SmartARM 處理器微架構之設計與改良

SmartARM-An Improved Microarchitecture Design for ARM Processor

羅宏毅
逢甲大學電機工程系
d8962729@webmail.fcu.edu.tw

王壘
逢甲大學電機工程系
leiwang@ fcu.edu.tw

摘要

承繼之前關於 ARM 架構之研究，本論文進一步的將此架構發展為類似 ARM9 的五階管線(pipeline)處理器: SmartARM。並在考慮過 branch 指令對 pipeline 造成的影響後，加入專用於計算 branch target address 的 fast adder 以改良 SmartARM 的 branch 機制。此外，針對 ARM 指令集中 data processing 指令的特殊 addressing mode: shift [Register]的需求，我們改良 SmartARM 中的 C-BUS 設計，使此類指令的執行週期減短，從而提升其效能。最後分別以 VHDL 實作與軟體數據模擬，進行電路設計驗證與提升效能統計，證明我們所提出的 SmartARM 處理器在不增加過多的硬體負擔下，即可獲得明顯的效能提昇，符合我們的設計理念。

關鍵詞: 嵌入式處理器，管線作業，微架構，指令集架構

Keyword : Embedded Processor, Pipeline, Microarchitecture, Instruction Set Architecture

第一章 前言

隨著電腦軟硬體製作技術的精進及電腦網路的日漸普及，電腦的應用除了傳統的快速資料處理平台外，已另行開展了一條需求更多、處理性質迥異的資訊家電(Information Appliances, IA)領域[1]。在此一領域中，嵌入式處理器(Embedded Processor)配合特殊改良的系統軟體及應用軟體，其發展將主導資訊家電應用的發展與影響層面。

由於嵌入式處理器的設計限制較傳統高效能一般目的處理器嚴格(如其在電力消耗、晶片面積及接腳的限制)，因此已有多家廠商提出了許多有別於傳統處理器的嵌入式處理器架構及產品，而也因為上述的諸多限制，使得這些處理器必須利用較簡單的硬體架構建立較原始而單純的處理引擎，如此自不免連帶的犧牲了許多效能提昇的機會。

ARM 處理器為一RISC 架構的微處理器，原先由 Acorn Computers Limited 在1983年所發展，它是第一顆商業用途所發展的RISC 微處理器。到了90 年代， ARM 公司更進一步的針對ARM 的技術加以開發，使它成為低功率、低花費之嵌入式應用的市場領導者。ARM 嵌入式處理器的市場應用範圍相當廣泛，包括汽車電子、消費性娛樂產品、影像處理、工業控制、高容量儲存、網路、安全系統、以及無線通訊等。其中，在無線通訊部分，ARM 核心目前在全球數位式行動電話中擁有75%的市場佔有率。對於Java 的應用，ARM在2000 年10 月所推出的Jazelle 技術，是ARM 為Java 所設計的微處理器核心延伸，支援在硬體和軟體上直接執行Java 位元編碼(Byte code)。新一代的ARM 整合了記憶體管

理單元(Memory Management Unit,MMU)，並能支援Symbian、Linux 以及Window CE 等主要作業系統。

ARM RISC 架構微處理器系列目前主要分為幾個家族產品，分別為ARM7 Thumb 家族、ARM9 Thumb 家族、ARM10 Thumb 家族、ARM11 家族:

ARM7 Thumb 嵌入式系統微處理器家族主要是針對中低階市場應用所設計的，都採用ARM7TDMI 的微處理器運算核心，管線設計為三階，指令集架構為v4T(支援Thumb 指令集)。[2][3]

ARM9 Thumb 嵌入式系統微處理器家族都採用ARM9TDM 的微處理器運算核心，管線設計為五階，指令集架構支援v4T、v5TE(J)(E 表示支援延伸DSP 指令集、J 表示支援Java application 使用Jazelle 技術)，v4T是支援ARM9TDMI 的微處理器運算核心，v5TE(J)則是支援ARM9E 的微處理器運算核心。[4][5]

ARM10 Thumb 嵌入式系統微處理器家族是ARM 公司針對更強大的多媒體市場及更有效率的即時資料處理所設計。具有六階管線，指令集架構支援v5TE(J)。增加VFP10(Vector Floating-point Coprocessor)以處理浮點數的運算，擁有許多向量化控制的解決方法。[6]

ARM11Thumb 嵌入式系統微處理家族特別適用於支援新一代無線通訊與消費性裝置，其可充分滿足這類型產品對系統效能與低耗電率的需求，包括2.5G 與3G 行動電話、PDA、多媒體無線通訊裝置、以及影像與數位相機等家用消費性產品。包括VoIP 以及寬頻數據機。指令集架構為v6，管線設計為八階，增加新型SIMD 媒體指令、高效能記憶體系統、以及針對未整理資料存取應用的硬體支援，以提升多媒體的效能。

1-1 研究動機

根據我們之前進行的 ARM7 實作研究[7]，發現其簡單的三階管線設計並無法真正有效的達到管線重疊執行(Overlap Execution)的效果。因此為提升處理器效率，在引進ARM-9TDMI 的技術規格後，我們設計此與ARMTDMI 指令集架構相容且具備五階管線架構，並加入中斷處理之完備功能的處理器，名之為" SmartARM"。[8][9][10][11]

SmartARM 架構近似於 ARM-9/Strong ARM，管線設計上由三階擴充至五階。由於管線的加深，在管線危障(Pipeline Hazard)的處理上必須做全盤的檢討並加以克服。同時我們也在此基本架構下，以不造成電路過重負擔為前提，針對使用 shift with register 定址技巧的指令，提出在 C-Bus 上予以改良的方案；而在分支指令(Branch Instruction)方面，也藉由加入一快速加法器，減少 DataPath 在運算分支指令目的位址時所造成的延遲。而在中斷的處理上，為配合 SmartARM 簡潔的管線特性，我們採用精確式中斷(Precise Interrupt)的控制方式完成其中斷處理，其優點為設計考慮上較單純，且硬體負擔較小，符合本研究之精神。

1-2 研究目的及方法

我們由嵌入式處理器之必要的硬體限制為出發，探討將新資訊技術加入嵌入式處理器架構的可能性，透過適度的理論分析及具代表性的媒體基準程式之執行模擬，觀察並檢討各項技術的效能提昇效果，使嵌入式處理器能在既有的硬體考量下產生更大的效能潛力，並且將研究改良後之理論電路架構以硬體實作來驗證其效能提昇。

隨著半導體微電子積體電路技術之進步快速，使得上萬個電晶體可以集成在一小片的矽晶片上，積體電路之技術從 LSI(Large Scale Integrated Circuit)，約數千 Gate 進步到超大型積體電路(VLSI，Very Large Scale Ics)，在單晶片上的電晶體數目可達百萬以上，加上考慮電

路成本及電磁干擾問題(EMI)，所以數位電路設計不再是用傳統 TTL 及畫電路圖的設計方法，而是使用電腦來輔助電路設計(EDA，Electronics Design Automation)加速設計及可靠性。

而 Xilinx Foundation 提供了從設計、模擬、合成到燒錄的完整設計環境，且學術單位可透過 CIC(國家晶片系統設計中心)向 Xilinx 免費申請軟體，並且可在設計後實際燒錄於晶片中，常見的晶片有 Spartan、Virtex、XC4000 系列。

運用硬體描述語言(VHDL)完成電路設計，我們實現此研究的 Gate-level 設計，並利用 Xilinx foundation 軟體模擬驗證其正確性。

第二章 論文相關議題

延續之前所進行的相關研究，本論文所提出之處理器 SmartARM 在指令架構與電路設計上均有不同於 ARMTDMI 的改良。本章將介紹我們在過去研究中已完成的電路成果與加入的特殊指令，並說明 SmartARM 的微架構設計。

2-1 SmartARM 的指令集架構及改良

SmartARM 處理器除了支援 ARM 原有的指令功能外，亦配合之前研究而新增了四個附加指令:一種為支援多媒體運算之平行加乘指令 PMLAV、PMLA[12]；另一種為支援加密查表功能的 Load.extract、Load. ex.eor 指令。[13]

由於傳統 ARM 指令集架構中之乘法指令只提供 32-bits 資料之乘或乘加動作，且一次只處理一筆資料之乘或乘加運算，如果是進行小 bits 數之資料乘或乘加動作，亦是利用位元擴展(Sign-extension)將之擴增成 32-bits 再進行乘或乘加動作，並無針對小 bits 數資料型態，如 16-bits 之多媒體資料的平行乘加動作，所以對於大量的多媒體資料進行乘加動作之速度及效率相當的有限，對於目前及未來要求即時的(real-time)、大量的多媒體資料處理能力尚有相當大的改進空間，所以我們新增兩個 Parallel SIMD MAC 指令(PMLAV、PMLA)，以改善多媒體資料的處理效能。

新增之指令不同於傳統 ARM 乘加指令 MLA 針對 32-bits 資料進行乘加的動作，它加入 SIMD(Single Instruction Multiple Data)平行處理的技巧，可同時平行處理兩筆 16-bits 多媒體資料的乘加動作，達到可即時的處理大量的多媒體資料乘加運算，所以可以增進多媒體資料的處理效能。

Load.extract 指令則能將索引的展開、索引 Scaled、以及記憶體的存取合併至單一指令內。如果在既有硬體架構下，能夠不增加額外的執行延遲(Latency)，可以顯而易見這類指令必然會提昇執行之效能。新指令可將有效位址的計算合併為一個指令，因為是針對 ARM 的架構，所以固定對 32-bits 的值做 Scaled，也就是左移 2 個 Bits。這個指令的動作除了將資料由實際記憶體位址讀進 Rd 外，最重要的是還包括了有效位址的計算，將 Ra 的第 n 個 Byte 取出，再向左位移兩個位元，最後加上 Rb 的基底位址，以得到有效位址。

由於加解密演算法中，Load 的資料往往緊接著會作 XOR(eor)運算，如果能與 Load 合併運算，則更能進一步提昇加密效能，考量 ARM7 處理器之 DataPath 及 Pipeline 設計，我們發現該 XOR 動作可加入 ARM 原有的執行路徑且不影響整體微架構之設計，因此本研究亦提出了第二個名為 Load.ex.eor 之新增指令。而在我們的 SmartARM 處理器中，為支援此新增指令，我們亦在不增加過多電路下的前提下對 datapath 做出修正。

2-2 SmartARM 微架構

SmartARM 處理器，其微架構如圖 1 所示，主要可分為以下八大部份：

(1) 暫存器組(Register Bank)：暫存器組負責儲存微處理器所需資料，其中暫存器至少具有三個讀取埠(Read Port)與二個寫入埠

(Write Port)以供存取暫存器之用。

(2) 乘法器(Multiplier)：使用 8-bits 的 Booth's algorithm 實現整數乘法，如此一來，乘法器每個週期可以做 32*8 的乘法。而承繼我們之前關於乘法器的研究，SmartARM 處理器將具有 SIMD 的平行乘法功能

(3) 桶型位移暫存器(Barrel Shifter)： barrel shifter 負責執行位元位移或旋轉，且支援指令在做 ALU 運算前先完成 shift 運算。

(4) 算術邏輯運算單位(ALU)：ALU 負責執行算術與邏輯運算

(5) 位址暫存器(Address Register)與累加器(Incrementer)：我們利用位址暫存器與累加器來選擇並維持記憶體定址功能，產生連續或非連續的記憶體位址。

(6) 指令暫存器/資料暫存器(Instruction Register/ Data Regsiter)：SmartARM 處理器中將記憶體儲存指令的部分與儲存資料的部分分開，分別有其獨立的 BUS 接線，並利用簡易的 latch 功能進行讀寫資料時的暫存。

(7) 指令解碼器（Instruction Decoder）：指令解碼器用以產生與指令相對應的控制信號。

(8) 資料位址累加器（Address Incrementer）：為因應連續讀寫指令(LDM/STM)之需求，利用此自動累加器獲得此類指令所需之位址。

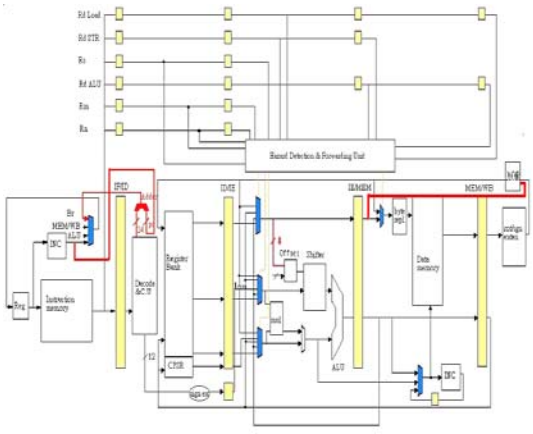

圖 1: SmartARM 微架構

除了上述的八大部份外，SmartARM的暫存器結構則與原先的ARM系列大致相同，CPSR(Current Program Status Register)為狀態暫存器，專門儲存旗標狀態(Flag bits)、中斷禁能狀態(Interrupt disable bits)、和執行模式(Mode bits)的暫存器，並且在不同的Mode下，有不同的SPSR(SaveProgram Status Register)，提供儲存CPSR 的功能。此外，不同的mode有其專屬的Banked Registers，是其他mode 下不能存取的，目的就是為了達到快速的內容值交換(Context Switch)，這是設計Banked Registers 的最大優點。

SmartARM 處理器採用在 ARM9 之後的版本所提出的五階管線(圖 2、圖 3)，五階段分別為:Fetch、Decode、Execute、Memory Read/Write、Write Back。

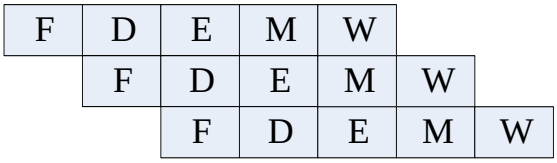

圖 2:SmartARM single-cycle instruction pipeline operation

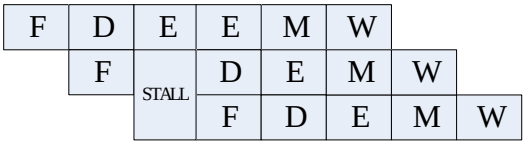

圖 3: SmartARM multi-cycle instruction pipeline operation

當管線由 ARM7 的三階加深至五階時，將使結構危障(Structure Hazard)、資料危障(Data Hazard)以及控制危障(Control Hazard)同時出現。在 SmartARM 的架構中，採取硬體鎖定(stall)的方式處理結構危障。控制單元(Control Unit)在進行多週期指令執行時，強制暫停可能產生資源競用的接續指令至該指令完成，再繼續執行後續指令以克服結構危障；在 DataPath 中加入 Forwarding 單元的電路設計，以完成資料危障之克服；此外我們藉著在控制單元內簡易的 Flush 電路，克服控制危障。

2-3 SmartARM 的中斷設計

SmartARM 的中斷觀念是當中斷發生，程式正常流程必須暫時被停止，目前的處理器狀態被保存，然後進入對應該種中斷的模式，再將程式流程指向處理中斷的中斷服務程式。當中斷服務程式完成後，再回復原先保存的處理器狀態並返回中斷前的程式流程。

SmartARM 之中斷系統，具有一中斷控制單元與一中斷偵測單元。中斷控制單元將於處理中斷時暫時取代CU控制datapath 以完成中斷處理的動作，而中斷偵測單元負責接收中斷訊號，排定中斷優先權順序並傳送正確中斷種類至中斷控制單元，再由中斷控制單元完成指令流之變更、原始程式狀態的保存以及mode 的切換。當mode 切換完成且進入中斷處理程式之後，再將datapath控制權回復至CU，由CU 執行中斷處理程式以處理中斷。

中斷偵測單元之架構如圖4所示。SmartARM 架構下的中斷來源可分為內部中斷與外部中斷二大類，其中SWI/BKPT/Undefined Instruction 這三種為內部指令的問題，由CU 產生訊號至中斷偵測單元，而Reset/FIQ/IRQ/Abort 這四種為外部區塊發生問題，中斷偵測單元須經由外部訊號線得知。當任何一種中斷發生時，中斷偵測單元將保存此中斷訊號，再經由Priority Encoder 比較其優先順序，其結果則輸出至中斷控制單元以進行中斷處理。若是複數的中斷訊號同時被偵測到，優先權較低的中斷則被保留至前一中斷處理完成時再予以處理。

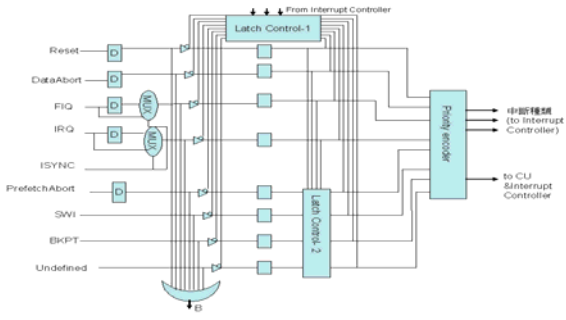

圖4:中斷偵測單元架構圖，其中latch control控制各中斷訊號線之輸入，priority encoder判斷優先權

中斷控制單元內部結構如圖5所示，Interrupt receiver 用來接收中斷訊號，Signal Generator依據Counter、中斷編號產生在完成中斷前置工作時，每個Cycle 下所需的控制訊號。而Normal Signal Unit 以及Change Signal Unit，分別發送正常程式流下的訊號或剛觸發中斷時的訊號。

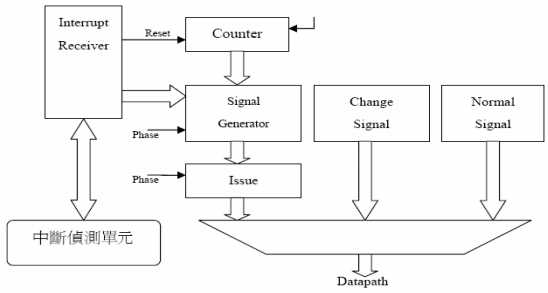

圖5:中斷控制單元架構圖，在收到中斷訊號後產生對應的控制訊號

在了解SmartARM的設計規格後，我們分別以VHDL語言完成相關的電路設計，以Gate level的層次將SmartARM實作完成。而在下一章中，將針對SmartARM不同於ARMTDMI規格的電路改良部分加以介紹。

第三章 SmartARM 微架構之改良

除了前一章所述，SmartARM 在指令集架構上針對 ARM 指令集所做的改良與五階管線微架構及電路設計外，本論文透過細心的觀察及完整的設計考量，發現原本 ARMTDMI 架構的五階管線可作諸如附加 fast adder 以提早完成分支指令的執行，改進 C-Bus 的功能使 R-R type 的移位運算指令提早一個週期完成。

3-1Branch fast adder

就管線技術的觀點而言，由於分支指令(branch) 需計算分支位址(branch target address)，使管線延遲抓取(fetch)到正確指令的

時間，造成控制危障(control hazard)。以 ARM9 的五階管線架構為例，如圖 6，分支指令發生時，分支位址必須在 execution 階段透過 ALU 運算獲得，導致下一指令的抓取必須延遲兩個週期(Cycle)才能抓取到正確的指令並使管線效能損失。由於分支指令的執行結果取決於正確分支位址的取得時間，若在 decode 階段提前完成分支位址的運算，則分支指令將可減少一個週期的代價(penalty) ;但若分支指令提前在 decode 階段使用 ALU 運算分支位址，管線將產生結構危障。因此，為在不發生結構危障的前提下改善控制危障，我們另行加入一組快速加法器，取代 ALU 計算分支位址。

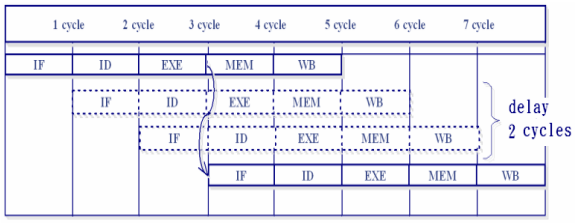

圖 6：分支指令在 ARM-9 管線架構中的執行流程

3-1.1 設計規格定義

在 ARM 的指令集中，branch 指令共有兩種：B、BL，指令格式如圖 7，其中 0-23bit 為 branch 位移量，L bit 代表此指令為 B 或 BL。ARM9 處理 branch 指令的流程如下：Fetch 階段時 CU 由記憶體中讀出指令; Decode 階段中辨識出指令為分支指令並將 24 bit 立即值 sign-extension 成 32 bit 以產生分支所需的位移量;execution 階段時 CU 將位移量與 program counter 中所儲存的 pc 值透過 ALU 進行相加，得到分支位址。由於 ARM9 的管線設計與電路架構所致，此時 program counter 中所存的 pc 值將為 branch 指令的指令位址+8。若執行指令為 BL 時，則在計算分支位址並修改位址暫存器之外，尚需將接續於 BL 指令之後的指令位址存入 Link Register(R14)中。

31 28	27 26	25	24 23	0
cond	**101**	**L**	**Signed_immed_24**	

圖 7: B、BL 指令格式

我們注意到 branch 指令的分支位址主要經過兩次運算產生：位移量的有號擴充與跟 PC 的相加，而在 ARM9 的設計中，使用 ALU 完成位移量與 PC 相加的運算導致分支指令需等待至 execution 階段結束方能獲得分支位址。為減少分支指令延遲，觀察 ARM 的分支指令執行流程後，本論文加入專門用於處理分支指令的快速加法器，在 decode 階段提前完成分支位址的運算。考慮到分支位址的計算僅使用簡易的加法，我們不需複製完整的 ALU，一組 32 bit 的加法電路即可完成此 fast adder 的設計。

3-1.2Data path 上的改變

新增快速加法器電路前，必須先考慮原先 ARM9 架構的 branch 相關電路設計。ARM9 指令集中，除 B 指令之外，data processing 指令與 Load 指令也具有修改 address register 以改變指令流程的功能。因此在原始的 ARM9 架構中，address register 需由 CU 控制多工器選擇下一週期的輸入為正常累加、ALU 運算結果或由記憶體 Load 所得結果，如圖 8 所示。新增快速加法電路後，branch 指令的位址輸入將不再由 ALU 獲得，而改由快速加法器產生。因此，在 address register 的部份將新增一條由快速加法器產生的輸入結果供 branch 指令改變指令流程使用。同時為配合 ARM9 的設計規格，快速加法器電路中的兩個來源運算元將固定由 CU 與 increment 電路提供，確保第二運算元為 branch 指令的指令位址+8，而運算完成的結果直接連接至 address register 供後續指令使用，如圖 9。

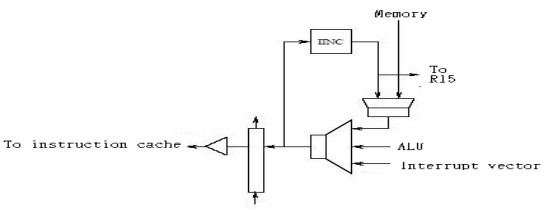

圖 8: ARM9 address unit data path

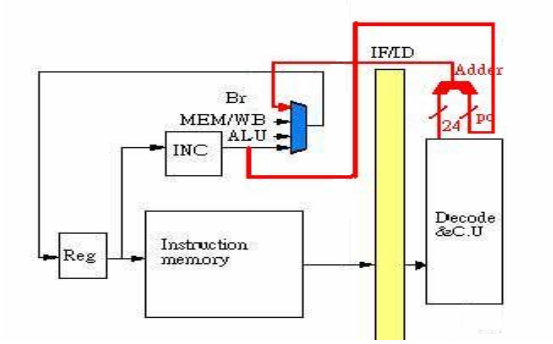

圖 9: 加入快速加法器後的 data path 相關區塊圖

3-2 C-Bus 改良

在 ARM 指令集中，運算元進入 ALU 中運算前，尚可藉由桶型位移器(barrel shifter)進行移位處理，其位移量與位移方式一般直接編碼在指令內，控制單元可在一個週期內完成指令解碼與位移量產生。但特殊的位移編碼格式 shift with register 中，位移量與位移方式存在指令編定的暫存器內，需要一個額外的週期將暫存器值讀出以進行位移量產生，如圖 10。

我們在觀察 ARM9 的暫存器組後，由於 ARM9 的暫存器組(register bank)中具有專為 store 指令使用連接至 C-BUS 的讀取埠，而此讀取埠在執行 data processing 指令時不會用到。本研究修改原本閒置的 C-BUS，並配合快速產生位移量的解碼電路，藉以減少執行 shift with register 動作時所需延遲。

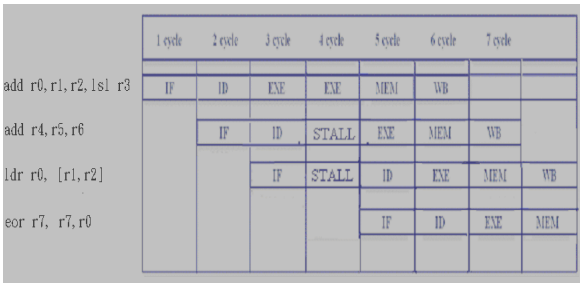

圖 10: shift with register 格式指令在 ARM-9 管線架構中的執行流程

3-2.1 設計規格定義

Data processing 指令共有 11 種定址法，如表 1

表 1 Addressing Mode I

	#<immediate>	Immediate
2.	<Rm>	Register
3.	<Rm>, LSL #<shift_imm>	Logical shift left by immediate
4.	<Rm>, LSL <Rs>	Logical shift left by register
5.	<Rm>, LSR #<shift_imm>	Logical shift right by immediate
6.	<Rm>, LSR <Rs>	Logical shift right by register
7.	<Rm>, ASR #<shift_imm>	Arithmetic shift right by immediate
8.	<Rm>, ASR <Rs>	Arithmetic shift right by register
9.	<Rm>, ROR #<shift_imm>	Rotate right by immediate
10.	<Rm>, ROR <Rs>	Rotate right by register
11.	<Rm>, RRX	Rotate right with extend

格式 1、2 為最基本的 data processing 指令，第二運算元直接由 CU 獲得立即值或由 register bank 讀取暫存器值(Rm)，不額外使用 barrel shifter，執行週期為 1 cycle。格式 3、5、7、9、11 的執行模式亦不複雜，CU 透過指令定址方法判斷位移方向後，直接產生位移量控制 barrel shifter 將 Rm 進行位移，再與 Rn 做 ALU 運算，執行週期亦為 1 cycle 。其餘的格式即為 shift with register，CU 需耗費 1 cycle 讀取 Rs 暫存器以產生位移量，在下一個週期方能完成 Rm 移位與 ALU 運算，共需 2 cycle 的執行週期。

shift with register 編碼格式額外所需的週期為 B-BUS 讀出 Rs 並產生位移量之用，且此週期中僅由 register bank 將 shift register 資料傳入解碼電路以獲取位移量，其餘 function unit 皆未受到使用。由於 ARM9 架構中為處理 store 指令而將 register bank 的讀取埠增加為三個，使單一指令在 execution 階段最多可同時獲取三組暫存器資料。我們發現，若使用此新

增的讀取埠將 shift register 資料讀出並另以小型解碼電路產生位移量，指令將可在一個週期內完成執行。修改第三組讀取埠所連接的 C-bus 後需配合位移量解碼電路，由於此解碼電路只負責根據 shift register 中的最低 8 bit 資料產生位移量及 barrel shifter 的 carry out 控制訊號，電路設計將不至於太過複雜。

3-2.2 Datapath 上的改變

我們首先考慮讀取埠的改變所造成的影響:在 ARM9 的設計中，連結至 C-BUS 的讀取埠為 store 指令在 execution 階段傳遞資料所用，使用時間為一個週期。由於 store 指令本身並不支援此 shift with register 定址技巧，且修改後的 shift with register 格式指令與 store 指令均只需一個週期的 execution，可確定不會因資源競用的問題而對 pipeline 造成不必要的傷害。另外一個考量是 barrel shifter 的位移量來源除了位移量解碼電路之外，還有一般指令經 CU 解碼後所獲取的位移量。因此我們在 barrel shifter 的位移量輸入之前加上一個由 CU 控制的多工器選擇是否使用解碼電路的輸出結果，確保其餘的位移格式可以正確的運作。圖 11 為 ARM9 的原始電路，圖 12 則為修改後的電路設計，其差異僅在 Rs 的輸入方式由 B-BUS 改為 C-BUS。

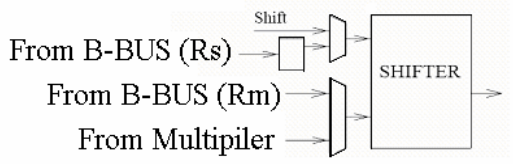

圖 11: ARM9 的 barrel shifter 設計，shift register 需由 B-BUS 讀出，透過解碼電路產生位移量

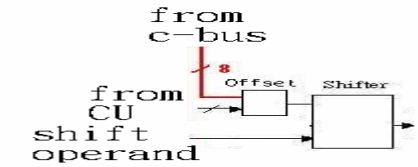

圖 12:修改後的 barrel shifter 設計，shift register 改由 C-BUS 讀出

3-3 Smart ARM 微架構

在加入以上的電路修改後，我們提出此與 ARM9 指令集相容的電路架構 SmartARM，如圖 3。除原有 ARM9 架構中之功能電路與我們加入的改良外，為解決資料危障中 RAW(Read After Write)問題，我們亦導入 forwarding 電路設計。藉由偵測執行指令的目的暫存器編號與來源暫存器編號，forwarding 電路在 RAW 發生時可以將已完成運算但尚未寫回 register bank 的資料前饋至其他指令提供執行使用。完整的 SmartARM 架構除了可以支援 ARM9 指令集架構所需的各種功能，配合我們所提出的兩項改良設計，更可以在分支指令與 shift with register 指令的執行上獲得效能提升。在下一章中，我們將以 xilinx 軟體驗證我們的電路設計正確性，並以軟體模擬統計本研究的效能提升。

第四章 模擬評估

為了驗證本論文的電路設計與評估其效能提升，我們利用 VHDL 語言進行 SmartARM 處理器之實作，並以 xilinx 4.2i 軟體驗證其邏輯正確性。同時藉由 SimpleScalar LLC 所提供的模擬軟體，評估我們所提出的 SmartARM 設計所能提昇的效能。

4-1 電路設計驗證方法與流程

本論文所提出的 SmartARM 處理器，是使用 VHDL Code 以 Gate Level 所設計而成。由於指令集是 32 位元的長度，因此指令的格式繁多，再配合不同的定址法，產生出來的模擬指令數將相當的多，因此在驗證方面，我們僅列出與本研究相關的改良部分。

我們使用的設計軟體為 CIC(國家晶片系統設計中心)教育中心所提供的 Xilinx Foundatuon 4.2i，軟體內有完整的設計與模擬驗證流程如圖 13，其步驟如下：

=> (1) VHDL Code 設計

=> (2) 功能上的驗證模擬 (Function

Simulation)

=> (3) 電路合成 (Synthesis)

=> (4) 佈局與繞線 (Place & Route)

=> (5) 時序驗證模擬 (Timing Simulation)

=> (6) 晶片燒錄 (Programming)

我們透過軟體的幫助，對我們所設計的各個邏輯元件以至整個 CPU 單元進行功能和時序的模擬驗證。其中關於各功能單元的部份，我們已在 91 年度的相關研究中進行過模擬驗證，故本論文中將不再贅述，而以本研究相關的電路改變為主。

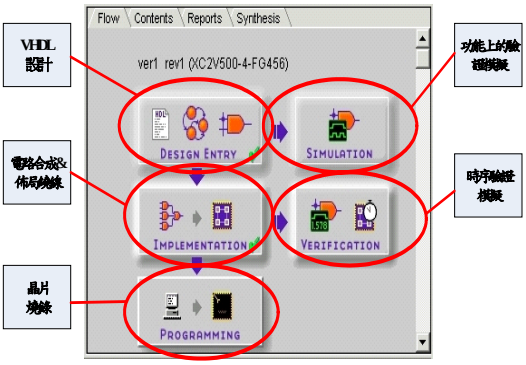

圖 13: 設計&驗證流程

在驗證方法上，我們使用之前研究中所整理的「指令格式集和控制訊號表」，做為模擬時的輸入端資料庫，並製作一個輸出端預期資料庫，然後與經由模擬出來的輸出端波形進行比對，來驗證其正確性，再經過不斷的除錯、修改，再反覆模擬驗證，以求得到最接近實際電路的設計。以下列舉兩個模擬驗證的結果。圖 14 為加入快速加法器後 CU 相關的控制訊號之軟體模擬驗證結果，輸入 CU 之 B 指令機器碼為 6A800000，其中 6A 為 opcode 部份，0X800000 為分支位移量。由於加入快速加法器後 B 指令不再使用 ALU 進行運算，CU 僅需對快速加法器提供位移量並控制 address register 之選擇，其餘的功能單元則需保持無動作。圖 15 為 C-Bus 改良後所需的解碼電路驗證圖。Shifter with register 指令格式的位移/旋轉方式可分為邏輯左移、邏輯右移、算數右移與向右旋轉四種，可由 CU 的解碼單元判斷，故解碼電路僅負責產生位移量與 carry out 的輸出選擇。而解碼電路根據位移的方式與位移量暫存器 Rs 所提供的 8-bit 輸入，方可產生正確的位移量。

經由 xilinx 4.2i 的軟體模擬，驗證了我們的電路設計確實符合我們的要求。修改後的電路雖較原本的設計複雜，但增加的延遲與電路負擔皆不至於影響整體效能。

4-2 軟體模擬與效能評估

本研究除以VHDL語言完成電路設計外，另以ARM的模擬程式配合Benchmarks評估改良後的電路所能提升的效能。在使用模擬器執行各Benchmark程式後，我們可以統計出指令的總執行數與各指令在總執行指令數中所佔的比例，並計算電路改良後各可增進的效能。本研究所使用的Benchmark：Mibench ，是由密西根大學電機工程計算機實驗室（The University of Michigan Electrical Engineering and Computer Science ）以 EEMBC (EDN Embedded Microprocessor Benchmark Consortium)為基礎，針對嵌入式系統及提供學術研究上用途所發展的Benchmarks。

Mibench 共將 benchmarks 分為 Automotive、Industrial Control、 Consumer Devices、Office Automation、Networking、Security、Telecommunications 六大類，各類Benchmarks如表2示之。

本研究從表中所列出的Benchmark中，在各分野內選取較具代表性的數個Benchmark：bitcount、quicksort、jpeg encode/decode、dijkstra、blowfish encrypt/decrypt、rijndael encrypt/decrypt、CRC32、FFT/IFFT等，進行模擬分析。在模擬程式方面，本研究採用 SimpleScalar LLC 所提供之 SimpleScalar Version 4.0 test released－SimpleScalar/ARM simulator作為模擬平台，將各Benchmark輸入之後統計其執行結果，俾以評估我們改良後的電路可提升的效能。

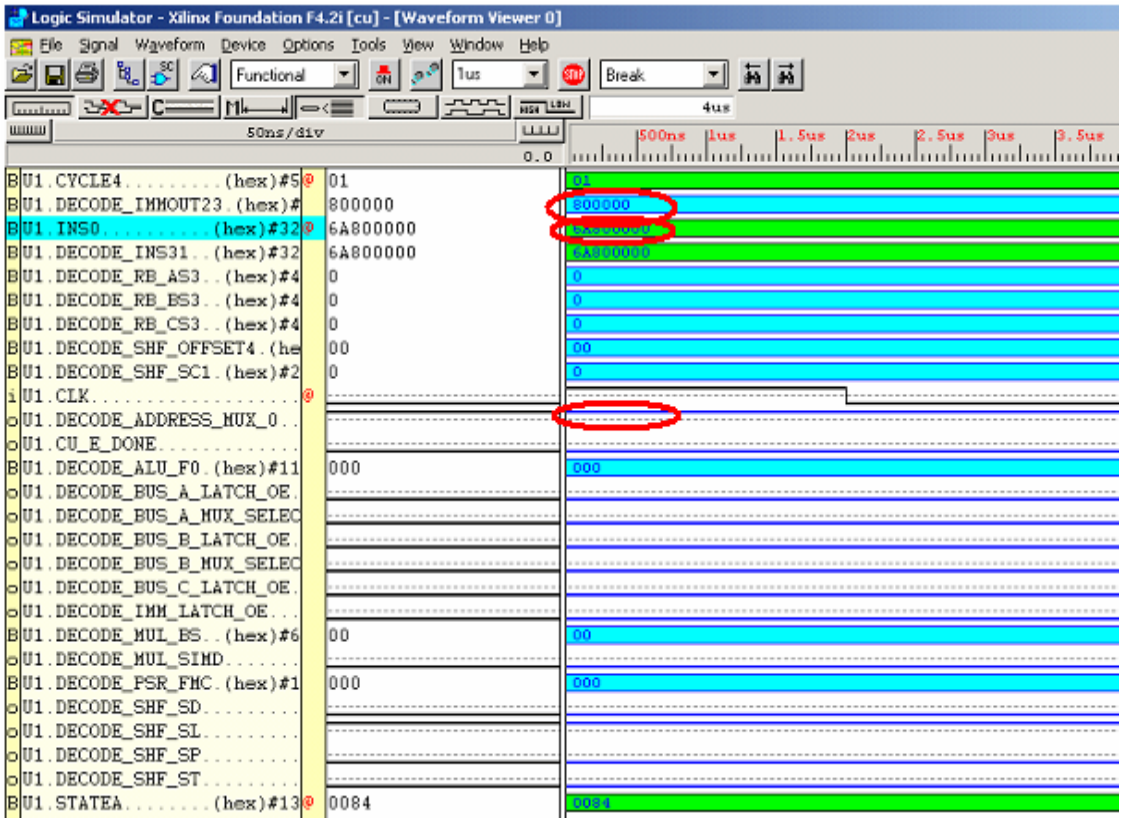

圖 14: 加入 fast adder 後的 CU 解碼電路，主要控制訊號為位移量輸出與 address register 控制

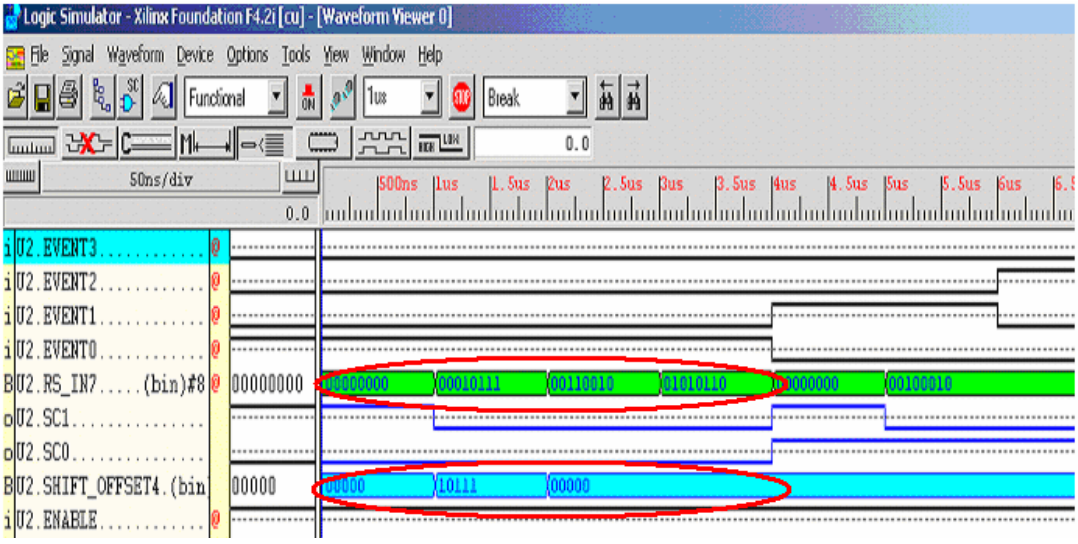

圖 15: 改良 C-BUS 後 shifter 所需之解碼電路，相同的旋轉方式在 Rs 輸入不同時，其位移量亦有所不同

表2 : Mibench 分類

Auto/Industrial	**Consumer**	**Office**
basicmath bitcount	Jpeg enc Jpeg dec	ghostscript ispell
quicksort	lame	rsynth
susan(edges)	mad	sphinx
susan(corners)	tiff2bw	stringsearc
susan(smoothing)	tiff2rgba	
	tiffdither	
	tiffmedian	
Network	**Security**	**Telecomm**
dijkstra	blowfish enc	CRC32
patricia	Blowfish dec	FFT
(CRC32) (sha)	pgp sign pgp verify	IFFT ADPCM
(blowfish)	rijndael enc	ADPCM
	rijndael dec	GSM enc
	sha	GSM dec

SimpleScalar 模擬環境最初是由 Todd Austin 於攻讀 University of Wisconsin - Madison博士學位時所開發，可供使用者針對於所需的硬體架構自由的修改。此外，亦提供專為特殊用途或一般常用模擬工具供使用者直接使用或修改後使用。為了評估SmartARM對分支指令執行的效能提昇，本研究使用Sim-outorder模組模擬分支指令執行電路改良的效能提昇。我們首先將相關參數設定為ARM9-like的系統，並排除cache等不相干的因素，藉以比較加入fast adder後，branch penalty由1降至2時的效能差異。模擬結果顯示branch penalty 降至 1 時整體平均IPC(Instructions per cycle)提升了4.338634%，如圖16所示。

而在 shifter 與 C-BUS 的改良方面，由於改良後的電路在執行 shift with register 指令時，可以減少一個週期的浪費，我們僅需統計在各 benchmark 中使用 shift with register 定址技巧的指令所佔的比例，便可估算改良後的電路所能提升的效能，如圖 17。

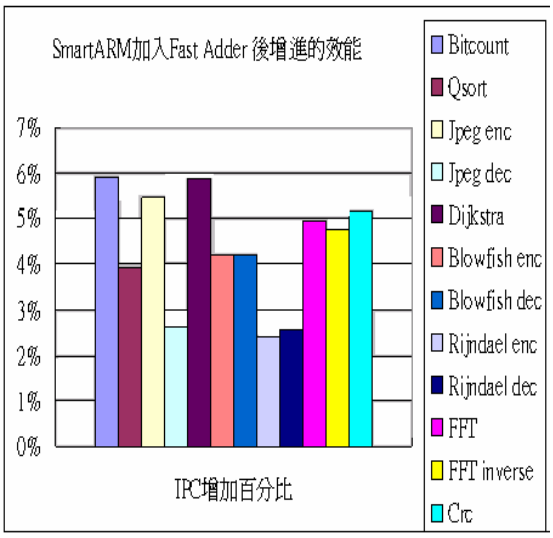

圖 16:分支指令執行電路改良之效能提昇

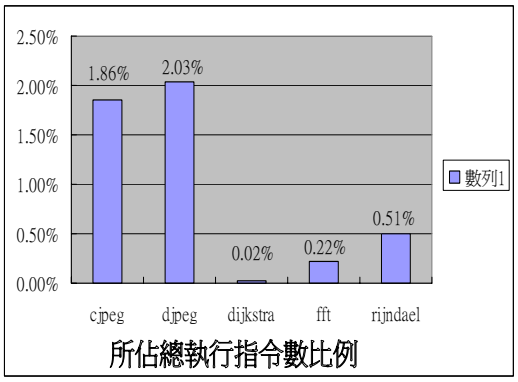

圖 17:各 benchmark 中 shift with register 指令之動態執行次數百分比

我們注意到使用 shift with register 定址技巧的指令主要應用在 djpeg/cjpeg benchmark 中，可預期改良 C-BUS 與 shifter 後的 SmartARM 架構將對類似 djpeg/cjpeg benchmark 行為特徵的應用程式有相當的效能提升。

由於 SmartARM 處理器支援我們在之前的研究中所提出的新增指令並進一步透過微架構的改良，使分支指令與 shift with register 指令上產生明顯的改進，使相關的程式應用能獲得更大的效能改善。新增的 Load.extract 與 Load.ex.eor 指令已被證明能在資料加解密部

分有效減少指令執行數目，而導入 SIMD 技術的平行乘加指令在多媒體程式的執行上亦可帶來 10%~20%的效能提昇。至於一個 fast adder 的加入，能直接將分支指令的延遲代價由二個 cylce 減少成一個 cycle，產生 2.5%~6%的效能改善。而僅僅是 C-BUS 的連線改善，亦可使影像處理的應用提升 1.8%以上的效能。這些效能評估的統計，不僅證明了本研究所設計之 SmartARM 的有效性，更激勵了本研究成員的信心，相信藉由對指令集架構/處理器微架構的細心檢討及改良，即使如目前十分成熟普遍的 ARM 處理器，也能在維持相容性，不增加大量成本的前提下，產生明顯而有效的效能改善。延續之前的研究成果，我們所提出的 SmartARM 處理器在不增加過多的硬體負擔下，可以獲得明顯的效能提昇，符合我們的設計理念。

第五章 結論

ARM7簡單的三階管線設計並無法真正有效的達到管線重疊執行(Overlap Execution)的效果。因此為提升處理器效率，在引進ARM-9TDMI的技術規格後，本論文設計一個與原有ARM7指令集架構相容，又具備ARM-9架構特性，並加入中斷處理之完備功能的處理器，名之為”SmartARM”。並在考慮過branch指令對pipeline造成的影響後，加入專用於計算branch target address的fast adder以改良SmartARM的branch機制。此外，針對ARM指令集中data processing指令的特殊addressing mode: shift [Register]的需求，我們改良SmartARM中的C-BUS設計，使此類指令的執行週期減短，提升其效能。最後分別以VHDL實作與軟體數據模擬，進行電路設計驗證與提升效能統計，證明我們所提出的SmartARM處理器在不增加過多的硬體負擔下，即可獲得明顯的效能提昇，達到本研究“以最少量的硬體成本，擴充處理器能力以因應未來之應用趨勢”的設計目的。

由於SmartARM處理器支援我們在之前的研究中所提出的新增指令，並進一步透過微架構的改良，使分支指令與shift with register指令上產生明顯的改進，使相關的程式應用能獲得更大的效能改善。新增的Load.extract與Load.ex.eor指令已被證明能在資料加解密部分有效減少指令執行數目，而導入SIMD技術的平行乘加指令在多媒體程式的執行上亦可帶來10%~20%的效能提昇。至於一個fast adder的加入，能直接將分支指令的延遲代價由二個cylce減少成一個cycle，使IPC產生2.5%~6%的效能改善。而僅僅是C-BUS的連線改善，亦可使cjpeg/djpeg之類的程式應用提升1.8%以上的效能。

目前本研究仍持續進行中，在後續的研究中，我們的目標將朝向引入ILP(Instruction Level Parrellism)技術[14]使SmartARM處理器更加有效率，同時將在評估現有的Branch Predication技術後[15]，在SmartARM中加入適合的Branch Predication機制，以去除控制危障(Control Hazard)。

誌謝

本研究接受國科會計畫編號 NSC93-2213-E-035-020 之部份經費補助

參考文獻

【1】 G. Martin and F. Schirrmeister, “A design chain for embedded systems,” Computer,Vol.35, Issue 3, pp.100 –103 , March 2002.

【2】 ARM7TDMI Technical Reference Manual, Document Number: ARM DDI0210B, ARM Limited 2001.

【3】 ARM7 Data Sheet, Document Number: ARM DDI0020C, ARM Limited 1994.

【4】 ARM9TDMI Technical Reference Manual, Document Number: ARM DDI0180A, ARM Limited 2000.

【5】 S. Segars, “The ARM9 family-high performance microprocessors for embedded applications,” International Conference on Computer Design: VLSI in Computers and Processors, 1998, pp.230 –235.

【6】 ARM1022E Technical Reference Manual, Document Number: ARM DDI0237A, ARM Limited 2001.
【7】 范佐毅,王壘 ”具特殊指令支援的ARM 處理器之電路設計”中華民國九十二年全國計算機會議(NCS2003).(NSC 91-2213-E-035-020)
【8】 David Seal, Architecture Reference Manual, 2nd Edition, Addison Wesley Longman Inc, 2000.
【9】 A. van Someron and C. Atack, The ARM RISC Chip : A Programmer's Guide, Addison-Wesley, Wokingham, England, 1993.
【10】 Stephen B. Furber, Steve B. Furber, ARM System-on-Chip Architecture, 2nd Edition, Addison Wesley Longman Inc, 2000.
【11】 Steve Fuber, ARM System Architecture, Addison Wesley Longman Inc, 1996.
【12】 L.Wang, Leo Fang , and H.Y. Hsu "Equipping the SIMD MAC operations into embedded RISC processor," proceeding of the international computer symposium , 2002 . (NSC-90-2213-E-035-028)
【13】 Chia-Hua Liu, Larry Wang, Hong-Yang Hsu," Enhancing the Performance of Encryptionfrom the View of ISA," 14th National Information Security Conference2002 (ISC2002), pp. 265-271, Taiwan, May 2002 .(NSC-90-2213-E-035-028)
【14】 J.L. Hennessy and D.A. Patterson, Computer Architecture: A Quantitative Approach, Morgan Kaufmann, SanFrancisco, CA, 1996
【15】 M. Evers and T.-Y. Yeh "Understanding Branches and Designing Branch Predictors for High-Performance Microprocessors" , proceedings of the IEEE Vol. 89, No.11, November