國立臺灣大學電機資訊學院資訊工程研究所

碩士論文

Graduate Institute of Computer Science and Information Engineering
College of Electrical Engineering and Computer Science
National Taiwan University

Master Thesis

嵌入式計算機系統效能與功率分析之模擬環境設計

A System-Level Performance and Power Simulation Environment for Embedded Computer Systems

陳江睿

Chiang-Ruei Chen

指導教授：洪士灝 博士

Advisor: Shih-Hao Hung, Ph.D.

中華民國 97 年 7 月

July, 2008

國立臺灣大學電機資訊學院資訊工程研究所

碩士論文

Graduate Institute of Computer Science and Information Engineering

College of Electrical Engineering and Computer Science

National Taiwan University

Master Thesis

嵌入式計算機系統效能與功率分析之模擬環境設計

A System-Level Performance and Power Simulation Environment for Embedded Computer Systems

陳江睿

Chiang-Ruei Chen

指導教授：洪士灝 博士

Advisor: Shih-Hao Hung, Ph.D.

中華民國 97 年 7 月

July, 2008

誌謝

兩年的研究所生涯中，首先我要感謝我的指導教授洪士灝博士，洪教授悉心的教導以及豐富的學經歷帶領著我完成此篇論文，從題目的發想、問題的解決、報告與寫作上的表達等，皆給予我許多指導及寶貴的建議，使我在完成論文的過程中獲益匪淺；另外洪教授在生活經驗上的分享更是讓我有寬闊的視野，待人處世及面對生活的態度都是我學習的典範。此外，還要感謝口試委員郭大維教授、周承復教授、施吉昇教授以及林風教授在百忙之中抽空參加我的口試且不吝指教，使得論文更臻完善。

實驗室的大家也是這一段時間中重要的回憶。我要感謝博士班學長涂嘉恆，提供許多研究上的經驗以及處理問題的方法。還要感謝碩士班學長姐們的經驗傳承，讓我更快速跟上實驗室的腳步。同一屆的夥伴們：昶竣、俊文、承祐、豐旭、彧宏和冠儒，很感謝兩年來的幫助、陪伴以及包容，一起度過充滿歡笑和淚水的研究生涯，很開心可以跟你們一起學習和成長。還有實驗室的學弟妹們，感謝你們的幫忙與搞笑，實驗室裡有你們總是氣氛融洽。

最後我想特別感謝我摯愛的家人，父親、母親、姊姊和妹妹在求學生涯中不曾停斷的鼓勵與關懷，讓我無後顧之憂的學習與生活，可以恣意的在學海中發揮，點點滴滴的辛勤付出造就了今天的我。另外感謝女友郁雯兩年來的陪伴以及對我任性的包容。還有感謝許多朋友們的加油打氣，儘管身處異地都還是不忘關懷。你們的支持讓我每一天都充滿希望與動力，由衷感謝。

摘要

由於消費性電子產品的高度需求帶動了嵌入式系統的蓬勃發展，在快速導入市場的壓力之下，系統設計者在早期開發階段對於系統效能評估以及預測顯得特別重要。另外在這些電子產品中，可攜式裝置佔了大部分，系統功率消耗也成為了設計時期必須納入考量的因素。因此在本論文中，要探討的主題即為系統層級的效能及功率評估方法，透過評測分析應用程式的行為建立而成的應用程式模型及功率模型，放入根據目標平台參數所建立的硬體結構平台，進行交互模擬，提供一個快速的評估環境。我們使用 Mibench 作為目標應用程式，在效能部分，使用 Cycle-Accurate 的 ARM SoC Designer Simulator 當作比較對象；功率部份，我們著重在記憶體系統的功耗，並且使用 DRAMsim 進行驗證。從實驗結果得知，模擬速度平均約有 15 倍至 25 倍的加速，效能評估誤差率在 15%以下，功率評估誤差則在 6%以下。從結果來看模擬的速度有顯著的提升，未來將致力於更精準的應用程式及硬體結構模型建構，以及加入其他功率消耗元件。

關鍵字：*行為模擬、效能分析、功率評估*

Abstract

Due to the high demand of the consumer electronics, embedded systems generally have short product design cycles. To shorten the time to market, system designers need a way to quickly validate the performance for the entire system in the early design stage. For battery-powered devices, power consumption is also critical issue.

Thus, we propose a simulation framework to estimate the performance and power consumption of memory system jointly using behavioral models. Our results showed that our simulation scheme reduced simulation time by up to 25X , compared to the ARM SoC Designer, a cycle-accurate simulator, while the error was controlled under 0.67%, 8.16% and 11.78% for single-core, dual-core and 4-core systems respectively. For the power consumption of the memory system, the error was under 6%. Thus, our framework provides a significant improvement of simulation speed over the traditional method and still delivers reasonable accuracy for joint performance and power consumption evaluation.

Index Terms：*behavioral simulation*、*performance analysis*、*power estimation*

目錄

圖目錄

表目錄

第1章　序論

近年來，由於消費性電子產品以及相關的嵌入式裝置需求大量提升，帶動了嵌入式系統架構的蓬勃發展，從製程技術的進步到各式各樣推陳出新的微系統以及硬體架構，可看出系統單晶片是大勢所趨。這樣的系統隨著日益成熟的發展，系統效能與成本，以及產品開發時間往往是關鍵性的因素。對於硬體架構設計者而言，若能在開發的早期階段中快速得到效能估計，有便於決定設計方向，可能進而節省了不必要的錯誤開發成本及時間，由此可知快速的效能估計方法將成為開發階段重要的參考方針。然而，功率及能量的消耗速度，不僅關係到攜帶式電子系統的持續使用時間，在今日重視節能的趨勢之下，降低能量消耗，除了節省系統運作時的電力成本之外，還可以減少環境資源的損耗。

根據上述，我們了解到高計算效能以及低功率消耗將是系統單晶片的兩大需求，然而兩者在系統設計上往往是互相對立的，例如高效能的處理器雖然帶來快速的運算及處理能力，供電量除了與硬體的複雜度成比例，也隨操作頻率而提高，反之，架構簡單、低頻率的處理器所須的功率相對較低。這樣的現象也帶起了多核心架構的發展，多顆簡單的核心比起單一複雜的核心如多條處理超純量亂序結構（Wide-Issue Out-of-Order Superscalar）能夠更有效且節能的執行某些應用[1, 2]，然而如何在兩者之間取得平衡並做出適當的設計成為重要的課題。

為了在系統開發設計早期階段得以同時考慮效能與功耗這兩個關鍵因素，許多研究利用模擬來評估及衡量系統,這種技術在研究領域及產業界都被廣泛利用，主要的原因在於從發想、設計到實作出成品的程序複雜繁瑣並且耗時，開發製作成本大，又往往等到成品製作出來才進行軟體的開發或移植，一旦效能不符合原先預期，又必須重新一個設計製作循環，不僅人力物料成本的浪費。精確且快速的模擬在早期開發階段提供了設計者使用較小的成本完成上述流程，並且方便能

夠在廣大的設計空間中找尋出較適合的集合。然而，隨著軟硬體逐漸複雜化，模擬的複雜度也提高不少，如何維持精確並且具備速度的模擬流程，成為許多研究者致力的目標。目前大部分研究使用行為層級的模擬方法[3, 4]，將系統元件包括軟體、作業系統到硬體結構等抽象化，因應低階電路或結構模擬所產生的模擬複雜度，提升整體模擬速度，然而目前多半研究都專注於特定程式或是單一結構領域的模擬[5, 6]，又或是單純考慮效能或功率，軟硬體整合並且考慮全系統因素的效能及功率模擬架構並不多見。

有鑑於此，我們設計出一個可供系統開發者使用的工具及方法，以系統行為層級模擬的方式，在系統尚未完全產生之前，快速評估目標系統的效能及功耗；在效能方面，可以預測架構上的改變所帶來的效能影響，供給硬體設計開發者做為評估與發展的依據。在功率上，我們專注在記憶體模組的量測，讓開發效能之際同時得到記憶體模組的功耗資訊，了解應用程式行為所產生能量消耗的情形。

同時，我們提出的工具及方法，可以讓軟體、硬體以及系統設計者都可以用相同的語言建構各自的規格與模型，達到軟硬體協同設計及模擬的目的。除了在快速導入市場的需求之下，讓系統設計抽象化，成為一個初步的效能及功率評估工具之外，更可讓軟硬體的細部設計或原始碼得以保持隱密，使得合作者之間不必顧慮機密外洩的問題。

本論文之其他章節安排如下：第二章開始探討及分析相關的模擬方法及研究。第三章說明本次模擬方法及環境，透過 SystemC 建置軟體模組與硬體模組，進行協同模擬，並且說明該模擬環境的速度與準確性。第四章則介紹如何在模擬環境中建置記憶體功率模組以及函數功率資料庫，在效能模擬的同時也可以提供記憶體功率等資訊。第五章是相關實驗方法介紹，主要針對記憶體功率模組的準確率提出結果及驗證。第六章則總結整篇論文及未來展望。

第2章　背景及相關研究

第一章曾提到開發者經常利用模擬的方法來評估衡量系統，模擬即利用軟體程式來仿真一個系統元件的行為，模擬器的開發是諸多研究學者的重點，至今已出現許多具代表性的模擬器，例如單一處理器效能模擬器中的 simplescalar[7]，它模擬著重在單處理器架構上的行為，如指令層級平行度、分支預測、管線結構、快取等；多處理器模擬器如 Rsim[8]，提供共享記憶體多處理器架構中的記憶體系統效能資訊，亦能模擬快取一致協議（Cache Coherence Protocol）；上述兩者屬於專精於部分元件的模擬器，另外還有全系統處理器如 Simics[9]，它提供系統中更多的周邊元件模型，包括處理器、記憶體、磁碟、網路、匯流排、作業系統等，這類的模擬器將整個系統元件的交互影響納入考慮，可以提供須全系統評估的設計者充分的效能資訊，但是這類的模擬器也面臨同樣的限制，就是模擬的速度緩慢，對開發者造成一些限制；另外還有一些暫存器傳輸級（RT Level）的低階模擬，利用低階的硬體描述語言如 Verilog 或 VHDL 寫出目標硬體，然後加入時脈訊號進行週期模擬，這樣的低階模擬方式提供了相當的準確度，卻也有模擬速度緩慢的問題。

從上述的相關研究，我們了解若需要對系統做全面性的評估衡量，就必須將系統中相關元件加到模擬環境中，另外在模擬速度上如果能加以提升，對於早期開發者而言將方便許多。SystemC 即為了達到快速的軟硬體系統層級模擬而發展出的語言，它是一種基於 C++的高階程式語言，透過高階的語言抽象化描述低階硬體元件並仿真其行為，設計者可以遵循公開的函式庫自行定義目標系統中的周邊元件，達到系統層級的快速模擬， 2.1 中會介紹 SystemC 的歷史及相關研究。

另外，近年來嵌入式系統的應用逐漸延伸到可攜式的消費性電子產品，在設計的考量上，除了效能之外，功率相關議題的重要性也逐漸提升，各種軟硬層級的技術導入，舉例來說，許多處理器使用了 Dynamic Voltage Frequency Scaling

（DVFS），即結合了節能演算法以及動態調節技術；記憶體模組或是儲存磁碟在硬體架構上直接支援了多種運轉模式，依照存取頻率等作為依據來轉換運作模式。為了提供這些技術的改進與研究，功率模型的製作及模擬也漸漸的成為大家探討的重點之一。在 2.2 中，也會討論一些功率模組的建置方法及相關研究。

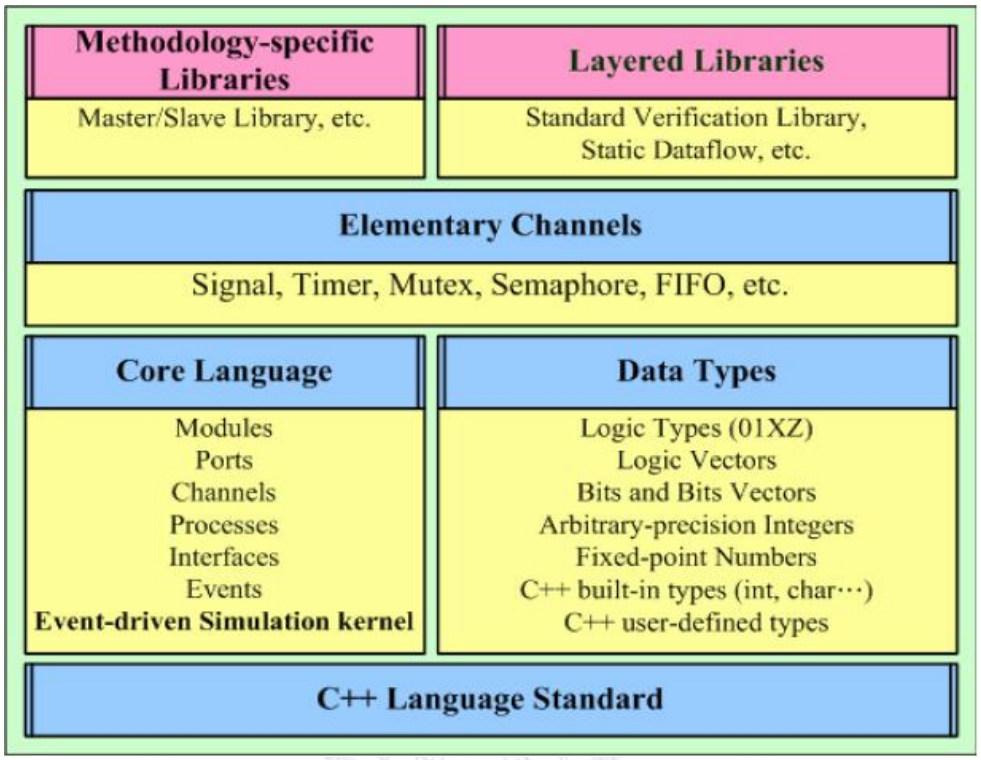
2.1 SystemC

SystemC[10]是 1999 年 9 月時由 Cadence Design Systems, Inc.、CoWare、Fujitsu、ARM、NEC、Motorola、Synopsys,Inc.等組成的 OSCI (Open SystemC Initiative)組織所訂定的硬體描述語言。OSCI 於 2000 年 3 月推出了 SystemC 1.0，加入了支援硬體模擬用的建構元件(Constructs)，如埠(Ports)、訊號(Signals)及硬體資料型別等，所以此版本主要是做硬 體 RTL 及行為(Behavior)層次的設計。到了 2002 年 4 月推出了 SystemC 2.0.1，新增通道(Channels)、介面(Interfaces)及事件(Events)等系統層級模擬用類別。而最新一版的 SystemC 2.1 於 2004 年 10 月推出，它可以讓使用者自己擁有自己的 main() Funtion 而加入 SystemC 程序，如 sc_main(argc, argv)、支援動態程序、多了 sc_event_queue CLASS 可以 notify 更多的 event、支援 MacOS X 作業系統與 POSIX THREAD。

以 C++為基礎的 SystemC 系統設計語言，包含適合描述軟硬體及系統各部份功能的類別函式庫(Class Library)，能在軟硬體間做交互運作溝通(Communication)，透過事件驅動的模擬核心，建構整個設計的階層關係。圖表 2-1 為 SystemC 語言的架構圖，C++ Language Standard 上的即為目前 SystemC 開發至今的核心、資料型別及各種類別函式庫。

另外，SystemC 還能描述不同抽象層次的設計，當系統規格功能改變時，將能夠快速修改並加以驗證，也能夠做系統的效能分析並選擇較佳硬體架構。學姊張筱薇的論文[11]即利用此特性，使用 SystemC 建構一個初步的系統效能分析模擬環境，改善了軟硬體複雜度提升，設計時間拉長以及模擬速度趨緩等問題，模擬方

法從較精確的週期精準指令層級模擬發展至系統層級的行為模擬，透過抽象化（Abstracting）及模型化（Modeling）的概念，達到提升模擬速度的目的，[5, 6]的研究即致力於模型化特定應用程式以及特定平台，這部分將在第三章中做詳細的介紹與說明。



圖表 2-1 SystemC 語言架構

2.2 功率模組相關研究

功率模型化及分析估量部分，從[12, 13]中，可以大致分兩大層級：

1. 較低階層級例如邏輯閘層級（Gate level）或暫存器傳輸層級（RT level）的評估模型，這個層級的模型化通常需要各種元件包括邏輯閘、迴路等基本的電壓電容資訊，並透過訓練得知在其元件之上的活動（Activity）例如電路開關或多工器使用所會產生的對應功耗，建構出一個分析模型（Analytical Model），此方式會得到較精準的功率預測，但模擬時間也較長。
2. 另一個則為系統層級的行為模擬量測，這個方式抽象描述系統行為（Behavior），

利用評測工具追蹤出應用程式中會產生的活動，例如有追蹤程式中有多少算術指令[14, 15]，並且估量出處理器中執行這些算術指令所須的功耗，經由這樣的功率消耗模型及監控元件追蹤，最後分析加總算出總功耗，這樣的方式雖然準確度較低，但相對的準確度卻是可信的，實作起來成本也較低。同樣也有一些研究是基於指令集模擬器本身所擁有的特性進而整合具有功耗評估能力，例如Wattch[16]、SimplePower[17]、Sim-Panalyzer[18]。另外，巨集化模型（Macro-Modeling）[19-21]也是常用的高層級功耗模型，利用較低層級量測出的功耗資訊或是第三方工具本身所帶有的能力，得到所需的功耗資訊，採用預先特徵化模型並描述之，並以較高層次的系統或方程式概念型式提供能耗估量。這種做法經常運用於嵌入式系統各式各樣的元件中，主要是因為大部分的嵌入式系統元件行為較固定，便於歸納出巨集動作。

我們的功率模型比較類似第二種作法中的巨集化模型，對程式中的函數特徵化並進行功耗評測。從諸多系統層級的模擬分析估量結果得知，雖然這些功耗分析準確度不如較低層級，但速度上卻快上數十倍甚至數百倍，相對誤差也在容忍範圍之內，在早期開發階段，足夠提供設計者作為設計參考依據。

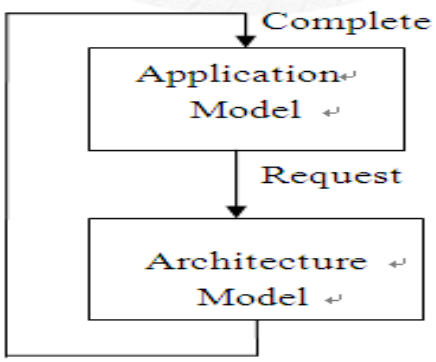
綜合以上，我們針對嵌入式系統的特性，並基於[11]提出的框架，針對記憶體系統將行為層級的功率模型加入，延伸出快速的軟硬體整合系統層級效能及功率模擬環境。下一章將介紹這個系統層級模擬框架[11]。

第3章　SystemC 模擬環境介紹

本章我們將介紹初步建置出來的 SystemC 系統層級模擬環境，以及它相關運作模式與架構，並說明模擬環境之可行性與優勢。

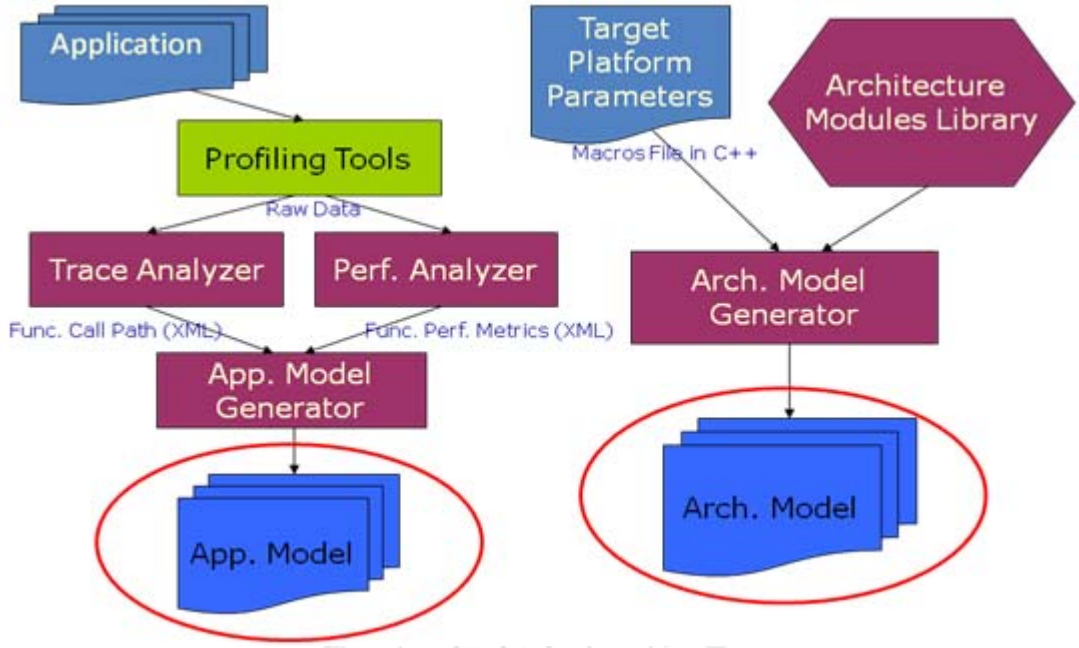
3.1 模擬環境及架構

在軟硬體開發早期階段，提供快速的效能量測與粗估是很重要的，尤其時常需要評估特定應用程式跑在特定平台上的效果。我們了解到應用程式的行為特性常常可以代表該應用程式，並且利用其行為特性達到模擬該應用程式的目的，這樣抽象化的描述降低了模擬設計的複雜度。同樣的硬體結構也有相同的特性，具有一些參數來代表該硬體結構的特性，例如頻率、頻寬、延遲等等，將之抽象化再搭配抽象化之應用程式模型來做行為層級（Behavior Level）的模擬，比較起指令層級模擬（Instruction Level）甚至是暫存器傳輸層級模擬（RTL Level），行為層級模擬在準確度上雖不及其他較低層級的模擬，但是卻能省去低層級模擬必須花費的冗長時間成本。如何建構出一個快速的模擬環境並且提供一個合理範圍之內的準確度來滿足早期開發之需求是本章的重點。有鑑於此，我們利用獨立的應用程式模型（Application Model）與結構模型（Architecture Model）來表示各自的行為模式以及狀態，仿真應用程式與硬體結構，並利用訊號埠將兩者連結，達到協同模擬的效果，如圖表 3-1 所示。



圖表 3-1　模型驅動模擬架構

整個模型的建置流程如圖表 3-2，接下來的下兩節將介紹應用程式模型與結構模型的建置細節以及如何達成模型驅動模擬。



圖表 3-2　應用程式及硬體架構模型建構流程

3.1.1 應用程式模型

要簡單的建立一個應用程式模型，我們必須擷取出該應用程式的行為，例如函數呼叫路徑、迴圈次數、以及相關的效能資訊等等，利用這些資訊抽象化應用程式並且透過轉換的方式產生代表應用程式的模型。在這裡我們建立一個函數層級（Function Level）的應用程式模型，將整個應用程式拆解成相互呼叫的函數，在這條件之下，我們須蒐集函數呼叫路徑（Function Call Path）、以及各個函數平均的所經歷的週期數、指令數、記憶體讀寫數等效能單位資訊(Performance Metrics)。要蒐集這些資訊須對應用程式作深入的評測，可以透過 Intel Vtune Performance Analyzer[22]、Sun Dtrace[23]、GNU gprof[24]或是 Valgrind[25]等評測工具(Profiling Tools）取得，在這裡我們使用 ARM SoC Designer Simulator[26]的軟體評測工具，並將得到的應用程式資訊紀錄成 xml 格式如圖表 3-3，以便於產生 SystemC 模型。

```
<path>
<path_id>1</path_id>
 <frequency>1</frequency>
 <func_count>2</func_count>
 <functions>
   <function>
     <func_name>__Heap_Alloc</func_name>
      <count>1</count>
        <metric>
         <cycles>66</cycles>
            <mem_read>18</mem_read>
            <mem_write>16</mem_write>
            <inst_count>70</inst_count>
        </metrics>
  </function>
  <function>
    <func_name>__Heap_DescSize</func_name>
      <count>2</count>
         <metric>
          <cycles>5.5</cycles>
            <mem_read>0</mem_read>
            <mem_write>0</mem_write>
            <inst_count>6</inst_count>
          </metrics>
   </function>
 </functions>
</path>
```

圖表 3-3　以 xml 格式紀錄應用程式函數行為與效能資訊

我們以 xml 的方式儲存效能資訊，主要是因為 xml 格式的可擴充性以及結構化的表示方式，透過這樣的中介標記語言，日後便於我們的維護以及有效發揮資訊利用的彈性度。有了這些函數層級的效能相關資訊，接下來任務即利用這些資訊，將其打造成一個個函數任務區塊（Task Block），並建構成可執行的 SystemC 應用程式模型，這個應用程式模型仿真了每個函數的效能行為，包括上述所提的

週期數、記憶體讀寫數等，並且也同時仿真呼叫的路徑，達成應用程式行為的模擬，最後再透過 start 的函數啟動了應用程式的執行。資訊儲存於 xml 經轉換後產生的 SystemC 應用程式模型範例如圖表 3-4。

```
void thread_test::__Heap_Alloc(){
    cycles=(int)66;
    inst_count=(int)18;
    mem_read_count=(int)6;
    mem_write_count=(int)3;
    wait(suspended);
    __Heap_DescSize
    }
void thread_test:: __Heap_DescSize(){
    cycles=(int)5.5;
    inst_count=(int)0;
    mem_read_count=(int)0;
    mem_write_count=(int)6;
    wait(suspended);
    }
```

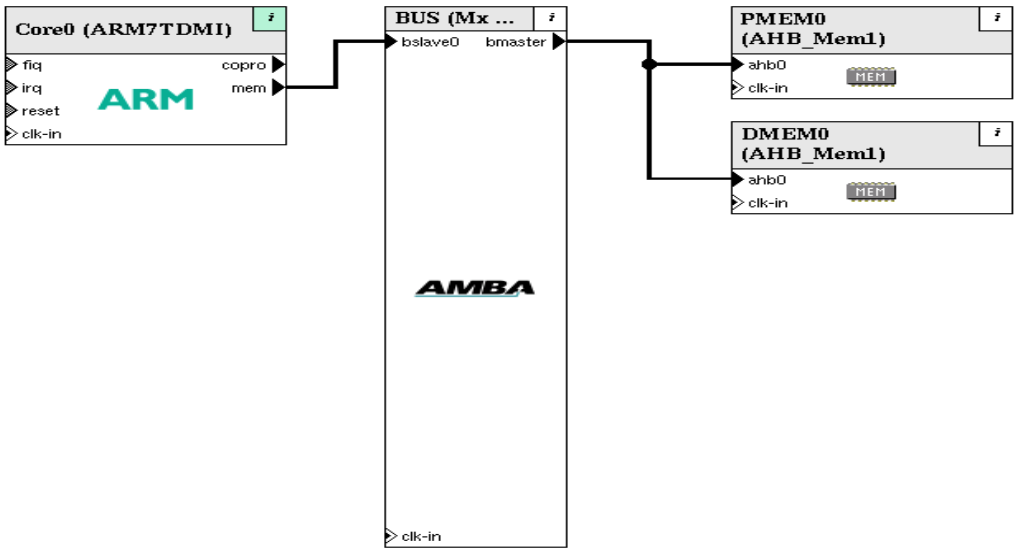
圖表 3-4　根據 xml 紀錄的應用程式效能資訊轉換成 SystemC 模型

範例中顯示函數__Heap_Alloc()需要 66 個計算週期、18 個指令數、6 個記憶體讀取及 3 個記憶體寫入，這些工作構成__Heap_Alloc 的任務區塊，待這些任務區塊內的工作被模擬執行完畢後，才會呼叫__Heap_DescSize()，繼續下一個函數級任務區塊的模擬。

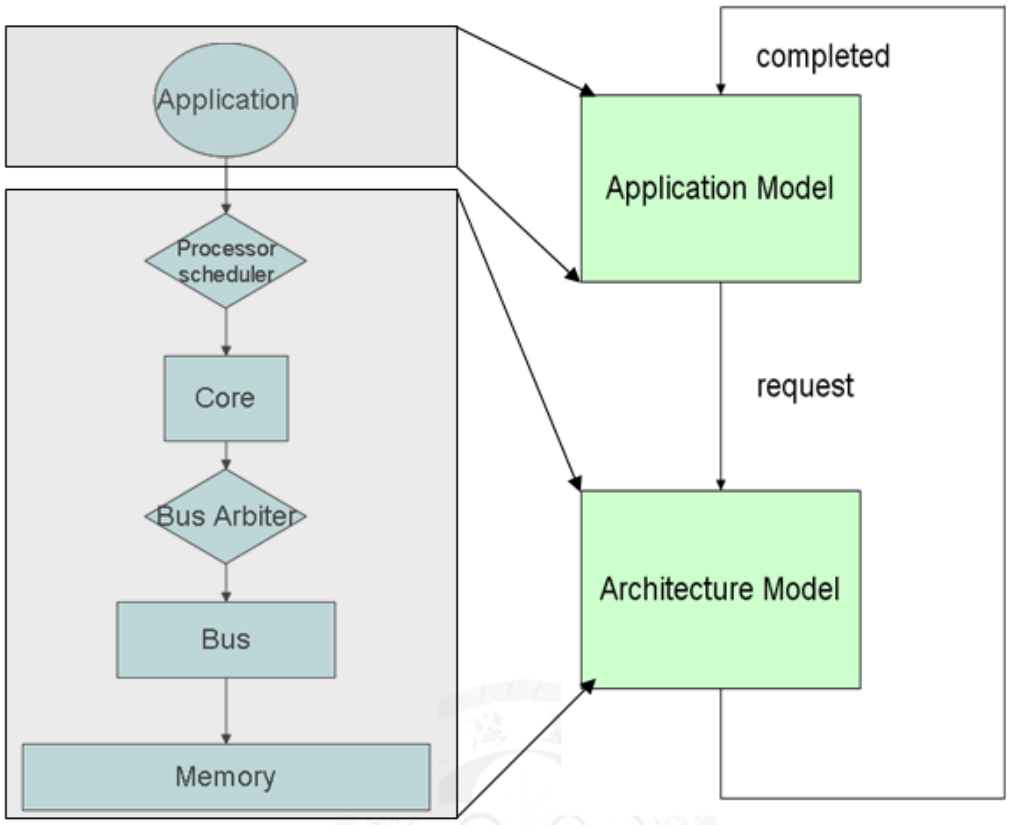
3.1.2 硬體結構模型

將應用程式模型與硬體結構模型獨立分開，目的是希望在早期開發時期硬體結構尚未成形的時候，透過改變硬體結構模型，搭配原本的應用程式模型進行協同模擬，達到預測該軟硬體運作的結果。建構硬體結構模型的概念與應用程式模

型相仿，我們需要對目標的平台進行評測並且抽出參數，例如核心的數量、指令擷取延遲、記憶體讀寫延遲、匯流排協定延遲等。圖表 3-5 是 ARM SoC Designer Simulator 模擬的嵌入式系統平台，也是我們的目標平台，圖表 3-6 左即為相對應的 SystemC 模組，我們建立了處理器排程器、處理核心、匯流排仲裁、匯流排以及記憶體模組，再利用 SystemC 提供的串接元件如埠與訊號，將各模組連成一個網絡，達到溝通及任務轉移的功能，這些獨立的模組分別模擬出該元件收到要求所產生的對應行為，例如 Processor Scheduler 負責接收任務並將之分配到 Core 中進行處理，當多個 Core 同時發出記憶體存取的要求，Bus Arbiter 則是負責匯流排主控權的仲裁，Bus 及 Memory 模組則模擬匯流排及記憶體的存取延遲時間。最後利用 3.1.1 所提出的應用程式模型當作輸入進行交互模擬，如圖表 3-6 右圖。



圖表 3-5　ARM SoC Designer 的模擬平台



圖表 3-6　SystemC 硬體模組與應用程式模組

3.2 模擬正確性及速度

文章先前曾經提到，為了滿足快速導入市場的需求，必須在開發早期階段快速衡量系統架構與軟體之間的效能，這也是我們使用系統層級模擬的目的。然而，通常為了滿足快速得到效能資訊，正確性往往是犧牲的對象，就如同若使用 RTL 層級的模擬，得到了詳盡的資訊，卻也必須忍受較慢的模擬速度一般。如何兼顧速度與可以接受的正確性，是系統層級模擬的關鍵。接下來將深入探討我們模擬框架的速度與正確性。

3.2.1 模擬的正確性

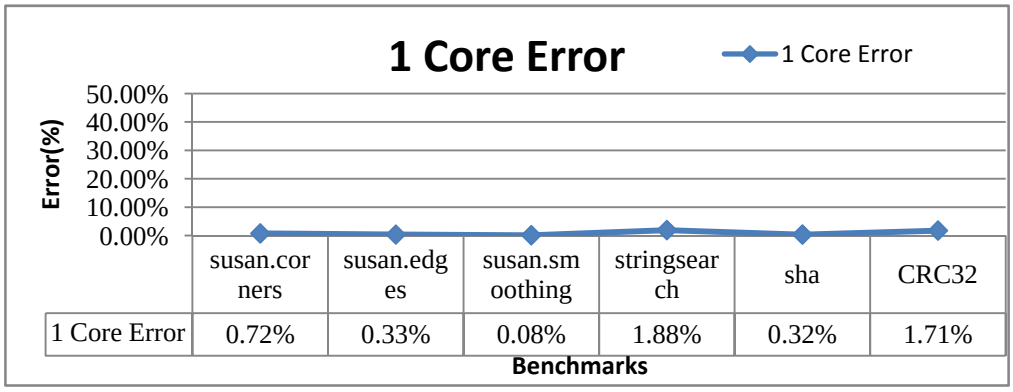
我們使用一套稱做 MiBench[27]的免費基準測試程式來當作我們衡量的標準，這套基準測試程式包含了六大類型：自動化工業控制（Automotive and Industrial Control）、網路（Network）、安全（Security）、消耗裝置（Consumer Device）、辨

公自動化（Office Automation）以及電信（Telecommunications），我們從中選擇六個測試程式如表格 3-1。

表格 3-1　MiBench 測試程式與指令數

workloads	IC
susan.corners	1.06M
susna.edges	1.83M
susan.smoothing	24.89M
sha	13.54M
string serach	0.15M
CRC32	52.83M

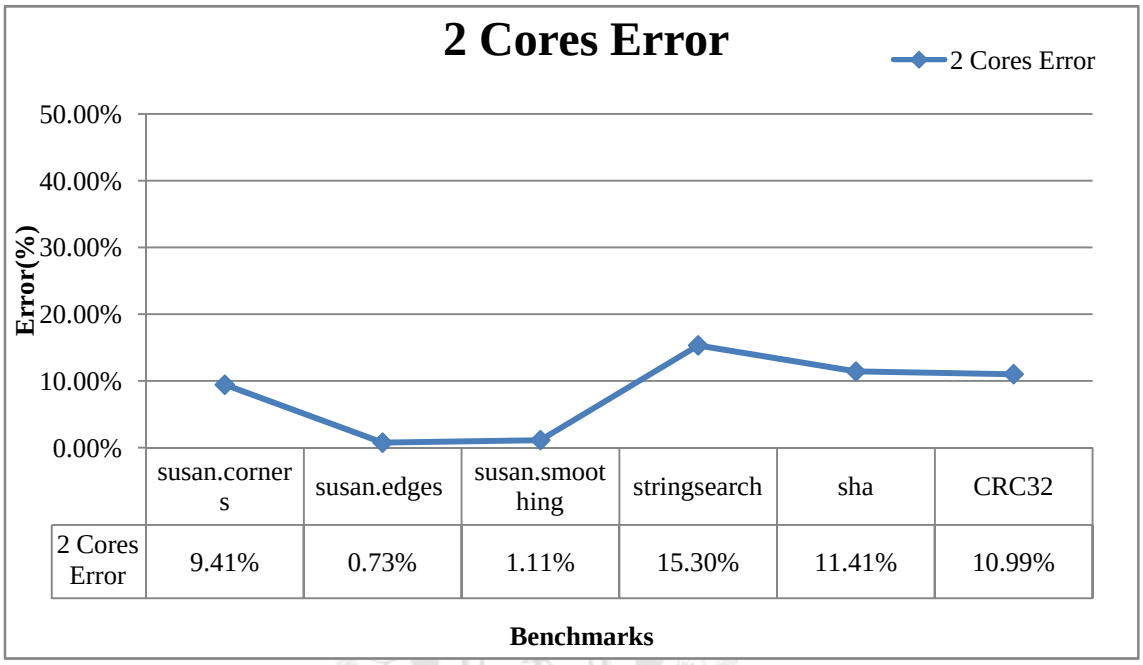
我們利用這六個測試程式跑在 SoC Designer Simulator 以及我們的模擬框架上，分別測試了單核心、雙核心以及四核心三種不同平台，比較兩者消耗的週期數來當做正確性的評估基準。圖表 3-7 為單核心的誤差圖，各個程式的跑在模擬框架上相對於 SoC Designer Simulator 平均產生的誤差大約為 0.67 %，幾乎接近週期模擬的準確度。



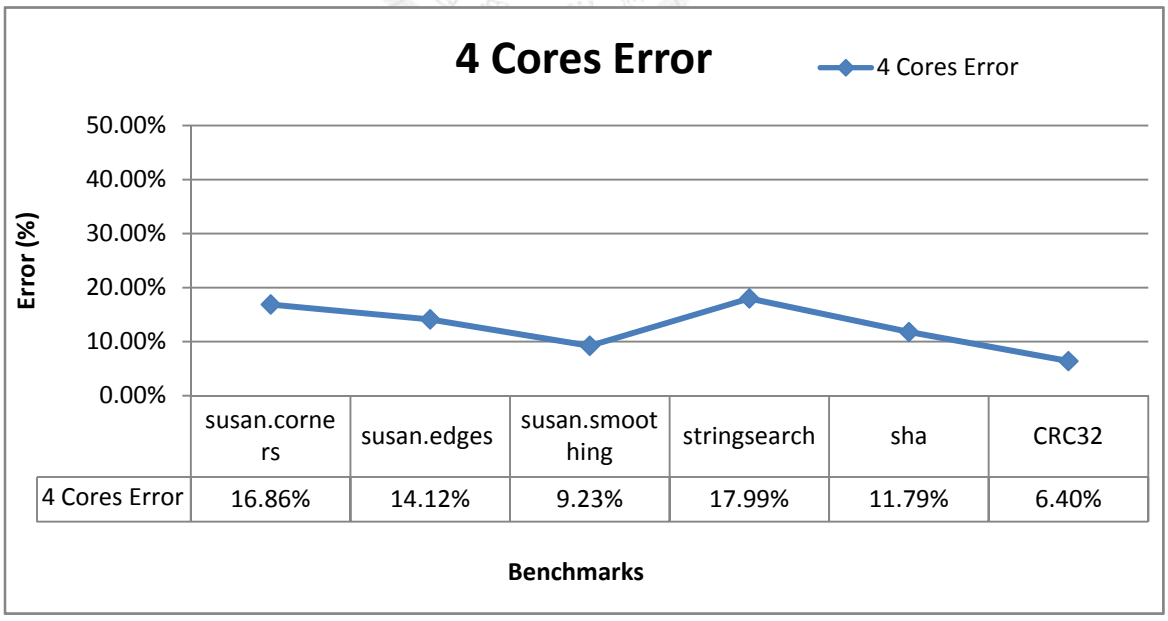
圖表 3-7　單核心之模擬框架相對於 SoC Designer Simulator 的誤差

圖表 3-8 與圖表 3-9 分別呈現出雙核心與四核心的誤差，雙核心的誤差約達 8.16%，四核心則為 11.78%，可以發現明顯大於單核心的誤差，原因可以追究到雙

核心與四核心的模擬複雜度比起單核心多了一些資源分配以及匯流排仲裁等議題，在目前的架構上用了一些簡單的方針來取代 SoC Designer Simulator 裡週期檢視，減低複雜度以增加模擬速度。在早期開發階段，這樣的誤差足夠提供設計者充分的資訊當作參考，並且在廣大的設計空間中（Design Space）找到適合的選項。



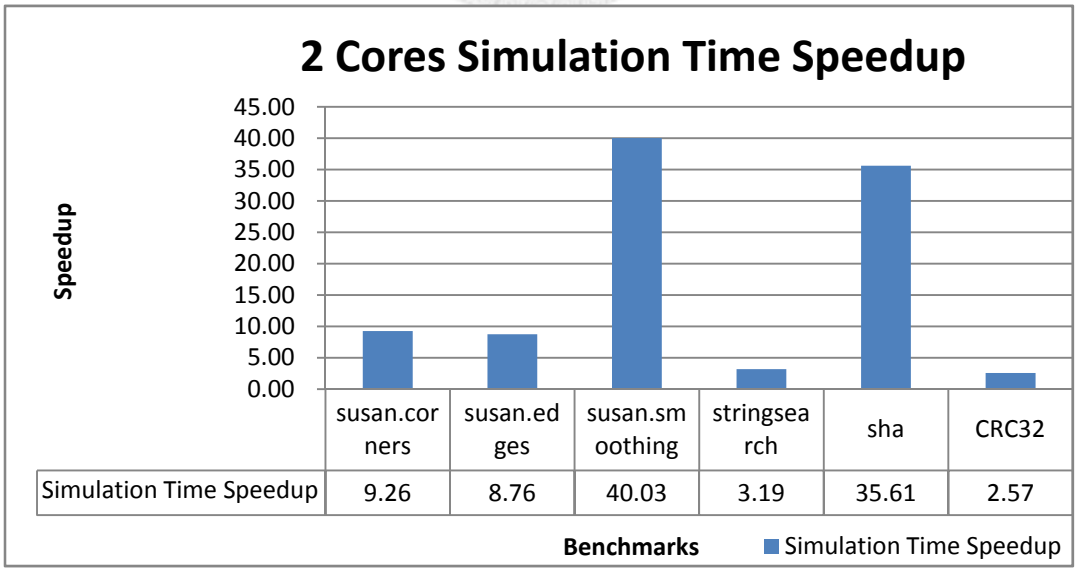
圖表 3-8　雙核心之模擬框架相對於 SoC Designer Simulator 的誤差



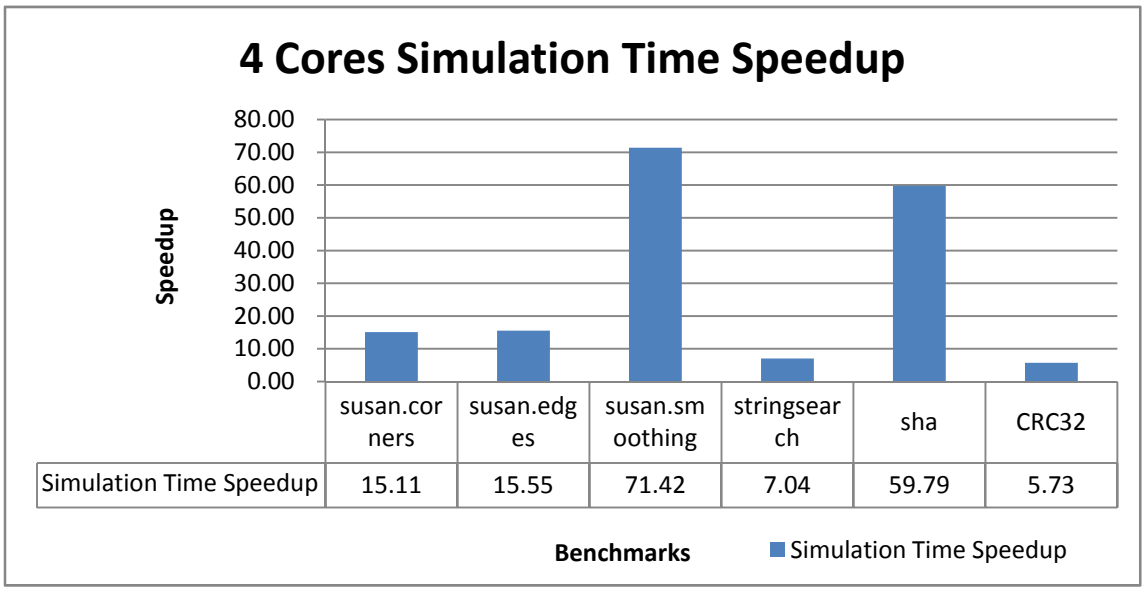
圖表 3-9　四核心之模擬框架相對於 SoC Designer Simulator 的誤差

3.2.2 模擬的速度

除了正確性之外，另一個我們要探討的就是模擬速度的議題。先前曾經提過為了要滿足快速導入市場之需求，希望在早期開發階段以最低成本提供軟硬體設計者足夠的資訊當作開發的方針，時間成本當然在這些成本中佔重要角色。快速的模擬搭配上容許範圍之內的誤差率，將有助於快速的從設計空間中擷取出合適的選項。圖表 3-10 與圖表 3-11 即為雙核心與四核心在模擬速度上相對於 SoC Designer Simulator 的增進，我們可以看到在雙核心的部份，平均會有 14.45 倍的提升，四核心則有高達 25.56 倍的提升，比較起週期模擬，系統層級模擬的確在速度上佔有優勢。另外從圖中注意到 susan.smoothing 與 sha 這兩個測試程式的加速特別顯著，其原因為這兩個測試程式的指令數都較大於其他的測試程式，從表格 3-1 我們即可發現指令數上的差異，再加上此二程式的執行時間也遠遠大於其他程式，可以發現系統層級模擬的速度優勢在執行時間較長的程式上有加成的效果。CRC32 的指令數雖然很多，但多無繁雜運算，所以執行時間其實非常短暫，故沒有太顯著的加速成效。



圖表 3-10　雙核心相對於 SoC Designer 之 speedup

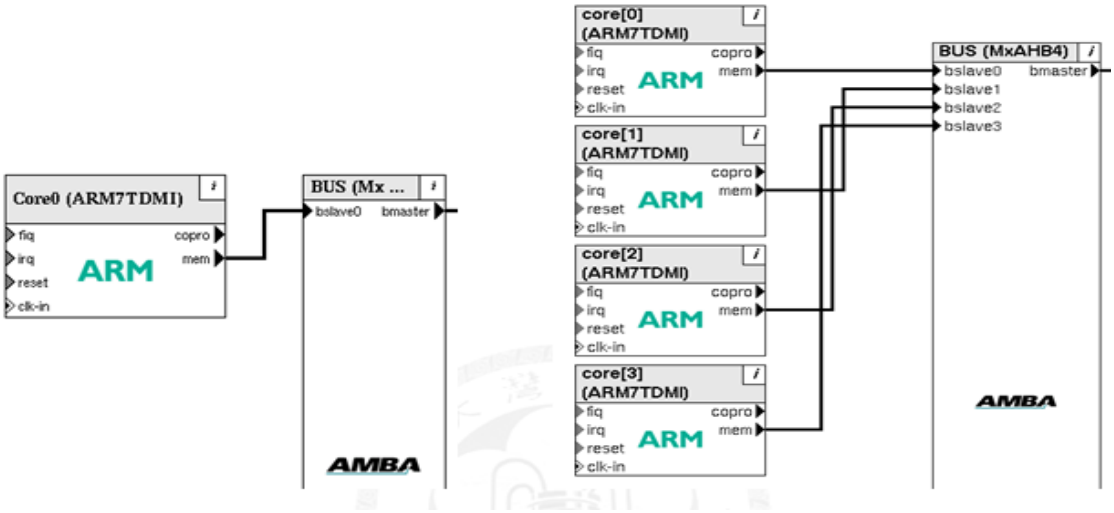


圖表 3-11　四核心相對於 SoC Designer 之 speedup

3.3 小結

本章介紹了一個系統層級的模擬環境，提供了軟硬體設計者更快速了解應用程式在不同硬體結構上的效能，另外，比較起在 EDA（Electronic Design Automation）Tools 上開發新的硬體結構，需要繁瑣的設計佈局，這樣的系統層級開發方式也能節省許多嘗試的時間，例如如果要設計一個多顆處理器架構的硬體，必須使用 ARM SoC Desinger Canvas 進行佈局設計，光是處理器連接匯流排的線路就必須多拉數條線，如圖表 3-12，更別說其他更複雜的架構。系統層級模擬希望可以透過建立硬體結構資料庫的方式，定義更多巨集，例如處理器的數目直接透過”#Define Core 4”的方式，方便快速的進行開發測試。

第一章曾提起現在開發嵌入式系統產品所關心的除了效能之外，功率也是逐漸被重視的課題之一，從耗電量到溫度的問題，無不牽涉到功率與耗能，如何在早期開發階段也注入相關資訊也成為開發工具所需具備的能力。在下一章中我們將專注在記憶體系統的功率資訊取得以及將其功能整合入我們的模擬環境，致力達成一個效能功率模擬分析的框架。



圖表 3-12　ARM SoC Designer Canvas 設計硬體結構示意圖

第4章　記憶體功率分析與評估

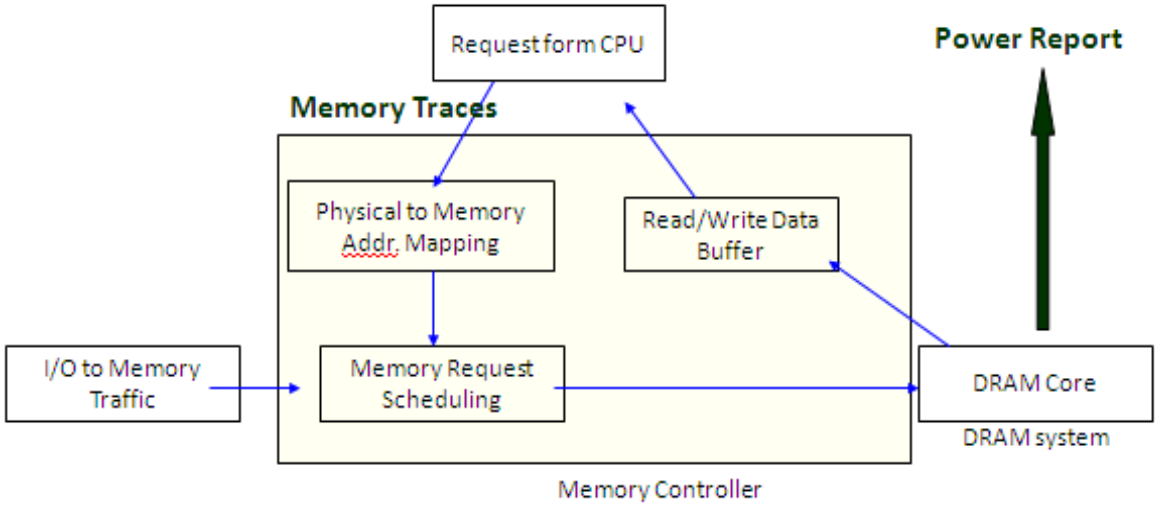
上一章介紹了系統層級的模擬環境，在本章中，我們加入針對記憶體系統功率的估算與分析。由於記憶體系統在整個嵌入式系統中逐漸成為主要的功耗元件[28]，甚至超過處理器，主宰著整個系統的能耗多寡，因此能夠對記憶體系統做進一步的評估及模擬有助於早期系統功率規劃，並且可以了解程式執行時的功耗變化情形，做更進階的動態調整或是節能策略。

在效能模擬方面，我們利用 ARM SoC Designer Simulator 所嵌入的 Profiling tools 對執行中的應用程式進行解析，利用解析出來的效能資料描述應用程式的特性，並建構出模擬環境中所需的應用程式模型。這個模型足以代表該程式執行時的行為特性，這曾在 3.1.1 小節中介紹。同樣的，如果需要模擬應用程式對記憶體系統所產生的功率與能量耗損，我們必須利用模擬器來取得應用程式對記憶體系統產生的功率資料，進而建立屬於該應用程式的功率模型 （Power Model）。我們在這邊選用 DRAMsim 來作為功率資料的評測工具。在 4.1 中我們會介紹 DRAMsim，4.2 將說明如何將應用程式透過 DRAMsim 的模擬取得該程式的功率資料，4.3 說明功率模型以及函數功率資料庫（Function Power Data Bank）。

4.1　功率評測工具 – DRAMsim

DRAMsim[29]是由馬里蘭大學的 Systems and Computer Architecture Lab (SCAL)提出，DRAMsim 可獨立為一個專門模擬記憶體系統的程式，也可與一些全系統模擬器進行整合，如 sim-MASE 或 GEMS。它可模擬大部分記憶體裝置，像是 SDRAM、DDR、DDR2、DDR3 SDRAM 等。透過 DRAMsim，我們可模擬記憶體系統行為，包括匯流排介面單元(Bus Interface Unit)、執行佇列(Transaction Queue)、列緩衝管理機制(Row Buffer Management Policy)及位址轉譯(Address Mapping)等記憶體控制器（Memory Controller）內部元件的行為，再加入記憶體模組操作的時間及功率參數，得到記憶體系統的效能與功率相關數據資料，如圖表 4-1。

驅動 DRAMsim 的方式有兩種，其中一種是提供一些 pre-programmed 的程式隨機產生一連串的記憶體要求(Memory Request)，另一種就是以位址記錄(Address Trace)當作輸入。我們透過第二種方式來使用 DRAMsim，若能取得應用程式的位址記錄，即可進行該程式對記憶體系統操作的模擬與評估。在下節我們敘述如何取得應用程式的位址記錄並且轉換成驅動 DRAMsim 的資料格式。



圖表 4-1　DRAMsim 模擬架構圖

4.2　應用程式位址記錄蒐集與轉換

如上小節所說，利用 DRAMsim 對應用程式進行功率評測前，須先得到該應用程式的記憶體存取位址記錄，這個位址記錄將代表應用程式對於記憶體系統使用的行為，驅動 DRAMsim 進行模擬。如何得到應用程式的位址記錄成為接下來討論的重點。先前提過 ARM SoC Designer Simulator 提供許多 software profiling tool，包括 software profiling、core profiling、bus profiling、transaction profiling 等。我們利用處理器與匯流排之間的 transaction profiler 來觀察處理器對匯流排產生的要求，進一步擷取對記憶體系統的存取資訊，記錄下的資料內容格式如表格 4-1 所示。

Cycle	Address	Operation Type	
396093	3736	2	⇦ Transaction 1
396094	3736	3	⇦ Transaction 1
396094	3740	2	⬅ Transaction 2
396095	3736	3	⇦ Transaction 1
396095	3740	2	⬅ Transaction 2
396096	3740	3	⬅ Transaction 2
396096	4294966672	2	
396097	3740	3	⬅ Transaction 2
396097	4294966672	2	
396098	4294966672	3	
396099	4294966672	3	
396102	3744	2	
396103	3744	3	
396103	3748	2	
396105	3748	3	

Legend for Operation Type
0 → Write Address
1 → Write Data
2 → Read Address
3 → Read Data

表格 4-1　ARM SoC Designer 剖析記憶體存取資料之輸出

Cycle 代表處理器產生要求的時間點，Address 代表要存取的記憶體位址，而 Operation Type 則是代表對匯流排所發出的要求型態，這裡有四種，write address 和 read address 將對 bus 的 address line 發出要求，告知 bus 即將要存取該 address line 上的資料，write data 和 read data 則完成讀寫資料的動作，這裡的每一個 operation 都代表著在該 Timestamp 發出讀或是寫某個記憶體位置資料的要求。然而記錄下的是未經處理的資訊，包含許多因為尚未得到匯流排主控權而重複送出的要求，必須經過處理，擷取出真正對記憶體系統送出的

要求；從表格 4-1 的例子中可以得知，在 cycle 396093 時，即將對 address 3736 進行 read 的動作，隨後的 cycle 396094 即取得 bus 的准許並對該記憶體位址進行資料讀取，直到 cycle 396095 才結束這一次的 transaction，我們稱為 transaction1。另外可以注意到，在 cycle 396094 時，另一個記憶體要求也同時產生，我們稱為 transaction2，由於此時的 bus 被 transaction1 佔據著，所以一直產生 read address 的要求直到 transaction1 結束於 cycle 396095，transaction2 才得以在 cycle 396096 進行 read data 的動作。所以擷取分析的過程，即為區分每一次不同的 transaction ，並且擷取出每一次 transaction 的第一個記憶體要求，得到每一個 transaction 要求的時間點並且記錄下來，我們稱這一動作為位址紀錄簡化（Trace Simplify）。表格 4-2 即為簡化後的位址紀錄。

Cycle	Address	Operation Type	
396093	3736	2	⇦ Transaction 1
396094	3740	2	⬅ Transaction 2
396096	4294966672	2	
396102	3744	2	
396103	3748	2	

表格 4-2　ARM SoC Designer 記憶體存取之簡化輸出

然而，這樣的位址記錄並不是驅動 DRAMsim 的輸入格式，我們必須經過一些轉換產生對應的格式。DRAMsim 的輸入格式範例如表格 4-3。

Address	Operation	Timestamp
0x00000E98	READ	396093
0x00000E9C	READ	396094
0xFFFFFD90	READ	396096
0x00000EA0	READ	396102
0x00000EA4	READ	396103
0x00002AE6	READ	396105

表格 4-3　DRAMsim 輸入格式範例

經過簡化之後的位址紀錄利用 Parser 將簡化之位址記錄轉譯成符合驅動 DRAMsim 的輸入格式，包括位址十進制轉十六進制以及 Operation type 的對應轉換，表格 4-2 所舉的例子經過簡化之後會成為表格 4-3 所示。如此一來，這份輸出即可代表應用程式對記憶體系統的操作行為，也就是說記憶體系統將會收到的記憶體存取指令，我們將利用它輸入 DRAMsim 進行功率評測，取得該應用程式對記憶體系統所產生的功率消耗。下一小節會討論如何得到應用程式的功率資訊。

4.3　應用程式功率量測

4.3.1　DRAMsim 功率量測方法

先前提到的DRAMsim可以模擬現存的記憶體硬體裝置，諸如SDRAM、DDR、DDR2、DDR3 SDRAM 等，它利用一配置檔案（Configure File）當作參數輸入來指出模擬的記憶體裝置結構，如表格 4-4 所示。檔案內容提供了記憶體型態資訊，包括 datarate、channel 數、rank 數、bank 數等，利用這些參數可以調配出預期的記憶體裝置。另外要使用 DRAMsim 量測功率，還必須提供另一個功率的配置檔，如表格 4-4 所示，配置檔包含記憶體芯片的工作電壓以及記憶體不同狀態時的電流等。兩者搭配即可利用 DRAMsim 模擬出 DRAM 裝置並且取得需要的時間及功率資訊。我們透過設定這兩個檔案，將目標裝置設定為一 50Mhz 的 2GB SDRAM 記

億體，並根據記憶體製造商 Micron[30]提供的電力規格當作功率設定的標準，利用此規格當作我們的目標記憶體裝置並進行模擬。

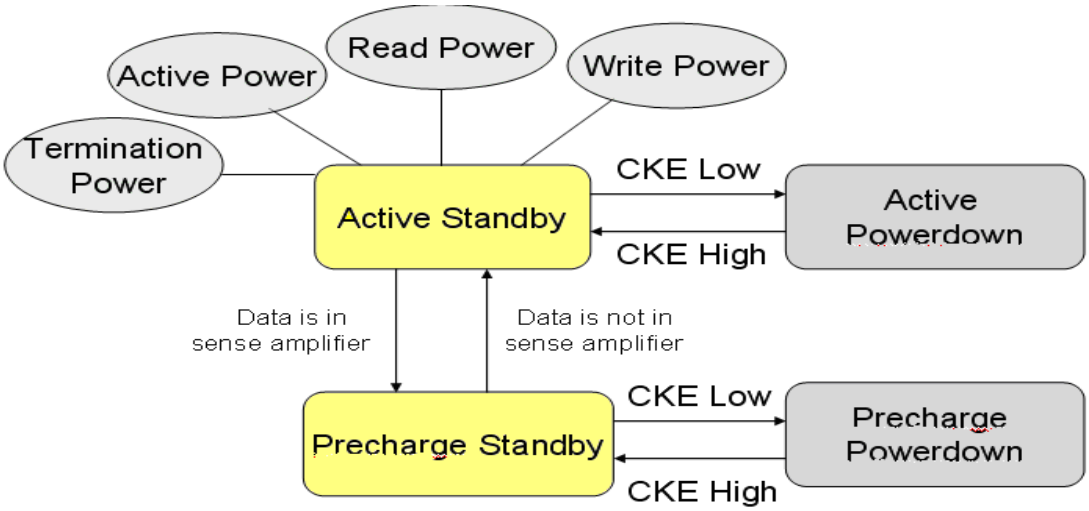
type	sdram
datarate	50
channel_count	1
channel_width	4
PA_mapping_policy	sdram_close_page_map
row_buffer_policy	open_page
rank_count	4
bank_count	8

表格 4-4　DRAMsim 之記憶體配置檔案

```
# DDR SDRAM Configuration and Data Sheet Parameters #
# corresponding to 512Mb 400 MHz x8 3-4-4 spd file    #
density    512                    # Mb
#DQ          8                    # per chip
DQS          1                    # per chip
max_VDD    2.7                    # V
min_VDD    2.5                    # V
IDD0       155                    # mA
IDD2P      5                      # mA
IDD2F      55                     # mA
IDD3P      45                     # mA
IDD3N      60                     # mA
IDD4R      190                    # mA
IDD4W      195                    # mA
IDD5       11                     # mA
```

表格 4-5　DRAMsim 之功率資訊配置檔案

DRAMsim 對於功率的模擬方法是根據記憶體裝置透過位址記錄被使用的情況來計算，位址記錄的一個記憶體存取指令如”0x00000E98 READ 396093”，會被記憶體微控制器（Microcontroller）分解成數個記憶體操作指令，如 Row Access Strobe、Column Access Strobe、Precharge 等，透過統計這些一連串實際針對記憶體裝置進行操作的指令，並累計記憶體裝置在各種狀態的時間比例，乘上在該狀態所須的功率，來估計所消耗的功率。記憶體裝置基本上分成四大狀態，如圖表 4-2，分別為 Active standby、Active powerdown、Precharge standby、Precharge powerdown，其中 standby 與 powerdown 的區別在於記憶體的 CKE（Clock enable signal）是否為 active，若 CKE 為 low，記憶體裝置的輸入緩衝區是關掉的，此時並不接受存取指令，裝置的狀態基本上是 powerdown 的。Active 與 Precharge 的差別則是取決於 snese amplifier 中是否有要被讀取的資料，如果有，則在 Active State，沒有的話則在 Precharge State。如果裝置狀態在 Active，則可以進行讀寫動作，所以狀態 Active 中又分出四種狀態，分別為 Write、Read、Active 與 Termination，這四種狀態基本上是一體的，在 Active Standby 時接收到記憶體操作指令，必先從 Active 開始，將資料送出到 sense amplifier，才可以進行讀或寫，最後結束該次的存取則是 termination power。記憶體處在各種狀態上就會有相對應的功率值，在根據在各狀態上的時間比例，綜合計算並加總出最後的總功率。

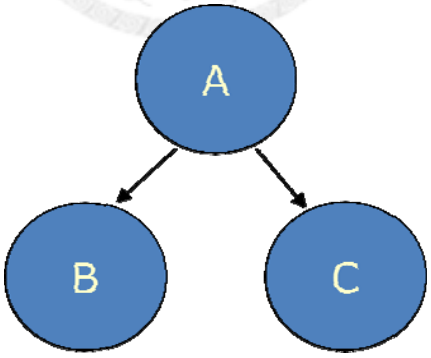


圖表 4-2　DRAMsim 中功率模擬之記憶體狀態

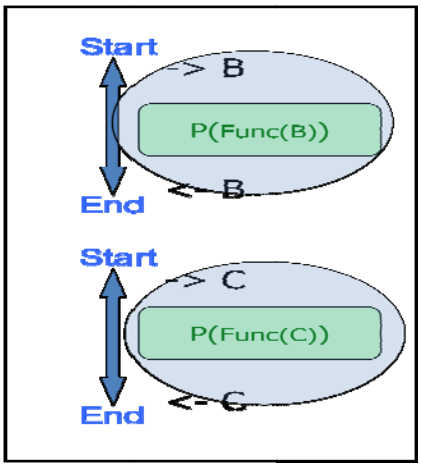
4.3.2 函數層級功率量測與函數功率庫的建立

介紹完 DRAMsim 進行功率量測的方式後，我們預計將應用程式的位址紀錄輸入 DRAMsim 中，得到該應用程式對記憶體系統的功率消耗。3.1.1 節中我們曾經提到我們的將應用程式拆解成相互呼叫的模型，利用函數中的行為資訊建構一個個屬於該函數的任務區塊。在功率模型這方面，我們也採取相同的方法，把應用程式對記憶體系統所產生的功耗當作是應用程式對系統產生的一個行為，以函數層級來進行估測和建構，達到提高模擬速度的效果並且整合進原本的系統層級模擬框架中，過去也可以看到類似的研究應用在處理器上[31]。

4.2 節中說明了如何將欲模擬的應用程式記憶體位址紀錄轉換成驅動 DRAMsim 的格式，這份輸入代表著該應用程式一連串對記憶體系統的操作指令，欲達到上段提到的函數層級功率模擬，我們必須知道各函數的執行起始點，然後對應到相關的記憶體系統操作指令。舉例說明，有一個函數呼叫如圖表 4-3 所示，要得到函數 B 及函數 C 的功率，必須提供函數的起始及結束的時間點如圖表 4-4，DRAMsim 將模擬這段期間的記憶體要求並估計這段時間的平均功率。



圖表 4-3 函數呼叫路徑圖



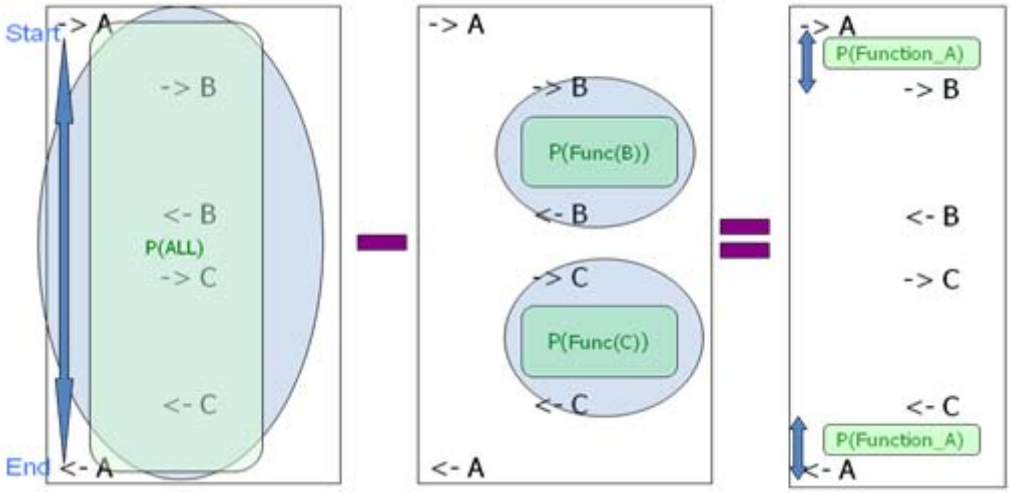
圖表 4-4　函數 B 及函數 C 功率估量示意圖

函數 B 及函數 C 透過最直觀的方式即可取得各自的功率值，乘上時間可得到能量消耗，公式如下：

- *E(Func(B)) =P(Func(B)) * T(Func(B))*
- *E(Func(C)) =P(Func(C)) * T(Func(C))*

其中 E(Func(B))代表函數 B 所消耗的能量，P(Func(B))代表函數 B 執行時期的平均功率，T(Func(B))則表示函數 B 的執行時間。接下來要得到函數 A 的功率，無法透過直接給定其開始及結束的時間點直接輸入 DRAMsim 作估量，原因在於在這段期間還呼叫了函數 B 及 C，並非純屬於函數 A 的記憶體要求，所以我們透過加入運算的方式取得，如圖表 4-5 所示，我們可以得到整個程式的功率及能耗，即 P(ALL)以及 E(ALL)，再利用上述得到的 P(Func(B))、P(Func(C))、E(Func(B))、E(Func(C))，經過公式如下即可得到函數 A 的功率及能耗：

- *E(Func(A)) = E(ALL) - E(Func(B)) - E(Func(C))*
- *P(Func(A)) = E(Func(A)) / T(Func(A))*



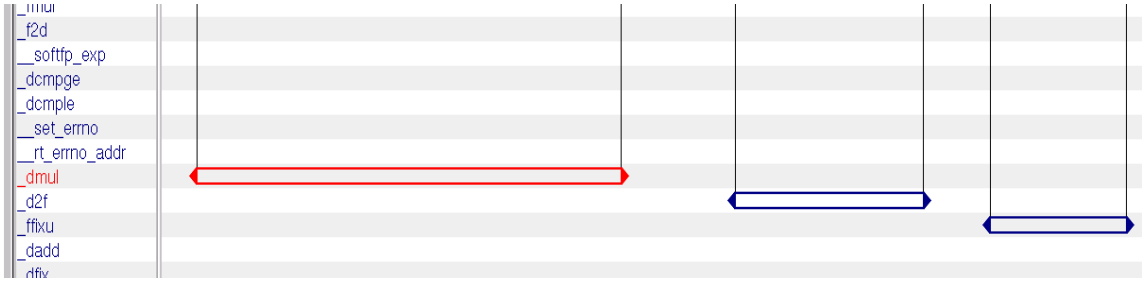
圖表 4-5　函數 A 功率估量示意圖

透過上述範例描述的方法，我們可以得到目標應用程式所有函數的功率及能耗，儘管可能是較複雜的呼叫路徑。得到所有函數功率資料，搭配程式效能評測所得到的函數呼叫次數，可以進一步的估算出整個應用程式的平均功率，計算方式如下：

- *E(Func(i)) =P(Func(i)) * T(Func(i)) * Number of function calls*
- *E(Total)=∑E(Func(i))*
- *P(Total) = E(Total) / T(Total)*

其中 i 從 1 到 n，n 代表應用程式中的函數個數，E(Total)及 P(Total)則為整個應用程式的能耗及功率。

利用 SoC Designer Simulator 中所提供的軟體評測工具，它提供了函數的呼叫路徑圖，並且可以得到函數的起始點時間，如圖表 4-6 所示，搭配此資訊，我們在模擬應用程式對記憶體系統操作指令的同時，標記出該函數的起始點，再利用上述範例提出的方法，即可完成函數層級的功耗量測。

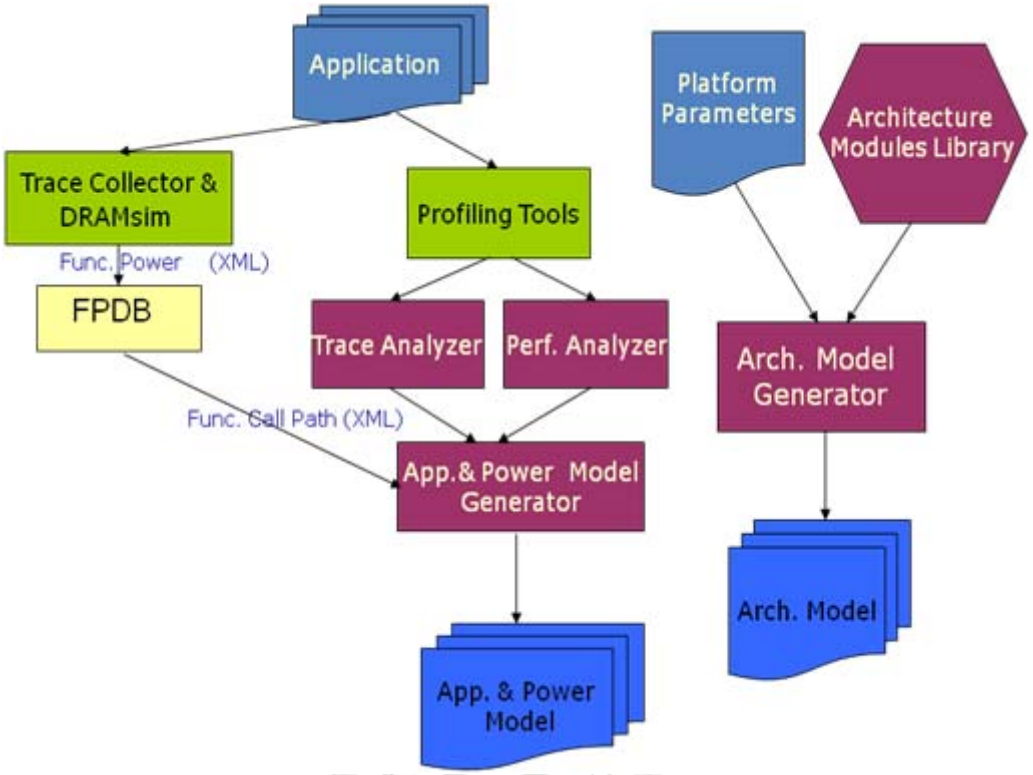


圖表 4-6　SoC Designer 軟體評測工具提供函數呼叫路徑功能

在量測各函數功率的過程中，我們發現在各個應用程式裡，擁有大部分相同的函數，這些函數通常為 ARM 標準 C 語言函式庫裡的函數，包括了許多初始化及配置的程序，尤其同一類型的應用程式如 susan.corner、susan.edge、susan.smoothing，同屬於圖像處理的應用程式，呼叫到許多相同的浮點數函數庫中的例行程序（Routine Process），如_dcmpge、_dcmple 等，這些程序的行為固定，對應的記憶體操作指令也大致相同，基於這樣的原因，我們將這些函數對記憶體系統產生的功率建成一個函數功率資料庫(Functions Power Data Base)，簡稱 FPDB，供應我們建構應用程式的功率模型。共用函數功率資料庫有一些優點，我們可以不必對每一個應用程式的每一個函數進行功率量測，甚至可以進行預測不同應用程式或是同一應用程式但不同工作集（Working Set）的功率情形。

加入功率函數資料庫後，整個效能及功率模型建置流程如圖表 4-7。

另外，我們在架構中增加一個功率估算模組，搭配函數功率資料庫，隨著應用程式模型的執行，即時的處理的平均功率以及能量消耗資訊並記錄下來，希望可以提供程式執行時的功率變化情形。目前我們建置的函數功率資料庫可供 stringsearch、susan.corner、susan.edge、susan.smoothing 這些應用程式使用，在下章節中將討論如何評估的這樣實作方法以及其準確度。



圖表 4-7 效能及功率模框架境示意圖

第5章　實驗及結果探討

上一章中說明了我們如何利用 DRAMsim 取得目標應用程式中各函數的功率資訊，然後透過函數層級的功率資訊估測整個應用程式的功率，估計這樣的作法或許會有一些誤差,需要設計一驗證方法評估其誤差率。5.1 節中說明實驗的平台、步驟以及使用的測試程式，5.15.2 將分析與探討實驗的數據。

5.1 實驗平台及步驟

我們實驗的目標記憶體系統為一時脈為 50Mhz 的 SDRAM，搭配時脈同為 50Mhz 的 ARM7 系列處理器組成系統平台，詳細的記憶體架構參數如表格 5-1 所示。

表格 5-1　目標記憶體系統之架構

DRAM Type	sdram	Bank count	8
Data rate	50Mhz	Row count	8192
Channel count	1	Column count	2048
Channel width	4	Row buffer policy	Open page
Rank count	4		

表格 5-2　Mibench 測試程式與指令數

workloads	IC
susan.corners	1.06M
susna.edges	1.83M
susan.smoothing	24.89M
string serach	0.15M

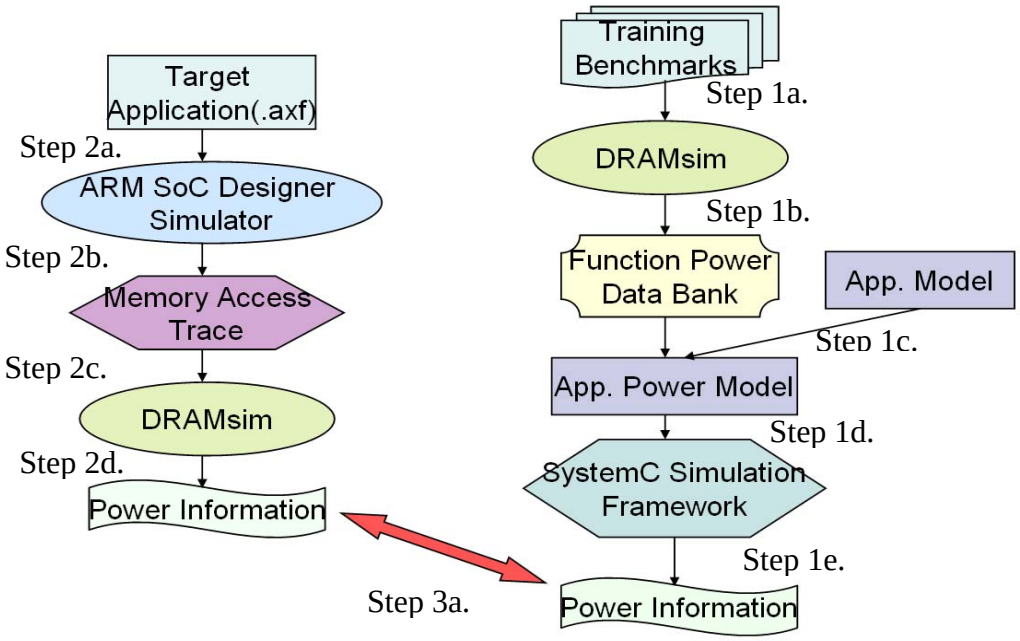
表格 5-2 為本次實驗所使用的測試程式，由 Mibench 中選出 stringsearch 及 susan 系列程式，在 5.2 中會分別介紹及討論。

圖表 5-1 為實驗步驟流程圖，分別有左右兩種得到記憶體系統功率資訊的流程：

1. 右邊的即為第四章中所提到的，利用 DRAMsim 訓練測試程式得到的函數層級功率資訊(Step 1a.)，建立成功率資料庫(Step 1b.)，功率資料庫將提供原本的應用程式模型參考，使應用程式模型中包含各種行為資訊，其中當然包括時間與功率(Step 1c.)，最後再放入整個模擬框架(Step 1d.)，得到整個應用程式的效能與功率資訊(Step 1e.)。

2. 左邊流程圖即為將欲模擬的目標應用程式放入 ARM SoC Designer Simulator(Step 2a.)，並利用其提供的評測工具得到完整的記憶體存取位址記錄(Step 2b.)，然後直接將整個記憶體存取記錄透過 DRAMsim 進行模擬(Step 2c.)，模擬完畢後即可取得該程式對記憶體系統所造成的功率能量消耗資訊(Step 2d.)。

3. 最後我們將左邊流程所得到個功率資訊當作標準，與右邊流程的結果做比較及驗證(Step 3a.)，評估函數層級功率採集的正確性以及函數功率資料庫的可行性。在 5.2 中將探討這部分的結果。



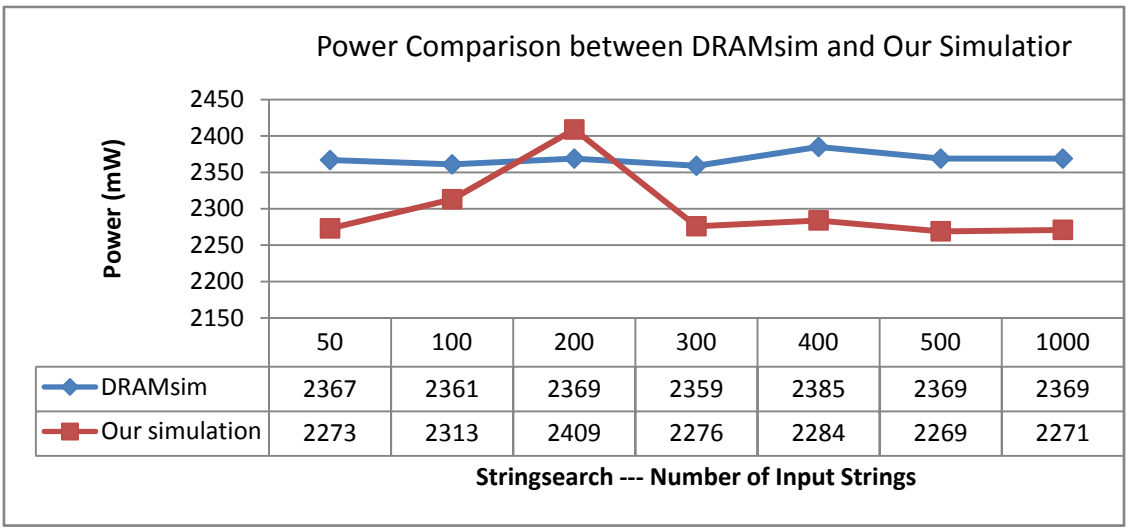
圖表 5-1　實驗與驗證流程圖

5.2 功率量測準確度探討

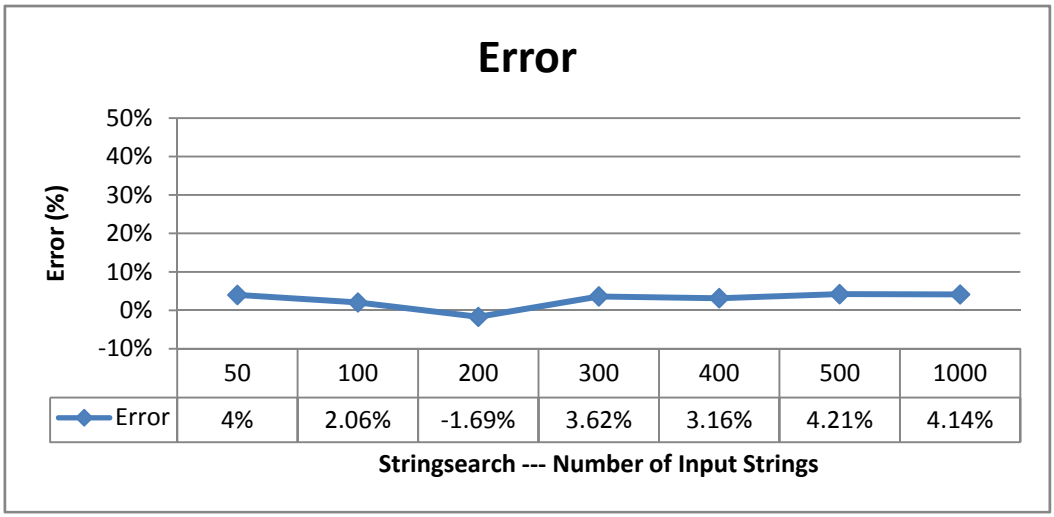
目前的函數功率資料庫能支援的應用程式包括 stringsearch、susan.corner、susan.edge、susan.smoothing，透過上小節的實驗流程，得知實作方式的誤差。5.2.1 及 5.2.2 將分別介紹並討論這兩類測試程式的相關實驗結果。

5.2.1 Stringsearch

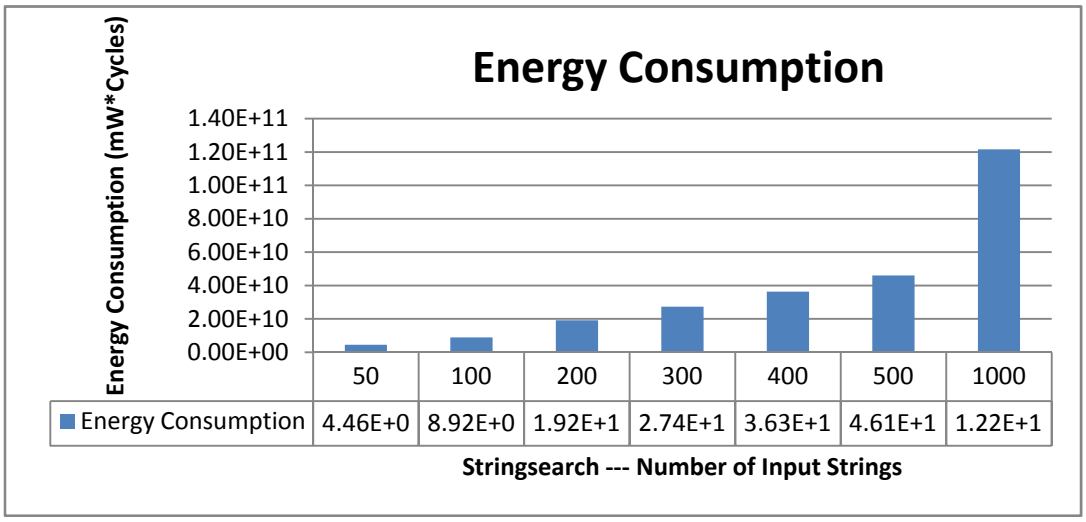
圖表 5-2 為應用程式 stringsearch 的結果，stringsearch 為一字串比對程式，輸入為欲比對的字串，若與預設的字串集合相同，則會被辨識出並標記。我們分別使用不同大小的工作集合（Working Set），也就是輸入不同數量的字串作測試。可以看到誤差範圍介於 20mW 至 100mW 之間，圖表 5-3 為其誤差圖表，誤差範圍約為 2% ~ 5%之間，可以發現針對不同的工作集合，功率的模擬及預測是可以被接受的。圖表 5-4 中則為不同工作集合所產生的能量消耗，隨著工作集合變大，記憶體能量消耗也呈比例上升。



圖表 5-2　應用程式 stringsearch 針對不同工作集合的功率



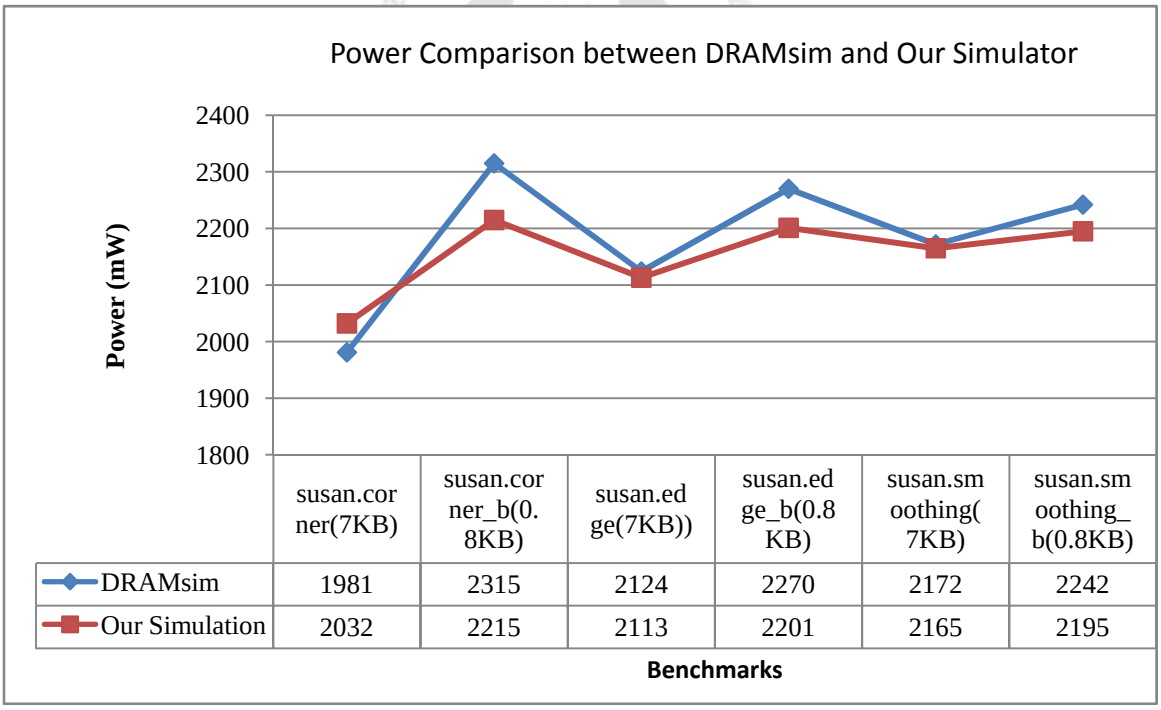
圖表 5-3　應用程式 stringsearch 針對不同工作集合的誤差



圖表 5-4　應用程式 stringsearch 針對不同工作集合之能量耗損

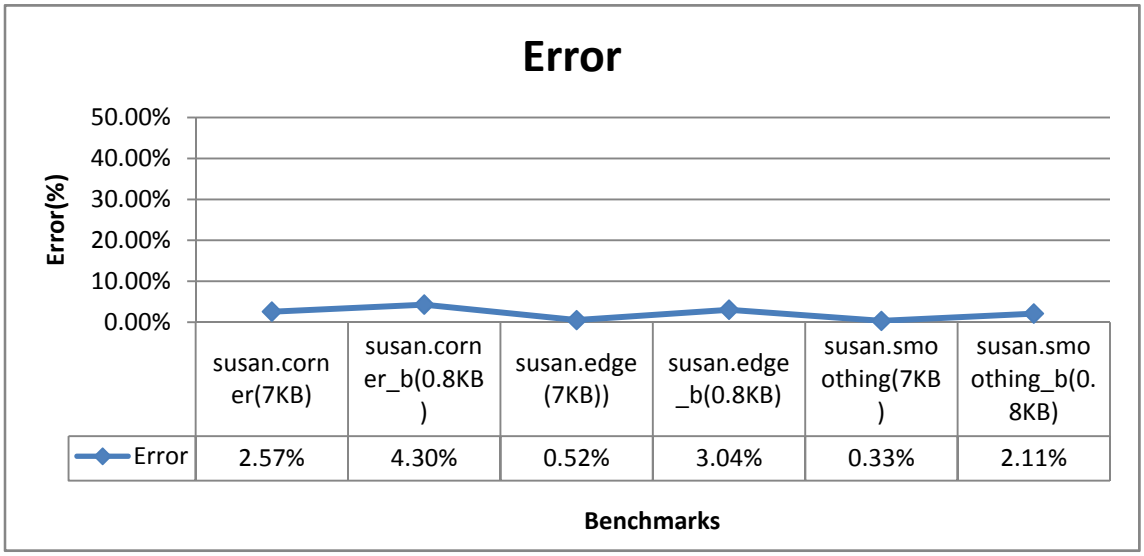
5.2.2 Susan 系列

圖表 5-5 及圖表 5-6 為 susan 系列應用程式的功率模擬結果及誤差， susan 系列為一圖形辨識及處理的程式，辨識輸入圖像的角（corner）、邊（edge）以及加入柔化的作用。由於同為對圖像作處理的應用程式，大量使用了相同的處理常式及函式庫，包括初始化和浮點數處理函數，功率函數資料庫的正確性及可行性可以透過 susan 系列程式得到評估。我們同樣投以兩種不同工作集合的輸入測試，一個為 0.7KB 的簡單單色圖樣，另一個則為較複雜的 8KB 多色圖像。功率值介於 1980mW 至 2320mW 之間，比較 DRAMsim 與我們的模擬，誤差最多至 6%。從 stringsearch 及 susan 系列應用程式，誤差率皆在可接受的範圍，證明了功率函數資料庫的可用性。



	susan.corner(7KB)	susan.corner_b(0.8KB)	susan.edge(7KB))	susan.edge_b(0.8KB)	susan.smoothing(7KB)	susan.smoothing_b(0.8KB)
DRAMsim	1981	2315	2124	2270	2172	2242
Our Simulation	2032	2215	2113	2201	2165	2195

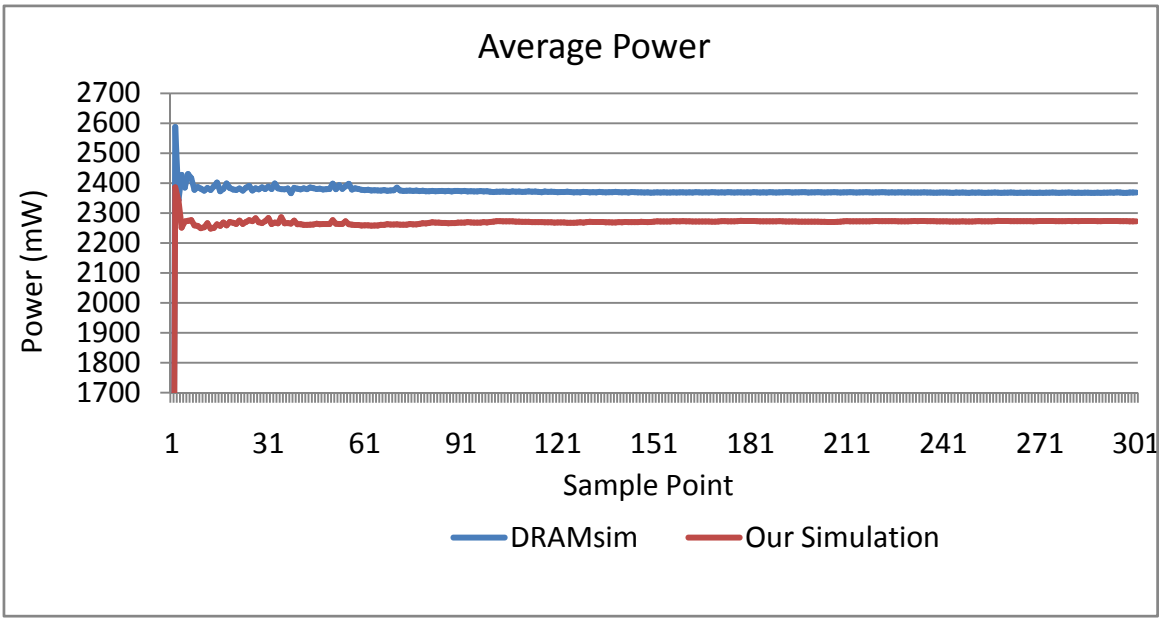
圖表 5-5 susan package 的功率模擬及預測結果



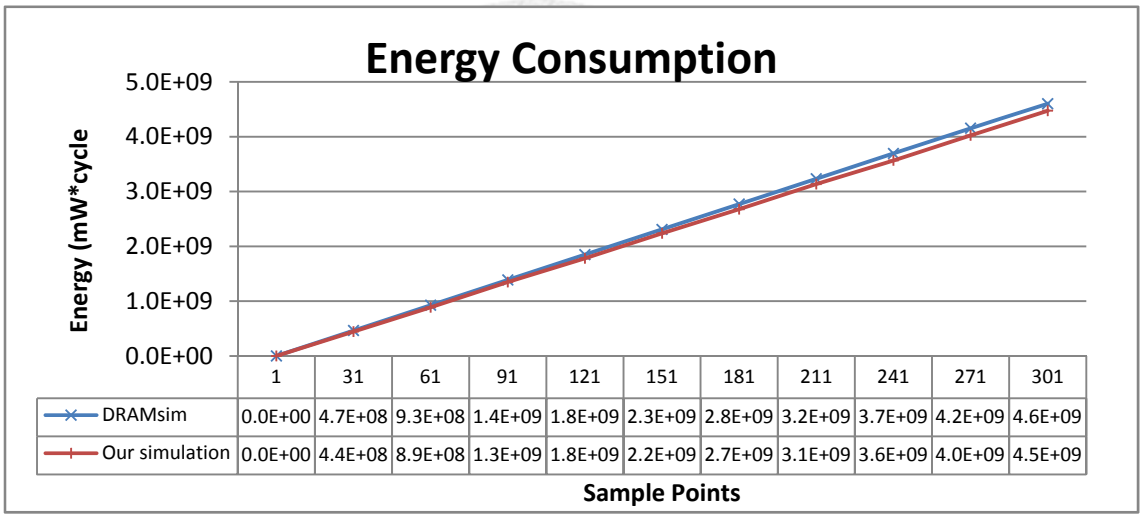
圖表 5-6　susan package 的功率模擬誤差

5.3 功率及能量消耗分析

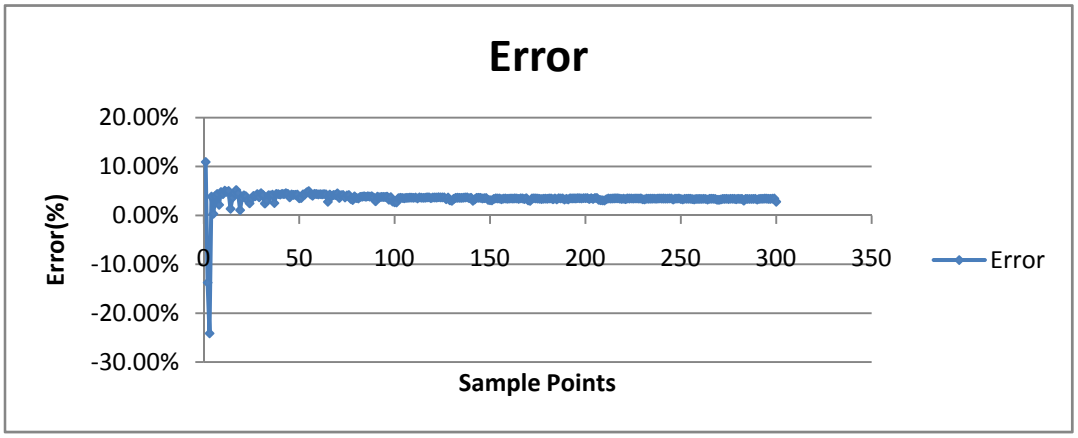
在程式執行的同時，如果能夠即時監控功率與能量消耗，對於軟體開發者甚至是硬體開發者而言，都有一些幫助，圖表 5-7 為應用程式 stringsearch 在程式執行時，取樣 300 次的功率情形，也就是每個一段固定時間區間即監控一次。橫軸即為取樣點編號，1 為程式開始點，301 則為程式結束點，縱軸則為功率值，可以看到平均功率表現大致與 DRAMsim 所得到的相仿，然而還是存在著系抽象化統行為層級模擬所帶來的誤差，大約 100 mW。圖表 5-8 則為隨著程式執行的累積能量消耗情形，縱軸為累積的能耗，橫軸為採樣點，可以看見與模擬出較大功率值的 DRAMsim 所估測的能量消耗值比較，我們得到的累積能耗皆較小，圖表 5-9 則是根據圖表 5-8 所計算出的兩者誤差，誤差在一開始的時候波動較大是因為起始的幾個函式可能不足以代表整個程式的行為，這種模擬方式隨著程式執行，漸漸表現出整個程式的行為，誤差也會趨於穩定，約收斂在 4%。



圖表 5-7　stringsearch 功率監測比較圖

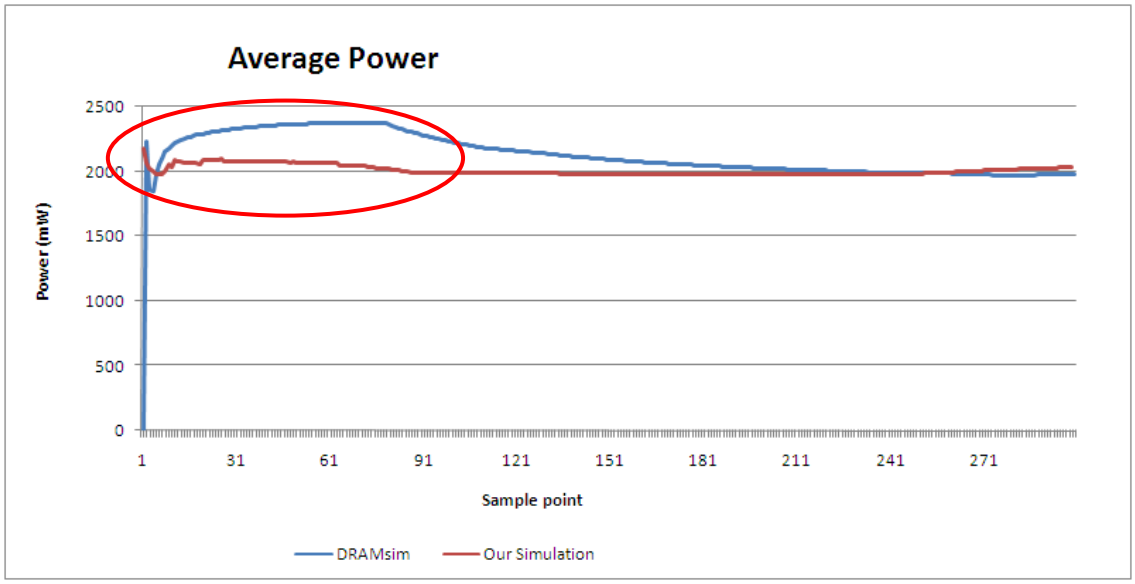


圖表 5-8　stringsearch 累積能量消耗比較圖

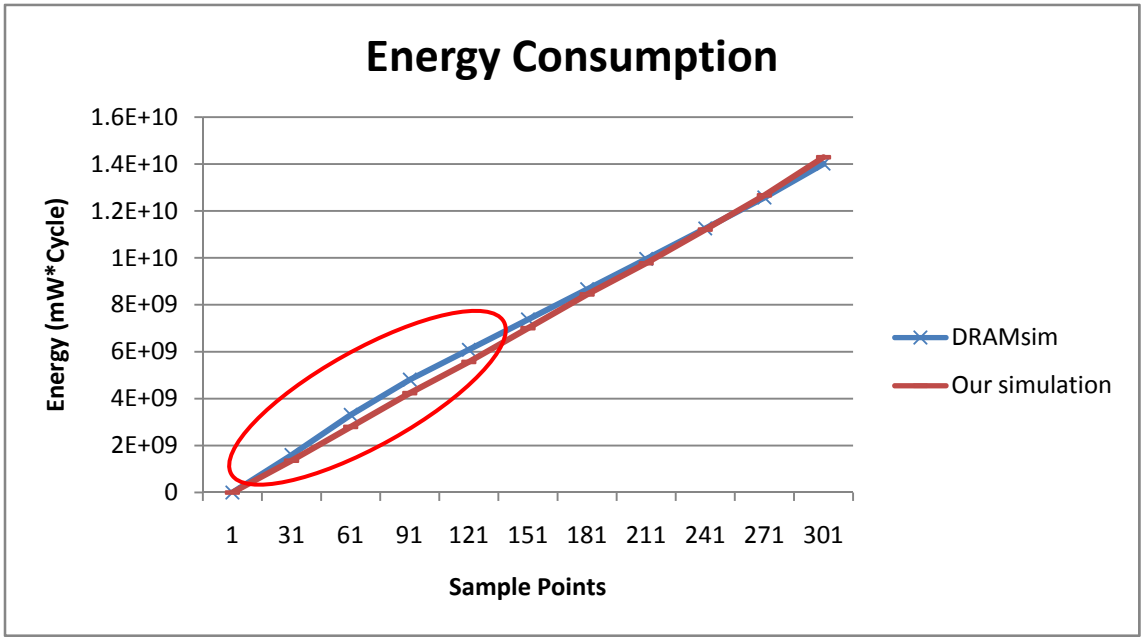


圖表 5-9　stringsearch 累積能量消耗誤差

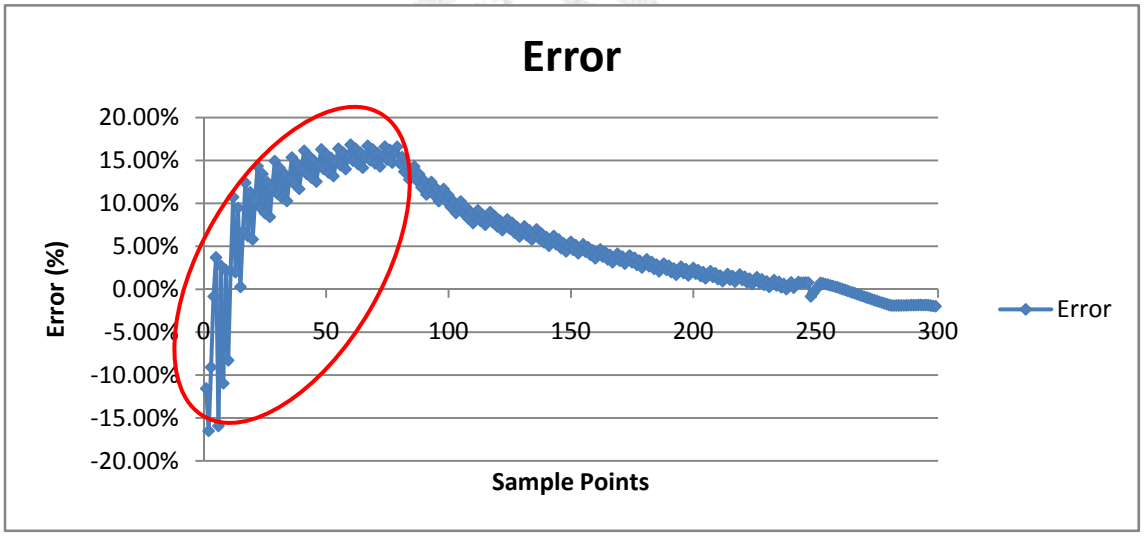
在 susan.corner 這個應用程式也可以看到類似的情形，DRAMsim 與我們的模擬結果比較，在程式剛開始時的功率估量差異較大如圖表 5-10，大約在第 90 個採樣點開始收斂，此時大約為程式執行前三分之一處(圈起處)，圖表 5-11 中可以看見能量消耗預估也在此處差異最大，圖表 5-12 為根據圖表 5-11 計算出的兩者誤差，在程式執行前三分之一時誤差在-20%至 20%之間波動，隨著程式的執行，逐漸收斂，最後約在-3%左右。經過探討以及相關研究指出，透過蒐集及取樣程式特性而建構的模擬量測，發生的現象就如我們的模擬所示，剛開始只是部分採樣所產生的效果，有可能會主宰當時的輸出資訊，待採樣點夠多或是足夠平均時，即可反映真實情形。在我們的模擬環境中，每一個函數就是相當於採樣點，待模擬完全部的函數，即可反映整個應用程式之功耗行為。



圖表 5-10　susan.corner 功率監測比較圖



圖表 5-11　susan.corner 累積能量消耗比較圖



圖表 5-12　susan.corner 累積能量消耗誤差

第6章　結論及未來展望

在此論文中，我們改進一個系統行為層級模擬環境，使之得以快速評估應用程式在不同系統結構上的效能以及記憶體系統功率資訊。模擬速度上，比起精確的週期指令集模擬器 ARM SoC Designer，抽象化應用程式行為與硬體結構作系統層級的模擬有速度上的優勢，平均約有 15 至 25 倍的提升；但是誤差也會隨著硬體架構的複雜化提升，雙核心架構的誤差約達 8.16%，四核心則為 11.78%，雖有些微提升，但仍在可作為參考的範圍之內，這是未來可以探討改進的空間。

另外在功率方面,我們提供函數層級的記憶體系統功率模型與能量相關資訊，並將應用程式裡各函數對記憶體系統的功率值建成功率函數資料庫，在效能模擬的同時，估量應用程式對記憶體系統的功率與耗能，並且評估其誤差，測試的應用程式誤差大約皆低於 6%，可見功率函數資料庫的可用性，

未來，隨著硬體結構的複雜化，軟體程式的多樣化，系統層級的行為模擬挑戰性也日趨上升。例如雙核心與四核心甚至更多核心的行為比起單核心的結構，必定多出許多溝通以及資源分享，包括之前提過的仲裁方針、資料一致性的協議等。因此，後續研究的重點是將這些複雜的資訊抽象化，設計出更多的軟體及硬體模組加入目前的模擬環境，改善其誤差率，提高可伸縮性（Scalability），提供更穩定且可信的模擬環境將是未來展望之一。

其次，功率的部份在本文中討論了記憶體系統部分，尚未提供處理器以及匯流排等相關資訊，未來可將系統其他元件考量入模擬環境中，提供設計者更全面的參考，做更多研究的應用，例如根據 SoC 中各元件所需的功率做最合適的區塊劃分，達到節省成本及能耗的作用等。此外，將函數功率資料庫擴充至微作業系統（Micro-OS）或是更多的應用程式也是未來展望之一，上述實驗可以得知函數功率資料庫的可用性，若應用至嵌入式系統中的作業系統，提供核心中系統呼叫及標準函式庫等的功率資訊，可以進而了解作業系統對於功率及能耗的需求，對

軟體層面所帶來的功耗影響做更近一步的分析及研究。

參考文獻

[1] Kunle, O., Basem, A.N., Lance, H., Ken, W., and Kunyung, C., "The Case for a Single-Chip Multiprocessor", *Proc. 7th International Symposium on Architectural Supportfor Programming Languages and Operating Systems*, vol. 31, no. 9, 1996.

[2] Jose, R., Karin, S., Luis, C., Wei, L., Smruti, S., James, T., and Josep, T., "Thread-Level Speculation on a Cmp Can Be Energy Efficient", in *Proceedings of the 19th annual international conference on Supercomputing*, 2005

[3] Sungjoo, Y., Iuliana, B., Aimen, B., Yanick, P., and Ahmed, A.J., "Building Fast and Accurate Sw Simulation Models Based on Hardware Abstraction Layer and Simulation Environment Abstraction Layer", in *Proceedings of the conference on Design, Automation and Test in Europe - Volume 1*, 2003

[4] In-Cheol, P., Sehyeon, K., and Yongseok, Y., "Fast Cycle-Accurate Behavioral Simulation for Pipelined Processors Using Early Pipeline Evaluation", in *Proceedings of the 2003 IEEE/ACM international conference on Computer-aided design*, 2003

[5] A.Snavely, L.Carrington, and Wolter, N., "Modeling Application Performance by Convolving Machine Signatures with Application Profiles", *Proc. IEEE Workshop on Workload Characterization*, 2001.

[6] M.Mahoney, and T.Elrad, "Modeling Platform Specific Attributes of a System as Crosscutting Concerns Using Aspect-Oriented Statecharts and Virtual Finite State Machines", *the 6th International Workshop on Aspect-Oriented Modeling as part of AOSD05 Chicago,USA*, 2005.

[7] "Simplescalar" http://www.simplescalar.com/

[8] "Rsim" http://rsim.cs.uiuc.edu/rsim/dist.html

[9] Magnusson, P.S., Christensson, M., Eskilson, J., Forsgren, D., Hallberg, G., Hogberg, J., Larsson, F., Moestedt, A., and Werner, B., "Simics: A Full System Simulation Platform", *Computer*, vol. 35, no. 2, 2002.

[10] "SystemC" http://www.systemc.org/home

[11] Chang, H.W., *A Rapid Simulation Environment for Application Performance Estimation on Parameterized Multi-Core/Multi-Threading Architecture Models(2007)*

[12] Enrico, M., Massoud, P., and Fabio, S., "High-Level Power Modeling, Estimation, and Optimization", in *Proceedings of the 34th annual conference on Design automation*, 1997

[13] Paul, L., "High-Level Power Estimation", in *Proceedings of the 1996 international symposium on Low power electronics and design*, 1996

[14] Tony, D.G., Frank, V., J, and rg, H., "Instruction-Based System-Level Power Evaluation of System-on-a-Chip Peripheral Cores", in *Proceedings of the 13th international symposium on System synthesis*, 2000

[15] C. Talarico, J. W. Rozenblit, V. Malhotra, and Stritter, A., "A New Framework for Power Estimation of Embedded Systems", *Computer*, vol. 38, no. 2, 2005.

[16] David, B., Vivek, T., and Margaret, M., "Wattch: A Framework for Architectural-Level Power Analysis and Optimizations", in *Proceedings of the 27th annual international symposium on Computer architecture*, 2000

[17] Ye, W., Vijaykrishnan, N., Kandemir, M., and Irwin, M.J., "The Design and

Use of Simplepower: A Cycle-Accurate Energy Estimation Tool", in *Proceedings of the 37th conference on Design automation*, 2000

[18] "Sim-Panalyzer Project" http://www.eecs.umich.edu/~panalyzer/

[19] Tan, T.K., Raghunathan, A.K., Lakishminarayana, G., and Jha, N.K., "High-Level Software Energy Macro-Modeling", in *Proceedings of the 38th conference on Design automation*, 2001

[20] Tan, T.K., Raghunathan, A., and Jha, N.K., "Energy Macromodeling of Embedded Operating Systems", *Trans. on Embedded Computing Sys.*, vol. 4, no. 1, 2005.

[21] Jinwen, X., Zhaohui, H., and Peixin, Z., "Energy Macro-Modeling of Embedded Microprocessor Using Systemc", in *Electro Information Technology, 2005 IEEE International Conference on*, 2005, pp. 6 pp.

[22] "Intel Vtune Performance Analyzer" http://www.intel.com/software/products/vtune.

[23] "Sun Dtrace" http://www.sun.com/bigadmin/content/dtrace/

[24] "GNU gprof" http://www.cs.utah.edu/dept/old/texinfo/as/gprof_toc.html

[25] "Valgrind" http://valgrind.org/

[26] "ARM SoC Designer" http://www.arm.com/products/DevTools/RealViewSystemDevelopment.html

[27] "Mibench" http://www.eecs.umich.edu/mibench/

[28] Kiran, P., Kyu-Won, C., Jun Cheol, P., Vincent J. Mooney, III, Abhijit, C., and Peeter, E., "System Level Power-Performance Trade-Offs in Embedded Systems Using Voltage and Frequency Scaling of Off-Chip Buses and

Memory", in *Proceedings of the 15th international symposium on System Synthesis*, 2002

[29] David, W., Brinda, G., Nuengwong, T., Kathleen, B., Aamer, J., and Bruce, J., "Dramsim: A Memory System Simulator", *SIGARCH Comput. Archit. News*, vol. 33, no. 4, 2005.

[30] "Micron Technology, Inc" http://www.micron.com/

[31] Gang, Q., Naoyuki, K., Kimiyoshi, U., and Miodrag, P., "Function-Level Power Estimation Methodology for Microprocessors", in *Proceedings of the 37th conference on Design automation*, 2000