

國立交通大學

電機與控制工程學系

碩士論文

低功耗相位式快取記憶體

之高效能管線設計

High-Performance Pipeline Design
for Low-Power Phased Cache

研究生：薛智文

指導教授：周志成 博士

林進燈 博士

中華民國九十六年七月

低功耗相位式快取記憶體

之高效能管線設計

High-Performance Pipeline Design

for Low-Power Phased Cache

研究生：薛智文

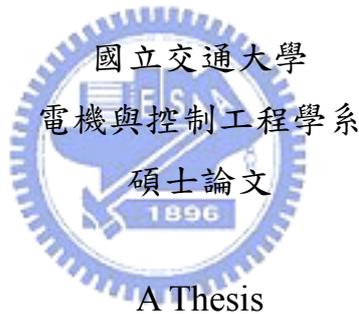
Student : Chih-Wen Hsueh

指導教授：周志成

Advisor : Dr. Chih-Cheng Chou

林進燈

Co-advisor : Dr. Chin-Teng Lin



Submitted to Department of Electrical and Control Engineering

College of Electrical and Computer Engineering

National Chiao Tung University

in Partial Fulfillment of the Requirements

for the Degree of Master

in

Electrical and Control Engineering

July 2007

Hsinchu, Taiwan, Republic of China

中華民國九十六年七月

低功耗相位式快取記憶體

之高效能管線設計

High-Performance Pipeline Design

for Low-Power Phased Cache

學生：薛智文

指導教授：周志成、林進燈教授

國立交通大學電機與控制工程研究所

中文摘要

隨著可攜式產品的應用日趨廣泛及全球能源觀念抬頭，低功率消耗成為微處理器研究中重要的一環。而快取記憶體佔了處理器一半左右的面積及功率消耗，依據 Amdahl 定律，如果能改善快取記憶體的功率消耗，就能帶給處理器可觀的低功率消耗的效果。快取記憶體一但失誤，需要花費大量的時間及能量跟外部記憶體存取，因此命中率嚴重影響效能及功率消耗，快取記憶體中為了提升命中率而增加的集合關聯性設計(set associative cache)，卻會造成多餘的功率消耗，因此相位式快取記憶體設計能去除不需要的集合關聯式記憶體存取，改善這個缺失，但卻會造成雙倍存取時間的缺失，造成效能下降。

本論文針對相位式資料快取記憶體，改良處理器管線，加速讀取指令其運作，以分擔相位式快取記憶體中的 Tag 存取相位，改善其耗時雙週期的缺點，去除管線暫停，只增加 6% 成本就達到 44% 低功率消耗又不增加時間負擔的設計理念。結合相位式快取及快速存取管線設計，並達到整體最高效益。為了驗證該演算法的正確性，本論文設計中結合本實驗室的嵌入式處理器，做為該資料快取記憶體的驗證平台，並整合成為一顆嵌入式低功耗處理器晶片。此晶片採用 TSMC 0.18 μm 1P6M 製程，以 Cell-based 方式設計，晶片面積約 $2.1 \times 2.1 \text{ mm}^2$ ，最大操作頻率在 100MHz，平均功耗 16mW。

High-Performance Pipeline Design for Low Power Phased Cache

Student : Chih-Wen Hsueh

Advisor: Chih-Cheng Chou

Co-advisor: Chin-Teng Lin

Department of Electrical and Control Engineering
National Chiao Tung University

Abstract

Low power and high performance design issues have played an important role among various portable systems and applications. In embedded processors, the cache design almost occupies half chip area and power consumption. According to Amdahl's Law, if we could reduce the power consumption of cache, the embedded processor can significantly save more power. The cache miss results in the penalty of thousands of cycles waiting and power consumption due to the increasing external memory accesses. Generally, the set associative cache design could increase the hit ratio, but also induces remarkable power consumption. On the other hand, the phased cache design can largely improve the power consumption which set associative cache wasted, but phase cache requires double access cycles compared with traditional one-access-cycle cache design.

In this thesis, we take advantages of the improved pipelined architecture without stalling and low-power phase cache to achieve high-performance and low-power embedded processor design. From experimental results, the proposed architecture could reduce 44% power consumption compared with traditional one-access-cycle cache and eliminate pipeline stalls incurred by phased cache with only 6% gate count overhead.

To verify the pipeline architecture, a RISC embedded processor is employed to be the verification platform for the proposed cache controller and pipeline design. The chip fabricated in TSMC 0.18 μm 1P6M CMOS process can operate at 100 MHz, where the whole chip area is 2.1x2.1 mm^2 . The average power consumption is around 16 mW.

誌 謝

兩年的研究所生涯隨著論文的完成劃上了句號，這兩年間，要感謝許多人的鼓勵和幫忙，使我獲得充實的專業能力並順利完成研究所的學業。

首先要感謝的是我的指導教授-林進燈老師。林老師是國內十分傑出的一位教授，在不同領域內都有相當好的研究成果。感謝老師提供了很理想的研究環境、豐富的資源及正確的引導，使我在研究上非常順利。在老師悉心的指導下，讓我學習到解決問題的能力及做研究應有的態度，使我獲益良多。

在學校裡，范倫達教授時常關心我學業上的研究，時常與我討論論文方向及進度。在實驗室裡，仁峰學長給予我最直接的教導，不管遇到課業上或研究上的問題，常常去請教仁峰學長，感謝學長不厭其煩地教導，使我增進了對積體電路設計上的專業知識，開拓了我的視野。也感謝實驗室所有的夥伴，經翔、紹航學長，德璋、俊傑、靜瑩以及實驗室的學弟妹，感謝大家在研究上及生活上的互相扶持及鼓勵。

最後要感謝家人媽媽、哥哥的支持，讓我能專心於學術上的研究，渡過所有難關，謝謝！

人生值得感謝的人其實很多，感謝老天、感謝許多親人、朋友和同學，在生命的旅途中，因為有你們，因為我們彼此珍惜、相互扶持，才能有無比的力量

目 錄

中文摘要	i
Abstract	ii
誌 謝	iii
目 錄	iv
圖 目 錄	viii
第一章 序論	1
1.1 簡介	1
1.2 論文架構	11
第二章 相位式快取記憶體之高效能管線設計	12
2.1 相位式快取記憶體之高效能存取管線架構與原理	12
2.1.1 快取記憶體控制器及高效能存取管線架構設計	12
2.1.2 快取記憶體控制器及高效能存取管線之原理與演算法	15
2.1.3 相位式快取記憶體及高效能存取管線之分析及改善	17
2.2 相位式快取記憶體之高效能存取管線之效能分析	24
2.2.1 相位式快取記憶體之高效能存取管線設計之功率分析	24
2.2.2 SimpleScalar 功率及週期分析	29
2.2.3 高效能之記憶體存取管線設計分析	32
2.3 結語	34
第三章 低功耗嵌入式處理器設計	35
3.1 低功耗嵌入式處理器架構	35
3.1.1 低功耗嵌入式處理器核心	35
3.1.2 低功耗嵌入式處理器指令集架構	38

3.2 功率感知之匯流排編碼解碼器	43
3.2.1 功率感知之匯流排編碼解碼器原理	43
3.2.2 功率感知之匯流排編碼解碼器效果	44
3.3 具有使用者可調性主從式指令快取記憶體控制器	44
3.3.1 主從式快取記憶體控制器原理	44
3.3.2 主從式快取記憶體效果	46
3.4 軟體開發環境	47
3.4.1 組譯器	47
3.4.2 模擬器	48
3.5 測試驗證	50
3.5.1 有限長度脈衝響應(Finite Impulse Response)	50
3.5.2 餘弦轉換(Discrete Cosine Transform)	51
3.5.3 索比爾運算(Sobel operator)	52
3.6 可程式邏輯閘陣列(FPGA) 驗證	54
3.7 結語	54
第四章 晶片實現與結果驗證	55
4.1 晶片製作	55
4.1.1 設計流程	55
4.1.2 合成結果	56
4.1.3 佈局與封裝	56
第五章 結論與展望	61
參考文獻	62
附錄	64

圖目錄

圖 1-1 : ARM 920T 消耗功率比例圖	1
圖 1-2 : 關聯性記憶體命中率及消耗功率分析圖	2
圖 1-3 : 相位式快取記憶體控制與傳統快取記憶體控制比較圖	3
圖 1-4 : Sentry tag 快取記憶體架構	4
圖 1-5 : 區塊預測關聯式快取記憶體	6
圖 1-6 : 低功率消耗關聯式資料快取記憶體分析	9
圖 2-1 : 相位式資料快取記憶體管線流程圖	12
圖 2-2 : 相位式快取記憶體架構	12
圖 2-3 : 高效能低功率消耗相位式資料快取記憶體管線流程圖	13
圖 2-4 : 高效能低功率消耗相位式快取記憶體的架構圖	13
圖 2-5 : 快取記憶體執行相位比較圖	14
圖 2-6 : 高效能低功率消耗相位式快取記憶體演算法	17
圖 2-7 : 512 block 之集合關聯式記憶體位址組成	19
圖 2-8 : critical path 時間模擬結果	20
圖 2-9 : 處理器管線前饋圖	20
圖 2-10 : 管線 REG 級前饋改善圖	22
圖 2-11 : Sentry tag 架構與相位式架構降低功率消耗比較圖	28
圖 2-12 : Sentry tag 架構與相位式架構降低功率消耗比例關係圖	28
圖 2-13 : Sentry tag 架構與相位式架構功耗跟命中率之關係圖	28
圖 2-14 : SimpleScalar 軟體介面	29
圖 2-15 : 2-Way 時 Sentry tag 與相位式架構降低功耗比較	30
圖 2-16 : 4-Way 時 Sentry tag 與相位式架構降低功耗比較	30
圖 2-17 : 8-Way 時 Sentry tag 與相位式架構降低功耗比較	31
圖 2-18 : 總周期改善比例與讀取指令比例	32
圖 2-19 : 快取記憶體讀取周期改善比例	32
圖 2-20 : 管線改良之分析圖	32
圖 2-21 : 高效能相位式快取記憶體設計改善比較圖	34
圖 3-1 : 處理器組成架構圖	36
圖 3-2 : 處理管線流程圖	36
圖 3-3 : MACHR 指令運算	42
圖 3-4 : 匯流排編碼架構流程圖	43
圖 3-5 : 多媒體傳輸編碼效果比較	44
圖 3-7 : 主從式快取記憶體操作演算法	45
圖 3-8 : 主從式快取記憶體的效能提升示意圖	46
圖 3-9 : 組譯器 (Assembler) 流程圖	47
圖 3-10 : 圖形化介面組譯器 (Assembler)	48

圖 3-11：利用倒寫管線以使用高階語言的寫法描述管線。	49
圖 3-12：圖形化介面模擬器 (simulator)	50
圖 3-13：FIR RTL 模擬及 simulator 執行結果	51
圖 3-14：1-dimension 8-by-8 DCT 演算法	51
圖 3-15：2-dimension 8-8DCT RTL 模擬及 simulator 執行結果	52
圖 3-16：Sobel 模擬結果	53
圖 3-17：處理器 FPGA 模擬 Sobel 結果圖	54
圖 4-1：晶片設計流程	55
圖 4-2：佈局圖	57
圖 4-3：腳位圖	57
圖 4-4：打線圖	58
圖 4-5：處理器 Post-sim 模擬 DCT 結果圖	60
圖 4-6：處理器 Post-sim 模擬 Sobel 結果圖	60



表 目 錄

表 1-1：集合預測架構與相位式快取記憶體架構時間及功耗分析.....	7
表 1-2：集合關聯式記憶體之低功耗設計分析.....	8
表 2-1：各種定址模式範例、意義及使用頻率.....	16
表 2-2：傳統、Sentry tag 架構、phase 架構功率公式比較表.....	26
表 2-3：標準效能評估程式介紹.....	30
表 2-4：整合設計效能、功率比較.....	33
表 2-5：本設計與其他低功耗快取設計分析比較.....	34
表 3-1：資料搬移指令列表.....	39
表 3-2：算數邏輯運算指令列表.....	40
表 3-3：跳躍指令列表.....	40
表 3-4：SIMD 指令列表.....	41
表 3-5：其他指令列表.....	42
表 4-1：合成的結果.....	56
表 4-2：晶片設計規格.....	58
表 4-3：快取記憶體功率計算.....	59



第一章 序論

1.1 簡介

隨著可攜式的產品應用日趨廣泛，普遍於人們的日常生活之中，如手機、PDA 數位相機，MP3 player 等等，帶給人們生活上很大的便利，於是也佔了市場上很大的一環。可攜式的產品相當重視其使用時間長短的表現，但是電池能源技術的成長卻相當緩慢，以至於電池能源部分佔了可攜式產品相當大的重量，如 NOKIA 6100 系列，電池重量佔了將近 30% 的重量。越長的使用時間也意味著電池所佔的比例越大，於是在能源無法提供更多的情形下，必須將電源使用的更有效率，所以低功率消耗便成為近代嵌入式處理器中，相當重要的研究。

處理器的低功率消耗設計相當廣泛，從 $P=fcV^2$ 中，不論是管線(pipeline)設計的複雜程度，暫存器的數目或快取記憶體的模式，到指令集架構的編排及匯流排的變動(bus activity)都是可以從中改善以獲得不錯的效益[1]。根據 Amdahl 定律，改善部份佔全體的比例，深深決定設計改良能帶來多少效益。也就是改善的單位效率還要乘以全體比例，才是全體的改良效率。

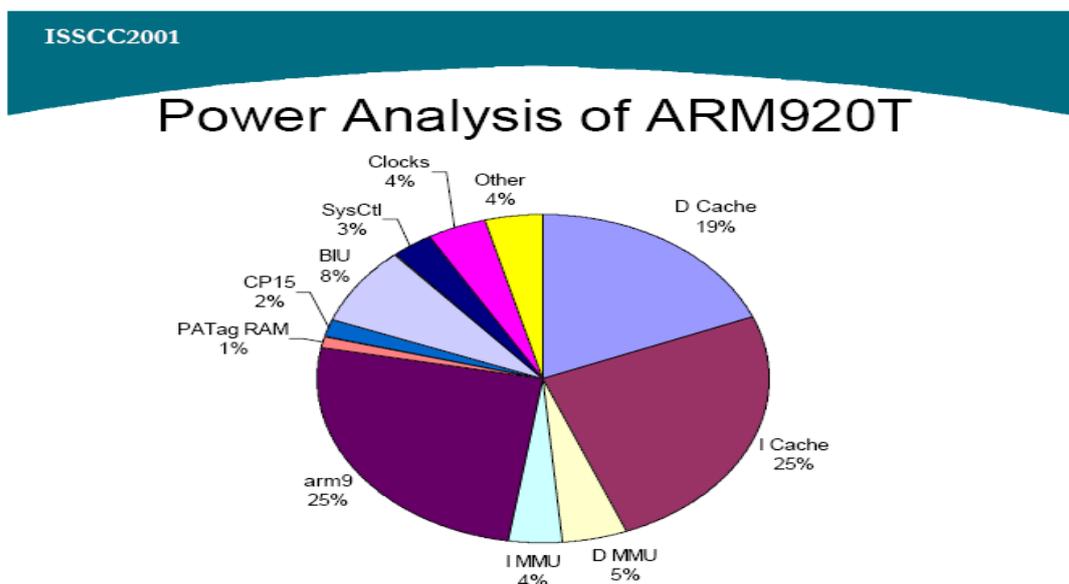


圖 1-1 :ARM 920T 消耗功率比例圖

一般來說，快取記憶體是處理器中經常使用的單元，佔了處理器五成的功率消耗，如圖 1-1 [1]，例如 Arm920T，快取記憶體佔了功率消耗 46%，StrongArm-1，快取記憶體佔了功率消耗 43%，所以快取記憶體的低功耗消耗設計是個值得研究的部份，可以從中獲得不錯的報酬率。

為了降低處理器中快取記憶體功耗的研究相當多元[2][3][4]，可以從提升快取命中率來降低整體功耗，或者 SRAM cell 的低功耗設計，又或者從快取架構上來著手，例如 直接定址快取記憶體(Direct Mapping cache)、集合關聯性快取記憶體(Set associative cache)和完全關聯性快取記憶體(Fully associative cache)[5]。

其中快取記憶體相當重視命中率(hit ratio)，因為 1 次的快取 MISS 必須到外部記憶體存取正確的資料，平均需要花費數倍到數十倍存取週期，而外部的記憶體存取相對於晶片內部快取記憶體除了耗費更多的存取時間之外，所耗費的能量也是晶片內部快取記憶體的數倍，所以提高命中率在效能及功耗上都能獲得改善，其中最基本、簡單的方法就是使用集合關聯式快取記憶體，例如完全關聯式快取記憶體，4-way 集合關聯式快取記憶體，都能有效改善快取衝突性失誤(conflict miss)，提高命中率，但這個技術當然也有相對上的缺失，例如，4-way 集合關聯式快取記憶體，會在 1 次存取中，同時啟動 4 個 way 集合，讀取 4 筆資料，如圖 1-2，再將命中的集合區塊資料讀取到處理器使用，也就是在這 4 個集合中，最多只會有一筆區塊資料是我們所需要的，最少有 3 個以上的區塊存取是浪費的，這就是很值得改善的空間。

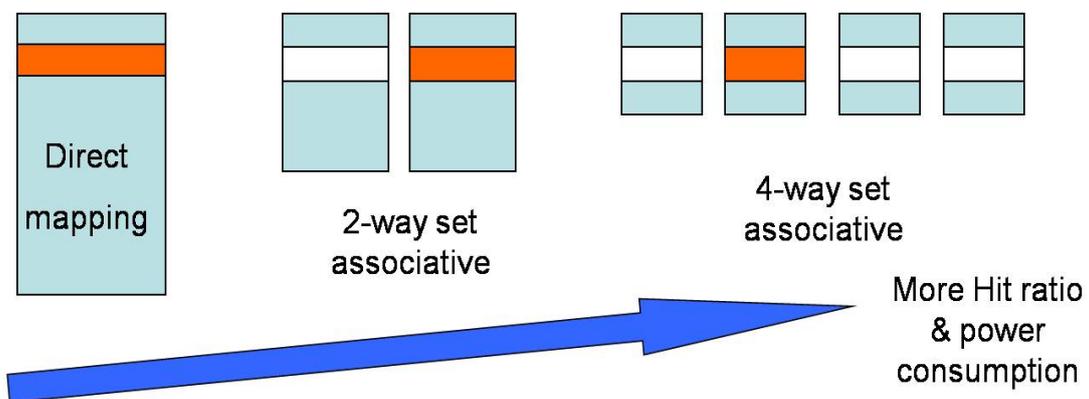


圖 1-2 :關聯性記憶體命中率及消耗功率分析圖

針對以上提出來的現象，已有相當多的研究試著解決集合關聯式快取記憶體不必要的功率消耗，其中早期提出的觀念是相位式快取記憶體(Phased cache) [6][7]，如圖 1-3 所示[10]，藉由先執行快取記憶體中 Tag 的存取及比對判別出不需要執行的 way 集合，也就是 Cache MISS 的 Way 集合的資料存取，藉由這個快取記憶體控制的演算法，可以將集合關聯式記憶體中不必要的集合資料區塊存取完全的省略掉，避免不必要的功率消耗。但此設計有個嚴重的缺失，因為將原本快取記憶體一個步驟就提取出 Tag 跟資料的做法，變成分兩步驟執行，第一步驟先執行 Tag 的存取，比對記憶體位址是否正確，第二步驟在將正確的資料讀取出來，也就是需要耗費 2 倍的記憶體存取時間，而這多出來的時間浪費，會造成管線暫停，降低效能，處理程式所花費的時間越多也代表著需要耗費更多的功率消耗，所以雖然此演算法能省下快取記憶體約略 40%-70%左右的功率消耗(依關聯式設計集合數多寡影響)，但也造成多出的管線暫停及管線功率消耗。

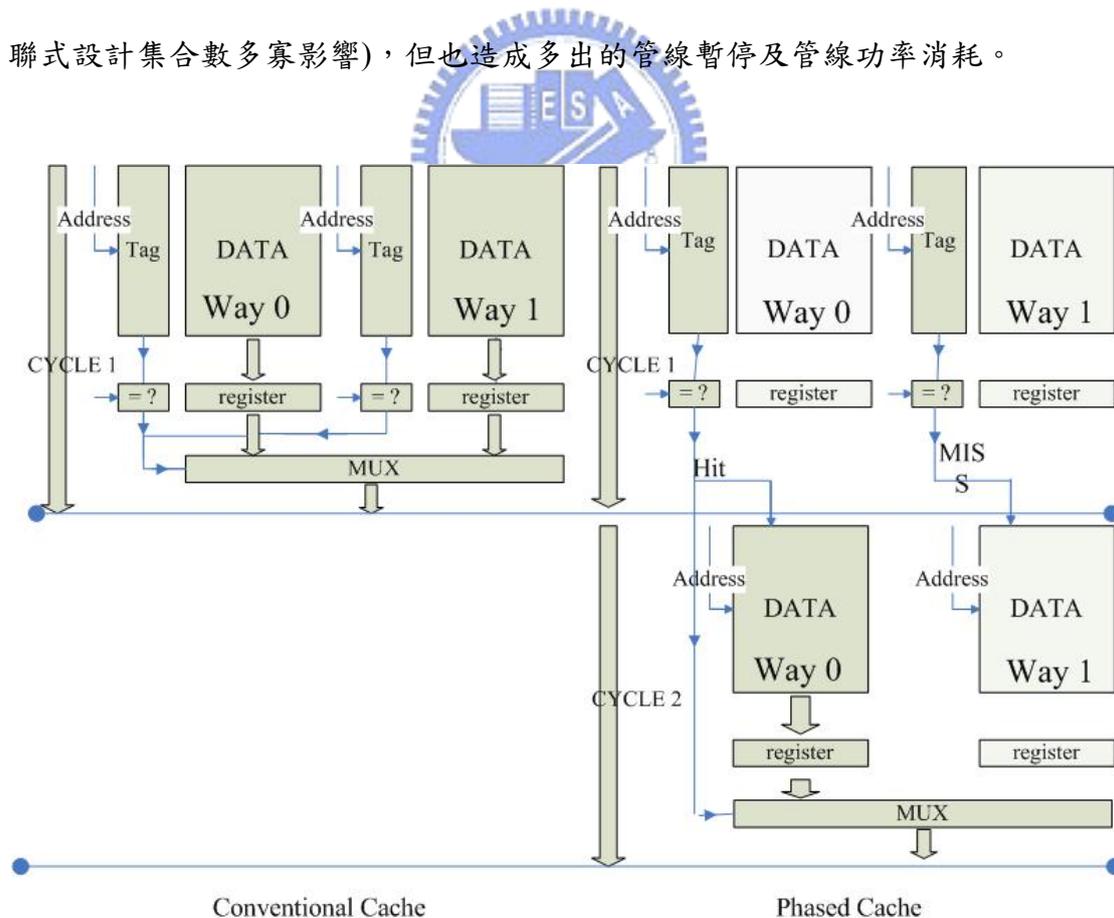


圖 1-3：相位式快取記憶體控制與傳統快取記憶體控制比較圖

因此也有相當多的研究試著去改善相位式快取記憶體在時間上的缺失，例如 Sentry tag architecture [8]，這篇論文試著將相位式快取記憶體中的兩個步驟，濃縮在 1 個記憶體存取時間內完成，這樣就能達到相位式快取記憶體中避免不必要的集合資料存取，又能改善相位式快取記憶體需要耗費兩個記憶體存取時間的缺失，如圖 1-4 所示[8]。其將一部分的 Tag bits 存放在 Sentry tag cache 這個 table 之中，這樣再處理快取記憶體存取時，先經由 Sentry tag cache table 快速存取出部分 Tag bits，與記憶體位址做比對，經由此步驟可以有效過濾部分不必要的集合存取，又只需要增加短暫 Sentry tag cache table 存取時間。藉由使用內容定址記憶體 CAM(Content Addressable Memory) 9-T cell 來實現 Sentry tag architecture 所需的高速存取需求，但 CAM 採用的是類似 Full associative cache 的存取方式，也就是對記憶體內部每筆資料都做比對，需要驅動所有 CAM cell，也就是需要耗費大量的功率消耗[9]。

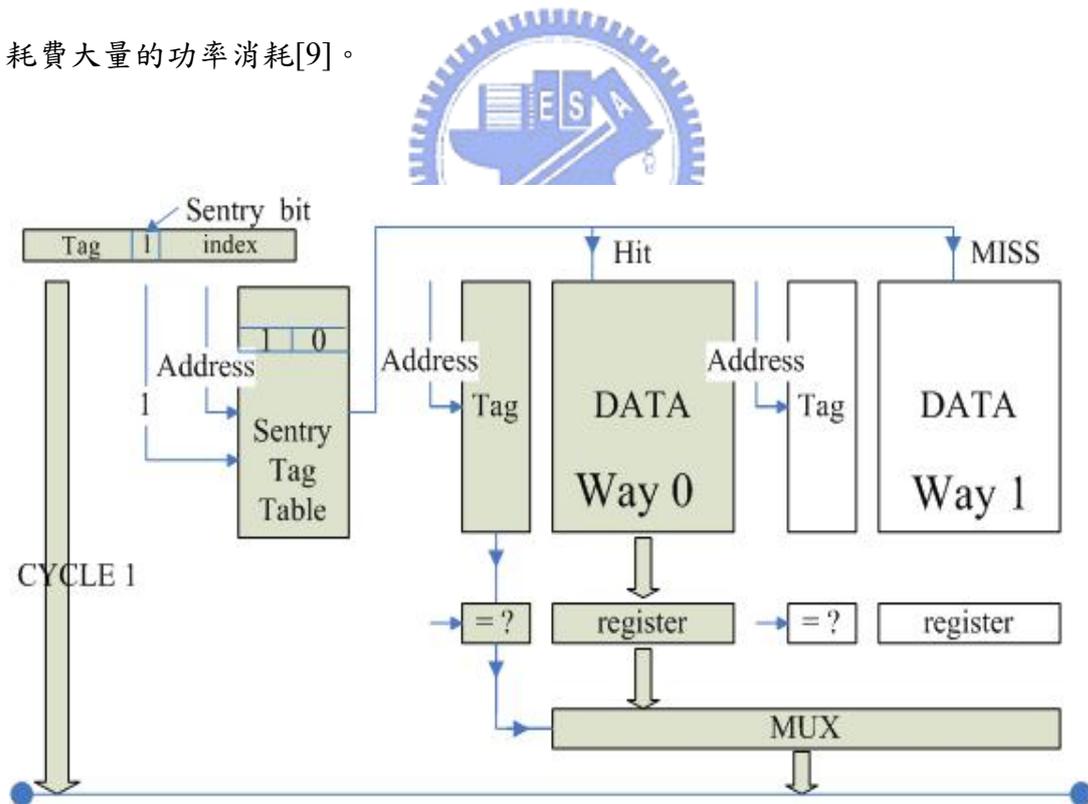


圖 1-4 : Sentry tag 快取記憶體架構

但其實深入探討可以看出此設計仍有改善的空間，這些設計必須使用快速的特殊記憶體元件將 2 個步驟濃縮在一個記憶體存取時間內，除了有可能會降低記憶體存取頻率之外，因為使用快速的記憶體元件 CAM，因為 CAM 會隨著面積、區塊增加而增加存取時間的性質，因此 CAM 有著面積、區塊上的限制，不得不只能紀錄 2-4bit 的 Tag bit，也因為比對的 Tag bit 數少，不能完全的過濾掉不必要的資料存取，只能降低約略相位式快取記憶體八成的功率消耗，且考慮到 CAM 快速存取但必須全部比對需要驅動所有 cell 且單位面積比約為 SRAM 兩倍大[9]，耗費的單位功率損耗比 SRAM 多出 1 倍以上，所以此特殊記憶體單元設計也耗費掉可觀的功率消耗。

另外因為 Sentry tag architecture 在過濾快取記憶體非必要執行集合的效果是機率性的，隨著測試程式的不同，區域性(locality)的不同，會影響其降低功率消耗的效果，不像 Phased cache architecture 過濾快取記憶體非必要執行集合的效果是 100%，所以因為資料快取記憶體的區域性相對於指令快取記憶體較低，Sentry tag architecture 使用在資料快取記憶體上的效率是比使用在指令快取記憶體上的效果較差。

除了相位式的方法，還有一種也是以降低集合關聯式快取記憶體的功率消耗為目標的設計，Way-predicting set-associative cache [10]，有別於相位式快取記憶體經由 Tag 快取部份存取比較後的訊息，此記憶體經由 MRU(Most Resent Used)演算法來預測命中的 Way 集合的訊息，因此在執行快取讀取時，只需同時執行預測命中的 way 集合的 Tag 及資料的存取，即可達到避免不需要的 WAY 集合的存取，降低功率消耗，如圖 1-5[10]所示，因為此演算法為事先預測，於是在執行快取存取時，已有關聯式記憶體集合命中、失誤的訊息，只需要單一週期存取，不會造成時間上的負擔。但此設計也有必需付出的代價，若是預測的訊息錯誤，不但要多付出 1 個記憶體存取週期去存取其他 way 集合的資料，且完全沒有降低功率消耗，所以依然會增加平均處理時間，且降低功率消耗的效率也不如相位式快取記憶體。

因此預設率的高低決定此設計的效果，在指令快取記憶體中也因為指令的區域性，連續性高，因此猜測率高，降低功率消耗效果與相位式快取記憶體相近，但在資料快取記憶體中會因為預測率降低，降低功率消耗的效果變差，也增加了處理時間。此外預測率也會隨著關聯性快取記憶體的集合數增加而降低。

另外此設計有個問題需要來思考的，就快取處理時間上來分析，集合預測的演算法確實能將快取處理時間降到平均一個周期左右，但從管線的角度來思考，在指令快取記憶體中，預測命中時，在快取處理時花一個週期的處理時間，但若下一筆快取存取預測失誤，卻需要花兩個週期的處理時間，就管線流程的角度，此時反而會造成管線暫停。雖然能降底整體快取處理時間，但因為其處理週期的不確定性，反而造成整體的週期增加，因此其時間上的效果在指令快取中，是沒有效果的。

就如表 1-1 [10]顯示 Phased cache 與 way prediction cache 的比較一樣，可以看出 Phased cache 需要花費較多的快取處理時間，但功率消耗較 way prediction cache 為低。

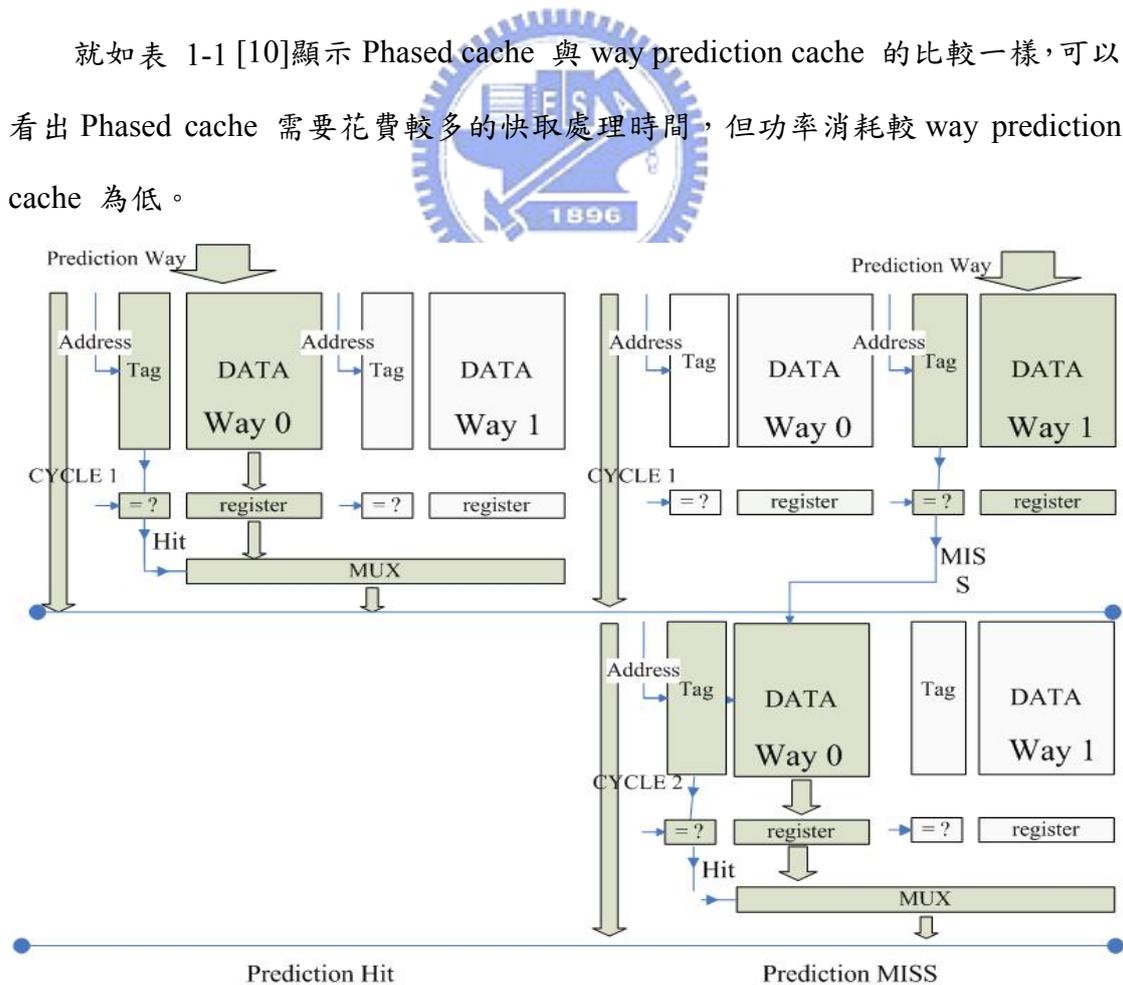


圖 1-5：區塊預測關聯式快取記憶體

表 1-1：集合預測架構與相位式快取記憶體架構時間及功耗分析

Benchmarks	I-Cache				D-Cache			
	Phased(4-way)		Predict(4-way)		Phased(4-way)		Predict(4-way)	
	Time	power	Time	power	Time	power	Time	power
099.go	198%	30%	105%	29%	199%	30%	119%	39%
126.gcc	197%	30%	108%	31%	197%	30%	113%	35%
129.compress	200%	30%	100%	25%	195%	29%	108%	31%
102.swin	200%	30%	102%	27%	182%	26%	150%	62%
101.tomcatv	199%	30%	108%	31%	198%	30%	112%	34%
103.su2cor	200%	30%	104%	28%	193%	29%	107%	36%

本論文針對以上各論文的研究，演算法及數據上的分析，綜合出表 1-2 的結果，首先從降低功率消耗上來探討，雖然各論文的實驗規格數據不一致相同，但從過濾非必要集合執行上來比較，Sentry tag 架構及集合預測架構都是機率性的過濾非必要執行集合，而相位式快取記憶體架構卻能 100%過濾非必要執行資料集合部分，雖然與 Sentry tag 架構及集合預測架構相比無法省略 Tag 部分的執行，考量到 Sentry tag 架構需花費 CAM cell 的 Sentry tag cache 的功率消耗，而集合預測架構在失誤時需要耗費的大量功率消耗，相位式快取記憶體架構在降低功率消耗的效果仍是比其他設計較為有效。

從處理時間週期上去探討，Sentry tag 架構增加了 Sentry tag cache 的處理、資料比較的時間，集合預測架構增加了失誤時增加的 10%~20%快取處理週期，而相位式快取記憶體架構需要增加 1 週期的處理時間，因此容易造成管線暫停，增加處理時間。在處理週期上相位式快取記憶體需要花費最多的處理時間。

綜合深入探討分析，相位式快取記憶體可以最有效的降低功率消耗，且準確率最高，但卻有耗時的缺點，而 Sentry tag 架構可以改善相位式耗時的缺點概念很好，但卻有特殊元件的限制及時間和功率上的小缺失，而預測集合關聯式快取記憶體雖然能使用事先預測的方法降低功率消耗，卻有猜測率失誤的風險及付出的代價。因此若是能結合最低功率消耗的相位式快取記憶體的概念，又能使用事先的判斷來改善其耗時的缺點，但又能維持判斷率的 100%命中，就能達到低功率消耗且不影響效能的高效能低功率消耗快取記憶體控制器設計。

表 1-2：集合關聯式記憶體之低功耗設計分析

	Conventional Cache	Sentry tag Cache	Way Prediction Cache	Phased Cache
降低快取 功耗效果	Low(0%)	Middle(34%)	Middle(30%~40%)	High(44%)
快取延遲/ 暫停	Low(1 cycle)	Low(1 cycle)	Middle(1~2 cycle)	High(2 cycles)
設計複雜 度/成本	Low	High (CAM)	Middle	Middle

本論文提出一個將相位式快取記憶體改善的另一途徑，將此相位式快取記憶體與前級管線結合，分擔相位式快取記憶體的 Tag 存取步驟，在快取記憶體存取之前級就判別出 Tag 比對的結果，這樣就能省略 MISS 的集合資料存取，有效降低五至七成左右的功率消耗，且在快取記憶體處理前得得到命中、失誤的訊息，於是能在正常處理時間內完成所需資料的存取，不會造成管線暫停，改善相位式快取記憶體耗時的缺失。

圖 1-6 為高效能低功率消耗相位式快取記憶體管線設計的原理，比照其他設計將相位式濃縮至單一週期的原理達到低功率消耗卻不造成時間負擔的效果，本設計理念亦以此為目標，因此提出將其提前一級去執行相位式中之 Tag 存取比對的步驟，就能在讀取快取資料時依前級比對的訊息驅動所需執行之集合的動作，又不會延後快取存取資料的時間點，這樣就不會影響管線的執行，達到目標。

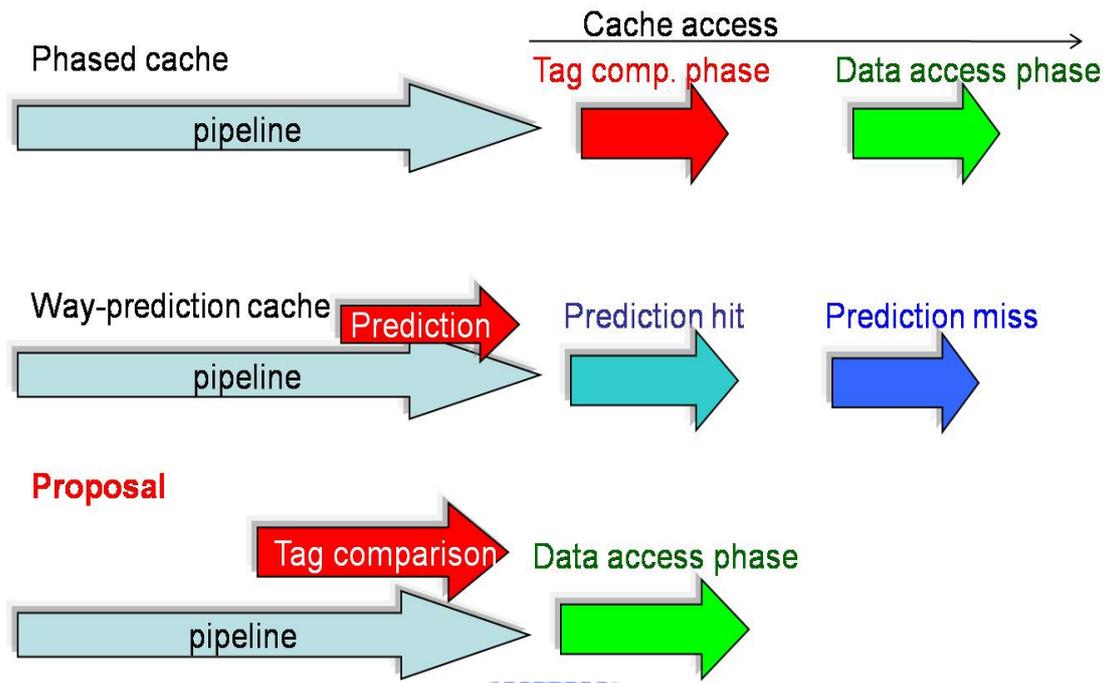


圖 1-6：低功率消耗關聯式資料快取記憶體分析

如圖 1-6 所示，此設計原因因與管線結合記憶體讀取優化來達到高效能的效果，在設計上能改善資料快取記憶體的效能，對於指令快取記憶體卻沒有另外設計，除了考量到指令快取記憶體多為 L1 直接映射快取記憶體(Direct-Mapped cache)設計，較少集合關聯式設計而且最多也是 2-Way 集合關聯式快取記憶體設計，且指令快取命中率高，相反的資料快取記憶體，命中率較低，資料被取代性高，有較高衝突性失誤，需要使用較深的關連性快取記憶體設計，2-Way、8-Way 以上的集合關聯式快取記憶體架構來降低失誤率，所以針對快取記憶體降低關聯式快取記憶體功率消耗的相位式設計，在資料快取記憶體上的使用是較指令快取記憶體實用的，詳細會在 2.1.3 節有介紹說明。

本論文所設計的快取記憶體與高效能存取管線設計具有以下特性:

1. 降低快取記憶體功率消耗:

將快取記憶體分成兩個相位，事先處理 Tag 與記憶體位址的比對，判別出關聯式記憶體所需的集合區塊命中、失誤訊息，將失誤的關連性記憶體集合關閉，達到低功率消耗的效果。

2. 改善相位式快取記憶體耗時兩倍記憶體時間的缺失:

因為相位式快取記憶體將存取 Tag 跟資料分成兩步驟依序執行，雖然能有效降低快取記憶體的功率消耗，但卻需要花費 2 倍的讀取記憶體的時間，因此本設計將相位式資料快取記憶體結合管線分擔 Tag 比對判別的相位步驟，雖然仍需耗費 2 倍記憶體時間，但提早一級執行此步驟，與一般資料快取記憶體同時完成資料存取的執行，所以在時間上並沒有造成負擔，改善相位式快取記憶體耗時造成的管線暫停的缺點，達到較高效能。

3. 設計單純、低成本:

此設計只需增加些微管線硬體電路及快取記憶體控制器成本，佔整體不到 10% 的 gate count。

且有別於 Sentry tag 架構快取記憶體控制器，需要使用特殊設計快取記憶體單元，如 CAM 或者加入其他邏輯單元，來改善相位式快取記憶體耗時的缺點，須準備 2 種晶片內記憶體單元及其控制單元，且這些快速記憶體單元在面積上及功耗上皆比 SRAM 高出數倍，本設計因為不需將相位式濃縮至單一週期，所以可以採用傳統一般 SRAM 記憶體作為資料快取記憶體，除了不增加製作難度，且不會造成面積及功耗上的取捨。

為了驗證該演算法的正確性、可行性及可靠性，本論文設計中結合本實驗室的嵌入式處理器，做為該資料快取記憶體控制器及管線的驗證平台，並整合其他低功耗設計成為一顆嵌入式處理器晶片。此晶片採用 TSMC 0.18 μm 製程，以 Cell-based 方式設計，晶片面積約 $2.1 \times 2.1 \text{ mm}^2$ ，最大操作頻率在 100 MHz，平均功率消耗為 16 mW。

1.2 論文架構

本篇論文中，第二章介紹低功率消耗相位式資料快取記憶體控制器及管線設計，從硬體架構到效能測試有詳細的說明。第三章介紹快取記憶體控制器其平台設計，從嵌入式處理器架構到開發工具製作、低功耗設計的整合和測試程式驗證。第四章介紹晶片實現的過程，包含模擬驗證方法及結果，晶片製作。最後，在第五章做總論。



第二章 相位式快取記憶體之高效能存取管線設計

本章介紹高效能之低功率消耗相位式快取記憶體控制器及其管線的設計，包含其原理、架構、效能測試、特點及比較。

2.1 相位式快取記憶體之高效能存取管線架構與原理

2.1.1 快取記憶體控制器及高效能存取管線架構設計

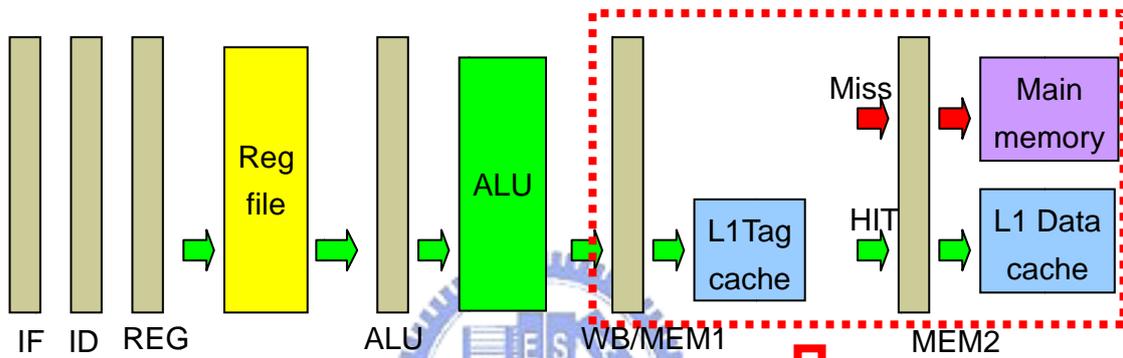


圖 2-1：相位式資料快取記憶體管線流程圖

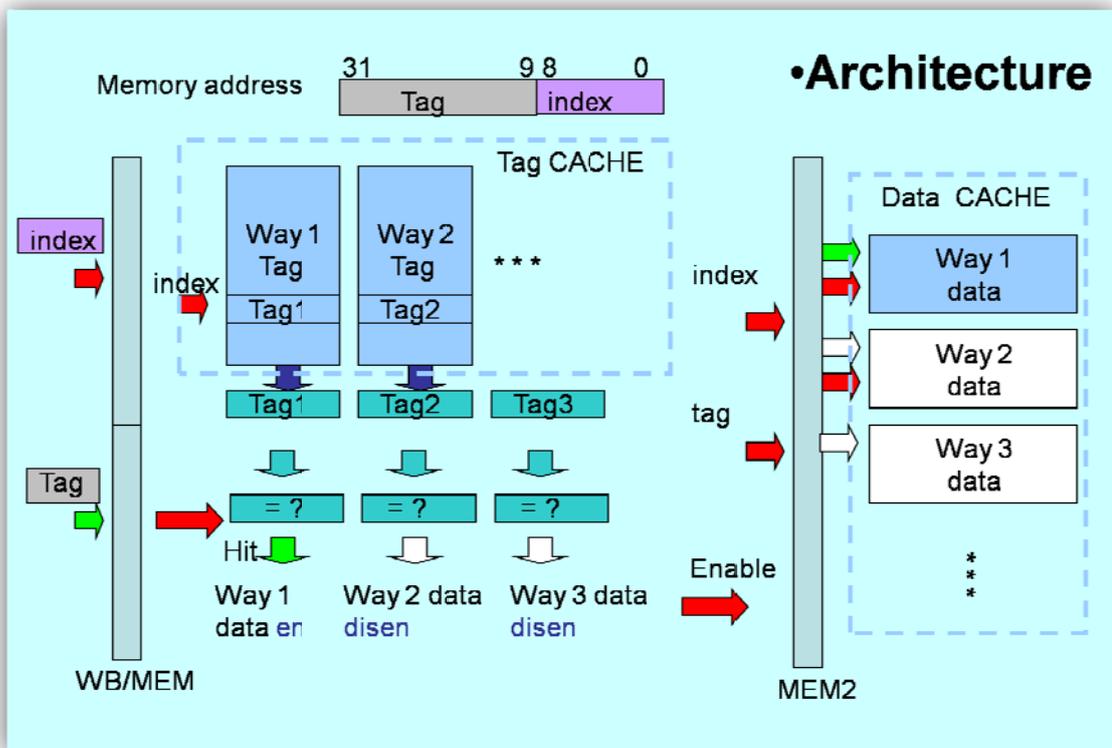


圖 2-2：相位式快取記憶架構

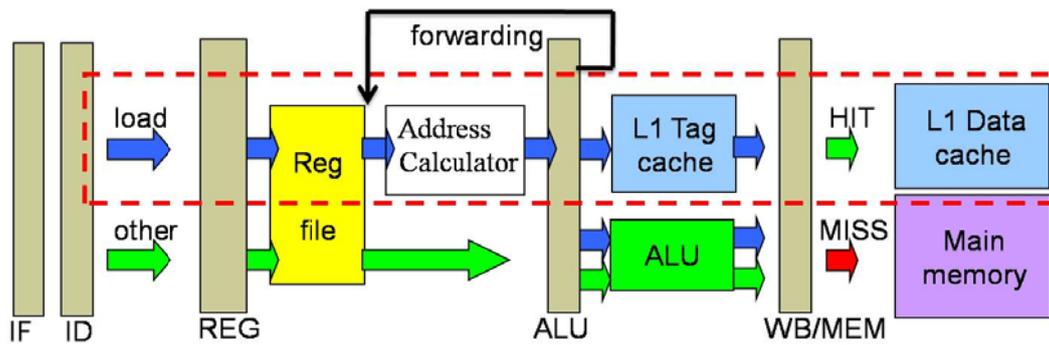


圖 2-3：高效能低功率消耗相位式資料快取記憶體管線流程圖

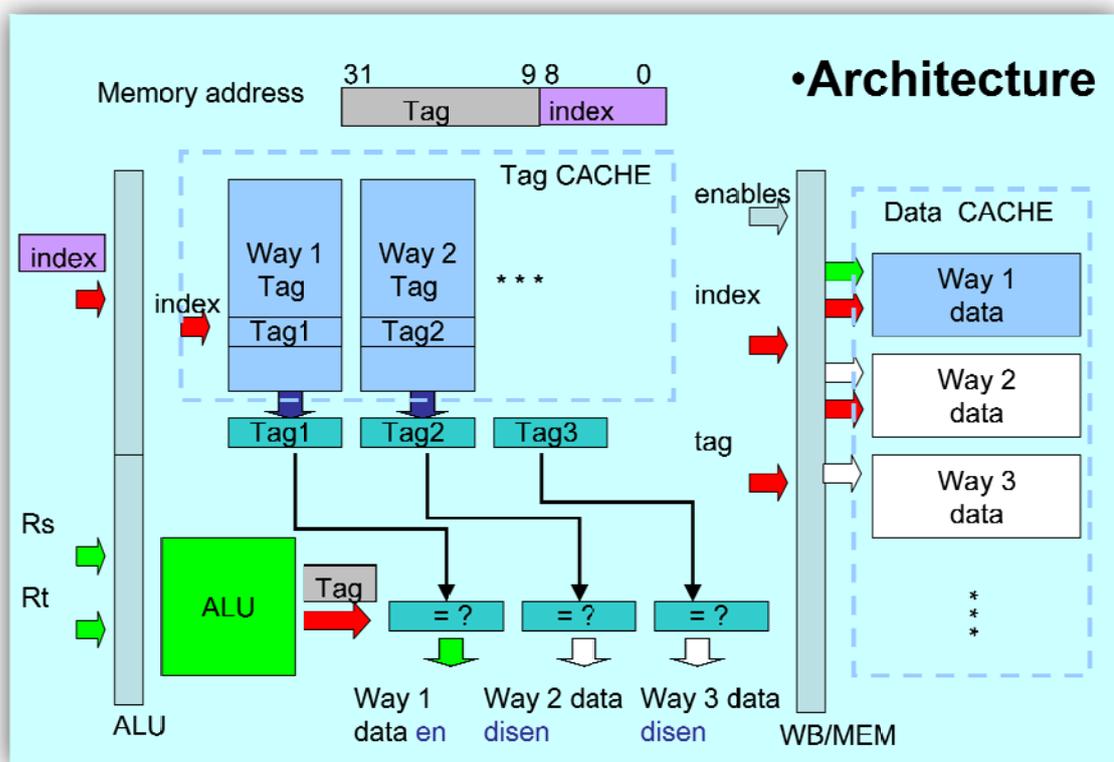


圖 2-4：高效能低功率消耗相位式快取記憶體的架構圖

圖 2-1 為傳統低功率消耗相位式資料快取記憶體管線流程圖，從 IF、ID、REG 讀取暫存器數值、ALU 計算記憶體位址再到相位式快取記憶體，相位式快取記憶體首先讀取 Tag cache 的數值，比對命中失誤訊息之後，下一週期在執行 Data cache 的存取，因此記憶體指令需要耗費兩個週期，如果有資料相依發生，就會造成管線暫停一個週期。

圖 2-2 為低功率消耗相位式快取記憶體控制器 MEM1 與 MEM2 級的架構圖，將 ALU 運算的記憶體位址分為 Tag 部分及 index 部分，依照 index 從 Tag cache 讀取出每一個 Way 集合的 Tag 區塊數值，再將其與記憶體位址中的 Tag 部份做比對，然後將命中的訊息作為 Data cache 驅動與否的判斷，以圖為例，在 MEM2 級只有 Way1 的 Data cache 會被驅動，其餘則會被關閉，降低功率消耗。

圖 2-3 為高效能之低功率消耗快取記憶體控制器與管線的架構圖。此控制器仿照相位式快取記憶體分為 Tag 讀取比對相位及資料讀取相位，且將 Tag 讀取比對相位提前至管線的前一級跟 ALU 平行去執行判斷，然後在 WB/MEM 級執行命中的區塊資料存取，跟圖 2-1(a)相位式快取記憶體管線比較，可以將快取記憶體存取時間點提前，得到較高的效能。

圖 2-4 為高效能相位式快取記憶體控制器在 ALU 級的改良架構圖，在此級我們就依照 REG 級得到的 index 數值讀取出 Tag cache 中每一個 Way 集合的 Tag 數值跟 ALU 單元在此級運算出來的記憶體位址做比對，判斷出命中的集合區塊並在 WB/MEM2 級驅動之，其餘失誤的區塊就會被關閉，達到相位式快取記憶體相同低功率消耗效果。

雖然本設計在相位式快取記憶體的處理時間上，仍然需要兩個週期才能分別完成 Tag 存取比對及資料存取，但在管線的處理流線上，如圖 2-5 所示，經由管線的提早執行快取的步驟，可以跟傳統快取處理器達到相同的時間點上存取的目標，因此不會像改良前的相位式快取記憶體般，因為需要多花費一個週期才能得到所要的資料，會造成管線暫停，降低處理速度。除此之外，因為提早運作 L1 快取的 Tag 比對得到的資訊，在命中時可以去除非必要執行的功率消耗，若判別為失誤時，更能提早處理下一級的記憶體存取，提高效能。因此若以 L1 快取記憶體需時 1-cycle，主要記憶體需時 4-cycles 為例，在我們提出的設計架構下，在 L1 失誤的情形下，可以將總記憶體存取所需時間從傳統的 5-cycles 改善至 4-cycles 的處理時間，進一步提升效能。

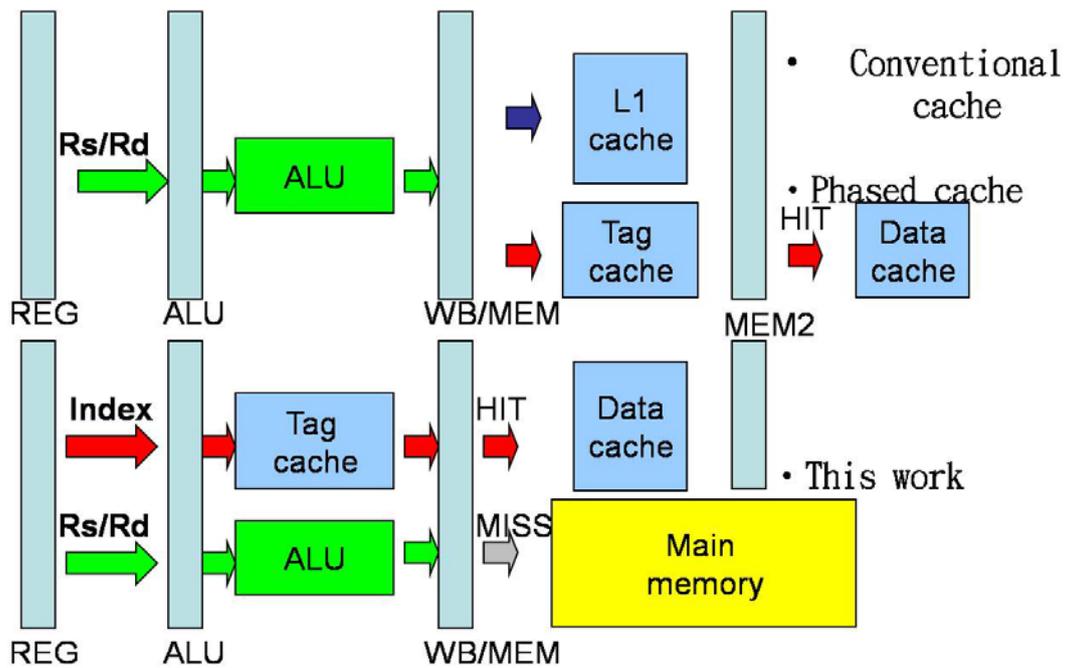


圖 2-5 快取記憶體執行相位比較圖

因此為了在 ALU 級執行 Tag cache 的存取，我們必須要 ALU 級前一級就獲得 Tag 存取所需要的資訊，也就是快取記憶體位址，於是本管線在 REG 級即須計算出快取記憶體位址，但原本應該是 ALU 級才產生的記憶體位址若要在 REG 級就計算出，勢必需要再管線上做些改變才能達到此目標。

2.1.2 快取記憶體控制器及高效能存取管線之原理與演算法

為了達到以上功能，本設計需要在 ALU 級之前獲得存取快取所需的位址資訊，依照記憶體存取指令分為直接定址、間接定址、暫存器定址等等模式去分析，隨著定址模式的不同以及指令集架構的設定決定而有所不同，若採用簡單定址模式如暫存器定址的指令集，因為記憶體存取指令不需經過運算，即可直接將暫存器數值做為記憶體位址，若是採用此指令集架構，則無須對處理器管線作任何改變即可提早獲得記憶體位址，實現本設計要求。

在此本設計利用記憶體位址計算簡單的特性，如表 2-1，只需進行 Reg+direct 運算或者直接是 Reg 或者 direct 的數值來計算記憶體位址，因為不需進行複雜的運算，因此可以將此步驟提前至 REG 級運算，針對特定記憶體指令增加硬體來計算記憶體位址，即可在 REG 級計算出記憶體位址的資訊，達到本設計的要

求。

表 2-1：各種定址模式範例、意義及使用頻率

位址模式	範例指令	意義	使用頻率百分比
立即值	Load R4,#3	$R4 \leftarrow \text{Mem}[3]$	39%(gcc),43%(Tex)
位移	Load R4,3(R1)	$R4 \leftarrow \text{Mem}[R1+3]$	40%(gcc),32%(Tex)
暫存器間接	Load R4,(R1)	$R4 \leftarrow \text{Mem}[R1]$	11%(gcc),24%(Tex)

如上表 2-1 所示，分別顯示位址模式及其意義和使用頻率，從頻率上可以看出此三種定址模式在記憶體指令中使用的多寡，可以包含 75%~99%的記憶體指令，是每一個架構應該具有的定址模式[5]。

因此本設計針對記憶體的存取指令設計，利用其運算簡單的特性，來改善相位式快取記憶體架構，在指令集中可以使用在此三大類定址模式，來提早執行記憶體位址的運算，達成我們的需求。

演算法流程如圖 2-6 所示，本圖架構以五級管線為例，分為 Fetch、Decoder、Register、ALU/Tag cache、WB/Data cache 五級，一開始指令從 Fetch 級輸出 Program Count 值至指令記憶體讀取出指令後，在 Decoder 級除了判別指令運算碼及其指令組成位元分佈，也針對記憶體相關的指令作判斷，輸出訊號給 REG 級做不同的處理。到了 REG 級會依造指令解碼產生的暫存器碼讀取相對的暫存器數值，讀取完畢後，若是記憶體相關指令會依照解碼產生的訊息對讀取出來的暫存器數值做不同的運算，有可能是暫存器加上暫存器或者暫存器加上直接定值又或者是直接定值，產生出快取記憶體位址供 Tag 快取記憶體做為存取所需要的訊息。在 ALU/Tag cache 級會依照指令的型態做不同的運算，若非記憶體相關的指令，則經由 ALU 做算數(arithmetic)或邏輯(logic)運算即可，若是記憶體相關指令，ALU 單元負責運算出完整的記憶體位址與快取記憶體讀取出來的 Tag 值做比對，將命中、失誤的訊息傳到下一級。而在 WB/Data cache 則處理 REG 的寫回或者命中的快取記憶體資料存取。

Algorithm

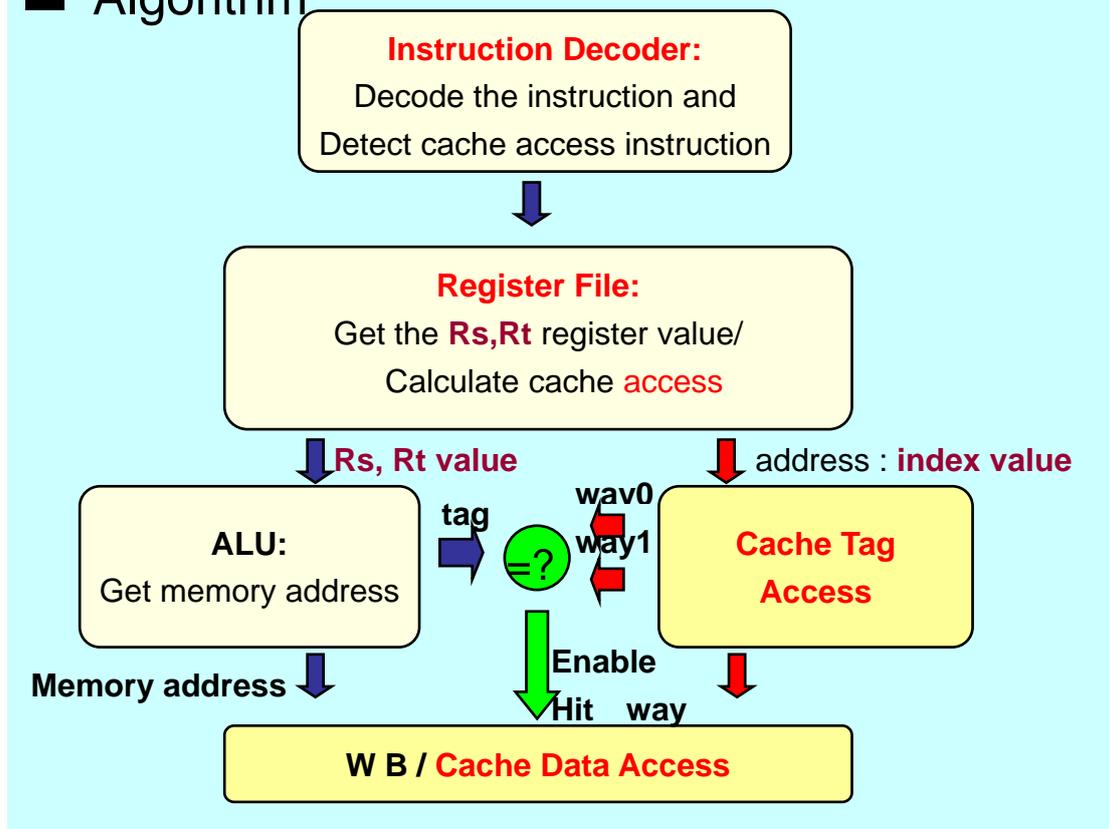


圖 2-6: 高效能低功率消耗相位式快取記憶體演算法

2.1.3 相位式快取記憶體及高效能存取管線之分析及改善

本設計要達到降低功率消耗又能兼顧效能的設計，需要對管線作出些改變，雖然能符合我們要求的目標，但也需要去克服一些困難以及必須犧牲的方面。

從益處、壞處分別去分析本設計

益處:

1. 降低功率消耗: 使用相位式快取記憶體，有效降低關聯性記憶體造成的功率浪費
2. 提高效能: 改善相位式快取記憶體造成的管線暫停以及在 L1 快取記憶體失誤發生時，提早處理下一級之記憶體單元，降低總處理週期
3. 設計單純不複雜: 不需使用其他高速元件，不造成額外負擔，只增加約 6~7%左右的管線硬體成本

壞處:

1. Critical path :對管線 REG 級增加計算流程，提高此管線級成為 Critical path 的風險，有可能造成時脈降低
2. Data Hazard : 需要在 REG 級使用暫存器的數值計算記憶體位址，若與上一指令有資料相依，會造成資料危障
3. 只能改善相位式資料快取記憶體的效能: 只能改善資料快取記憶體造成的管線暫停，對指令快取記憶體沒有效能改善。

以下將針對益處壞處的部分做出分析及改善方法

首先針對此設計對管線的改變，造成的問題，如 Critical path、Data Hazard，將在此小節，針對問題提出改善的方法，將傷害降到最低

- Critical path 部分

本設計需在管線中加入計算單元來提早計算此記憶體位址，可能是單純的加法運算，來處理的記憶體位址的計算，才能提供 ALU 級時的 Tag 快取記憶體存取，因此本設計在 REG 級中，在讀取完暫存器數值之後，加入加法器來做記憶體位址計算，這樣就能在 ALU 級前獲得所需的記憶體位址，但加入此加法器勢必會對原本的管線產生影響，增加此管線級成為 critical path 的風險。

考量到不希望因為增加的計算負擔造成此級成為 critical path ,也就是有可能降低處理器時脈的風險，本設計也提出許多對應的方法:將加法器所需要處理的位元數盡可能降到最低，這樣就能將 REG 級所增加的時間不至於太多，超過原本的 critical path(ALU 級)。

記憶體位址分成兩部分，分別為 Tag、Index 部分，如圖 2-7 所示，Tag 部分用來比對判斷關聯式記憶體中是否有命中的主記憶體中的資料，Index 部分用來讀取快取記憶體的位址中的資料。傳統的快取記憶體會一次擷取整段記憶體位址來運作，使用記憶體位址中 Index 讀出關聯式記憶體中的資料，在將這些資料跟記憶體位址中的 Tag 部份做比對，來判斷命中與否。

因為讀取快取記憶體中的 Tag 或資料只需要 Index 數值即可，所以為了降低 REG 級中加入的運算增加時間造成 critical path 的風險，所以在 REG 級中只去計算記憶體位址中的 Index 數值也就是較低的位元部份，大約可以減少一半的加法時間，將增加的時間負擔降到最低。其餘的記憶體位址則在 ALU 運算出完整的數值，以供利用。

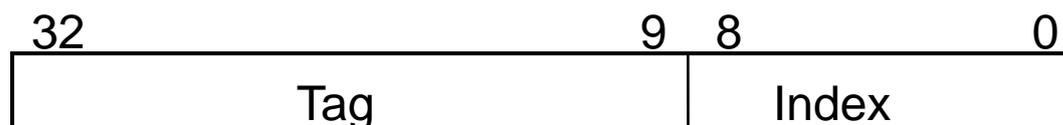


圖 2-7 : 512 block 之集合關聯式記憶體位址組成

本設計以 TSMC 0.18um cell-base 製程實作，經 Cadence tools 加入 time constrain 模擬結果分析時脈，結果如圖 2-8 顯示，左邊為 ALU 級的最長週期、右邊為 REG 級的最長周期，最長周期仍是以 ALU 級的運算為主，REG 級仍沒有成為 critical path，並沒有造成時脈上的降低，證明其實作上是可行的。

REG 級因為增加運算及選擇，造成 REG 級的處理週期增加，但可以選擇 Critical path 極大的處理器，例如 DSP，因為要在 1 個週期完成乘加，增加效能，所以此時 REG 級加入加法器也不會超過 DSP 的 critical path，所以也不會對處理器最高時脈產生影響。

因為本設計會隨著處理器的管線設計，架構，處理器應用需求，而有所不同，在此不詳加考慮、討論，隨著處理器設計者去抉擇。本論文以實驗室處理器驗證以第一種方法，即可解決此問題。

Startpoint: alu1/mb_reg_6			Startpoint: Reg_file_0/reg_r_data1_reg_1		
Endpoint: alu1/jump_addr_reg_7			Endpoint: Reg_file_0/reg1_foward_reg_30		
Path Group: CLK Path Type: max			Path Group: CLK Path Type: max		
Des/Clust/Port	Wire Load Model	Library	Des/Clust/Port	Wire Load Model	Library
CHIP	tsmc18_w110	slow	CHIP	tsmc18_w110	slow
Point		Incr Path	Point		Incr Path
...			Reg_file_0/reg_r_data1_reg_1/CK (SEDFFY4)	0.00 #	0.00 r
alu1/u_DW02_mult_0_U638/Y (AOI21X4)	0.28	3.55 r	...		
...			Reg_file_0/U3965/Y (XOR2X4)	0.27	8.66 r
alu1/U7046/Y (NOR2X1)	0.10	8.89 f	Reg_file_0/U4011/Y (AOI22X4)	0.09	8.75 f
alu1/U3213/Y (MX2X2)	0.25	9.14 f	Reg_file_0/U4023/Y (AOI21X4)	0.17	8.92 r
alu1/jump_addr_reg_7/D (SDFFXL)	0.00	9.14 f	Reg_file_0/reg1_foward_reg_30/D (SEDFHGX1)	0.00	8.92 r
data arrival time		9.14	data arrival time		8.92
...			...		
data required time		9.14	data required time		8.94
data arrival time		-9.14	data arrival time		-8.92
slack (MET)		0.00	slack (MET)		0.02

圖 2-8 critical path 時間模擬結果

- Data Hazard 部分

在管線執行部分，因為需要提前處理記憶體指令相關的資料，所以在執行記憶體相關的指令時，會因為資料尚未寫入暫存器而發生讀取之暫存器錯誤數值的資料危障，如圖 2-9 所示。

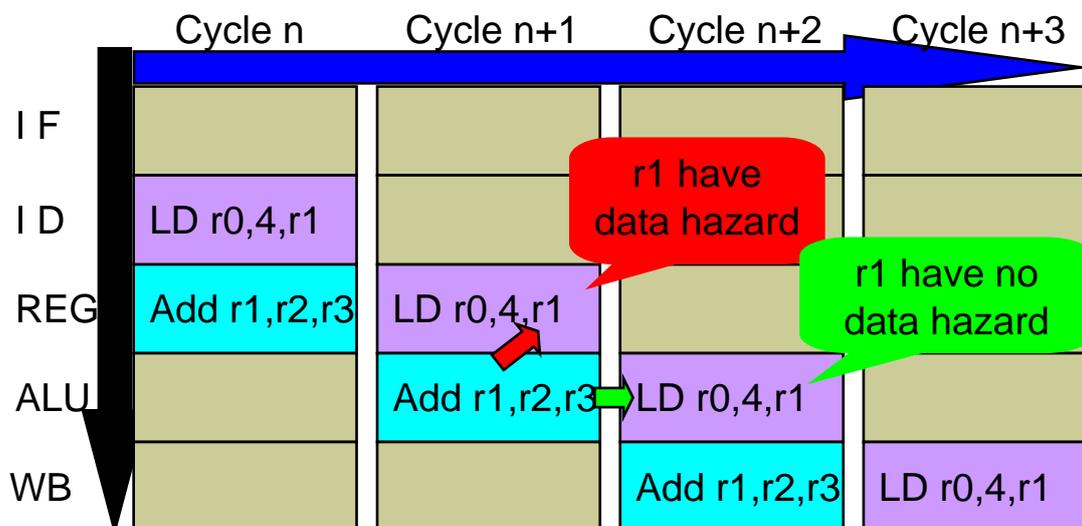


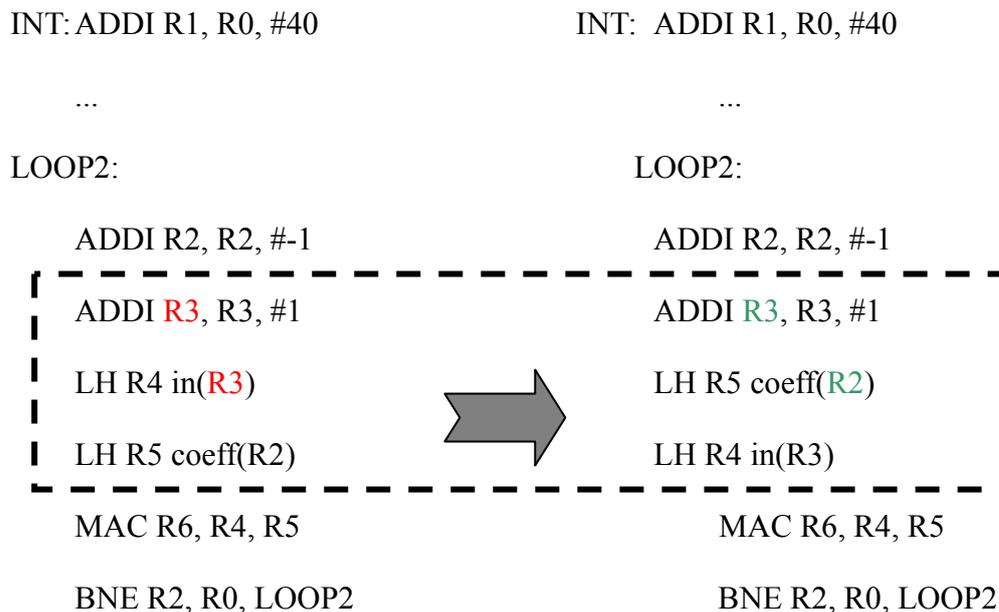
圖 2-9：處理器管線前饋圖

在正常的管線級中，Add r1,r2,r3 指令在 ALU 級產生的數值，需要前饋 (Forwarding)給下一週期的 ALU 級，去計算 LD r0,4,r1 的記憶體位址，以避免資料危障，這樣 LD 指令才能再 WB 級處理記憶體的存取，但是在我們的演算法中，本設計需要在 REG 級中計算記憶體位址的前半部的位元，但此時 r1 的值不但尚未寫入 REG 級中，甚至仍未經過 ALU 級的計算，也就是在 Cycle n+1 時，本設計需要再 REG 級計算 r1+4 的數值，但此時 r1 的數值正在 ALU 級中計算，此時 REG 級中讀取的 r1 數值是錯誤的，因此會發生資料危障，發生資料危障會造成管線暫停，增加總處理週期，造成效能的下降。

因此為了解決管線中因為記憶體相關指令提前計算而產生的資料危障，本設計也為此提出兩種解決方法，希望藉此能解決問題。

1. 藉由編譯器最佳化，避開資料危障的發生:

將會發生資料危障的部份，經由編譯器的重新編排，將發生資料危障的指令經由 slot 的技術，插入不相干的指令，在不影響指令效能及結果下，執行最佳化。例如 以下為 FIR 的一段程式，在 LH R4 in(R3)指令前，執行的 ADDI R3, R3, #1 因為跟 LH 指令有資料相依的危險，會發生資料危障，可藉由指令順序改善，避開因為讀取指令產生的資料危障。



2. 增加 REG 級前饋路徑

另外本系統也提供簡單的前饋系統去改善，加入特定的指令去前饋，例如 如果記憶體相關指令之前的指令有資料相依的情形發生，則我們使用之前 REG 級中加入的加法器去做運算提供 REG 級前饋資料，從 REG 級前饋至 REG 級就不會有資料危障的發生，如圖 2-10 所示，因為只有加法器單元，所以也只針對使用加法來計算記憶體位址的指令有前饋的效果，反之若不為加法的資料相依，也就是使用到乘法或邏輯位移的記憶體位址運算，則無法提供前饋，只能使用編譯器最佳化來避免資料危障。

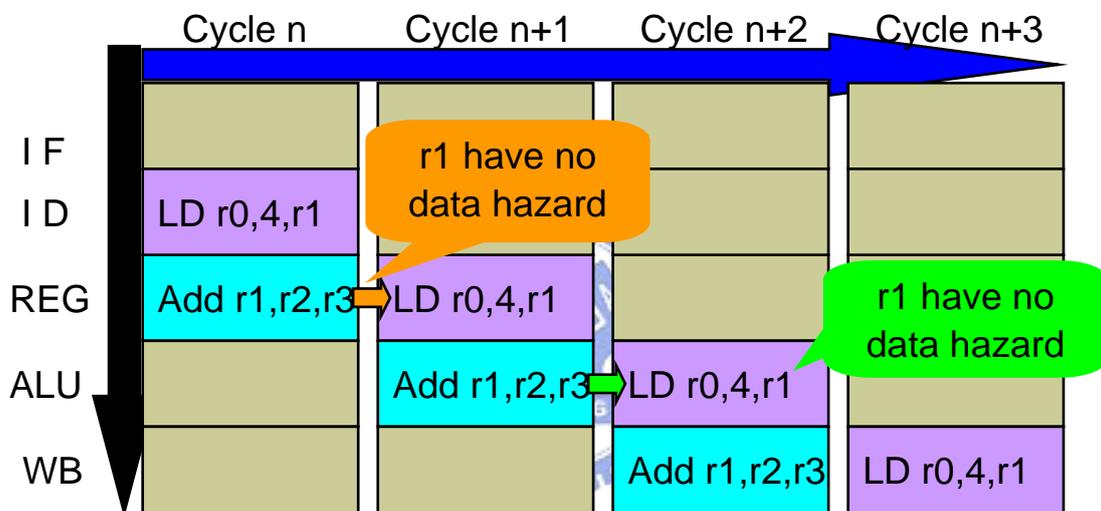


圖 2-10: 管線 REG 級前饋改善圖

本管線設計中 LH R4 ,in (R3) 會跟 ADDI R3,R3,#1 發生資料危障，但因為記憶體相關指令 LH 之前所處理的指令是為 ADDI，是可以使用 REG 級中的加法器去運算其 Index 部分所需要的位元的加法，再將計算的數值回饋至 LH 指令在 REG 級中執行的 R3+1 的低位元部份的加法，並不會造成資料危障的發生。但此方法只能解決因為特定排序的資料危障，若是乘法指令在記憶體指令之前所造成的資料危障，則必須經由編譯器來解決或者暫停管線。

● Instruction cache 部分

本論文提出之管線設計只改善資料快取部份效能，沒有針對指令相位式快取記憶體效能提升之改善去設計，有以下幾點原因

1. 指令快取記憶體增加相位式設計對於效能並沒有太大的影響:

因為指令快取記憶體所增加的相位週期會隨著管線的持續進行，而只在初始及發生指令跳躍時才會影響一個週期效能，並不是每次存取指令快取記憶體都會造成管線暫停，相較於相位式資料快取記憶體每次存取都有可能造成管線暫停而言，相位式指令快取記憶體本身可改善的空間不大，不如不要增加設計負擔。

2. 指令快取記憶體之關連性深度較資料快取記憶體淺:

指令快取記憶體因為指令有較佳的區域性及時間性，所以命中率本來就很高，最高可達到95%~98%左右的命中率，因此指令快取記憶體通常在關連性快取記憶體設計中，常見的通常直接使用直接定址快取記憶體設計或者最多只會使用到2-Way 關聯性設計左右，例如TMS320C6211 只搭配4KB的direct mapped L1指令快取記憶體，因此本設計重點在於改善關聯性快取記憶體設計之功率消耗上的浪費，所以針對相位式資料快取記憶體改善可以達到最高效益。

2.2 相位式快取記憶體之高效能存取管線設計效能分析

2.2.1 相位式快取記憶體之高效能存取管線設計之功率分析

在功率計算方面，仿效先前提到的論文中，以快取記憶體的驅動次數作為粗略功耗的公式的參數，來推導本設計估計功率消耗的公式，並加入面積的參數來評估各設計詳細的功率消耗。

$$\text{Activity Ratio} = \frac{\text{ACTsentry-tag}}{\text{ACTconventional}} \quad (1)$$

在公式(1)中[8]，ACTsentry-tag 表示在 Sentry tag 架構下，需要驅動的集合次數，ACTconventional 表示在傳統快取記憶體架構中，需要驅動的集合次數。並以兩者的驅動次數比例做為評斷功率消耗的比例。

$$\begin{aligned} W_{\text{Ave}} &= HR \times \left(1 + (W - 1) \times \frac{1}{2^S} \right) + (1 - HR) \times W \times \frac{1}{2^S} \\ &= HR + \frac{HR \times W}{2^S} - \frac{HR}{2^S} + \frac{W}{2^S} - \frac{HR \times W}{2^S} \\ &= \left(1 - \frac{1}{2^S} \right) HR + \frac{W}{2^S} \end{aligned} \quad (2)$$

公式(2)[8]為 Sentry tag 架構計算集合平均驅動次數的公式， W_{Ave} 表示集合平均驅動次數，HR 表示 Hit Ratio，W 代表集合數，S 表示 Sentry tag bit 大小，影響到過濾的非必要執行集合的機率，公式分為命中時及失誤時分開計算，因為命中時必定會有 1 個集合需要驅動，失誤時則全以機率去推導。

本設計根據演算法及前述論文的計算方式，並參考 Phased cache、Sentry tag cache 演算法推導出本設計消耗功率的公式表(3)，以面積及驅動次數作為評量功率的標準，將快取記憶體中的 Tag 跟 Data 兩部分面積做為功率基本單位，在累計其驅動 2 者的次數做統計平均。

$$\begin{aligned}
 Power_cache = & Hit_ratio \times Way \times (Tag_power \times H_T_enable_ratio + \\
 & Data_power \times H_D_enable_ratio) \\
 & + Miss_ratio \times Way \times (Tag_power \times M_T_enable_ratio + \\
 & Data_power \times M_D_enable_ratio) + Other
 \end{aligned} \quad (3)$$

- Power_cache** : 快取記憶體總功率消耗
- Hit_ratio** : 快取記憶體命中率，隨著測試程式不同改變
- Miss_ratio** : 快取記憶體失誤率，數值為 1-Hit_ratio
- Way** : 集合，快取的區塊個數
- Tag_power** : 快取記憶體中 Tag 部分功率消耗，隨快取記憶體大小而變，可利用 Cadence tools SRAM 模組測量
- Data_power** : 快取記憶體中 Data 部分功率消耗，隨快取記憶體大小而變，可利用 Cadence tools SRAM 模組測量
- H_T_enable_ratio**: 命中時，Tag 部分驅動的機率，依照設計有不同計算的方式
- H_D_enable_ratio**: 命中時，Data 部分驅動的機率，依照設計有不同計算的方式
- M_T_enable_ratio**: 失誤時，Tag 部分驅動的機率，依照設計有不同計算的方式
- M_D_enable_ratio**: 失誤時，Data 部分驅動的機率，依照設計有不同計算的方式
- Other** : 其他單元的功率消耗，在此表示 Sentry tag cache 使用 CAM 所帶來的功率消耗

如表 2-2 所示，本表以 4KB ,4-way(Way=4) ,8-bit Tag bit(tag= 8), 2 bit Sentry tag bit(S=2), 32-bit data bit(data=32) 的快取架構做為例子來比較，另外此圖表假設單位 CAM 所耗的 power 為單位 SRAM 的兩倍[9]作為評估功率消耗的比較。

表 2-2：傳統、Sentry tag 架構、phase 架構功率公式比較表

	Convention	Sentry tag	Phased cache/ This Work
Hit_power	Way(tag+data)	$(tag + data) + (Way-1) * (tag+data)(1/S^2) + Way * CAM * S$	Way*tag + data
Miss_power	Way(tag+data)	$Way * (tag+data)(1/S^2) + Way * CAM * S$	Way*tag
Power: Hit*Hit_power + Miss*Miss_power	$H * Way(tag+data) + M * Way(tag+data)$	$H * (tag + data) * (1 + (Way-1)/S^2) + M * (tag + data) + Way * CAM * S$	$H * (Way * tag + data) + M * Way * tag$
Power : (4-way) (95%Hit)	100%	51%	39%

Tag 跟 Data 分別表示 tag cache 跟 data cache 相位所消耗的功率，依快取記憶體大小不同而變化。將快取記憶體功率消耗分成命中時的功耗及失誤時的功率消耗，會依照快取記憶體使用的演算法而有所不同，以 4-Way 集合關聯性快取記憶體為例，傳統設計需要同時驅動 4 個 way 的 tag cache 及 data cache 部份，而 Sentry tag 架構在命中時約需消耗 1.75 個 way 的 tag 加 data 的 power，在失誤時約需消耗 1 個 way 的 tag 加 data 的 power，可比傳統快取記憶體架構省下約 49% 的功率消耗，此結果與 Sentry tag 架構論文數據大致相似。而使用 Phased cache 架構，雖然在處理 tag 部分都需要驅動 4 個 way 集合，但命中時只需要處

理 1 個 Way 集合的 data cache 部分，失誤時則不需要驅動任何一個 data cache 部分，於是比 Sentry tag 架構省下較多的功率消耗，大約能省下 61% 左右的功率消耗。

因為主要是針對集合關聯性快取記憶體降低功率消耗的設計，所以不同的集合關聯性快取記憶體條件下省略的功率消耗當然也是不同，如圖 2-11 所示在相同參數不同集合關聯數條件下，代入公式(表 2-1)去統計，以傳統的快取記憶體架構的功率消耗為標準，在 2-Way 關聯性快取記憶體條件下，Sentry tag cache 可以省下約 32% 的功率消耗，Phased cache 可以省下約 42% 左右的功率消耗，而在 4-Way 關聯性快取記憶體條件下，Sentry tag cache 可以省下約 48% 的功率消耗，Phased cache 可以省下約 61% 左右的功率消耗，而在 8-Way 關聯性快取記憶體條件下，Sentry tag cache 可以省下約 57% 的功率消耗，Phased cache 可以省下約 70% 左右的功率消耗，可以看出集合數越多越能達到降低功率消耗的效果，另外 Phased cache 跟 Sentry tag 架構在越多集合關聯性下，降低功率消耗的差距雖然有變小，如圖 2-12 所示，在 2-Way associative 條件下，Phased cache 降低功率消耗約為 Sentry tag 的 1.3 倍左右，4-Way 集合關聯性時約為 1.23 倍，8-Way 集合關聯性時約為 1.22 倍，16way、32-Way、64 way 集合關聯性時皆為 1.21 倍左右，表示在集合關聯性數越多的情形下 Phased cache 降低功率消耗的效果依然是 Sentry tag cache 120% 上下，並不會隨著 Way 數增加而有明顯效率變差。

測試程式的命中率不同，依照公式及演算法推演，也會帶來不同的效果，如圖 2-13 所示，命中率越高降低功率消耗越低，但相位式快取記憶體過濾非必要執行集合的效果是一樣的，只是需要驅動的集合次數增加，因此在降低功率消耗上效率有所不同。

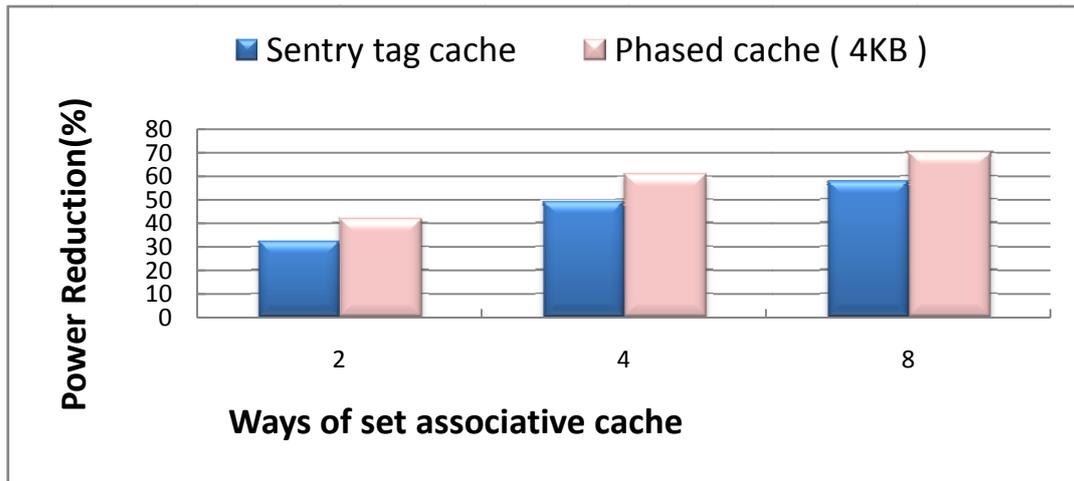


圖 2-11: Sentry tag 架構與相位式架構降低功率消耗比較圖

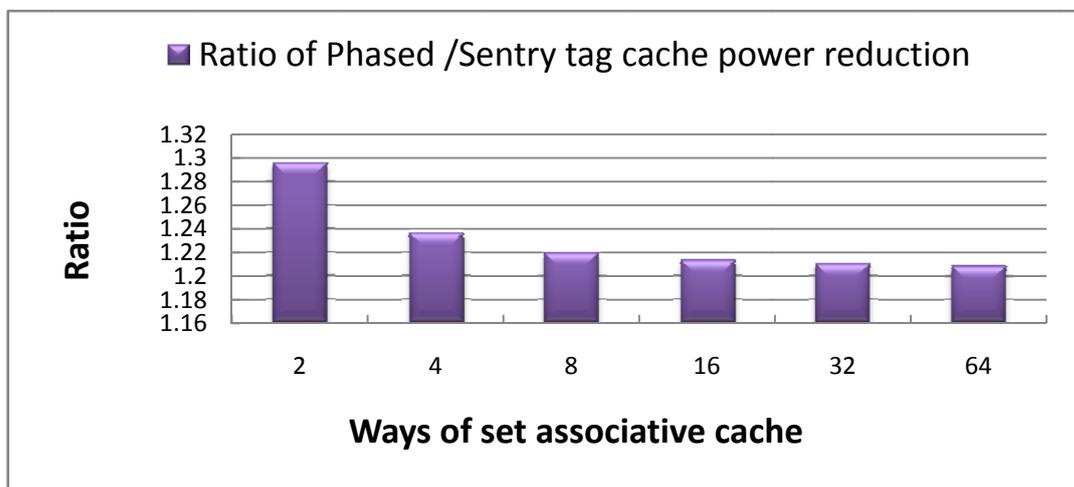


圖 2-12: Sentry tag 架構與相位式架構降低功率消耗比例關係圖

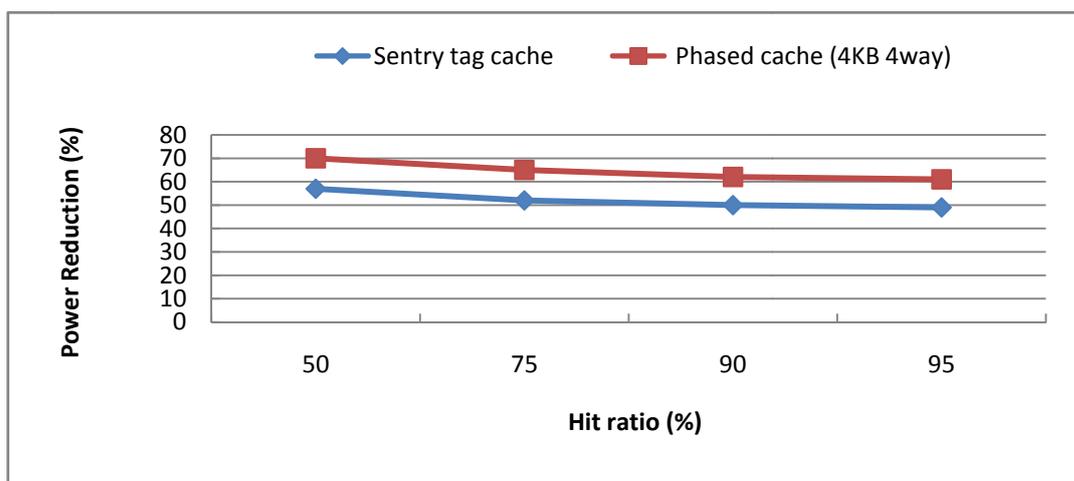


圖 2-13: Sentry tag 架構與相位式架構功率消耗跟命中率之關係圖

2.2.2 SimpleScalar 功率及週期分析

本論文仿照其他論文分析測量，採用 SimpleScalar[11]，圖 2-14 來模擬本設計及 Sentry tag architecture 演算法在處理標準效能評估程式 SPEC95 及 SPEC2000 (Standard Performance Evaluation Corporation)[12]的效果，隨機取用七種測試程式，包含整數標準效能評估程式及浮點數標準效能評估程式，表 2-3 有各標準效能評估程式的簡介，模擬結果如圖 2-15、2-16、2-17 所示，大致與推導相似，降低功率損耗平均結果分別為 Phased cache 2-Way:44%, 4-Way:62%, 8-Way:71% ,Sentry tag architecture 2-Way:34%, 4-Way:49%, 8-Way:58%，與 Sentry tag paper 所提結果 4-Way 46% 差距不大。

由以上公式推導，數據分析皆能證明 Phased cache architecture 在相同條件下，降低功率損耗的效果是比 Sentry tag architecture 較為優異。

而 Phased cache 跟 Way-prediction cache 的比較，圖 1-6 已有其比較結果，由圖 1-6 可以看出 Phased cache 在降低功耗的效果也是比 Way-prediction cache 優異。

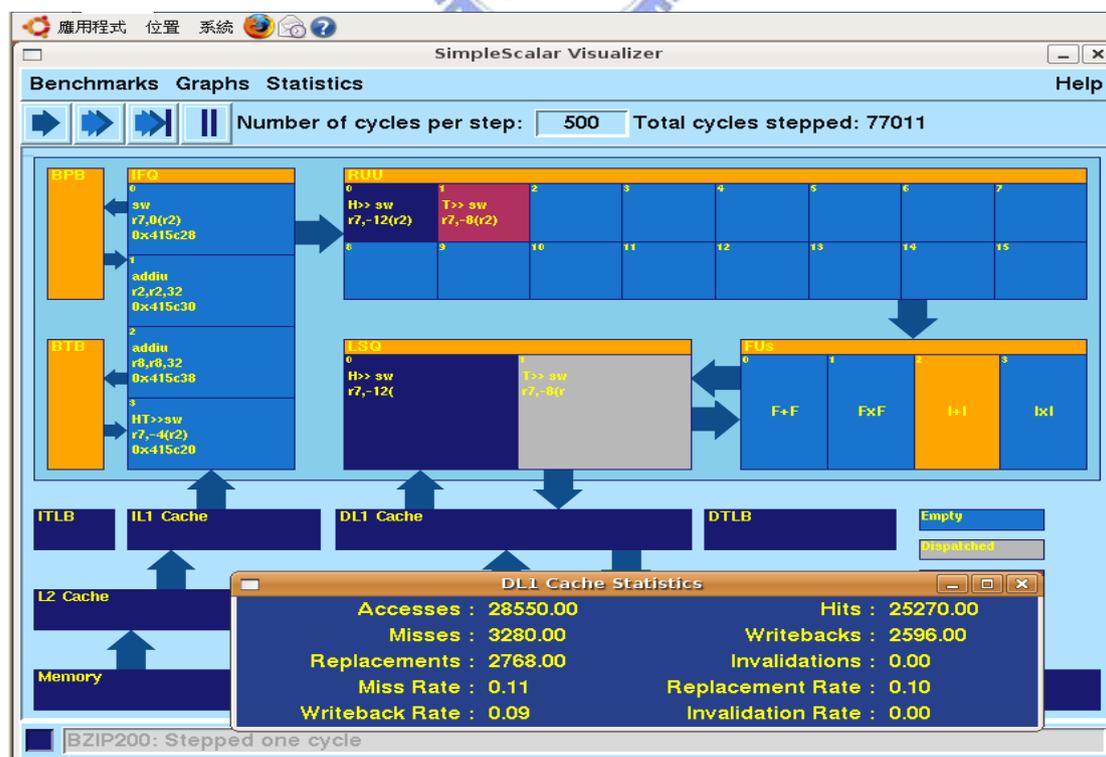


圖 2-14 : SimpleScalar 軟體介面

表 2-3：標準效能評估程式介紹

Items	SPEC	Description
compress95	CINT95	A in-memory version of the common UNIX utility.
li95	CINT95	Xlisp interpreter.
gzip	CINT2000	Compression
gcc	CINT2000	C Programming Language Compiler
ammp	CFT2000	Computational Chemistry
parser	CINT2000	Word Processing
bzip2	CINT2000	Compression

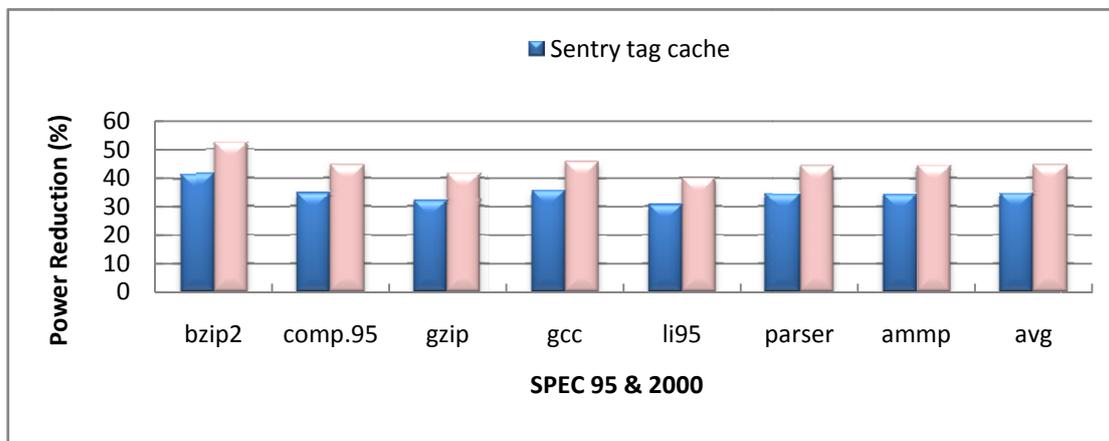


圖 2-15: 2-Way 時 Sentry tag 與相位式架構降低功耗比較

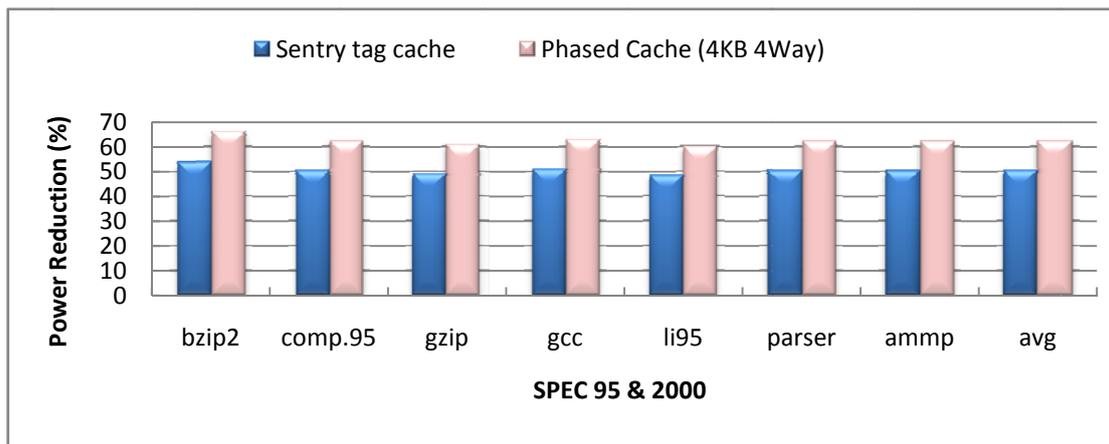


圖 2-16: 4-Way 時 Sentry tag 與相位式架構降低功耗比較

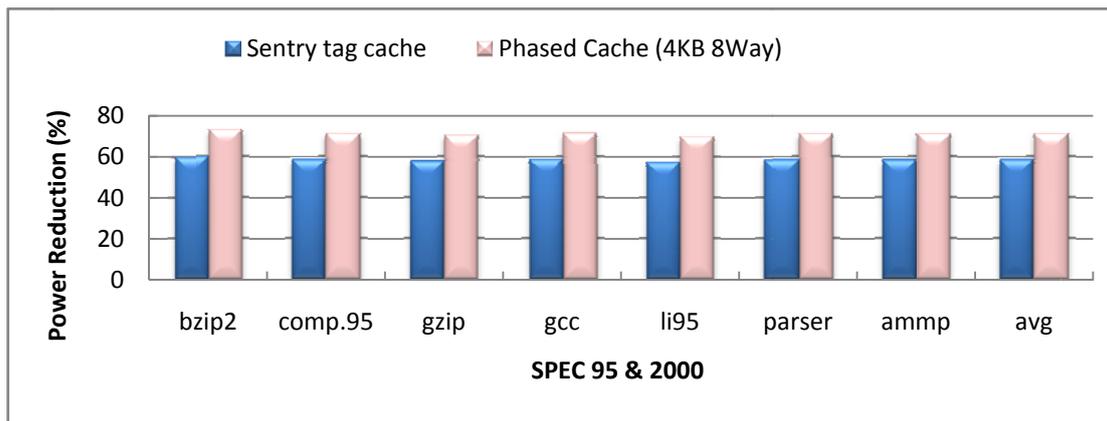


圖 2-17: 8-Way 時 Sentry tag 與相位式架構降低功耗比較

另外也使用 SimpleScalar 模擬此設計改善相位式快取記憶體時間效能的效果，使用依序執行(in order issue)，分別量測相位式快取記憶體與本設計改良相位式快取記憶體的效果，如圖 2-18 所示，在平均 20%的讀取指令中改善了總週期平均 8%的執行時間，相對就是減少了快取記憶體讀取時間的 39%左右，如圖 2-19 所示。分析此結果，雖然本設計將相位式增加的 1 週期延遲融入管線中，減少快取讀取的管線級數，去除管線暫停，但因為未改善前的相位式記憶體管線中，並非每次讀取相位式資料快取記憶體皆會造成管線暫停，必須視資料危障的距離週期決定，如以下程式為例，在相位式資料快取記憶體下，雖然 R4 跟 R5 都需要兩週期的讀取時間，但只有 R5 的讀取會造成管線兩週期的暫停，R4 的讀取時間會被 R5 讀取時間掩飾掉，因此這兩個讀取指令只有一個管線暫停週期，所以加入本高效能管線設計雖然能改善相位式快取記憶體造成的管線暫停，能完全減少相位式快取記憶體造成的週期增加，但從管線及程式指令架構考量，此數據結果是在合理的範圍之內。

LH R4 ,0(R2)

LH R5 ,0(R3)

MAC R6, R4, R5

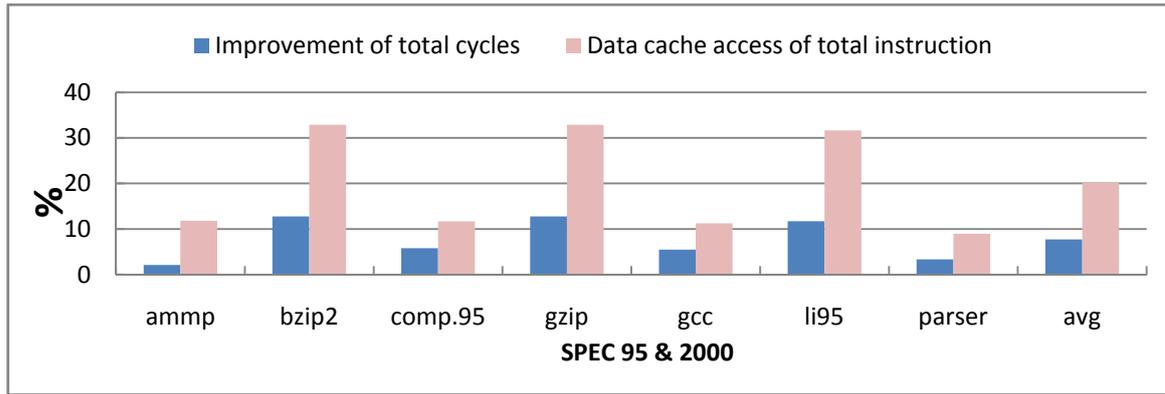


圖 2-18: 總周期改善比例與讀取指令比例

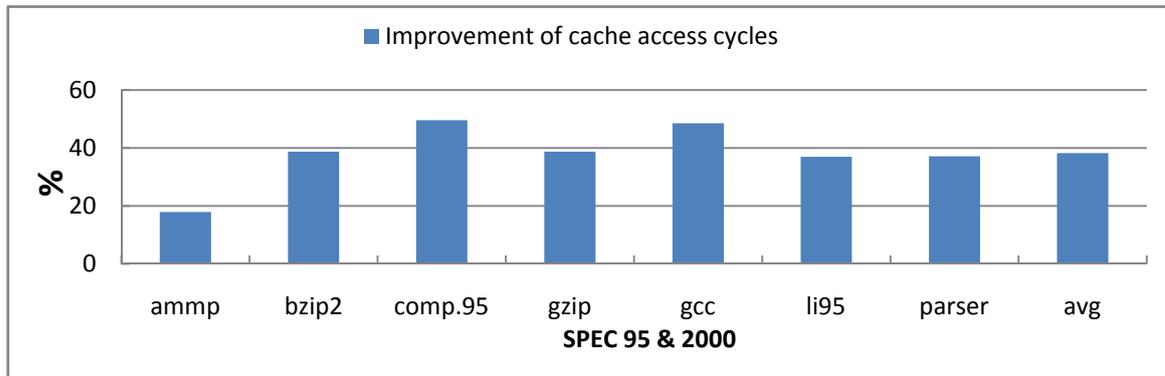


圖 2-19: 快取記憶體讀取周期改善比例

2.2.3 高效能之記憶體存取管線設計分析

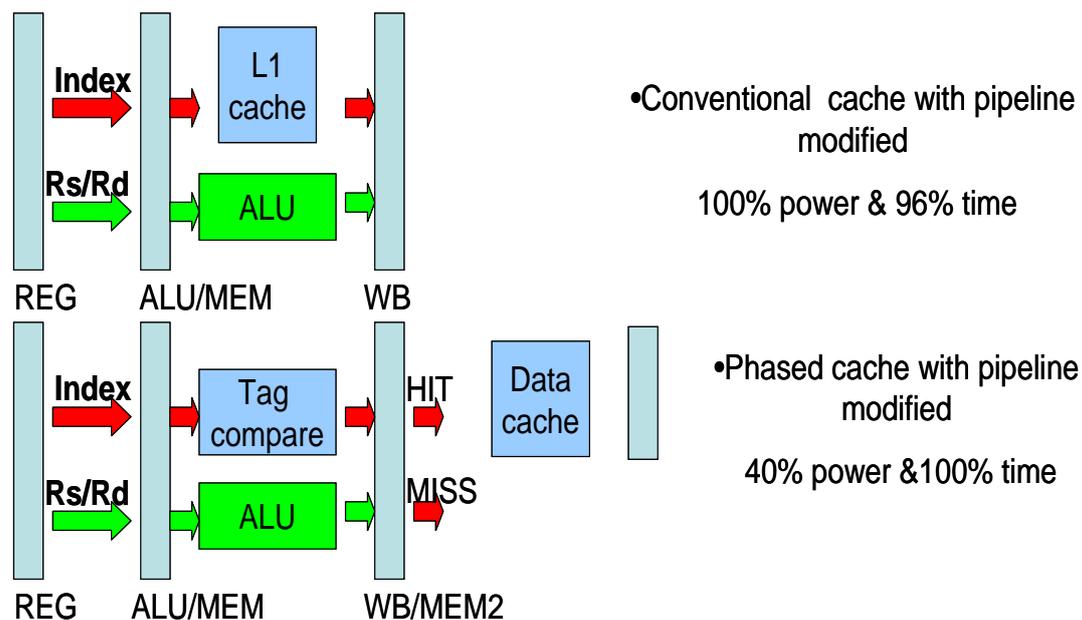


圖 2-20：管線改良之分析圖

從圖 2-20 來分析此管線設計的使用，若是單純只加入此高效能之記憶體存取管線設計，如圖 2-20 上半部管線，在 ALU 級就能獲得快取資料，在時間數據上，從上面的改善 37%管線暫停的數據中分析，可以得到只有 37%的 load 指令在 2 個週期以內會造成管線暫停，也就是有 63%的讀取資料指令不急著在 2 週期內讀取到資料，由此我們推測 37%指令中只有一半的會在 1 個周期內造成管線暫停，也就是 18%左右，也就是約略能加速 4%左右的整體效能。但若是結合相位式快取設計，除了能降低 40%~70%左右的功率消耗，又能改善相位式快取記憶體耗時的缺點，在 20%load 程式中，改善 37%快取讀取時間，改善 8%的整體時間。因此本設計結合相位式快取設計可以得到最大的效益，如表 2-4 所示。

表 2-4：整合設計效能、功率比較

	相位式快取	高效能存取管線	功率	執行時間
傳統	X	X	100%	100%
相位式快取設計	O	X	40%	108%
高效能存取管線	X	O	100%	96%
本論文設計	O	O	40%	100%

2.3 結語

本章節針對相位式快取記憶體及高效能存取管線設計描述其原理及其架構，從優缺點去分析，改善及測試，再從功率及效能上去評量，使用 SimpleScalar 軟體模擬結果，從解決相位式快取記憶體造成的管線暫停中，如下圖 2-21，改善 38% 左右的快取處理時間，也就是改善 8% 的整體效能，又能降低 40% 至 70% 左右的功率消耗，只需針對管線增加些微硬體電路，增加約 6% 的硬體成本，即可達到此效果，跟其他設計比較在功耗、效能、成本上都較為優秀，如表 2-5。

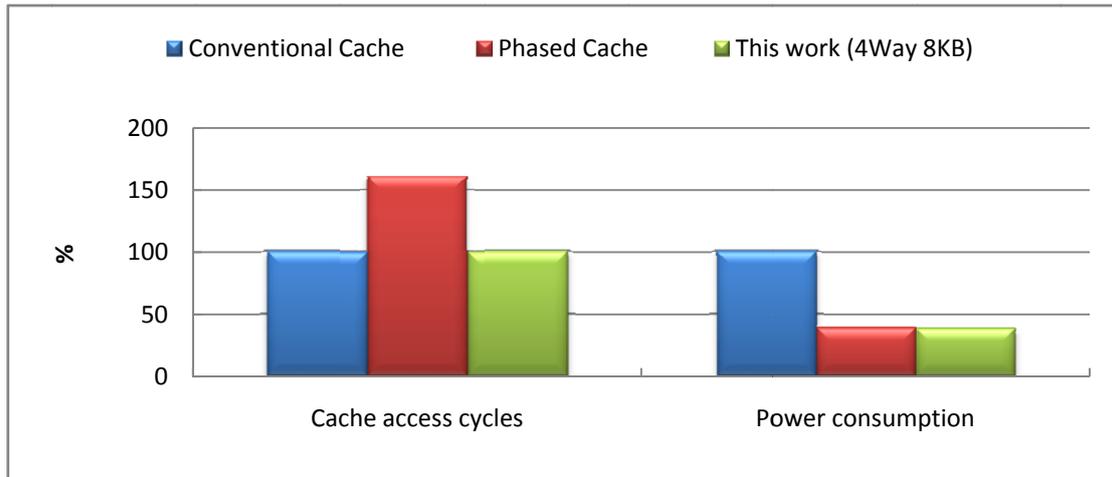


圖 2-21: 高效能相位式快取記憶體設計改善比較圖

表 2-5: 本設計與其他低功耗快取設計分析比較

	Conventional Cache	Sentry tag Cache	Way Prediction Cache	Phased Cache	This Work
降低快取 功耗效果	Low(0%)	Middle(34%)	Middle(30%~40%)	High(44%)	High(44%)
快取延遲 /暫停	Low (1 cycle)	Low (1 cycle)	Middle (1~2 cycle)	High (2 cycles)	Low (1 cycle)
設計複雜 度/硬體 成本	Low	High (CAM)	Middle	Middle	Middle (6% area)

第三章 低功耗嵌入式處理器設計

低功耗消耗相位式快取記憶體控制器及其管線設計硬體地位上不能單獨存在運作，需搭配一個處理器設計。因此，本論文也製作出一個 32 位元嵌入式處理器核心，以呈現完整的應用。以下章節將說明此處理器的特性、指令集、軟體開發環境及可搭配的其他設計作一介紹。

3.1 低功耗嵌入式處理器架構

3.1.1 低功耗嵌入式處理器核心

此處理器以低功耗為設計考量方向，主體以 RISC 架構為主，分別增加指令快取記憶體、資料快取記憶體控制器及匯流排編碼解碼器來降低處理器功率消耗的主要部份：快取記憶體及匯流排傳輸，指令快取記憶體設計為具有使用者可調性主從式指令快取記憶體控制器[13]，資料快取記憶體設計為本設計架構，匯流排編碼解碼器為功率感知之匯流排編碼解碼器設計[14]。希望能結合各自降低功率消耗的設計理念發展成一類以低功耗為訴求的嵌入式處理器。

本論文設計的低功率消耗嵌入式處理器擁有七級管線架構的設計[15]。當一個指令透過程式計數器 (Program counter)，指令快取控制、讀取，將指令抓取進入處理器內部後，會先對該指令進行解碼，接著到暫存器提取所需要的資料到 ALU 中執行，最後將結果存回暫存器或記憶體中。記憶體周邊裝置的存取透過快取控制處理器來規劃，周邊裝置(或 IP)可利用標準的 APB 規格掛上匯流排，將外部的取樣訊號透過 APB 匯流排，將資料存取於內部快取記憶體及外部主要記憶體之間，或內部運算的結果送到周邊裝置上輸出，架構如圖 3-1。

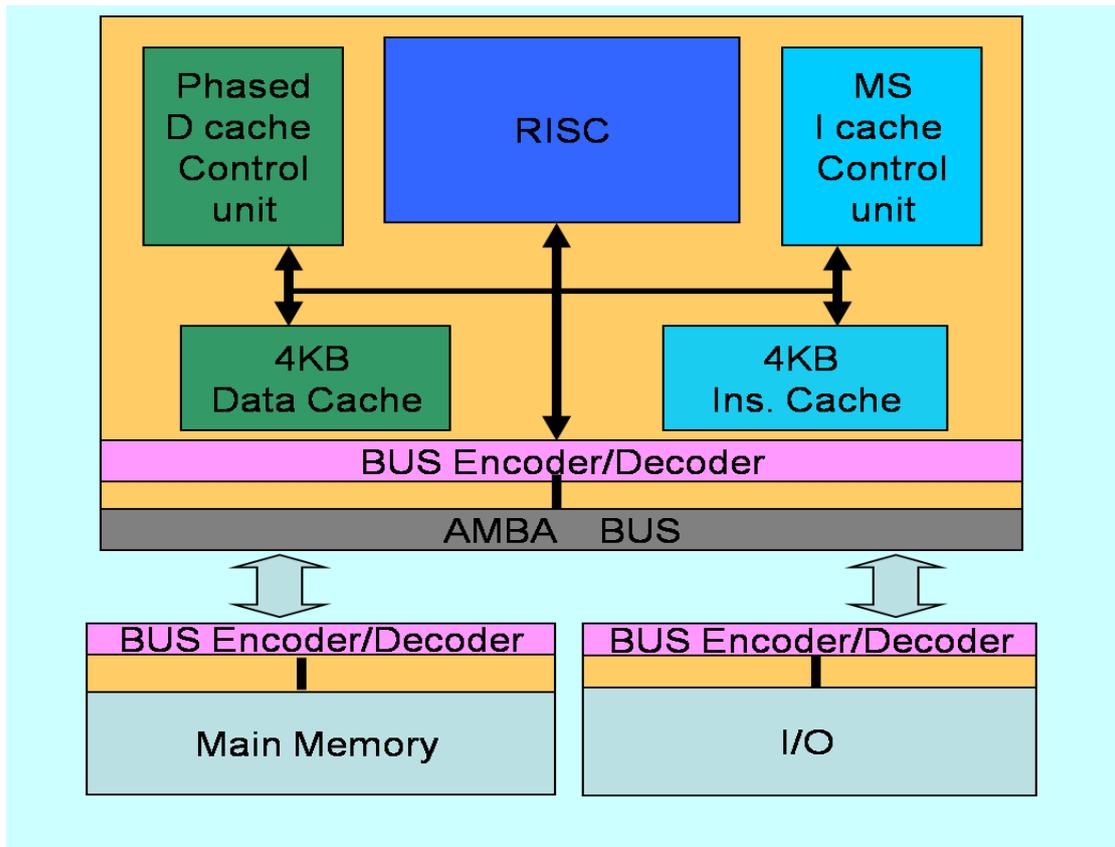


圖 3-1 :處理器組成架構圖

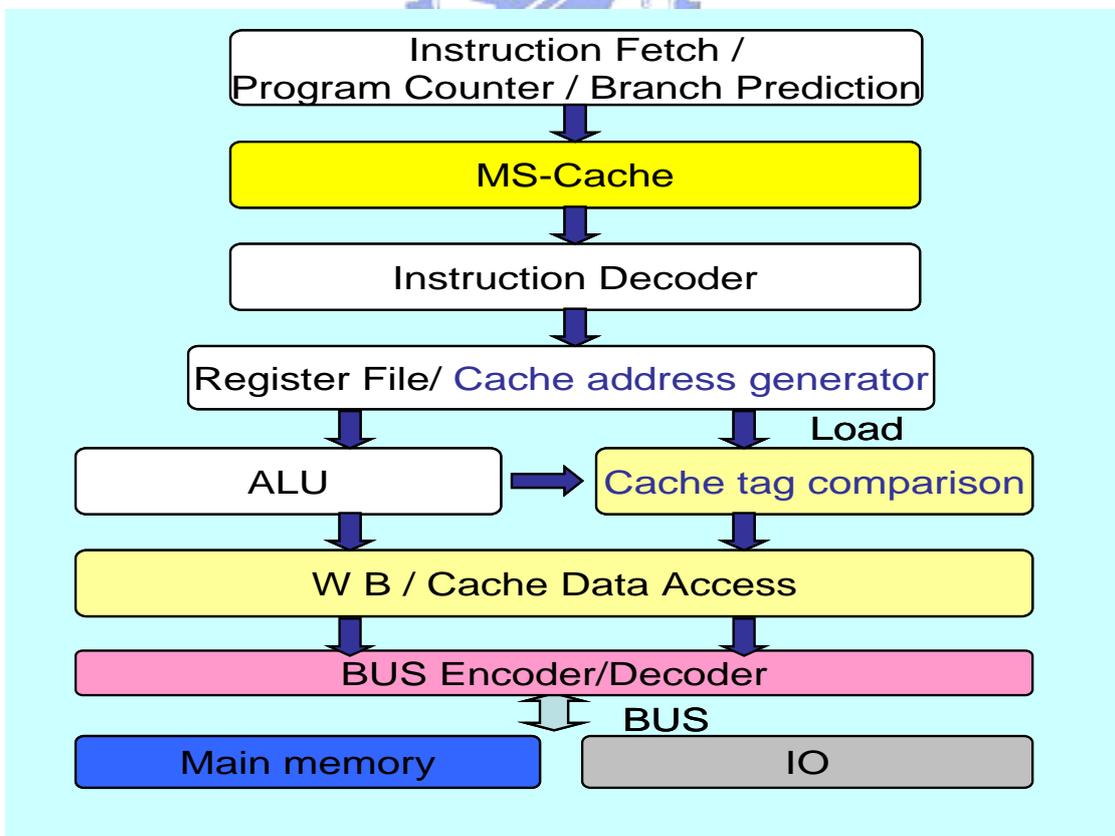


圖 3-2 :處理管線流程圖

圖 3-2 簡介此處理器管線的流程，此處理器擁有七級管線架構，七級管線包含程式計數/分支預測/指令抓取模組、指令快取記憶體模組(2 級)、指令解碼模組、暫存器模組、算數邏輯單元模組/快取 Tag 級以及寫回/快取資料級，以下分別描述其主要的工作簡述如下：

Program Counter/Branch Predict/ Instruction Fetch：在硬體架構最上層，目的是把程式計數器加一，並處理跳躍，最後決定程式計數器的值。再把程式計數器的值，轉成程式記憶體的位址，送進指令快取記憶體去抓取指令。其中，由於有些指令在 ALU 級時會需要處理指令中的程式計數器值，所以必須把程式計數器值一級一級地傳下去，因此，程式計數器值會傳進下一級的暫存器。在處理跳躍指令時，為了避免浪費跳躍指令發生後一個指令被抓取，設定兩個訊號做為防止跳躍發生時的指標，用以表示現在的管線是否在 Stall 狀態，決定要不要執行該級動作。

MS-I-cache：依照 PC 值讀取相對應的資料，若發生失誤，則至主記憶體搬移數筆資料取代。採用相位式設計，先經過 Tag 存取比對，再讀取命中的資料。並針對跳躍的指令加強，提高命中率。

Instruction Decoder：這一級的目的是把指令依照對應的指令碼去做解碼，指令中有兩個來源暫存器的位置，一個目的暫存器的位置。這些位置可以對下一級的暫存器組取得來源運算元並提供未來 ALU 級寫回的目的位置。

Register File：處理核心有 16 個通用暫存器。處理核心有額外的 16 個中斷暫存器來處理外部中斷、內部優先權中斷、與其他 IP 狀態設定所使用的暫存器，暫存器組為 2 讀 1 寫的格式，負責提供解碼器所解碼出的來源運算元值，並將 ALU 級算出的目的運算元寫回。在此模組中，有很多連接 ALU 級模組的輸出信號，目的是把這些 ALU 級要用到的信號經由管線級送往 ALU 級中，利用這些信號來完成 Data forwarding 的動作。此外針對記憶體相關指令，特別計算其記憶體位址的部份數值供下一級 Tag 快取存取之用。

ALU/ Tag Cache access：本級功能是計算出邏輯或運算值，為了 Data forwarding 的實作，Data memory 及 Write back 這兩級隱含在 ALU 級內。Data forwarding 的機制是利用前面一級一級傳回的信號實作而成，目的是為了減少 RAW hazard。此級也平行記憶體相關指令，讀取 Tag cache 的數值跟計憶體位址做比對，判別快取命中失誤等訊息。

Write-Back/ Data Cache access：由 ALU 或 Memory 寫回 Reg File 的動作，或由 ALU 寫入 cache 及 Memory。若為記憶體相關指令則依命中的訊息驅動集合的資料存取。

除了以上基本的管線模組設計外，還有一些特殊硬體設計需求被強調出來，包含以下五種說明：

SIMD support (Single-Instruction stream ,Multiple-Data stream):

支援 8/16 bit SIMD 指令集架構，加速特定多媒體運算，如 8-bit 圖形運算，16bit 語音運算。

Bit Reverse：針對 FFT 運算所增加的 memory 定址模式[16]，例如位址 (01101)可被轉成(10110)的位置儲存。

一個指令週期完成乘加運算。

良好的 Data Forwarding 機制。

Condition Branch：預測採用 Prediction-untaken 方法設計。

3.1.2 低功耗嵌入式處理器指令集架構

處理器指令集共分五大類：資料搬移、算數邏輯運算、跳躍指令、SIMD 指令及其他類指令，列表如下：

處理器共提供 Direct、Reg to Reg、Indirect、Displacement (base add)、Index，Bit-Reverse 六大類的定址模式。

表 3-1：資料搬移指令列表

資料搬移指令：

Instruction	Op code	Example	Mode
MOVRC	000001	MOV rd,data	Direct
MOVRR	000010	MOV rd,rs	Reg-Reg
MOVVM	000011	MOV rd,address	Direct
MOVMR	000100	MOV address,rs	Direct
MOVVMRR	000101	MOV @rs2,rs	Indirect
MOVRRM	000110	MOV rd,@rs	Indirect
MOVARR	100010	MOV rd(a),rs(b)	Reg-Reg
MOVB	101111	MOVB rd,base(rs)	Displacement
MOVI	110000	MOVI rd,rs1(rs2)	Index
MOVREVRM	101010	MOV rd,address	Bit Reverse
MOVREVMR	101011	MOV address,rs	Bit Reverse
MOVREVMRR	101100	MOV @rs2,rs	Bit Reverse
MOVREVRM	101101	MOV rd,@rs	Bit Reverse

表 3-2：算數邏輯運算指令列表

算數與邏輯運算指令：

Instruction	Op code	Example
ADDRR	001000	ADD rd,rs1,rs2
SUBRR	001010	SUB rd,rs1,rs2
MULRR	001100	MUL rd,rs1,rs2
ADDRC	000111	ADD rd, data

SUBRC	001001	SUB rd, data
MULRC	001011	MUL rd, data
MACR	100111	MAC rd,rs1,rs2
MACC	110001	MAC rd,rs1,data
ANDRR	001110	AND rd,rs1,rs2
ORRR	001111	OR rd,rs1,rs2
XORRR	010000	XOR rd,rs1,rs2
INVR	010001	INV rd,rs

表 3-3：跳躍指令列表

跳躍指令：

Instruction	Op code	Example
JMP	010010	JMP address
JMPR	010011	JMP @rs
JBE	010100	JBE rs1,address
JNE	010101	JNE rs1,address
JMB	010110	JMB rs1,address
JLB	010111	JLB rs1,address
JBER	011000	JBER rs1,rs2,address
JNER	011001	JNBR rs1,rs2,address
JMBR	011010	JMBR rs1,rs2,address
JLBR	011011	JLBR rs1,rs2,address
CALL	100011	CALL address
RET	011110	RET

Address 的部分也可以是 Label，組譯器會自動轉換成對應的位址。

表 3-4：SIMD 指令列表

SIMD 指令：

Instruction	Op code	Example
MOVHLRC	110001	MOVHLRC rd, direct
MOVHURC	110010	MOVHURC rd, direct
ADDHRR	110011	ADDHRR rd,rs1,rs2
SUBHRR	110100	SUBHRR rd,rs1,rs2
MULHRR	110101	MULHRR rd,rs1,rs2
MACHR	100110	MACHR rd,rs1,rs2
ANDHRR	110110	ANDHRR rd,rs1,rs2
ORHRR	110111	ORHRR rd,rs1,rs2
XORHRR	111000	XORHRR rd,rs1,rs2
ADDBRR	111001	ADDBRR rd,rs1,rs2
SUBBRR	111010	SUBBRR rd,rs1,rs2
MULBRR	111011	MULBRR rd,rs1,rs2
ANDBRR	111100	ANDBRR rd,rs1,rs2
ORBRR	111101	ORBRR rd,rs1,rs2
XORBRR	111110	XORBRR rd,rs1,rs2

SIMD 指令集分成 8bit 以 B 表示，16bit 以 H 表示，分別對暫存器中的 32bit 資料，以每 8bit or 16bit 做運算，可以加速 8bit 及 16bit 的運算。

例如 MACHR Rd, RA, RB 指令，圖 3-3 顯示此指令的乘加運算，善用 32 位元的暫存器，一個指令周期完成原本需要兩個週期的運算：

$$Rd = ACC = ACC + A1 * B1 + A2 * B2$$

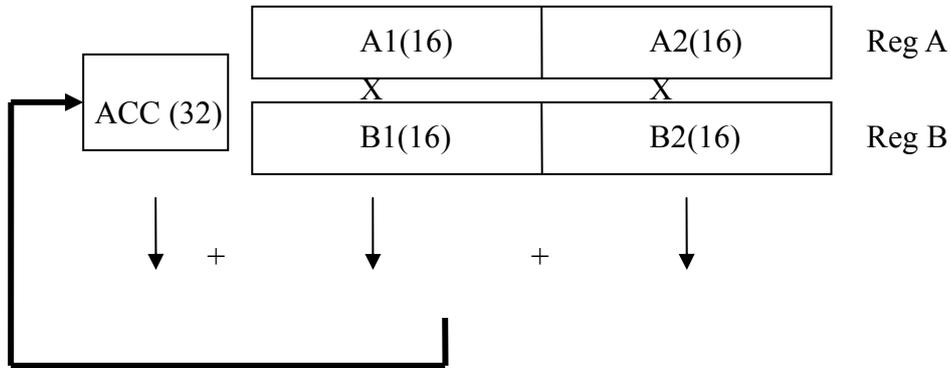


圖 3-3 : MACHR 指令運算

表 3-5 : 其他指令列表

其他指令：

Instruction	Op code	Example
SET	011100	SET A,rs
INTOK	011101	INTOK
SHR	100000	SHR rs
SHL	100001	SHL rs
ENDC	011111	ENDC

SET 指令是當嵌入式處理器沒有連接匯流排時，利用這指令可以設定兩個 16-bit 的 I/O port，與外界溝通；INTOK 是處理軟體中斷的指令，利用這指令可發出軟體中斷；SHR 將 rs 向右移一位元；SHL 將 rs 向左移一位元。

3.2 功率感知之匯流排編碼解碼器

3.2.1 功率感知之匯流排編碼解碼器原理

本處理器加入低功率消耗匯流排編碼器[14]，希望能經由此編碼器降低對外部資料、位址傳輸耗費的大量位元轉換功率消耗(switch activity power)。此編碼器可針對不同的資料流特性，自動採取較佳的編碼方式，使之編碼具備廣泛使用在各種資料傳輸皆能有效的降低匯流排傳輸的位元轉換功率消耗，且無論在編碼器或者解碼器所付出的硬體成本只需整顆處理器 1%左右成本，就可進一步加強本快取記憶體設計與外部記憶體溝通、傳輸資料的傳輸功率消耗。

如圖 3-4 所示，資料傳輸前會先跟前一筆傳輸資料的位元作比對，也包括控制訊號的比對，統計使用各演算法何者有較低的變化率，其演算法可以綜合各種編碼方式，取用較常使用的編碼作為比對選擇之一，以達到較佳的編碼效果。

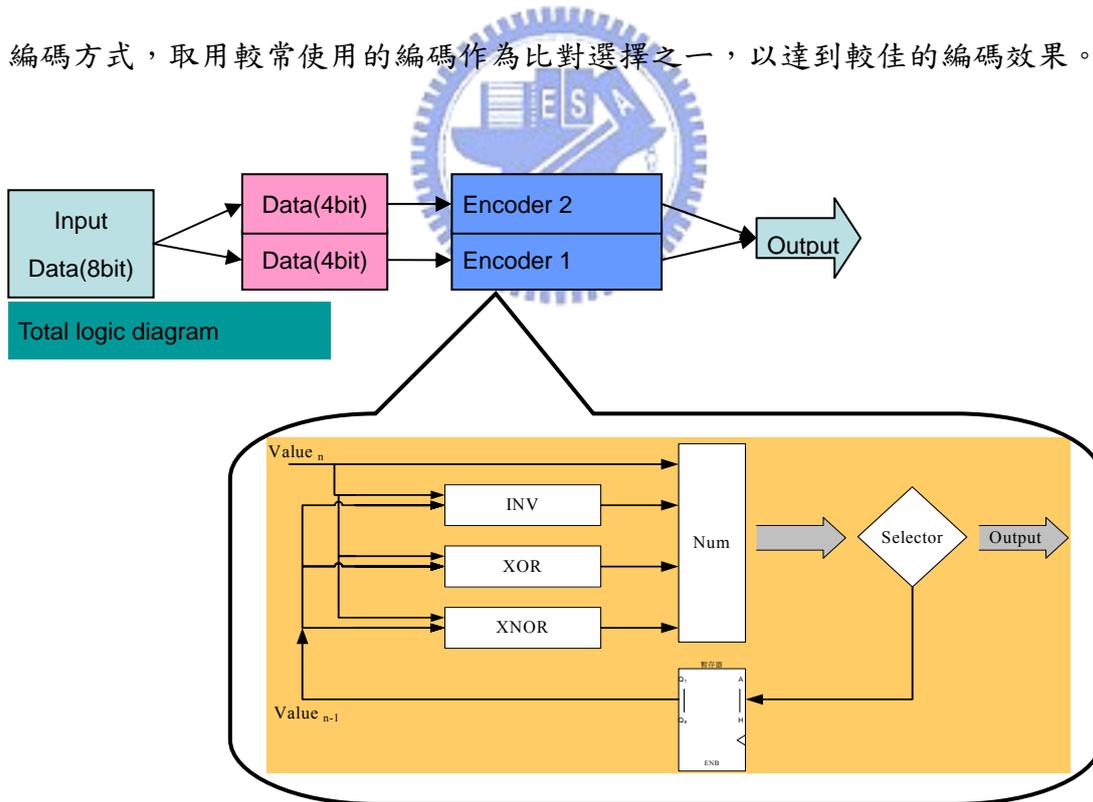


圖 3-4 匯流排編碼架構流程圖

3.2.2 功率感知之匯流排編碼解碼器效果

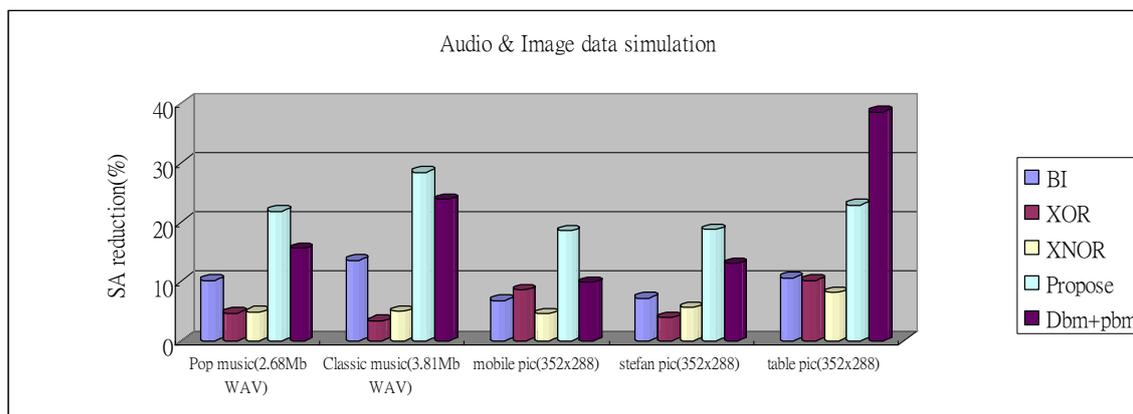


圖 3-5：多媒體傳輸編碼效果比較 (SA :Switch Activity)

經過多媒體語音及圖像資料傳輸模擬，如圖 3-5，透過此編碼演算法的編解碼動作，可以有效降低平均 20%左右的匯流排傳輸功率消耗，並且勝過其他編碼演算法。

3.3 具有使用者可調性主從式指令快取記憶體控制器

如圖 1-1 所示，指令快取記憶體佔全體處理器功率消耗將近 25%左右，因此在本處理器指令快取記憶體內，加入相位式的設計，在影響效能不大的情形下達到低功耗效果。下面將針對該快取記憶體控制器的功能，架構，使用流程，作一簡介，詳細內容請參考 [13]：具有使用者可調性主從式指令快取記憶體控制器之多核嵌入式處理器。

3.3.1 主從式快取記憶體控制器原理

圖 3-7 顯示具有使用者可調性主從式指令快取記憶體控制器的演算法流程，此設計藉由切換主副快取記憶體集合，能有效降低指令快取存取在大幅位址跳躍情形下的失誤率及事先存取的功能提高命中率。除此之外，本指令快取記憶體設計也加入相位式快取記憶體設計，達到低功率消耗效果。

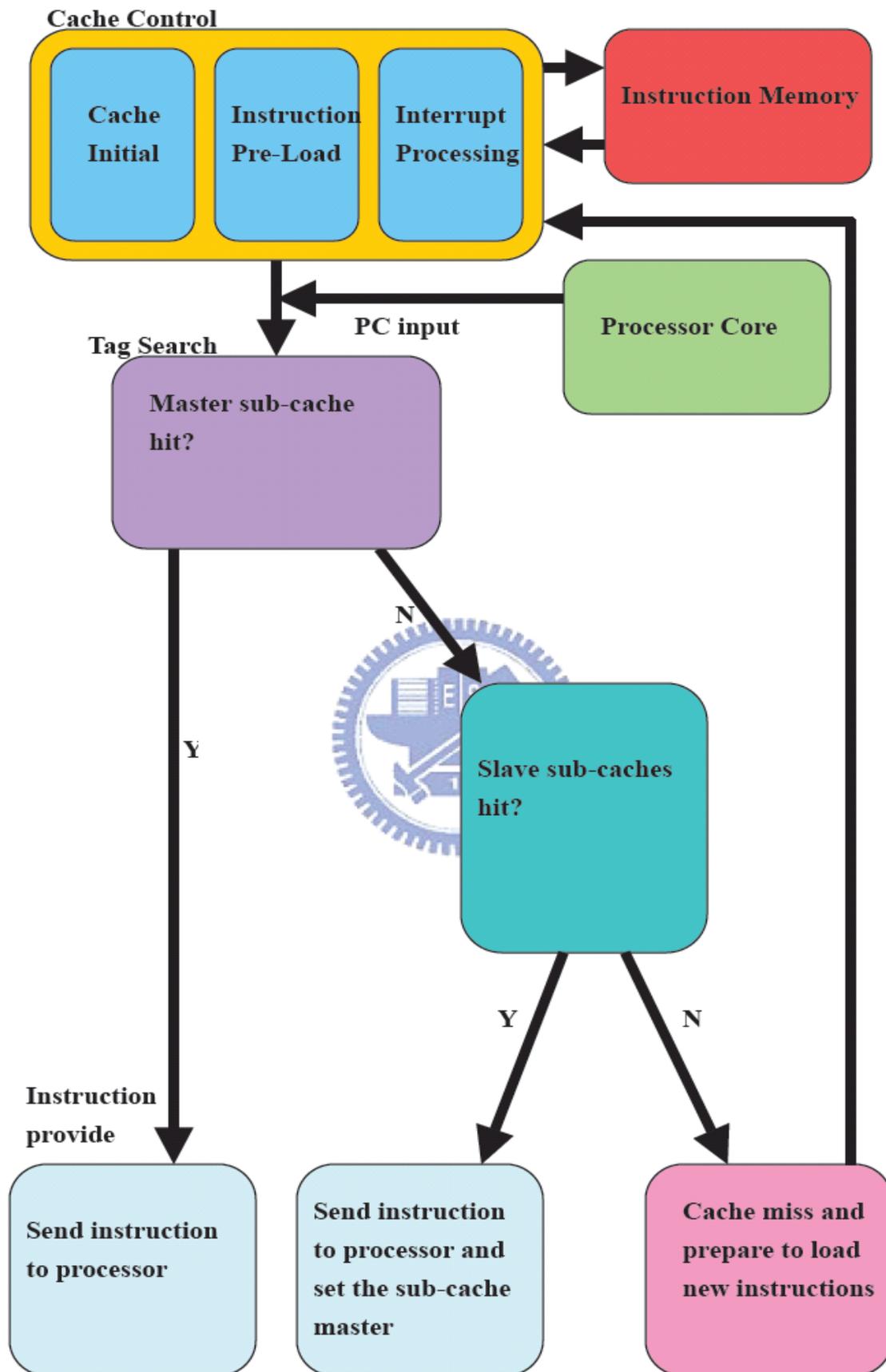


圖 3-7：主從式快取記憶體演算法

3.3.2 主從式快取記憶體效果

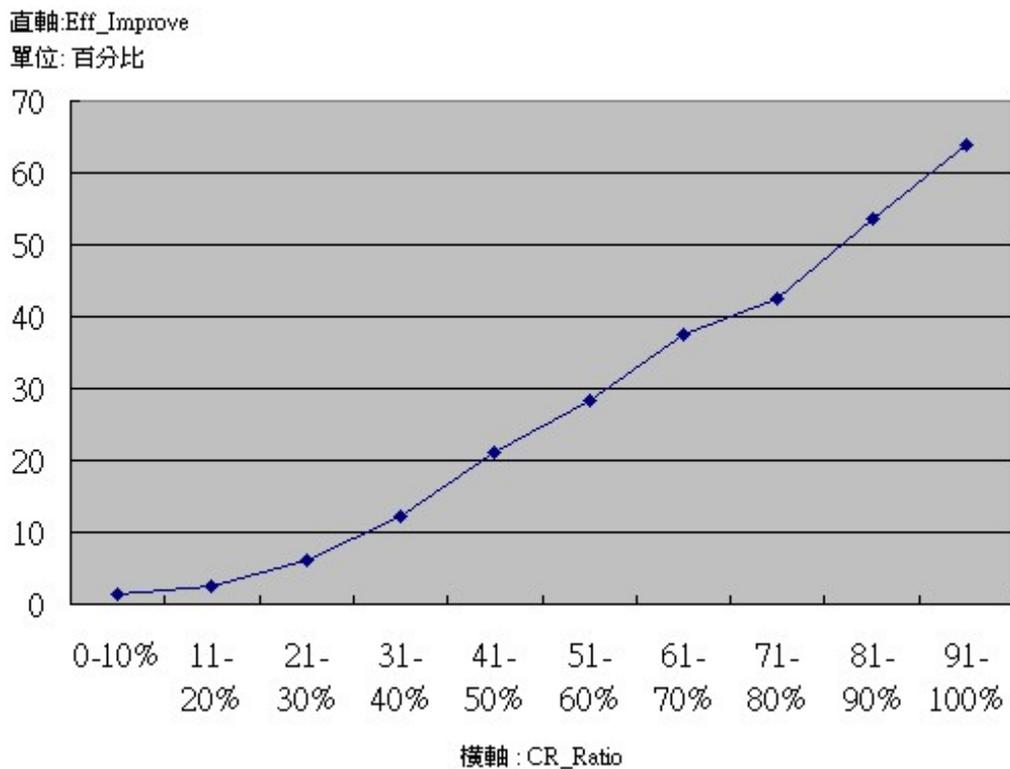


圖 3-8：主從式快取記憶體的效能提升示意圖。

CR_Ratio :有返回動作的跳躍在程式跳躍指令中佔的比例

Eff_Improve:快取記憶體架構整體的效能提升參數定義名稱為

加入主從式快取記憶體設計，依照不同的指令跳躍比例，提高指令快取記憶體的命中率，進而提升整體效能，如圖 3-8 所示。

另外在此設計中加入相位式快取記憶體的概念，先經過 Tag 比對再存取命中資料，可以達到同上一章節提到的降低功率消耗效果，在此可降低約 44%左右的功率消耗(2-Way)。

3.4 軟體開發環境

3.4.1 組譯器

硬體設計外，處理器的軟體支援是相當重要，為此發展了圖形化介面的組譯器 (Assembler)，提供機械碼 (Machine code) 的轉譯、程式記憶體 (Program rom) 的產生、及錯誤資訊 (Debug information)，讓使用者能夠利用以上資訊來除錯及產生 Testbench，如圖 3-9。

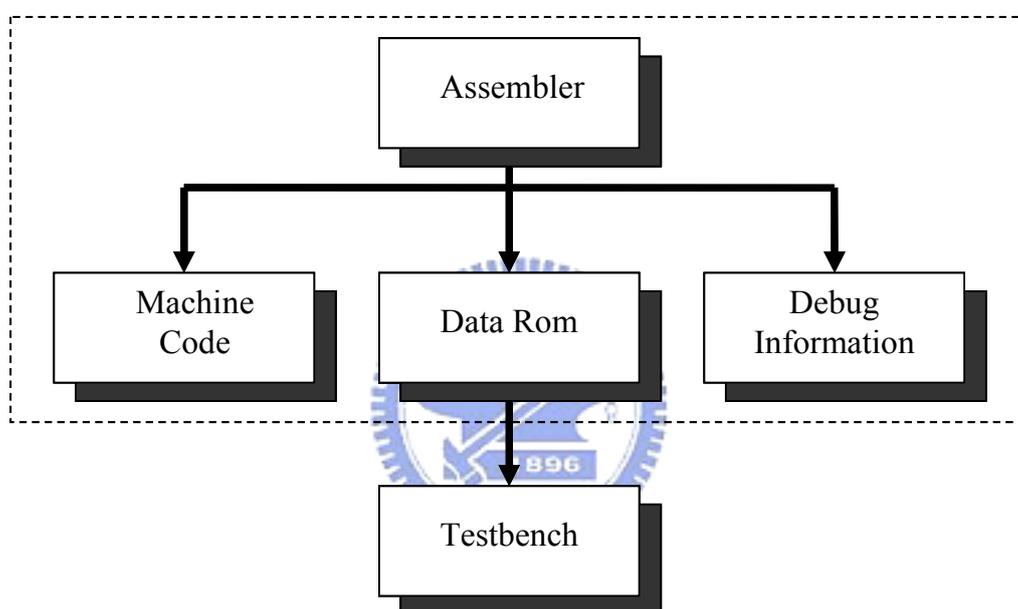


圖 3-9：組譯器 (Assembler) 流程圖

圖型化介面組譯器如下圖 3-10，使用 Visual C++ 完成演算法部分，使用 VB 做組譯器的圖形介面，並用 dll 作連結，並提供編輯檔案的能力。當完成編輯檔案後，須先經過 compile 的動作，確保無錯誤產生。最後經由 Build 的動作，產生所需的檔案：

pop.txt：產生十六進位的程式碼，提供晶片測試的資料。

bin.txt：產生程式記憶體的資料，提供模擬及後續測試必備的資料。

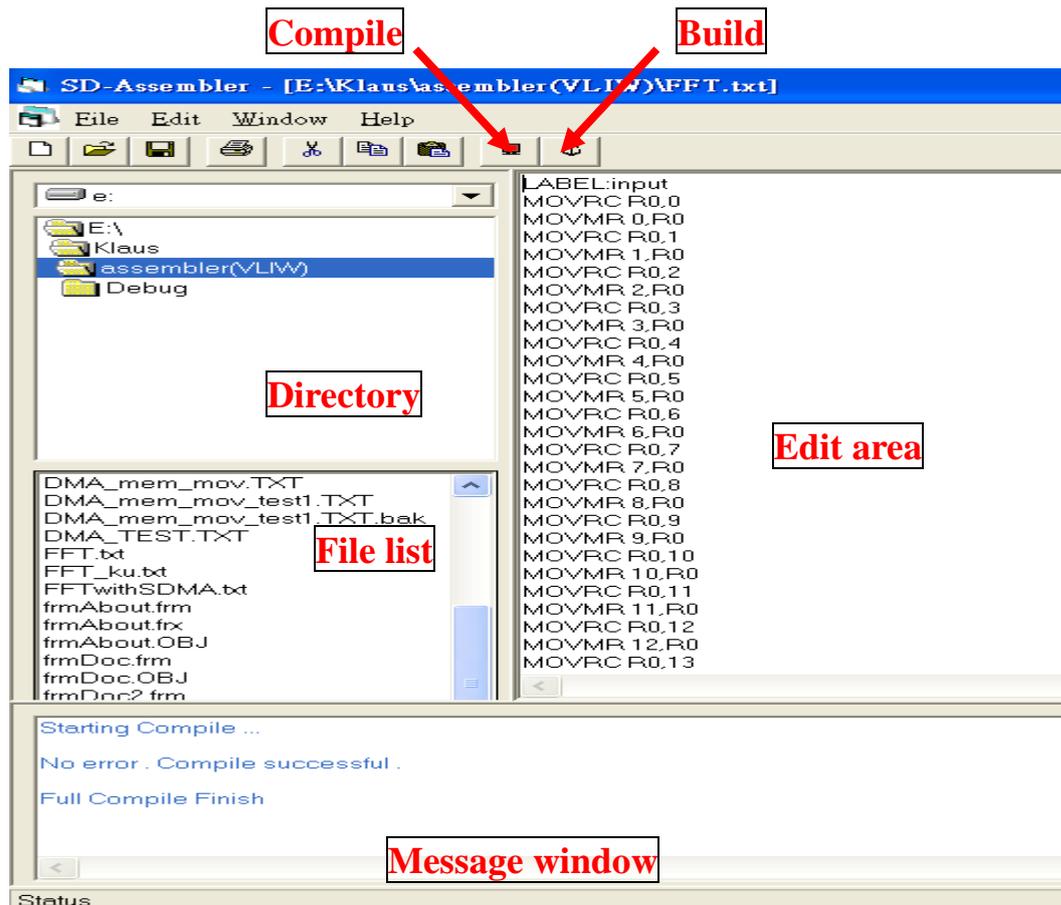


圖 3-10：圖形化介面組譯器 (Assembler)

3.4.2 模擬器

本論文也針對該處理器以高階語言(C++)設計了一個模擬器，以便能快速驗證各種測試程式，並實作此模擬器 GUI 介面。而寫這樣的模擬器困難的地方就是在於要拿一個本身沒有平行處理概念的語言 C/C++，去模擬一個有平行處理概念的管線。本論文製作的模擬器採用的設計方式是類似 software pipeline 的觀念將程式每個 iteration 倒著選一個指令然後送到硬體去循序。

由於 code 是循序的。所以如果把要用來處理管線的迴圈倒著寫的話，就可以達到該效果，以一個五級的 RISC 管線架構為例，如下圖 3-11 所示：

```

For(cycle++)
{
    //級五 Write Back
    ....
    //級四 ALU
    ....
    //級三 Reg_File
    ....
    //級二 Decoder
    ....
    //級一 Fetch
    ....
}

```

執行方向



圖 3-11：利用倒寫管線以使用高階語言的寫法描述管線。

這種實作方法類似在資訊科學領域演算法中 tail-recursion 的概念，遞迴的時候會發現把處理動作倒著寫會較好處理，因為遞迴本身是用到先進後出的堆疊，邏輯上和 software pipeline 作 iteration 的取樣很相似。

利用此模擬器可以得到每個測試程式在每一個 cycle 時暫存器及記憶體的內容，並計算 hazard 個數及總共的 Penalty Cycle 個數，方便對程式開發者進行效能上的分析，也方便除錯的工作。

利用此模擬器模擬處理器運算各種程式的結果，在跟 RTL code ，gate-level code 做數值上的驗證、週期的快速比對。

如圖 3-12 所示，顯示程式組語、記憶體數值、暫存器數值、運算總週期及運算指令數，方便做統計及比較。

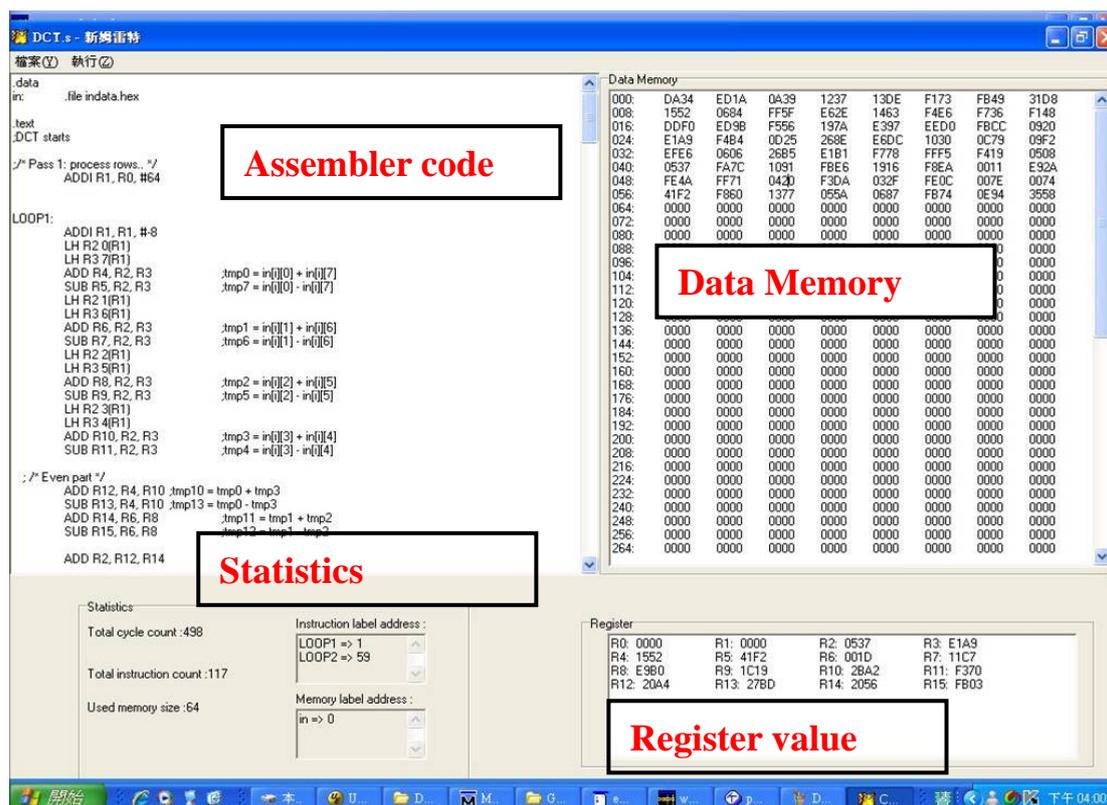


圖 3-12：圖形化介面模擬器 (simulator)

3.5 測試驗證

為了證明處理器設計的架構及指令集，需要去執行應用測試程式來確認設計是正確無誤的，為了符合 ISS (Information Systems and Sciences)，本處理器以有限長度脈衝響應 (Finite Impulse Response) 及餘弦轉換 (Discrete Cosine Transform) 2 種測試程式以及驗證快取在大量資料搬移情形下的索比爾運算 (Sobel operator)，加上模擬器的模擬結果，來證明本設計的功能無誤，以下將介紹 3 種測試程式內容及測試驗證結果。

3.5.1 有限長度脈衝響應 (Finite Impulse Response) Filter

FIR filtering 是常見的通訊及多媒體應用程式，因此在這個程式中，我們會執行一個 16tap 脈衝響應 FIR filter，經過 56 筆資料的輸入這個 FIR filter，得到 40 筆資料輸出。

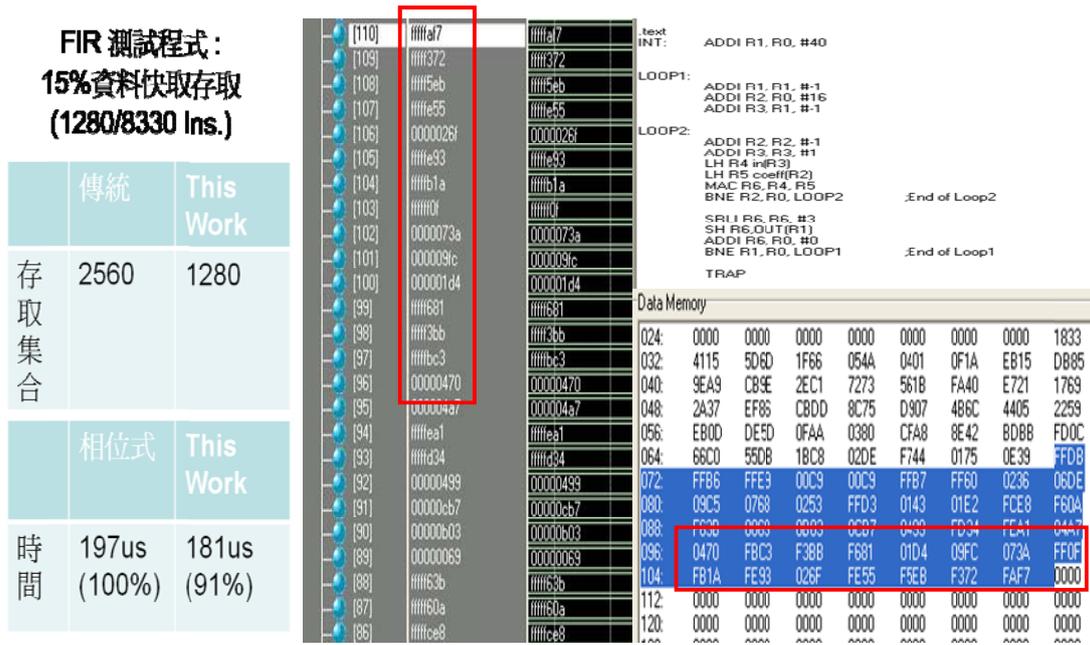


圖 3-13 : FIR RTL 模擬及 simulator 執行結果

3.5.2 餘弦轉換(Discrete Cosine Transform)

而在 DCT 部分，我們實現 2-dimension 8-by-8 DCT 運算，如圖 3-14 為 1-dimension 8-by-8 DCT 演算法，經過 row-by-row and column-by-column，2 次運算可以得到 2-D 8-by-8 DCT 結果，如圖 3-15 模擬結果。

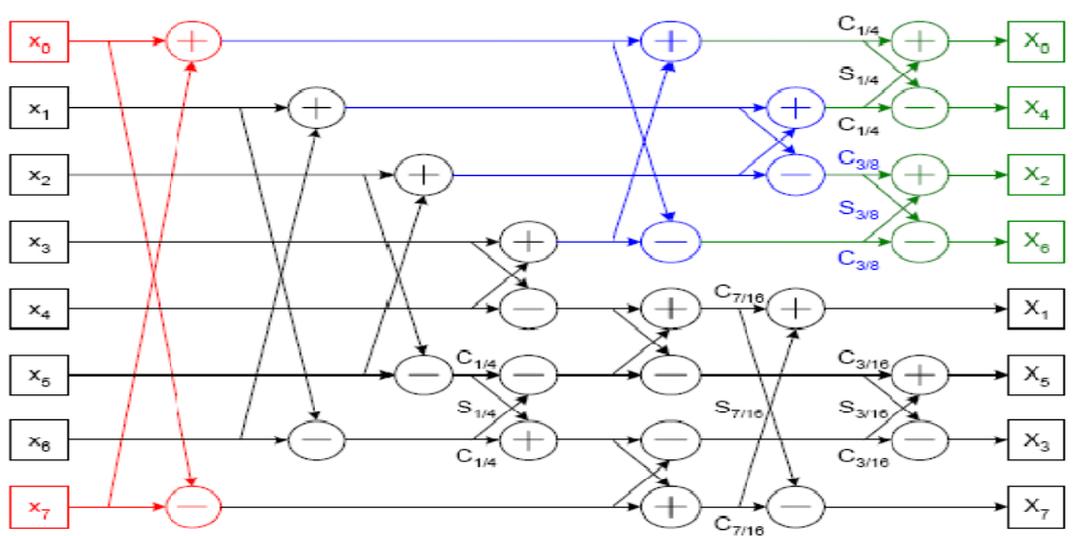


圖 3-14 : 1-dimension 8-by-8 DCT 演算法

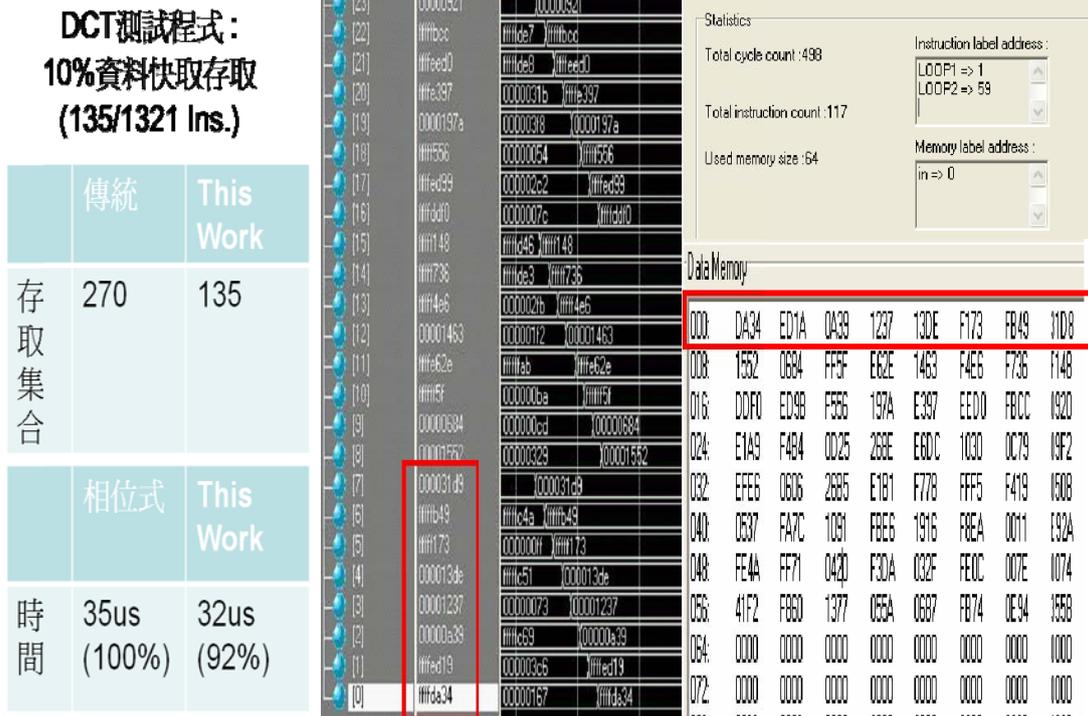


圖 3-15 : 2-dimension 8-8 DCT RTL 模擬及 simulator 執行結果

3.5.3 索比爾運算(Sobel operator)

本設計實現需要大量資料搬移傳輸之圖形邊緣檢測演算法-Sobel 來驗證本快取記憶體在執行大量資料搬移時，快取記憶體之功能並無發生失誤。

- 索比爾運算

索比爾運算是圖像處理中的運算之一，主要用作邊緣檢測(Edge-detection)。在技術上，它是一離散性差分運算，用來運算圖像亮度函數的梯度之近似值。在圖像的任何一點使用此運算，將會產生對應的梯度向量或是其法向量

- 索比爾邊緣檢測原理

該算子包含兩組 3x3 的矩陣，分別為橫向及縱向，將之與圖像作平面卷積，即可分別得出橫向及縱向的亮度差分近似值。如果以 \mathbf{A} 代表原始圖像， \mathbf{G}_x 及 \mathbf{G}_y 分別代表經橫向及縱向邊緣檢測的圖像，其公式如下(4)(5):

$$G_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} * A \quad (4) \quad G_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & +1 \end{bmatrix} * A \quad (5)$$

圖像的每一個像素的橫向及縱向梯度近似值可用以下的公式(6)結合，來計算梯度的大小，對算出的梯度大小取一臨界值，大於臨界值之像素設為1，表邊緣像素，反之為0，表背景像素。

$$G = \sqrt{G_x^2 + G_y^2} \quad (6)$$

本測試實現 64* 64pixels 影像 edge detection，共搬移 3 * 16 KB 以上資料，顯像結果與 MATLAB 處理結果相同



圖 3-16 : Sobel 模擬結果

3.6 可程式邏輯閘陣列(FPGA) 驗證

從網路產品、通訊設備、工業系統，到汽車電子系統的開發與應用設計上，都已經普遍採用現場可程式化邏輯閘陣列(Field-Programmable Gate Array; FPGA) 晶片，由於這種類型的晶片在實際應用時具有高度的靈活性、可讓使用者組成其邏輯功能，從簡單的布林函數運算式、暫存器功能，到內嵌記憶體、複雜功能 IP，加上在佈局上可以輕易達到重新配置的目的，可以輕易配合各種應用，因此成為上述各種電子產品所不可缺少的基本元件。

FPGA 在一般應用來說，速度要比 ASIC 來得慢，在整體設計上也較為精簡，不過主要優點在於可以快速的完成成品，或者是在初期開發時，採用 FPGA，等到完成整個設計與驗證的過程，再將之轉移到 ASIC 中，因此本設計使用 FPGA 來驗證功能，並架構為 IC 驗證平台。

FPGA 驗證雖然無法以較高的速度來測試功能，但其與 post-sim 不同的是，FPGA 使用的是實際的 Clock 相位來驗證，可以真實模擬實際的 Clock 相位下的

運作情形，可以提供更完整的驗證。

FPGA 使用 Altera APEX20KE EP20K1500EBC652-1X，利用軟體 Quartus II 經過平面規劃 (floorplanning)、擺置 (placement) 以及繞線 (routing)、燒錄 (burning) 等等流程，完成設計功能驗證動作，結果如下所示。

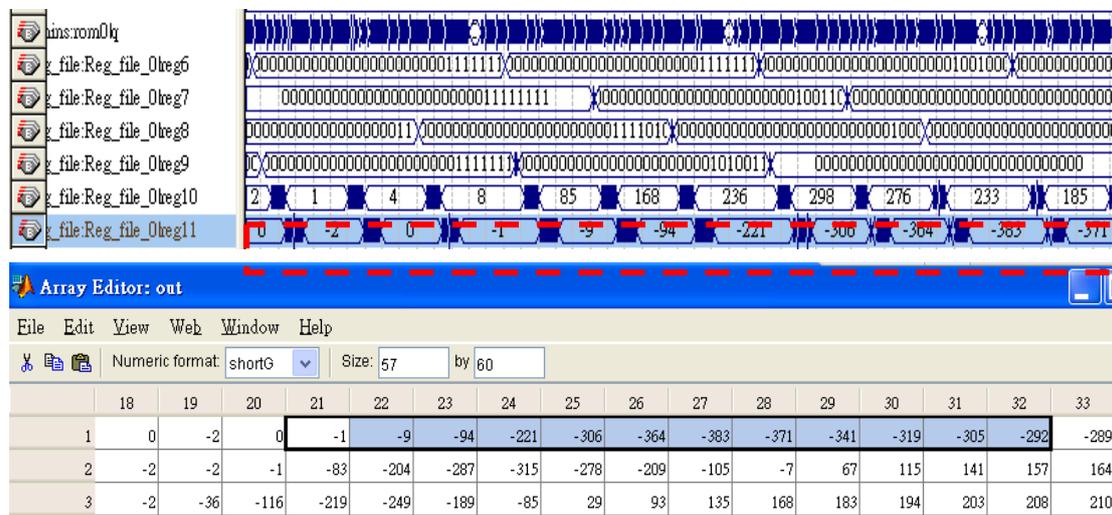


圖 3-17：處理器 FPGA 模擬 Sobel 結果圖

3.7 結語

本章節介紹了高效能相位式快取記憶體控制器及高效能存取管線的驗證平台：一個 RISC 的嵌入式處理器的架構及提高指令快取記憶體命中率且加入相位式概念的設計：具有使用者可調性主從式指令快取記憶體控制器和降低匯流排傳輸功耗的編碼器：功率感知之匯流排編碼解碼器。RISC 處理器搭配上具低功耗效果的 3 個設計，達到整體的低功耗效能，並在下一章節裡整合成為一個 SOC 晶片。

第四章 晶片實現與結果驗證

4.1 晶片製作

4.1.1 設計流程

依照標準的CIC Cell-Based Design Flow設計硬體[16]，設計流程如下圖4-1：

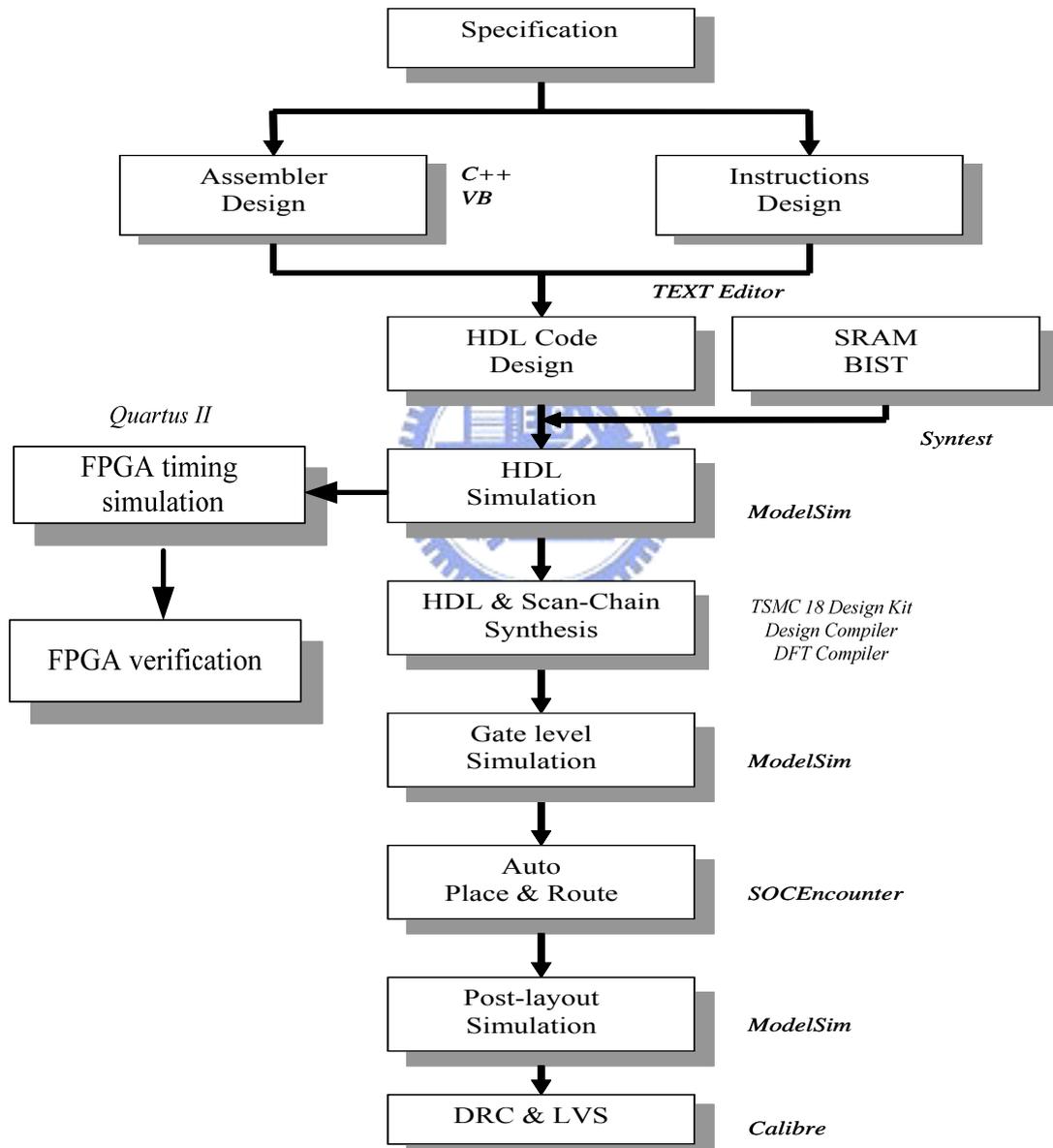


圖 4-1：晶片設計流程

4.1.2 合成結果

初期功能的設計，以 Verilog 硬體描述語言實作，搭配 Mentor 公司出的 ModelSim 進行功能驗證。待功能驗證正確後，以 Synopsys 公司出的 Design Compiler 進行電路合成，Library 使用 TSMC 0.18um 製程。經由合成軟體 Design Compiler 合成後，其結果如下表 4-1：

表 4-1：合成的結果

ITEM	Area (mm ²)	Timing	Fault coverage
Processor	497524.31	10 ns	98.24 %

4.1.3 佈局與封裝

採用 Cadence 公司出的 SOC Encounter 佈局軟體，將合成後的電路放在晶片上，並依據速度的要求，自動最佳化並繞線，結果如下：



CHIP name : BP_v1

Technology : TSMC 0.18um 1P6M CMOS

Package : 160 CQFP

Chip Size : 2.114× 2.114 mm²

Gate Count : 49K gate count

Power Dissipation (PrimePower) : ~16mW

Max. Frequency : 100MHz (10 ns)

使用Prime Power量測功率，跑驗證功能的DCT，FIR，Sobel測試程式，得到平均消耗功率約16mW。圖4-2為佈局圖，圖4-3為對應的腳位圖，圖4-4為打線圖，以160-CQFP的方式包裝。

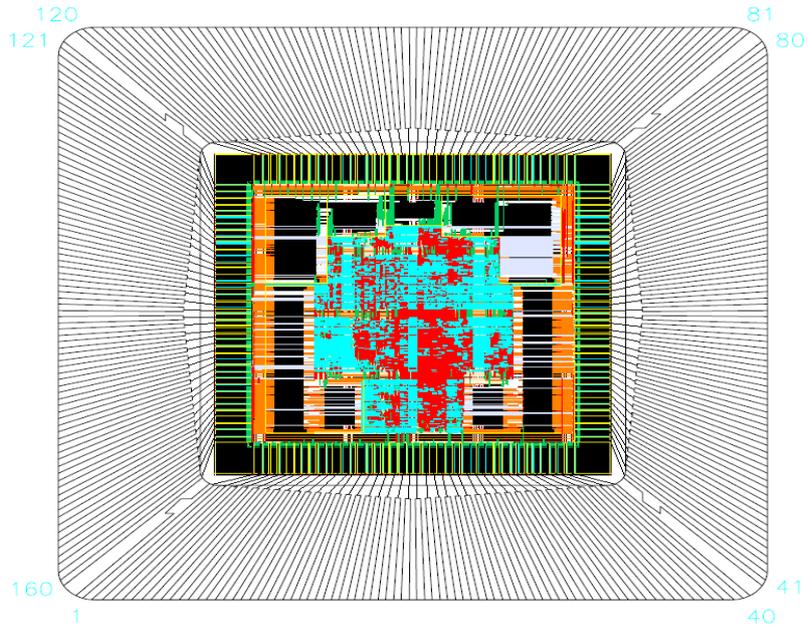


圖 4-4：打線圖

表 4-2：晶片設計規格

Technology	Description
Process	TSMC 0.18 μ m 1P6M Mixed Signal
Architecture	7-stage pipeline, 32bit RISC
Synthesis	Cell-based library
Gate Count	49K
Embedded Cache	RAM(512x2)x1(FIFO), RAM(512x8)x4(Tag), RAM(512x32)x4(Data)
Die size	2.114 \times 2.114 mm ²
Supply	1.8V/3.3V \pm 10%
Input Delay Time	Max 0.714ns/ Min 0.543ns
Power consumption	16mW (PrimePower)
Operating Frequency	100MHz

晶片設計規格如表 4-2。

而在後續的佈局驗證部分，使用Calibre的DRC（Design Rule Check）及LVS（Layout VS Schematic）也驗證無誤。

記憶體部分採用Cadence Memory generator 產生，分為4個Tag cache、4個Data cache 分別存放 Tag address及Data ，功率計算如下表4-3。

表4-3 :快取記憶體功率計算

120.00MHz operation

Power of Data SRAM (512x32) Condition	Fast@0C Process 1.98V, 0°C	Typical Process 1.80V, 25°C	Slow Process 1.62V, 125°C
	Value (mA)	Value (mA)	Value (mA)
AC Current	17.599	15.476	13.747
Peak Current	401.846	277.411	148.331
Deselected Current	4.447	3.788	3.290
Standby Current	0.004	0.002	0.013

Power of Tag SRAM (512x8) Condition	Fast@0C Process 1.98V, 0°C	Typical Process 1.80V, 25°C	Slow Process 1.62V, 125°C
	Value (mA)	Value (mA)	Value (mA)
AC Current	6.620	5.785	5.145
Peak Current	141.236	91.737	50.408
Deselected Current	2.144	1.810	1.557
Standby Current	0.002	0.001	0.007

由 $P=VI$ 推導，可以推算出Data部分記憶體功耗約為27mW，Tag部分記憶體功耗約為10mW(此兩數值為假設Input,Output50%變化的推算值)，佔了處理器功耗蠻大的部分，以本架構2-Way快取為例，傳統快取架構 $2 \times (27+10)=74\text{mW}$ 跟使用本設計架構 $27+2 \times 10=47\text{mW}$ 比較，可降低37%功率消耗與推導的44%約有7%的差距，此差距是因為記憶體功耗並沒有隨著位元數成正比增加的緣故造成。由上表也可以看出在Deselected 及Standby模式下，記憶體功耗可以大幅降低，以上幾點都能證明本論文設計降低快取功率消耗的效果。

經由 CIC TSMC 0.18um 合成、P&R 之後，在 Post-simulation 中，加入 delay time 模擬，功能無誤如圖 4-5 所示，左圖表示在讀取快取記憶體值時，經由 Tag 比對後只驅動 Hit 的 Way0 信號讀取資料，而不驅動 Miss 的 Way1 信號，達成我們所需的過濾功能、低功率消耗效果。右圖結果經由模擬器比對也驗證無誤。

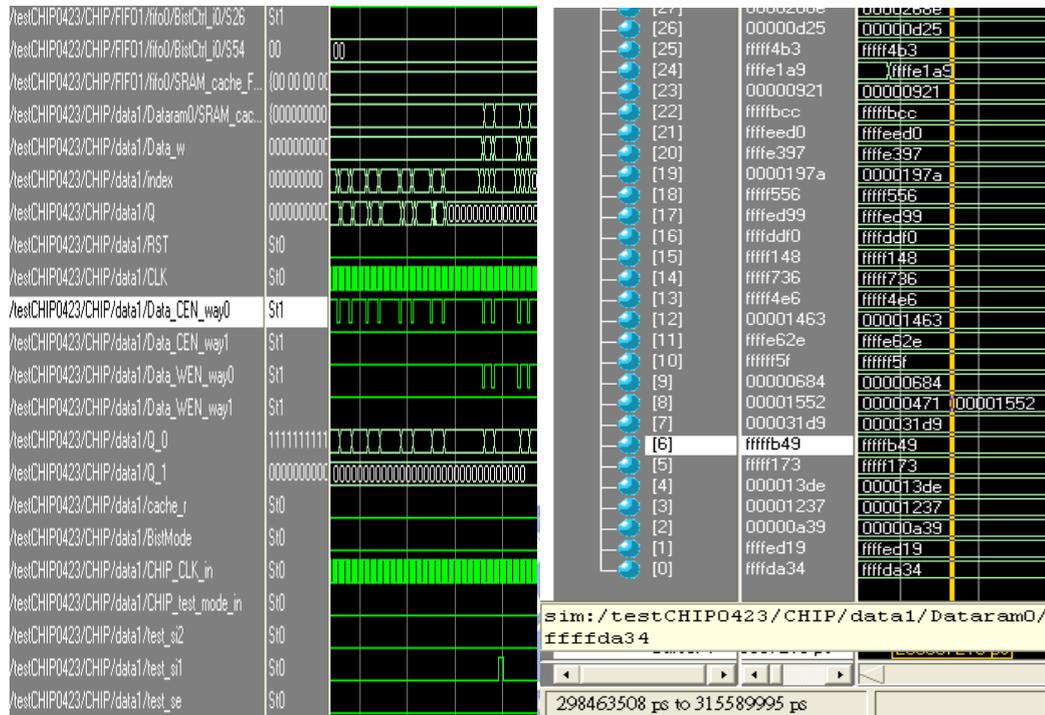


圖 4-5：處理器 Post-sim 模擬 DCT 結果圖

圖 4-6 顯示 Post-sim 模擬 Sobel 的運算結果，其結果與 MATLAB 陣列結果相同，經過圖形轉換，也與圖 3-16 有相同的數值，顯示功能無誤。

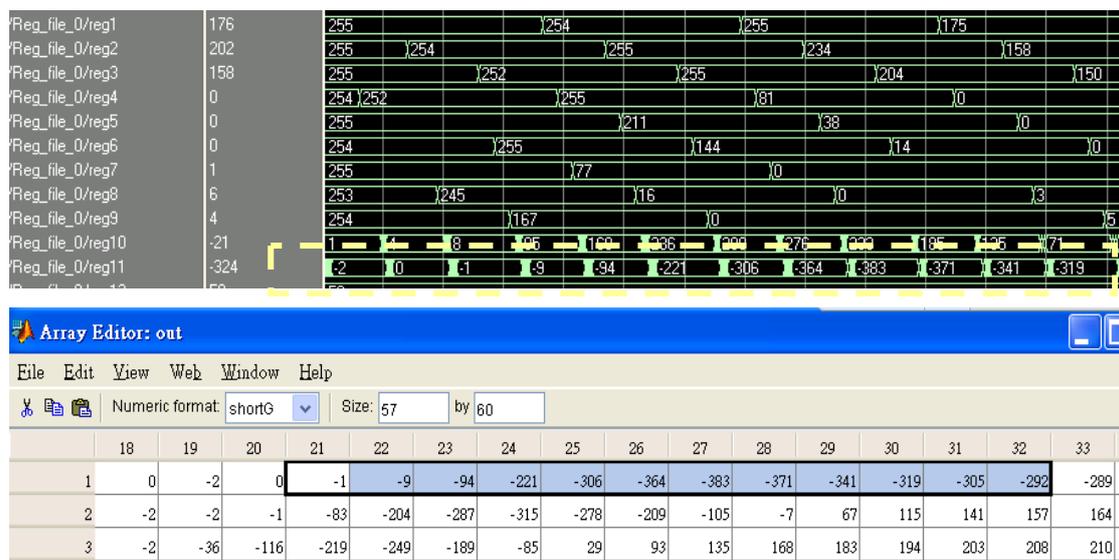


圖 4-6：處理器 Post-sim 模擬 Sobel 結果圖

第五章 結論與展望

本論文提出結合相位式快取記憶體及其高效能存取管線來達到低功耗無時間延遲的設計，利用記憶體位址計算上簡單的特性及快取記憶體位址的組成來達到提早存取相位式快取記憶體的目的，改善相位式快取記憶體耗時增加 37% 的缺失，並維持相位式快取的低功耗效果，且只付出 6% 的面積增加。

結合各低功耗設計的 RISC 處理器做為一實現平台，分別以功耗比例較重的指令及資料快取記憶體、匯流排傳輸為低功耗設計方向，降低快取存取次數及匯流排位元變化量來改善整體的功耗，達到低功耗要求。

此處理器未來希望能以 ARM6、ARM11 及 MIPS_DSP_ASE 為目標，以 RISC 架構為主，增強其多媒體功能、DSP 運算能力，相對於 RISC/DSP 的處理器組合，可以去除兩處理器資料傳輸的功率消耗，達到較低功耗、且具有較低研發成本。

本論文設計一顆通用 RISC 處理器，整合三種低功耗設計，以少量成本成功降低快取 44%(2-Way) 功率消耗，匯流排 20% 功耗，未來此顆晶片可用作低功耗的 RISC 處理器使用，或以結合其他 IP 的方式，整合系統成為一個 SOC 系統。

參考文獻

- [1] Simon Segars :“Low Power Design Techniques for Microprocessors,” ISSCC2001, Feb 4th 2001
- [2] Chuanjun Zhang, F. Vahid, Jun Yang, W. Walid, “A Way-Halting Cache for Low-Energy High-Performance Systems,” Computer Architecture Letters, IEEE Volume 2, Issue 1, pp.5 – 5, Jan. 2003
- [3] Hoon Choi, Myung-Kyoon Yim, Jae-Young Lee, Byeong-Whee Yun, Yun-Tae Lee, “Low-power 4-way associative cache for embedded SOC design,” Proceedings. 13th Annual IEEE International ASIC/SOC Conference, Volume13, Issue16, pp.231 - 235, Sept. 2000
- [4] D. Nicolaescu, A. Veidenbaum, A. Nicolau, “Reducing power consumption for high-associativity data caches in embedded processors,” Design, Automation and Test in Europe Conference and Exhibition, pp.1064 - 1068, 2003
- [5] John L. Hennessy and David A. Patterson, Computer Architecture, 3rd Edition, Morgan Kaufmann, 2003.
- [6] J. Kin, Munish Gupta, Mangione-Smith, “The filter cache: an energy efficient memory structure,” Microarchitecture, Proceedings Thirtieth Annual IEEE/ACM International Symposium on, Volume1, Issue3, pp.184 --193,Dec. 1997
- [7] Ching-Long Su, Alvin M. Despain, “Cache design trade-offs for power and performance optimization: a case study,” International Symposium on Low Power Electronics and Design, pp. 63 - 68,1995
- [8] Yen-Jen Chang, Shanq-Jang Ruan, Feipei Lai, “Design and analysis of low-power cache using two-level filter scheme,” Very Large Scale Integration (VLSI) Systems, IEEE Transactions on, Volume 11, Issue 4, pp.568 -

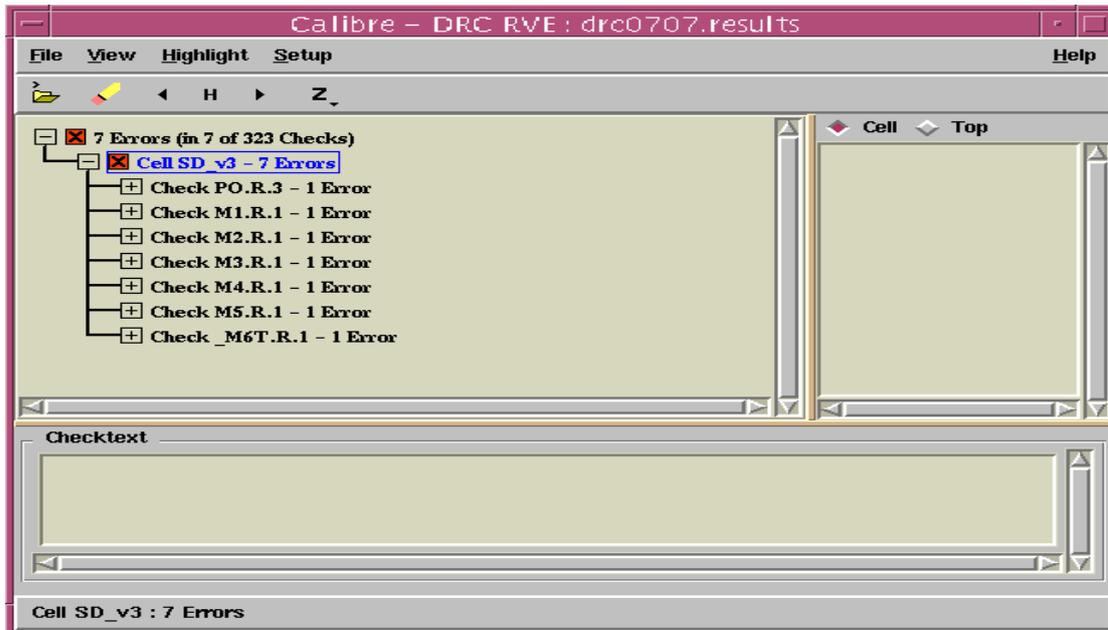
580, Aug. 2003

- [9] K. Pagiamtzis, A. Sheikholeslami, “Content-addressable memory (CAM) circuits and architectures: a tutorial and survey,” IEEE Journal of Solid-State Circuits, Volume 41, Issue 3, pp.712 – 727, March 2006
- [10] K. Inoue, T. Ishihara, K. Murakami, “Way-predicting set-associative cache for high performance and low energy consumption,” Proceedings International Symposium on Low Power Electronics and Design, pp.273 - 275,1999
- [11] T. Austin, E. Larson, D. Ernst, “SimpleScalar: an infrastructure for computer system modeling,” Computer, Volume 35, Issue 2, pp.59 – 67, Feb. 2002
- [12] <http://www.spec.org/>
- [13] 周經翔,「具有使用者可調性主從式指令快取記憶體控制器」,國立交通大學,碩士論文,民國95年
- [14] 黃德璋,「功率感知資料匯流排編碼解碼器設計」,國立交通大學,碩士論文,民國96年
- [15] John L. Hennessy and David A. Patterson, Computer Organization & Design : The Hardware / Software Interface,2nd Edition, Morgan Kaufmann Publishers,1998
- [16] Jae Sung Lee , Myung H. Sunwoo, “Design of new DSP instructions and their hardware architecture for high-speed FFT,” Proceedings of Kluwer Academic Publishers, pp. 247-254, 2003.
- [17] Nian Shyang Chang, Cell-Based IC Physical Design and Verification, Chip Implementation Center, July, 2004.

附錄

A. 佈局驗證結果說明

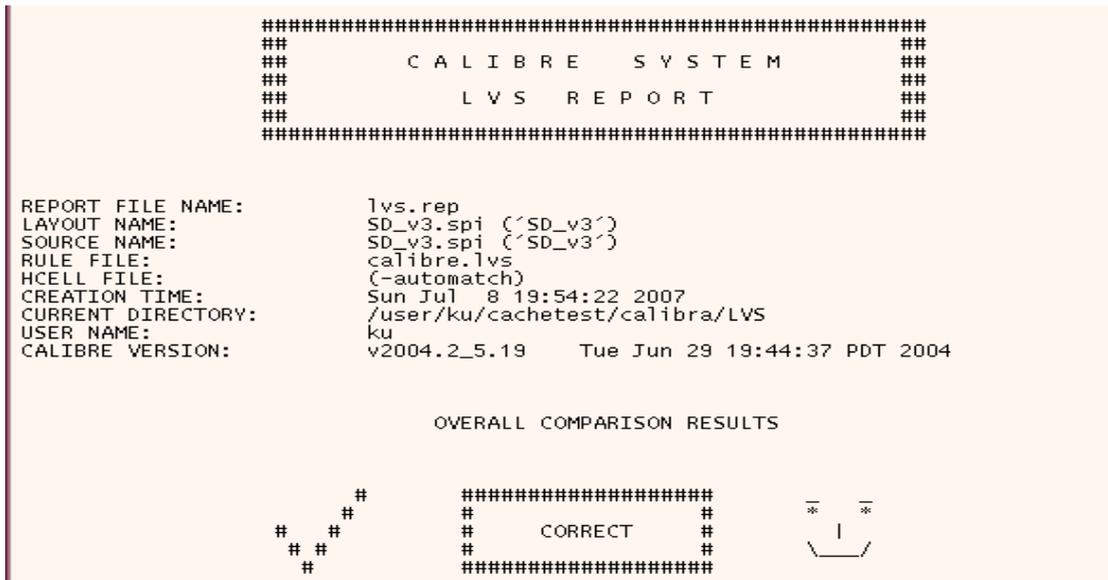
1. DRC



經詢問 CIC TSMC cell-base 負責人，cell-base 製程下的 Metal Ratio error 皆可忽略

DRC 驗證無誤

2. LVS



LVS 驗證無誤

B. CIC Tapeout Review Form(for Cell-Based IC)

1. 晶片概述：

1-1. 專題名稱：具有快取及匯流排低功耗設計之嵌入式處理器

1-2. Top Cell 名稱：SD v3

1-3. 使用 library 名稱：

CIC_CBDK35

CIC_CBDK25

CIC_CBDK18

版本：v1.0

1-4. 是否使用 CIC 提供之 Memory? Yes

1-5. 工作頻率：100 MHz

1-6. 功率消耗：16mW

1-7. 晶片面積：2114 X 2114

2. 設計合成：

2-1. 使用之合成軟體？Synopsys design compiler

2-2. 是否加入 boundary condition：

input drive strength、 input delay、 output loading、 output delay

2-3. 是否加入 timing constraint：

specify clock (sequential design)

max delay、 min delay (combinational design)

2-4. 是否加入 area constraint? Yes

2-5. 合成後之 report 是否有 timing violation? No

有 setup time violation、 有 hold time violation

2-6. 合成後之 verilog 是否含有 assign 描述? No

2-7. 合成後之 verilog 是否含有 *cell* 之 instance name? No

2-8. 合成後之 verilog 是否含有反斜線 \ 之 instance name 或 net name? No

3. 可測試性設計(前瞻性晶片必填)：

3-0. 使用之設計軟體？DFT compiler

3-2. 使用之 ATPG 軟體？Tetramax

3-3. 使用 Embedded memory 數量: SRAM 5 , ROM 0

Memory 大小：512x32 (Word x bit)x2 512x8 (Word x bit)x2 512x2

(Word x bit)x1

測試方法: BIST Yes , or 其他測試方法 N/A

若使用 BIST,其 Test Algorithm 為何?Moving Inversion (13N March)

- 同時有多個 memory，是否共用 BIST controller Yes，BIST controller 數量 1
- 3-4. Scan Chain Information
 Flip-Flop 共有多少個？ 2280
 Scan chain 的數量共有多少條？ 3
 Scan chain length (Max.) ? 25561.840
- 3-5. Uncollapsed fault coverage 是否超過 90% ? Yes，為多少？ 98.24%
 ATPG pattern 的數目為多少？ 272
 註：若使用 Synopsys TetraMAX 來產生 ATPG pattern，請使用 set faults
 -fault_coverage 指令指定 TetraMAX 產生 fault coverage information
 若使用 SynTest TurboScan 之 asicgen 來產生 ATPG pattern，請以 atpg pessimistic
 fault coverage 的值為準
4. 佈局前模擬
- 4-1. gate level simulation 是否有 timing violation ? No
 ___ 有 setup time violation、___ 有 hold time violation
5. 實體佈局
- 5-1. 使用之 P&R 軟體？ Apollo、v SE
 5-2. power ring 寬度？ 8 是否已考量 current density(1mA/1um) ? Yes
 5-3. 是否考慮 output loading ? Yes
 5-4. 是否加上 Clock Tree ? Yes
 5-5. 是否加上 Corner pad ? Yes
 5-6. 是否加上 IO Filler ? Yes
 5-7. 是否加上 Core Filler ? Yes
 5-8. 是否加上 Bonding Pad ? Yes
- 以下(A-1)為使用 Apollo 者才須回答
- A-1. 是否執行 Fill Notch and Gap 步驟？ _____
- 以下(S-1 至 S-2)為使用 SE 者才須回答
- S-1. power ring 上是否有 overlap vias ? No
 S-2. 是否確定 IO Row 和 Corner Row 互相貼齊？ Yes
6. 佈局後模擬
- 6-1. 是否做過 post-layout gate-level simulation ? Yes
 STA(static timing analysis) 軟體？ Primetime / Modelsim
- 6-2. 是否做過 post-layout transistor-level simulation ? No
- 6-3. 已針對以下環境狀態模擬：___ SS、___ TT、___ FF
- 6-4. 晶片取得時將以何種方式進行測試？ P600 of Agilent 93000
- 6-5. 模擬時是否考量輸出負載影響？ Yes

7. DRC/LVS 驗證

7-1. 是否有 DRC 錯誤? No 錯誤原因: _____

驗證 DRC 軟體? Calibre

是否有不作 DRC 的區域? No

7-2. 是否有 LVS 錯誤? No

驗證 LVS 軟體? Calibre

是否有非 CIC 提供的 BlackBox? No

設計者簽名: 薛智文/黃德瑋

指導教授簽名: 林進燈

