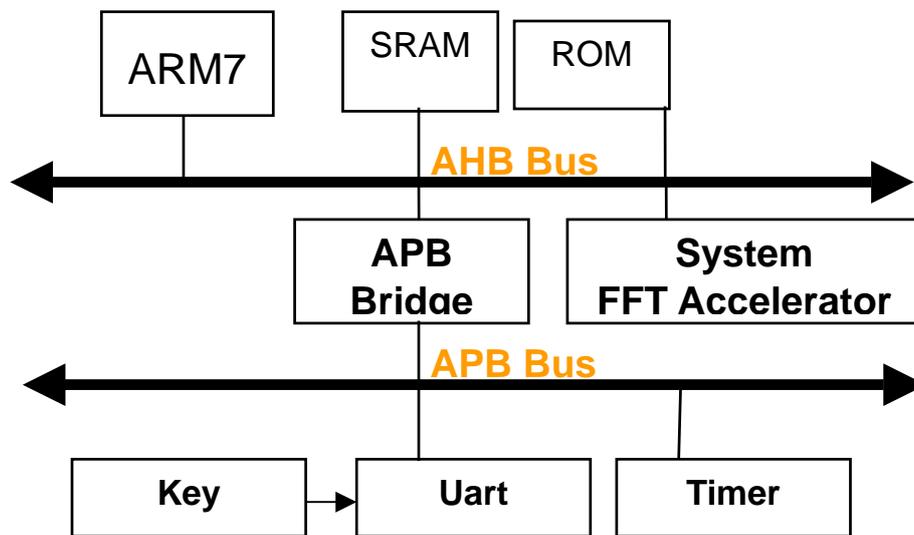


Figure 3.2 FFT Platform



如圖 3.3 所示，我們如何在 SID 平台上建立 FFT 硬體加速器，其設計步驟，說明如下：

步驟一:功能需求定義

這一步是發展一個元件去定義它功能的需求，如匯流排的界面和暫存器、輸出埠...等等。

步驟二:設計這個模組

我們將設計這個元件的溝通界面，如兩個元件間的端點連接，如輸出埠及端點。

步驟三:定義這個元件

我們將定義這個元件必要的輸出入埠連接方式及模型的暫存器的初始值。

步驟四:定義輸出入埠

這個 FFT 元件有兩個輸出控制接腳和一個資料匯流排，這二個輸出控制接腳 (control/status) 包含狀態旗標及控制旗標，資料匯流排將傳送或接收 ARM 微處理器與 FFT 加速器之間的資料傳遞。

步驟五:定義元件優先權時脈信號

一般來講我們在 SID 平台上，不需要定義時脈的訊號，但在此例 FFT 加速器我們需要知道運算時間，所以我們需要建立此基頻的時脈信號。

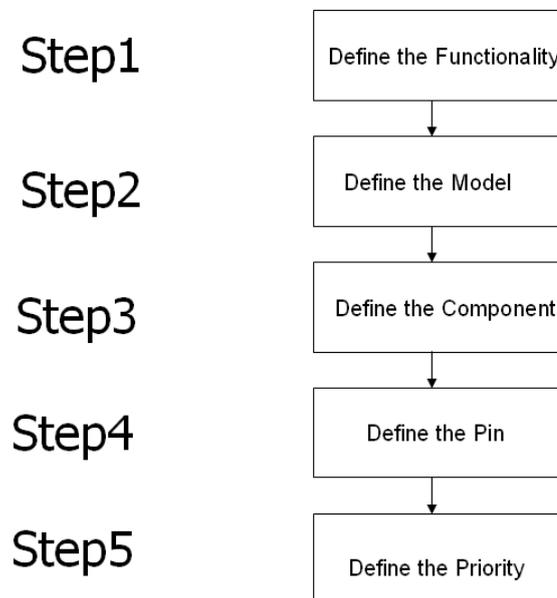


Figure 3.3 CDK Design Flow

完成上述步驟後，並進行儲存、編譯、下載然後執行。如圖3.4至圖3.6所示，當我們建立好執行檔後，下一步是確認此硬體元件的功能是否正常，這些工作需要先替嵌入式系統建構硬體，然後將執行檔的機器碼傳送到嵌入式硬體中。在本論文中所使用SID模擬器，它能夠提供非常有彈性的硬體平台，在這個平台上你就能夠得到嵌入式軟體的執行結果，而不必實際使用硬體測試。舉例而言，如圖3.5所示，當我們成功的建構好ARM target之後，接著要啟動除錯工作模式(debug session)和執行模擬器(simulator)，這個模擬器會定期更新畫面，便可同時觀察到CPU 的status、register內容與system memory 的內容，並且即時反應硬體實際的情況，必要時重複上述步驟，直到你滿意此軟體的執行結果為止。此外，模擬器還提供能協助偵錯的關鍵功能，例如支援紀錄程式碼的執行資訊，而這種能夠直接測量執行期間行為的能力，使得模擬器成為最關鍵的偵錯及輔助設計工具。

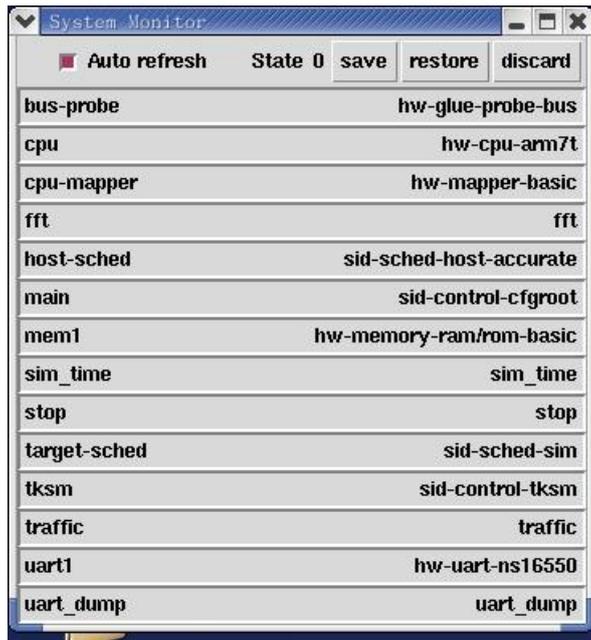


Figure 3.4 SID System Mointor

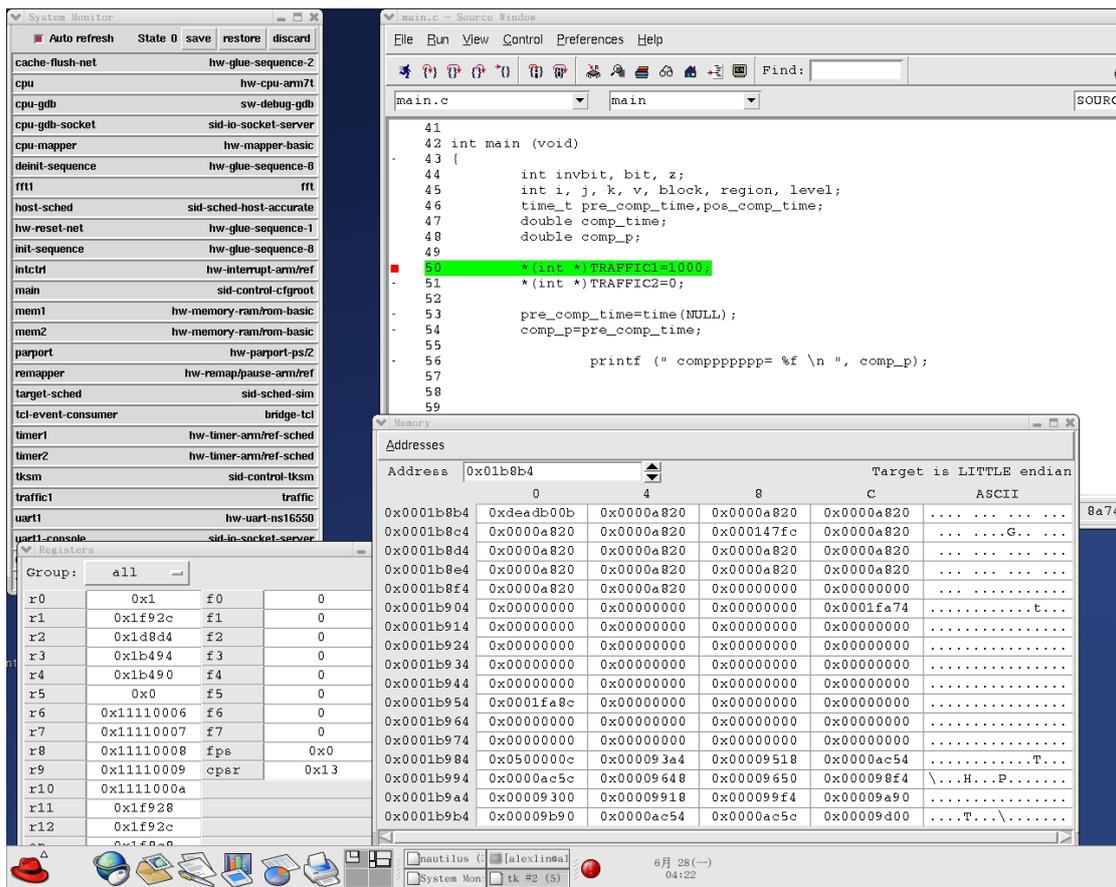


Figure 3.5 Trace Program

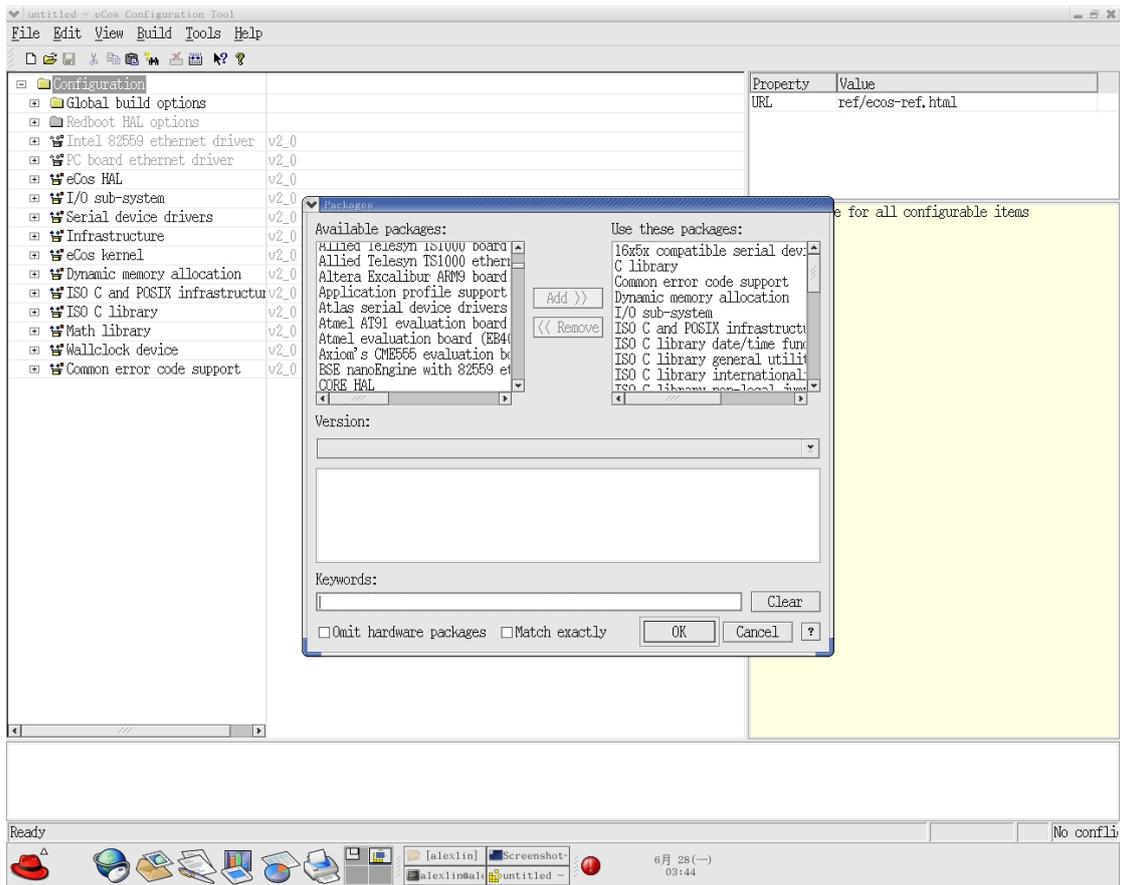


Figure 3.6 Simulation Tool

3.3 Sw/Hw Interface Modeling

在有限狀態機(FSM)模型中，我們以一組可能的狀態來描述系統行為，系統在任一時間只能處於一狀態；我們也描述可能的狀態轉移，這些轉移取決於輸入值，最後我們描述可能的狀態轉移時所發生的動作。由圖 3.7 和 3.8 中，我們藉由四種不同的硬體狀態及硬體暫存器，來描述我們軟硬體(SW/HW)溝通方式。首先我們設計了一個軟硬體溝通介面，並利用狀態機運行來詮釋其軟硬體間溝通的模式。在此軟硬體運作模型中，圖 3.7 左側為 FFT 軟體模型，圖型右

側是硬體加速器模型。軟體系統第一個狀態為系統等待狀態 (idle state)，第二個為系統傳輸狀態(transmit state)，第三個為系統接收狀態(receive state)；右側硬體系統第一個狀態為系統等待狀態 (idle state)，第二個為系統接收狀態(receive state)，第三個為系統運算狀態(compute state)，第四個為系統傳輸狀態(transmit state)。兩者由 Idle 狀態開始進行初始化動作，接下來軟體系統進入傳輸狀態，在這個狀態軟體會傳送資料給硬體 FFT 加速器，當軟體系統傳完資料後直接切入 receive 狀態，而且硬體 FFT 加速器在接收完資料後開始進行計算，等計算完畢，會通知軟體系統去抓取計算好的結果，軟體系統接收完資料後，就會通知硬體 FFT 加速器一起恢復至 Idle 狀態，整個系統即自動運行到下一個運算週期。

圖 3.8 為硬體 FFT 加速器狀態機描述，起始狀態為 Idle 狀態，此時 control 與 status 位元皆設為 0，FFT 加速器停留在 Idle 狀態；當軟體系統開始傳送資料時 control 位元會被軟體設定為'1'；硬體 FFT 加速器於是進入 receive 狀態並開始進行資料接收及進入 Compute 狀態進行運算，然後我們在硬體加速器內部有一個點數計數器，會根據軟體系統傳送過來的點數資料進行計算，當硬體加速器計算完畢，硬體加速器會將 status 位元設定為'1'；而軟體系統偵測 status 訊號為'1' 時，軟體系統即到暫存器內去抓取計算好的結果；抓完後

control 位元會被清除為'0'；此時硬體加速器轉移到 Idle 狀態並將 status 位元設為 0；而整個軟硬體且自動運行到下一個週期的運算；以上說明為軟硬體(SW/HW) 狀態機動作描述。這個詳細的暫存器配置說明如下：我們在加速器內動態配置有 $2*4*N$ 個位元組大小的暫存器，做為計算及輸出輸入資料的暫存區，其暫存器配置區分為實數及虛數部分，每筆資料需使用 4 個位元組，舉例而言如果有 1024 筆資料則需要 $2*4*1024$ 個位元組，去做為資料暫存區，其中包含有實數及虛數部分，另外有控制位元及狀態位元暫存器 (register) 被用於軟硬體間的溝通，也就是說當兩個系統建立連絡時，就交換預設控制信號或字元程序 (handshaking)。

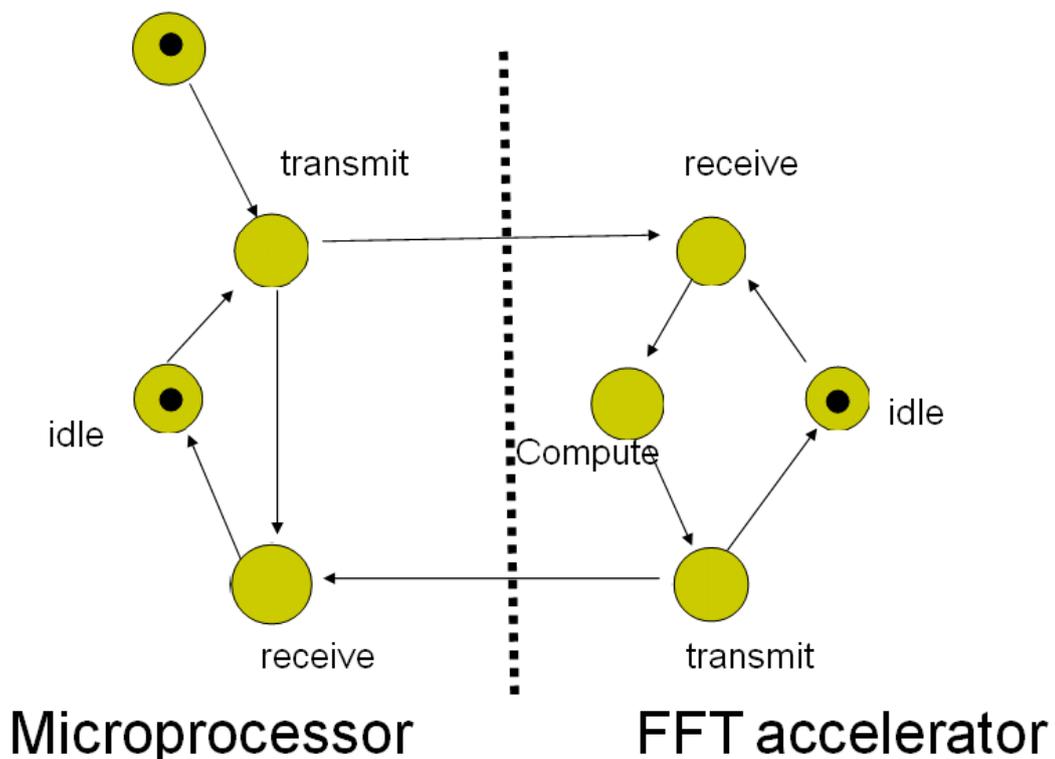
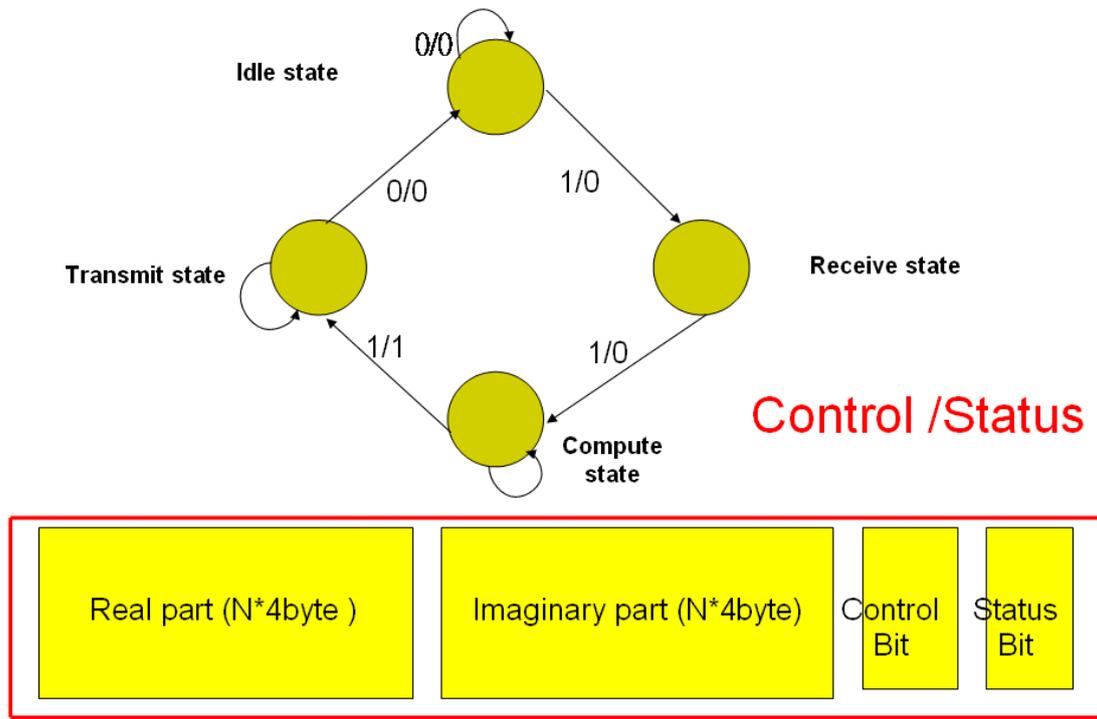


Figure 3.7 FFT SW/HW Model



22

Figure 3.8 FFT Accelerator Register/State Machine Model

3.4 FFT Software Pseudo Code

我們將簡單介紹下面軟體快速傅立葉轉換(FFT)的簡易程式碼，首先我們先要宣告硬體 FFT 加速器位址和幾個全域變數，這些變數方便我們待會於 FFT 計算時，當作記憶體來使用，並可於資料計算完畢後，可同時確認資料的正確性，第一步我們要輸入所需要的 FFT 點數及 FFT 的序列數據資料，並且將定義 FFT 區塊(block)大小及形態；第二步我們將根據這個 FFT 大小，讀寫 FFT 的資料然後再查表得到這個旋轉因子(twiddle factor)，並加入計算，上述中所有的旋轉因

子皆先行計算並儲存在表格中，以便節省運算時間；第三步我們將與快速傅立葉轉換(FFT)硬體加速器進行硬體溝通(handshaking)，並將快速傅立葉轉換硬體加速器(FFT accelerator)的點數及資料寫入匯流排(bus)中，並進行計算，當硬體加速器計算完畢，硬體加速器會將 status 位元設定為'1'，而軟體系統偵測 status_ready 訊號為'1'時，軟體系統即到暫存器內去抓取計算好的結果；第四步將等硬體加速器 (FFT accelerator) 計算完畢後，從匯流排中讀出資料；第五步將根據使用定義的點數，然後計算出我們需要經過幾次區塊運算，接著就會回到迴圈中再次執行上述的步驟，直到 block_size 小於 1 才會停止計算；第六步將根據使用定義的點數，然後計算出我們需要經過幾次運算，接著就會回到最上層迴圈中再次執行上述的步驟，直到運算次數達到 $\log_2 N$ 次才會停止計算；第七步將前述步驟中運算完畢的資料再進行位元反置(bit-reversal)運算。當整段程式執行完畢，我們可以在執行報告上紀錄一些可用的參考資料，例如運算時間或者是匯流排上的讀取次數，所以我們可以調整一下輸入點數或者是硬體加速器點數，然後重複上述的步驟再一次進行運算。

The general pseudo-code is as following:

```
#define FFT_DATA 0x500000C//Hardware address definition  
  
#define STATUS_WAIT 0
```

```

#define STATUS_COMPUTING 1

#define STATUS_READY 2//Hardware status definition

void FFT ( )
{
    Define block_size;

    Define a real part and imaginary part;

    Hardware data mapping;

    For n=1 to logN
    {
        while(block_size>1)
        {
            Put data to FFT hardware;
            Twiddle_factor lookup table;
            While(FFT_STAT!=Status_Ready)
            Wait until FFT computing complete;
            Get data from FFT hardware;
            block_size=block_size-1;
        }
    }

    Bit-reversal computing;
}

```

3.5 FFT Accelerator Circuit Design

在規定了硬體加速器的I/O 界面後，硬體設計者便可進行此硬體

加速器內部核心設計，首先是設計其主要的運算部份(Data path的部份)，也就其此硬體加速器最主要的內部行為(或功能)，接下來便是配合外界的環境設計其I/O 界面。本論文以Pipeline的方式完成FFT的硬體結構，以16點為例、每一階段運算所需要時間為11個時間單位(cycle)，而16點的FFT運算包含4個階段共需44個時間單位。在第一個時間週期內，讀取RAM及ROM內的資料並放置於暫存器一內，在第二個時間週期內，將暫存器一內的資料作乘法運算；然後放置於暫存器二內，第三個時間週期內；將暫存器二的資料依照順序寫入記憶體。執行傅立葉轉換運算首先將所需要的資料載入內建記憶體，然後進行傅立葉轉換再將轉換過的資料由記憶體輸出。我們的硬體FFT輸入與輸出的資料皆以17bit表示，負數則採用2的補數，執行傅立葉轉換所使用的radix-2 butterfly 乘法、包含有 $\log_2 N - 2$ 個乘法器和 $2\log_2 N$ 個加法器和 $N - 1$ 個移位暫存器。這個FFT硬體加速器可分為五個主要的區塊，是記憶體、位移暫存器(shift register)、乘法器、加法器(adder)和多工器(multiplexer)。如圖3.9所示在這個例子裡Radix-2 SDF是使用在我們FFT硬體結構中，其中FFT的點數 N 可以從8點至8192點進行模擬調整，我們使用Verilog去描述硬體的行為，在完成上列步驟後，我們將此電路區分為幾個獨立的區塊，經由Verilog-HDL[27]所描述之RTL-level電路透過邏輯模擬器(Logic simulator)進行功能驗證，而此RTL-level的電

路透過一test bench來驗證其功能上的正確性，當完成功能驗證後，接下來便是進行邏輯合成，將RTL-level Verilog-HDL 轉成Gate-level的Netlist。在這個實驗中，我們使用TSMC 0.25 um元件庫(cell library)和Synopsys 合成軟體(design compiler)去設計分析我們FFT加速器，最後可由Synopsys 報表中得到面積的參數，我們可以經由上列步驟，執行產生各種不同的點數的面積報表，方便爾後模擬時可對應查表得到不同的面積基本資料。

在圖中 3.10 中，顯示模擬的波形，包括 FFT 輸入信號、FFT 輸出信號、控制信號；在表格 3.11 中，詳細的定義所使用的介面及資料位元數，包括有 FFT 輸入信號、FFT 輸出信號、控制信號。

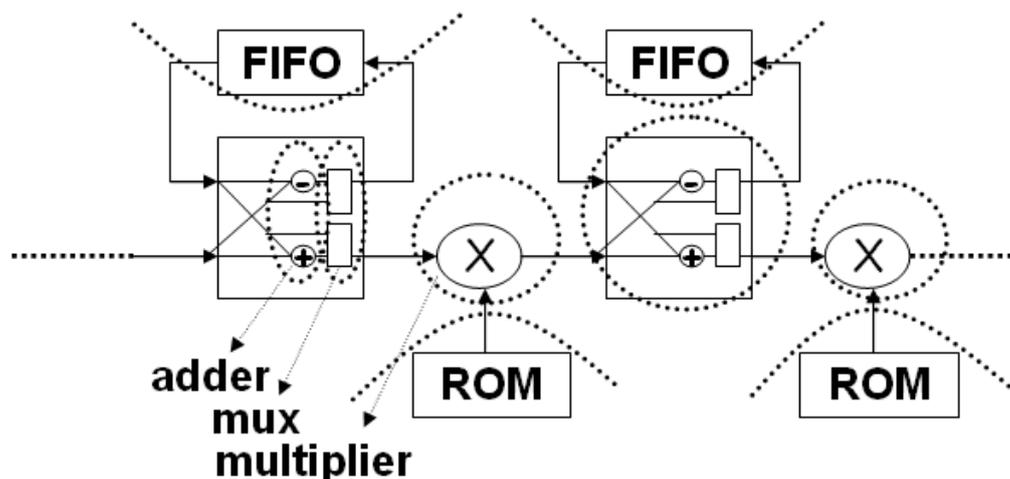


Figure 3.9 R2SDF Component

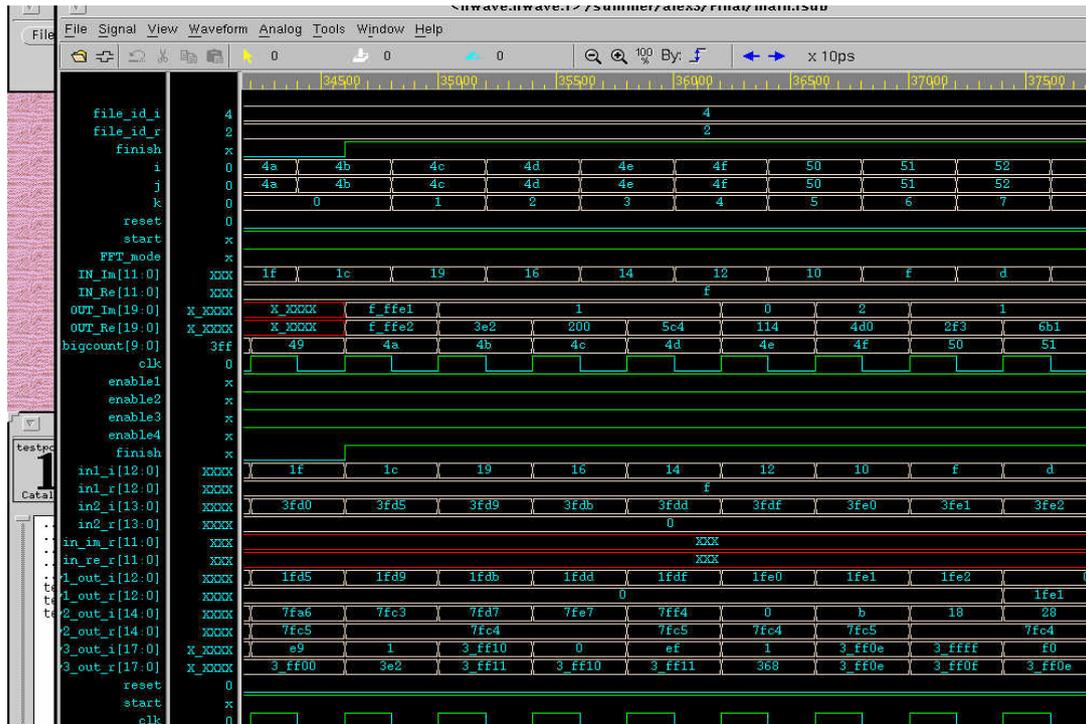


Figure 3.10 The Simulation Waveform

FFT Accelerator I/O Interface

| NAME | TYPE | Width | Description |
|--------------|--------|-------|-------------------------------------------------------------------|
| IN_re<16:0> | Input | 17 | Real component of the complex number input data, 16 bit wide |
| IN_Im<16:0> | Input | 17 | Imaginary component of the complex number input data, 16 bit wide |
| ENABLE | Input | 1 | Active high enable |
| RST | Input | 1 | Asynchronous reset, active high |
| Start | Output | 1 | Indicates the first the valid FFT data is on the output port |
| Finish | Output | 1 | Indicates the last sample on the FFT data |
| OUT_re<16:0> | Output | 17 | Real component of FFT data |
| OUT_im<16:0> | Output | 17 | Imaginary component of FFT data |

Figure 3.11 The FFT Accelerator I/O Interface

因為一個管線化作業的快速傅立葉轉換的程式碼，提供快速低成本的最佳化解決方案，也由於其管線化作業(pipelined)的緣故，我們可以執行多個動作，提昇其運算速度，並縮短執行時間。因此，我們使用管線化作業去架構我們 FFT 加速器。如圖 3.12 所示，左邊為資料輸入，右邊為資料輸出，我們定義 FFT 輸入當作第一級，緊鄰者為第二級依此類推，直到計算完最後一級即停止運算。我們在此實驗的背景是採用 DIF FFT 架構，當 N 從 8 點到 8192 點，我們使用 TSMC 0.25um 元件資料庫及 Synopsys Design Ware[22]去計算我們快速傅立葉轉換 (FFT) 加速器的硬體面積，然後利用表格 3.1 這個完整的 FFT 加速器面積模型，讓它很容易可以根據系統效能，計算 FFT ASIC 點數的大小，以達到系統之效能規格並且使用面積最小的 ASIC 為目的。

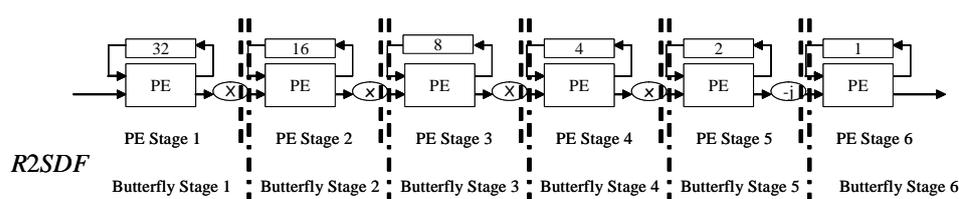
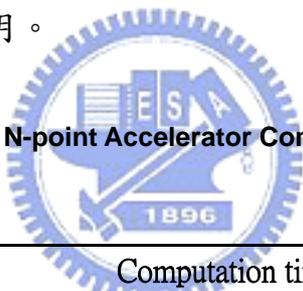


Figure 3.12 R2SDF Architecture (N=64)

如圖表格 3.1 所示，我們將比較不同 FFT 的點數所需要的運算時間，其中我們將硬體加速器的點數等於 FFT 運算的總點數，然後去估計整體的運算時間，看那部分的運算佔用了最多的運算資源。第一

行為變數定義及宣告陣列(array)所花費的運算時間，第二行為蝴蝶運算查表所花費的時間，上述中所有的旋轉因子皆先行計算並儲存在表格中，以便節省運算時間，第三行為 FFT 硬體加速器所花費的運算時間，至於列的部份，我們區分為不同的 FFT 點數所需要的運算時間，假如我們看 8192 點的部分，其中 FFT 硬體加速器所需要的時間佔整體 FFT 運算時間的極大部分，其他運算時間於軟硬體的溝通上或者軟體初始化的部分只佔小部分，所以我們可以簡單的推論，如果選擇適當的大小 FFT 加速器將可以提升整體的運算速度，在下個章節中我們將用數據去說明。

Table 3.1 FFT N-point Accelerator Computation Time



| Computation time Analysis | | | | | | | |
|---------------------------|-------------|--------------|--------------|--------------|---------------|---------------|---------------|
| (Cycle) | N=64 點執行週期數 | N=128 點執行週期數 | N=256 點執行週期數 | N=512 點執行週期數 | N=1024 點執行週期數 | N=4096 點執行週期數 | N=8192 點執行週期數 |
| Base Time | 210 | 220 | 240 | 280 | 520 | 840 | 1480 |
| Twiddle factor | 250 | 260 | 280 | 320 | 560 | 880 | 1520 |
| FFT Accelerator | 6289 | 13018 | 33224 | 71387 | 142895 | 574179 | 1416507 |
| Total | 6749 | 13498 | 33744 | 71987 | 143975 | 575899 | 1419507 |

Chapter 4

Experimental Results

在這個章節，我們將依照上節中的論點，經過實驗上的驗證呈現出完整的模擬數據結果。我們在此實驗的背景是採用 DIF FFT 架構，當 N 從 8 點到 8192 點，我們使用 SID 軟體平台去架構軟體快速傅立葉轉換及硬體 FFT 加速器，然後利用這個完整的模型，去模擬我們實際應用時所需要的硬體效能，並縮短設計所需要的時間。

如圖 4.1 中所示，我們取 64 點快速傅立葉轉換 (FFT) 為我們軟體 (FFT) 架構介紹，其中包含有控制邏輯單元及圖中一塊一塊的子單元是 8 點 FFT，我們需要一次又一次進行硬體溝通才能將 64 點的快速傅立葉轉換運算完畢。圖 4.2 中的硬體加速器是使用 DIF-Radix2 結構，左邊為資料輸入，右邊為資料輸出，我們定義 FFT 輸入當作第一級，緊鄰者為第二級依此類推，直到計算完最後一級即停止運算。

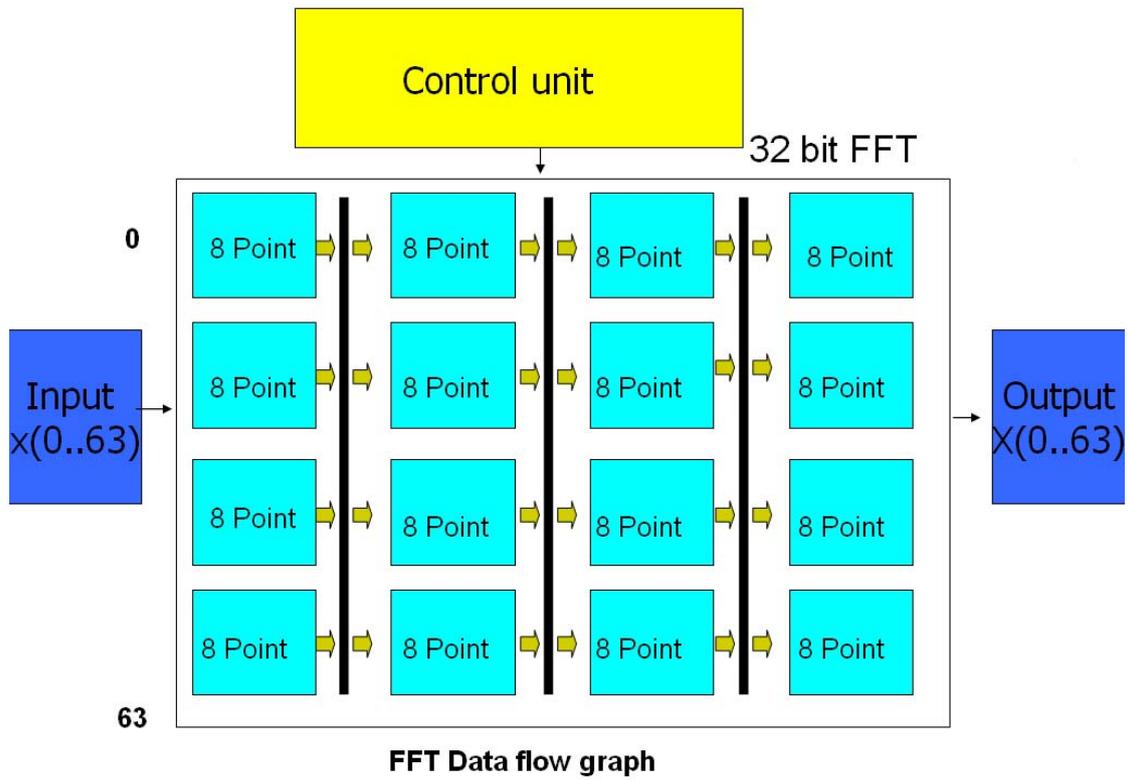


Figure 4.1 Dataflow for 64-Point FFT

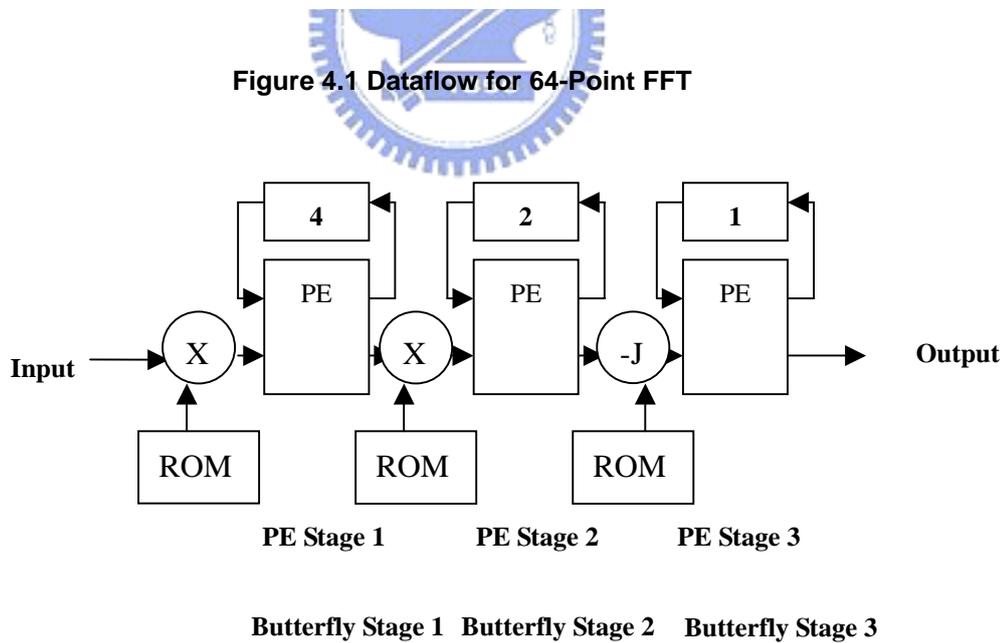


Figure 4.2 Simulation Structure of FFT Implementation

這個模擬的平台是在 Intel 2.53GHZ 微處理器及 1.0G Byte 記憶體及 Red Hat Linux 作業系統。首先我們開始說明我們的模擬環境，我們使用 TSMC 0.25um 元件資料庫及 Synopsys Design Ware[22]去計算我們快速傅立葉轉換（FFT）加速器的硬體面積，這個硬體加速器包含有加法器、乘法器、記憶體與多工器。在加速器元件結構上面，加法器使用 carry look-ahead 合成模組，乘法器是使用 Booth-recorded Wallace 合成模組，這個多工器使用 Synopsys 模組，然後我們透過 Synopsys 的設計分析器[23]（synopsis design analyzer）得到這個面積及耗電的綜合報表。

另一方面，這個記憶體模擬中包含有正反器及唯讀記憶體（ROM），當這個加速器（FFT）點數從 8、16、32 到 8192 點時，我們使用 Artisan 記憶體產生器去產生記憶體的模型而且選擇高密度雙固態記憶體（dual port SRAM）去模擬正反器，並且利用單固態記憶體（single port SRAM）去模擬唯讀記憶體（ROM）行為。

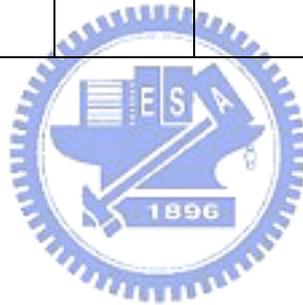
由於 FFT 整體運算時間為本論文研究的重點，我們以 4096 點 FFT 運算為例子，並且取硬體加速器的點數分別為 8 點、16 點、64 點，並以這三個部分相關性來對硬體加速器的效能來做一個比較。在表格 4.1 中為 FFT 整體運算所需的週期數，表中的週期數是由 ARM 的工作時脈 50MHZ 來計算，其中第一行表格的數字代表快速傅立葉轉換後的總點數，在第一列中 HW 數字代表硬體加速器所設計的點數，我

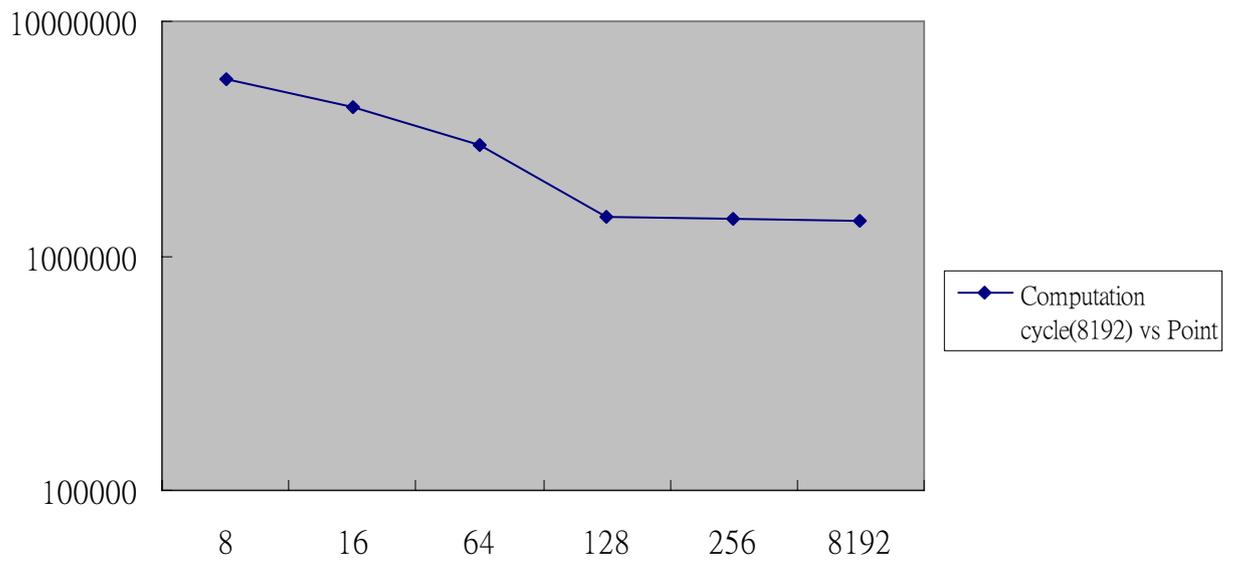
們用 N 來做說明，因此我們可以發現當 N 等於 8 與 N 等於 64 相互比較時，這個快速傅立葉轉換 (FFT) 整體運算時間將大幅度減少 50%，雖然這個模擬時間，將隨著這個 N 增加而增加，但也會在幾分鐘內模擬完成。在表格 4.1 中我們取第四行中純軟體運算時間的結果與 $N=64$ 點的硬體加速器效能來做一個比較，這個快速傅立葉轉換 (FFT) 整體運算時間將大幅度減少 90% 以上。事實上採用軟硬體共同設計之後之所以能夠比純軟體設計的執行時間節省，關鍵在於原本軟體運算太慢，造成執行上效能非常差。圖 4.3(a)(b)(c) 是 FFT 運算時間跟不同加速器點數比較的結果，我們能找到一些特點，例如當 N 在大於 128 點時，會出現比較沒有特別突出的結果，因為硬體間的運算時間，有一定的瓶頸，不可能無限制的提升效能，所以從此實驗數據中我們可以大膽的推論，當硬體加速器點數太大時，此運算時間將沒有明顯減少的趨勢；由圖 4.3(d) 中我們使用不同硬體加速器型態但同樣為 8192 點 FFT 運算，然後觀察軟硬間資料的傳遞是否成為此系統的瓶頸，我們發現了一個有趣的現象，從圖表中可以看出此系統於傳送和接收所花的週期數佔總週期數的比例並不多，所以我們可以說軟硬體的溝通的時間只佔 FFT 運算時間的極小部分。綜合上面實驗結果，我們以 8192 點 FFT 運算為例子，並且取硬體加速器的點數分別為 8 點、16 點、64 點，並以這三個部分相關性來對硬體加速器的

效能來做一個比較，假如我們選擇不同快速傅立葉硬體加速器轉換點數 (N) 和這個硬體所需要的面積做為我們的輸入變數，然後去估算整體快速傅立葉轉換的運算效能，我們能有一些特別的發現。例如：硬體快速傅立葉轉換加速器，當 $N=8$ 與 N 等於 16 或者是 $N=64$ 相互比較時，我們可以明顯的發現三者之間有極大的運算時間的差異，至於在單位時間的運算數量與硬體加速器面積比較結果，我們發現了一個有趣的現象，我們發現 $N=8$ 時將花費大量的運算時間於軟硬體的溝通上面。我們可以看到不同型態但同樣為 4096 點 FFT 運算，其運算時間有極大的差異，所以如果我們於實際設計前，事先選用適當的架構，將可以得到最佳的運算效能。由於本實驗採用 FFT 獨佔系統資源的方式進行模擬，所以軟硬體溝通的負載不太明顯，假如往後我們再設計系統時須注意小點數負載的問題。

Table 4.1 SW/HW FFT computation Time

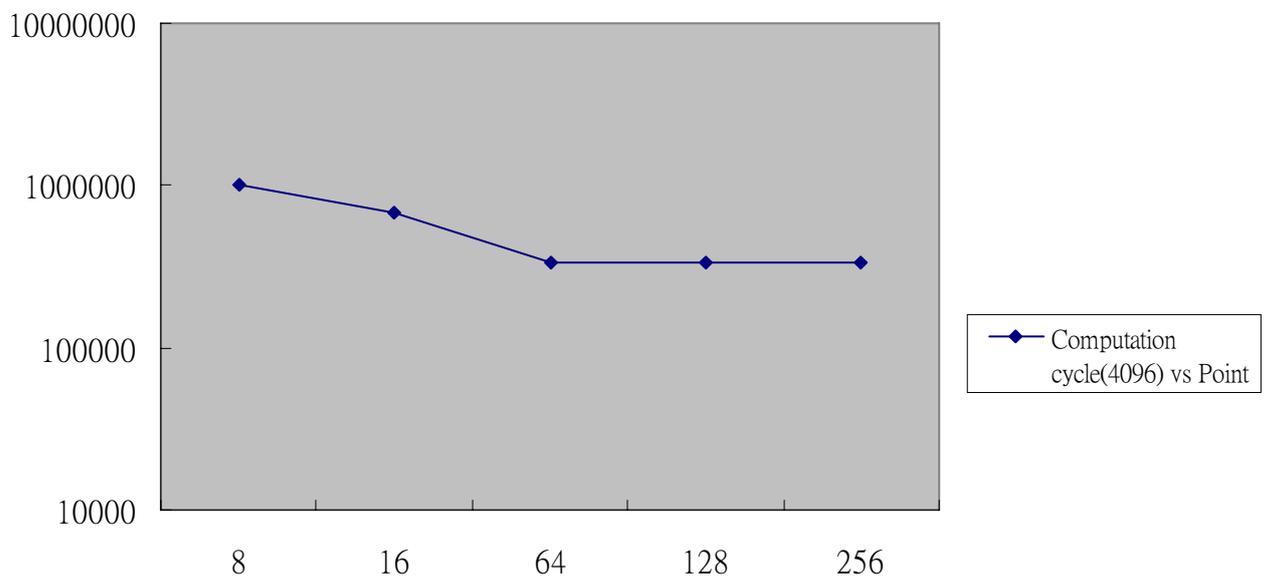
| SW/HW FFT Computation time comparison | | | | | | | |
|----------------------------------------------|-----------------------|------------------------|------------------------|------------------------|-------------------------|-------------------------|-------------------------|
| Cycle | 64 點執行 週期數 | 128 點執 行週期數 | 256 點執行 週期數 | 512 點執行 週期數 | 1024 點執 行週期數 | 4096 點執 行週期數 | 8192 點執 行週期數 |
| N=8 Area(258831um²) | 17,671 | 37,687 | 77,487 | 246,554 | 501,233 | 2,772,540 | 5,610,645 |
| N=16 Area(402693um²) | 7,582 | 16,122 | 78,425 | 161,149 | 331,246 | 2,112,225 | 4,294,097 |
| N=64 Area(733424um²) | 6,747 | 14,323 | 34,480 | 73,504 | 156,152 | 1,445,721 | 2,964,173 |
| Pure Software(SW) | 915,814 | 2,055,729 | 4,750,319 | 10,386,357 | 22,605,486 | 104,702,900 | 2,2405,334 |
| 1-(N=64)/SW Reduction rate(%) | 99.26% | 99.30% | 99.27% | 99.29% | 99.31% | 98.62% | 86.77% |





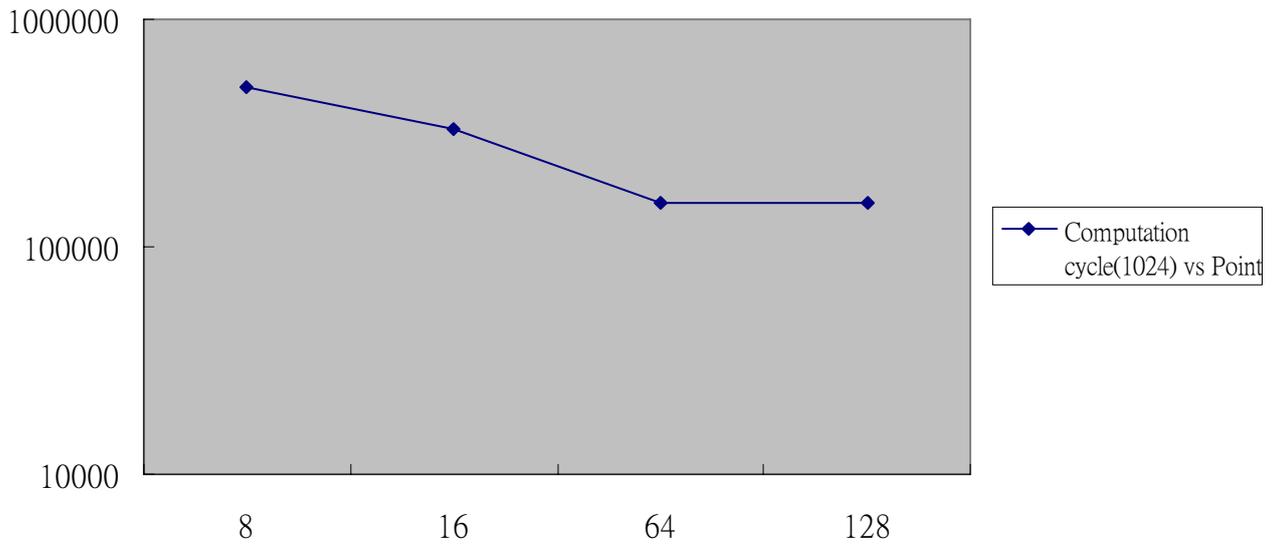
Cycle **FFT Accelerator Point**

Figure 4.3 (a) FFT 8192-point Computation Time vs. FFT Accelerator Point



Cycle **FFT Accelerator Point**

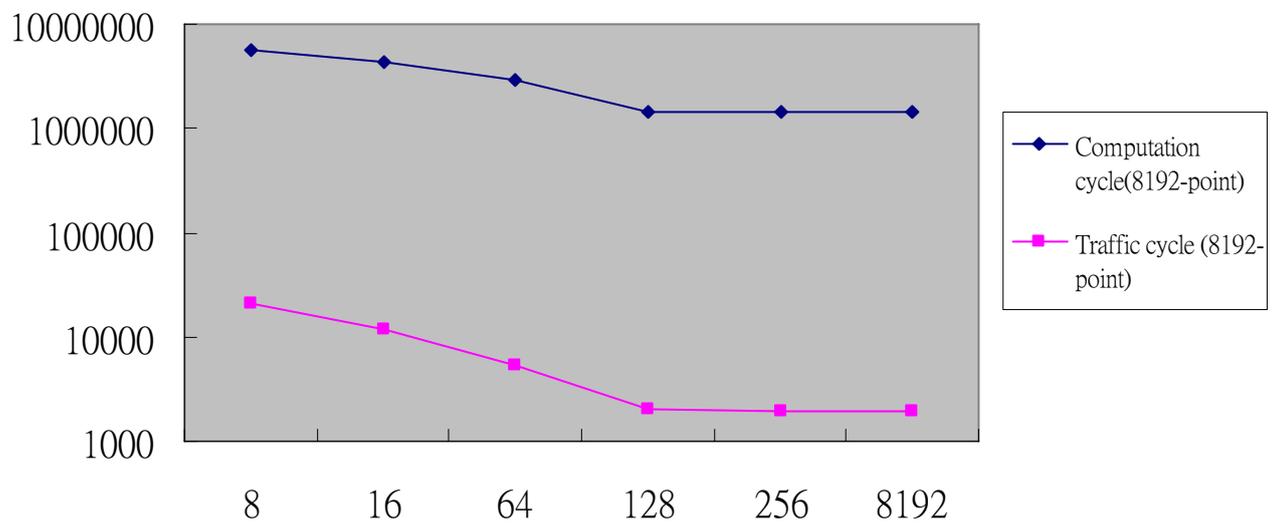
Figure 4.3 (b) FFT 4096-point Computation Time vs. FFT Accelerator Point



Cycle

FFT Accelerator Point

Figure 4.3 (c) FFT 1024-point Computation Time vs. FFT Accelerator Point



Cycle

FFT Accelerator Point

(d)

Figure 4.3 (d) Computation cycle vs Traffic cycle

Chapter 5

Conclusions

在此論文中，我們建立一個新的架構，結合 ARM 微處理器及一個小點數的 FFT ASIC，並提供設計者一個高階的模擬環境，讓它很容易可以根據系統效能，估出 FFT ASIC 點數的大小，以達到快速量產時程的最終目標。



在快速傅立葉轉換硬體架構下，我們建立了一種方法去做軟體與硬體間的模擬。我們使用 SID 硬體模擬器，以 ARM 微處理為核心去架構軟體快速傅立葉轉換(FFT)及硬體加速器，其中包含有作業系統與應用軟體的鏈結、微處理器與微處理器之間輸出入裝置、軟體快速傅立葉轉換和硬體加速器…等等。並根據使用者提供不同的限制，例如，面積大小或者是運算時間的需求去做最佳的分析，提供設計人員一個設計時的參考。因此我們能提供數位 IC 設計者於快速傅立葉轉換設計時，針對點數最佳化的選擇，提供一個方便的模擬環境，並根據軟

體的建議找到一個面積最佳化或者是運算時間最佳化結果。最後，我們將繼續朝著更快速，更精確的方向去努力。



Reference

- [1] J. W. Cooley and J. W. Turkey, "An Algorithm for Machine Computation of Complex Fourier Series", Math. Computation, Vol. 19, pp. 297-301, 1965.
- [2] Oppenheim, Alan V, and Schafer, Ronald W, "Discrete Time Signal Processing", Prentice Hall, pp.530-623, 1999.
- [3] Chao-Kai Chang, "Investigation and Design of FFT Core for OFDM Communication Systems", NCTU, Master Thesis, pp.31-35, 2002.
- [4] Shousheng He and Mats Torkelson, "A New Approach to Pipeline FFT Processor", Proceeding of the 10th International Parallel and Distributed Processing Symposium (IPDPS), pp. 766-770, 1996.
- [5] Shousheng He and Mats Torkelson, "Designing Pipeline FFT Processors for OFDM (de)Modulation", Proceeding of 1998 URSI International Symposium on Signals, Systems, and Electronics, pp. 256-262, 1998.
- [6] P. Dunamel, H. Hollmann, "Split Radix FFT Algorithm", Electronics Letters 5th Vol. 20, pp.3-10, January 1984.
- [7] E.H. Wold and A.M. Despain, "Pipelined and Parallel-Pipeline FFT Processors for VLSI Implementation", IEEE Transactions on Computers, pp.414-426, 1984.
- [8] L.R. Rabiner and B.Gold, "Theory and Application of Digital Signal Processing", Prentice-Hall, Inc, pp.73-130, 1975.
- [9] ARM Architecture Reference Manual, ARM Limited, pp.1-811, 2000.
- [10] ARM7TDMI Data Sheet, ARM Limited, pp.1-20, 2000.
- [11] GNU Pro Developers Kit, pp.1-10, 1990.
- [12] GNU Coding Standards, pp.1-50, 1990.
- [13] Embedded Configurable Operating Systems(eCos), pp.1-200, 2003.

- [14] Framework for Building Computer System Simulations (SID),pp.1-101,2003.
- [15] Simulator Component Developer's Guide(SID),pp.1-30,2002.
- [16] Simulator User Guide(SID),pp.1-28,2002.
- [17] RedHat 7.3 Customization Guide,pp.1-201,1998.
- [18] RedHat 8.0 Customization Guide,pp.1-300,2003.
- [19] RedHat 7.3 Reference Guide,pp.30-71,1998.
- [20] RedHat 8.0 Reference Guide,pp.21-63,2003.
- [21] Linux Kernel Documents,pp.11-230,1992.
- [22] Synopsys DesignWare, pp.1-501, 2003.
- [23] Synopsys Design Analyzer, pp.1-103,2002.
- [24] Artisan TSMC 0.25um Process High-Density Dual-Port SRAM (HD-SRAM-DP) Generator User Manual, release 1.0, June 2000.
- [25] Artisan TSMC 0.25um Process High-Speed Single-Port SRAM (HD-SRAM-SP) Generator User Manual, release 3.0, June 2000.
- [26] W. C. Yeh, "Arithmetic Module Design and its Application to FFT", PhD. Dissertation, National Chiao Tung University, R.O.C., Jul, 1, 2001.

Vita

Huang-Cang Lin was born in Taipei, Taiwan on Mar 18, 1973. He received the B.S. degree in Electrical Engineering from Chung Hua University in June 1996 and entered the College of Electrical Engineering and Computer Science, National Chiao Tung University in September 2000. His major studies were Electronic Design Automation (EDA) and VLSI design. He received the M.S. degree from NCTU in July 2005.



