

國立交通大學

電機與控制工程學系

碩士論文

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

Design of Multi-Core Embedded
Processor Using Configurable
Master-Slave I-Cache Controller

研究生：周經翔

指導教授：林進燈 博士

中華民國九十五年六月

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

Design of Multi-Core Embedded Processor Using
Configurable Master-Slave I-Cache Controller

研 究 生：周經翔

Student : Ching-Hsiang Chou

指導教授：林進燈 博士

Advisor : Dr. Chin-Teng Lin

國立交通大學

電機與控制工程學系



Submitted to Department of Electrical and Control Engineering

College of Engineering and Computer Science

National Chiao Tung University

in Partial Fulfillment of the Requirements

for the Degree of Master

in

Electrical and Control Engineering

June 2006

Hsinchu, Taiwan, Republic of China

中華民國 九十五年 六月

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


Design of Multi-Core Embedded Processor Using Configurable Master-Slave I-Cache Controller

學生：周經翔

指導教授：林進燈 博士

國立交通大學電機與控制工程研究所

中文摘要



無論是在一般應用的嵌入式處理器(General Embedded Processor)或是著重於運算的數位訊號處理器(Digital Signal Processor, DSP), cache miss幾乎都是影響效能的一個很大的因素。由於cache miss的penalty常常都是數千週期甚至上萬週期,所以如果可以設計一個可以減少cache miss次數及總miss penalty的快取記憶體控制器,便可以有效的提升處理器的執行效能。本論文設計了一個可實現於晶片中的指令快取記憶體控制硬體演算法,此演算法使用在一般的應用程式均能有效的減少指令快取記憶體(I-Cache)的總miss penalty,而在具有大量迴圈運算及函式呼叫的多媒體應用程式中尤有顯著成效,為驗證該演算法的可靠性及正確性,本論文也設計出一個VLIW架構的多核心嵌入式處理器,以做為該指令快取記憶體控制器的作用平台,並整合成為一顆嵌入式處理器晶片。此晶片採用UMC 0.18 μm 製程,以Cell-based方式設計,晶片面積約 $3.1 \times 3.1 \text{ mm}^2$, 預估最大操作頻率在 135MHz。

Design of Multi-Core Embedded Processor Using Configurable Master-Slave I-Cache Controller

Student: Ching-Hsiang Chou Advisor: Dr. Chin-Teng Lin

Department of Electrical and Control Engineering
National Chiao Tung University

Abstract

Cache miss is a very significant factor to affect its efficiency for the General Embedded Processor in general applications or the Digital Signal Processor(DSP) emphasizing on computing operations. The cache miss results in the penalty of wasting of thousands of cycles or more. For this reason, if we design a cache controller that can reduce the number of cache miss and save miss penalty, we will enhance the efficiency of the processor. In the paper, an I-Cache controller hardware algorithm that can be applied in Chip is introduced. When this algorithm is applied for general application program, it can efficiently reduce the total miss penalty of the I-Cache. Even more, we can see the significant effect when it is applied for the multi-media application that has many loop operations and function calls. In order to prove the dependability and the correctness of the algorithm, the thesis designs a multi-core embedded processor that has VLIW architecture. That can be used for the operation platform of the I-Cache controller, and be intergraded into a embedded processor chip. This Chip is fabricated in UMC 0.18 μ m process and designed in the way of Cell-based. The chip area is 3.1x3.1 mm², and the max operation frequency is operated at 135MHz.

誌謝

本論文的完成，首先要感謝指導教授林進燈博士這兩年來的悉心指導，讓我學習到許多寶貴的知識，在學業及研究方法上受益良多，也學習到許多做事的態度及技巧。另外也要感謝口試委員們的建議與指教，使得本論文更為完整。

其次，感謝資訊媒體實驗室的學長鍾仁峰博士、劉得正博士及資工系的范倫達教授的指導協助，同學峻谷、家昇及宗哲的相互砥礪，及所有學長、學弟們在研究過程中所給我的鼓勵與協助。尤其是鍾仁峰博士及范倫達教授，在計算機理論及下線流程上給予我相當多的幫助與建議，讓我獲益良多。

感謝我的父母親對我的教育與栽培，並給予我精神及物質上的一切支援，使我能安心地致力於學業。

謹以本論文獻給我的家人及所有關心我的師長與朋友們。



目 錄

中文摘要	ii
Abstract.....	iii
誌謝	iv
目 錄	v
圖 目 錄	viii
表 目 錄	ix
第一章 緒論	1
1.1 簡介	1
1.2 論文架構	4
第二章 主從式快取記憶體控制器	5
2.1 主從式快取記憶體控制器原理	5
2.1.1 主從式快取記憶體控制器硬體演算法	5
2.1.2 使用者可調性設計	8
2.1.3 低功耗設計	10
2.2 主從式快取記憶體控制器架構	11
2.2.1 程式初始指令讀取控制	11
2.2.2 位置比較器	12
2.2.3 發生cache miss時的處理	12
2.2.4 處理器中斷時的處理	13

2.3 主從式快取記憶體模擬器	15
2.3.1 主從式快取記憶體與L1快取記憶體控制模擬器	15
2.3.2 測試程式碼產生器	16
2.3.3 效能指標	16
2.3.4 效能測試	17
2.3.5 功率管理	24
2.3.6 延伸比較	24
2.4 結語	25
第三章 多核嵌入式處理器	26
3.1 VLIW處理器架構	26
3.1.1 VLIW處理器核心	26
3.1.2 VLIW處理器指令集	30
3.1.3 子核心特色	34
3.1.4 不同子核心間的資料讀取與寫入	34
3.2 軟體開發環境	34
3.2.1 組譯器	34
3.2.3 模擬器	37
3.3 智慧型DMA控制器	39
3.3.2 智慧型DMA控制器架構	41
3.3.3 智慧型DMA使用流程	42
3.4 整合	43
3.5 結語	46
第四章 晶片實現與結果驗證	47
4.1 晶片製作	47
4.1.1 設計流程	47
4.1.2 合成結果	48
4.1.3 佈局與封裝	48

4.2 測試驗證.....	52
4.2.1 餘弦轉換 (Discrete Cosine Transform)	52
4.2.2 快速傅立葉轉換(FFT).....	53
4.2.3 量化線性預估係數(LPC).....	54
4.3 效能比較.....	56
第五章 結論	58
參考文獻	59
附錄	64
A. 測試程式	64
B. 佈局驗證結果說明	69
C. CIC下線報告書(A).....	70
D. CIC下線報告書(B).....	73
E. Module I/O.....	76



圖目錄

圖 2-1(a)：主從式快取記憶體架構圖.....	5
圖 2-1(b)：L1 快取記憶體.....	6
圖 2-1(c)：主從式快取記憶體.....	6
圖 2-1(d)：master和slave切換後的圖.....	7
圖 2-1(e)：參數值的使用流程。.....	10
圖 2-2(a)：主從式快取記憶體控制器三級管線架構圖。.....	11
圖 2-2(b)：主從式快取記憶體預先存取流程圖。.....	13
圖 2-2(c)：主從式快取記憶體操作流程圖。.....	14
圖 2-3(a)：主從式快取記憶體的效能提升示意圖。.....	18
圖 2-3(b)：PLC對主從式快取記憶體的效能提升示意圖。.....	20
圖 2-3(c)：1k-entry時的比較結果。.....	21
圖 2-3(d)：2k-entry時的比較結果。.....	21
圖 2-3(e)：4k-entry時的比較結果。.....	22
圖 2-3(f)：8k-entry時的比較結果。.....	22
圖 2-3(g)：可利用主從式快取記憶體架構大幅改善的特殊例子.....	23
圖 3-1(a)：VLIW架構多媒體訊號處理器的組成結構.....	27
圖 3-2(a)：組譯器（Assembler）的用途.....	35
圖 3-2(b)：圖形化介面組譯器（Assembler）.....	36
圖 3-2(c)：平行處理指令的使用.....	36
圖 3-2(d)：利用倒寫管線以使用高階語言的寫法描述管線。.....	38
圖 3-3(a)：智慧型DMA 的架構.....	42
圖 3-3(b)：智慧型DMA的使用流程.....	43
圖 3-3(c)：VLIW處理器與智慧型DMA的整合架構圖.....	44
圖 3-3(d)：智慧型DMA與處理器及週邊的對應圖.....	45
圖 3-3(e)：VLIW處理器存取智慧型DMA暫存器-記憶體對映.....	45
圖 4-1(a)：晶片設計流程.....	47
圖 4-1(b)：晶片佈局圖.....	49
圖 4-1(c)：打線圖.....	50
圖 4-1(d)：腳位圖.....	50
圖 4-2(a)：DCT-2 Post-layout Simulation的運算結果.....	53
圖 4-2(b)：512 點FFT前 32 點的運算結果.....	54
圖 4-2(c)：多級向量量化C程式執行結果.....	55

表 目 錄

表 3-1(a)：資料搬移指令列表.....	30
表 3-1(b)：算數邏輯運算指令列表.....	31
表 3-1(c)：跳躍指令列表.....	32
表 3-1(d)：其他指令列表.....	32
表 3-1(e)：智慧型DMA控制指令列表	33
表 3-3(a)：智慧型DMA與DMA面積比較	41
表 4-1(a)：合成的結果.....	48
表 4-1(b)：晶片設計規格.....	51
表 4-3(a)：與其他DSP的規格比較表	56
表 4-3(b)：與DSP運算效能比較表	57



第一章 緒論

1.1 簡介

現行大部分的處理器在加速的部分都是採取 Super-scale[1]的方式，配合 Tomasula's algorithm 做動態排程[9]，以達到管線最佳使用效能。VLIW 架構受限於程式的相容性，因此一度成為處理器設計領域的冷門架構。然而德州儀器後來將 VLIW 的架構實現於要求效能的數位訊號處理器上，並成功地廣泛應用於多媒體系統中。VLIW 架構之所以能在數位訊號微處理器市場中成功，除了它以簡易的方式達到平行處理的目的[12][13]，更重要的，在講求效能的數位訊號處理上，使用處理器的韌體開發者大都會去把程式碼針對所使用的特定處理器進行最佳化排程[2][5]，以達到最好的執行效能。有些數位訊號處理器的程式開發人員甚至會手動做調整以達到最大的指令平行度，這使得原先 VLIW 的缺點：相容性及非動態排程都因為這個理由而消失了。而易於增加及減少處理核心的特點，在現今講求 IP 化的 IC 設計領域變的十分具有優勢，這也是 VLIW 架構的處理器在現今數位訊號處理器市場重新崛起的原因。

目前在現行大部份講求低功率、小面積的嵌入式處理器及數位訊號處理器中 [11]，指令快取記憶體控制都還是採用 L1-Direct Map 的架構[30]。少部份高價位的嵌入式處理器及數位訊號處理器則容許比較大面積的 L2 架構[19]，不過這只是為了讓快取記憶體在大容量上可作有效管理而產生的架構上的變化[22]，在第一層(L1)的快取記憶體的 control 模式幾乎都還是採取 Direct Map 策略。然而傳統的 Direct Map 的控制方式在遠距離函式呼叫返回及在跨越快取記憶體容量邊際的迴圈運算—也就是當只有部份指令在快取記憶體中的迴圈運算時，會發生大量的 cache miss，也直接的影響了 cache miss penalty。這對於常常使用函式呼叫及迴圈運算的多媒體程式是十分不利的，如果沒有好的編譯器對指令作最佳化輔助

[4]，那將會顯著的降低了原有處理器的效能。然而對於種類繁多的嵌入式處理器市場裡，要開發出能針對不同使用者採用的不相同的處理器及該處理器配置的不同大小的指令快取記憶體作最佳化的編譯器是相當不容易的，一般目前大部份的嵌入式處理器的編譯器都還是只能針對特定的處理器進行指令的最佳化。

無論是在一般應用的嵌入式處理器(General Embedded Processor)[25]或是著重於運算的數位訊號處理器(Digital Signal Processor, DSP)[24]，cache miss 幾乎都是影響效能的一個很大的因素。由於 cache miss 的 penalty 常常都是數千週期甚至上萬週期，所以如果可以設計一個可以減少 cache miss 次數及總 miss penalty 的快取記憶體控制器，便可以有效的提升處理器的執行效能。

影響 cache miss 發生的原因通常是快取記憶體的 entry 數量，在一般 RISC 處理器指令長度固定的情況下，快取記憶體的大小跟 cache miss 的次數會呈現反比例的關係，這個現象在指令快取記憶體(I-Cache)中尤其明顯[27]。然而就單一處理器於同一應用而言如果使用不同的快取記憶體控制器，由於在遇到 cache miss 時的處理方法不相同，所以不能只由 cache miss 的次數判定效能而應由真正反映在時間差異的 cache miss penalty 的週期數目來決定。對於比較不同的快取記憶體控制器，cache miss 的次數多卻不一定會造成比較多的 penalty cycle 的這個現象是要去注意考量的。

本論文設計了一個可實現於晶片中的快取記憶體控制硬體演算法並命名為主從式快取記憶體控制器，並在這篇論文中以指令快取記憶體的控制作效能實驗。此演算法使用在一般的應用程式均能有效的減少指令快取記憶體的總 miss penalty，而在具有大量迴圈運算及函式呼叫的多媒體應用程式中尤有顯著成效。為驗證該演算法的可靠性及正確性，本論文也設計出一個 VLIW 架構的多核心嵌入式處理器，以做為該指令快取記憶體控制器的作用平台。主從式快取記憶體的設計具有以下特點：

1. 具有保存跳躍前指令的機制：

相同於 L1 Direct Map 快取記憶體控制器所使用的快取記憶體的大小(這裡指的是 entry 數目)，主從式快取記憶體控制器將一樣大小的快取記憶體分為數等分，大小相等的 sub-cache。每次當 cache miss 發生時，原先的指令保存在當前提供處理器指令的 sub-cache 中，新的指令則取代掉最近最久未使用(未使用即未提供處理器指令)的 sub-cache(LRU sub-cache)，這樣的方法對於跳躍後還會返回的程式 ex:Function Call，就不會造成二次的 cache miss。

2. 提供使用者對參數作最佳化設定的介面

主從式快取記憶體控制器針對與 L1 Direct Map 快取記憶體控制器使用同等大小的快取記憶體作分割，每個 sub-cache 的大小只會是原本快取記憶體大小的 $1/n$ (假設切成 n 等分)，如此會導致每次 miss 時補充的指令也會只有原先的 $1/n$ ，在往後的執行反而容易因此造成 cache miss，這是這種設計的一個 trade-off[26]，為了降低這個 trade-off 所帶來的影響，在此演算法中設定了一個參數來讓使用者設定跳躍發生後經過多少個週期如果原先的跳躍還未返回，就判定不會返回，而可以提前對存放原先跳躍前指令的 sub-cache 進行新指令的取代，而這個參數可以利用處理器的模擬器先預先執行過觀察，或者實際用處理器晶片進行測試利用 ICE 來進行觀察測量得到。另外，此指令預先存取機制也可搭配處理器的 Branch-Prediction 機制一同作用[3]。

3. 方便用數位的方式對快取記憶體進行功率管理

常見快取記憶體降低功率的方法大都是在佈局圖上用 Full-Custom 的設計方式來作規劃。而在主從式快取記憶體控制器裡，可以對每個 sub-cache 分別進行開關控制，也就是說除了正在使用的 sub-cache 需要將它的時脈打開，其餘未使用的 sub-cache 可以將其時脈關掉，正在保存指令的 sub-cache

則可給予較低的時脈。用 gated clock 的方式來達成快取記憶體功率控制，如此就可以用數位的方式做到。相較於同等大小的 L1 快取記憶體，較常見的方法還是在佈局圖上以 Full-Custom 的方式進行細步設計才能達到效果。

1.2 論文架構

本篇論文中，第二章介紹主從式快取記憶體控制器設計，從硬體架構到效能測試有詳細的說明。第三章介紹主從式快取記憶體控制器與數位訊號處理器的整合，從 VLIW 數位訊號處理器架構到開發工具製作，及週邊矽智產的整合。第四章在介紹晶片實現的過程，包含模擬驗證方法及結果，晶片製作，及測試效能的比較。最後，在第五章做總論。



第二章 主從式快取記憶體控制器

本章介紹主從式快取記憶體控制器的設計，包含其原理、架構、效能測試、特點及比較。

2.1 主從式快取記憶體控制器原理

2.1.1 主從式快取記憶體控制器硬體演算法

圖 2-1(a)為主從式快取記憶體的架構圖。主從式快取記憶體主要是把 L1 快取記憶體的 entry 拆成若干等份。以一個擁有 1k entry 的 L1 快取記憶體(圖 2-1(b))為例，在分為兩個 sub-cache 的架構中，共有兩個 512 entry 的 cache，四個 sub-cache 的架構中，共有四個 256 entry 的 sub-cache，依此類推。而在

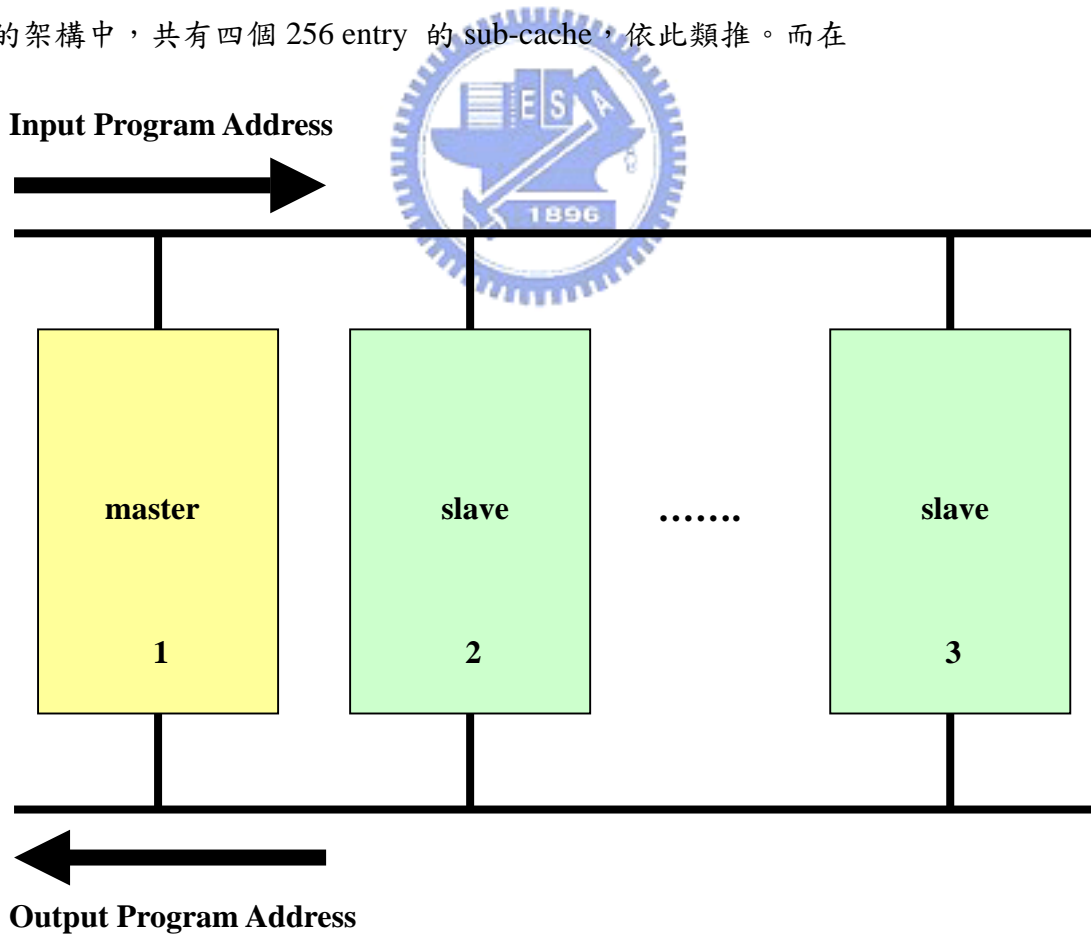


圖 2-1(a)：主從式快取記憶體架構圖

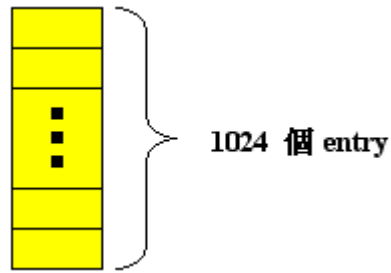


圖 2-1(b)：L1 快取記憶體

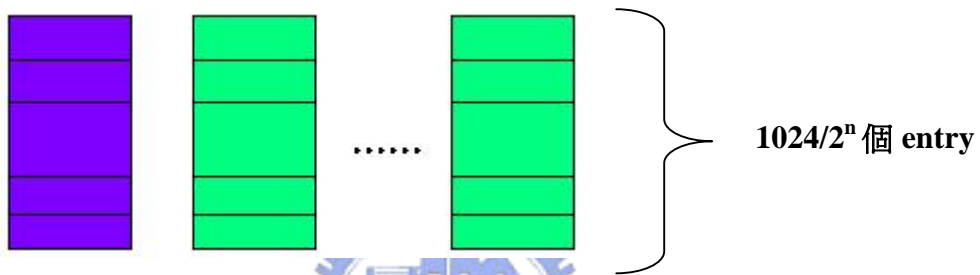


圖 2-1(c)：主從式快取記憶體

本篇論文的架構裡sub-cache的個數一定是2的冪次方(2^n) (圖 2-1(c))。

當一開始程式執行時，快取記憶體內還無任何指令。主從式快取記憶體從編號第一類的 sub-cache 開始填入指令，而編號第一類的 sub-cache 也將成為最初提供處理器提取指令的 sub-cache，在這個演算法裡，當下處理器提取指令的 sub-cache 稱為 master，其餘的 sub-cache 稱為 slave(圖 2-1(a))，這也是將此控制器命名為主從式快取記憶體控制器的原因。舉例來說，1024 個指令將從程式記憶體讀入，編號一的 sub-cache 將存入 512 個指令，編號二的 sub-cache 也將存入 512 個指令，依此類推。完成了整個快取記憶體的初始化後，快取記憶體發送信號通知處理器表示可以開始接受處理器的讀取，讀取的方式有兩種，在省電模式下是先比對目前被設為 master 的 sub-cache 的 Tag 部份是否有符合，如果符合就將指令輸出，若沒有符合則恢復其餘被設為 slave 的 sub-cache 的工作頻率，並

平行比對目前所有被設為 slave 的 sub-cache 的 Tag 部份，如果有符合的便將該 sub-cache 設定為 master，原先的 master 設定為 slave，並將指令輸出至處理器。而在高速模式下則是一開始就平行比對包含 master 及 slave 在內的所有 sub-cache 的 Tag 部份，符合的 sub-cache 設定為 master，原先的 master 設定為 slave，並將指令輸出至處理器。舉例如編號一的 sub-cache 的 Tag 部分沒有符合，然而在編號二的 sub-cache 中的 Tag 有符合，則將編號二的 sub-cache 設為 master(圖 2-1(d))。依上述演算法，在整個主從式快取記憶體的架構下，這種情況並不會造成真正的 cache miss，只是作為 master 的 sub-cache 改變了而已。

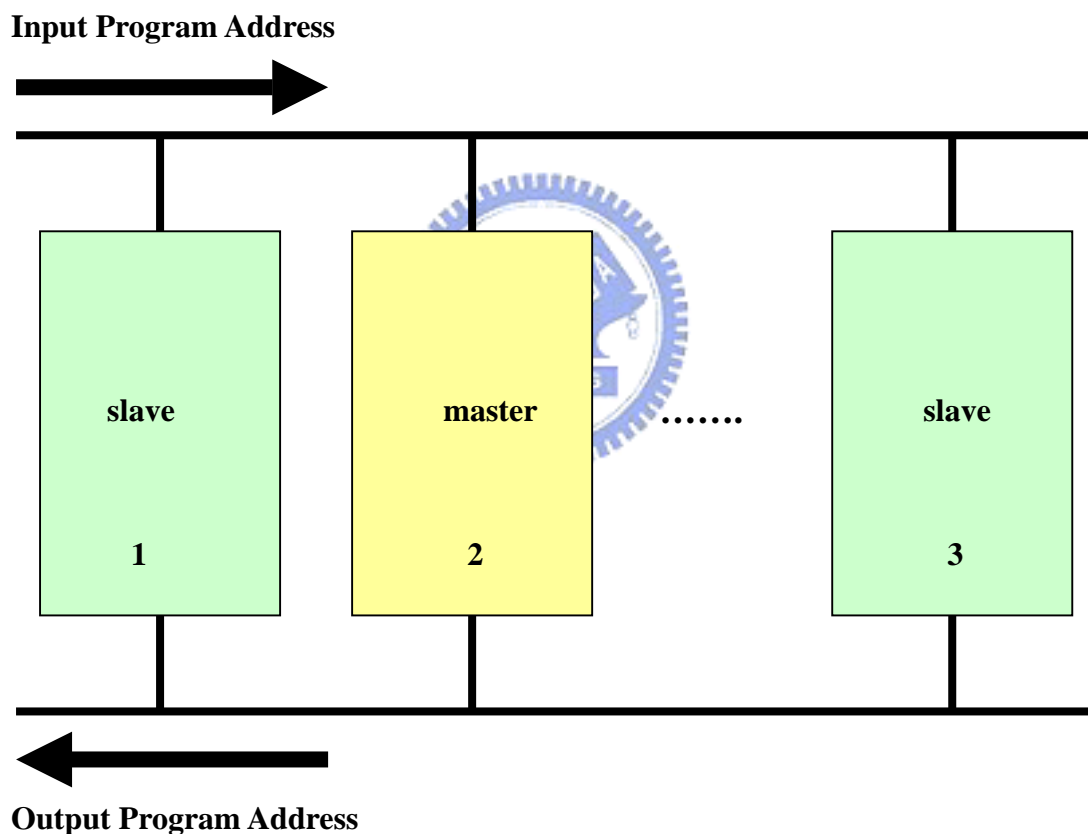


圖 2-1(d)：master 和 slave 切換後的圖

當真正的 cache miss 發生時(意思是在主從式快取記憶體架構下的所有 sub-cache 都 miss)，在 L1-Direct Map 機制裡，處理器被停下，等待 cache 將新指令從指令記憶體讀取進來取代現在存放在快取記憶體的指令[18][20]。而在主從

式快取記憶體控制器的機制裡，處理器被停下，然後對最近最久未使用的 sub-cache(LRU sub-cache)作指令的取代。此外，觀察指令快取記憶體的存取特性，由於指令都有地域性(locality)的現象，所以如果要在硬體中以比 LRU 更簡化的方式實現，又幾乎保持 LRU 的特性，可以選擇取代離 master 最遠的一顆 sub-cache。其理由為離 master 最遠的 sub-cache 應是對於現在執行的指令中地域性最差的，自然有很高的機會成為 LRU。以演算法而言，也就是對 master 的 sub-cache 的編號後一顆的 sub-cache 作指令取代(若 master 已經是編號最後一顆的 sub-cache，則取代編號第一顆的 sub-cache，採用環狀排序)。舉例來說，若現在 master 是編號第一顆，則對編號第二顆的 sub-cache 作取代，依此類推。而指令取代結束後，依據之前決定 master 的演算法，被 replacement 的那一顆 sub-cache 會成為 master。



由於是將原先 master 的後一顆 cache 作指令取代，所以原先的指令都還保留在原本作為 master 的 sub-cache(即 miss 前作為 master 的 sub-cache)裡面。所以如果是因為跳躍指令所造成的 miss，未來如果又跳回原本的程式區間，在主從式快取記憶體架構下就不會再次發生 cache miss，這種情況典型發生在函式呼叫和函式返回及跨邊界的迴圈運算，然而在 L1-Direct Map 架構下，這種情況仍舊會發生 cache miss。

2.1.2 使用者可調性設計

主從式快取記憶體控制器有一個 trade-off 在於當主從式快取記憶體控制器與 L1 Direct Map 快取記憶體控制器使用同等大小的快取記憶體作分割，每個 sub-cache 的大小只會是原本快取記憶體大小的 $1/n$ (假設切成 n 等分)。這樣會導致每次 miss 時補充的指令數量也會只有原先的 $1/n$ ，在往後的執行反而容易因此造成 cache miss。為了解決這問題，做法是提供使用者設定一個參數(如果在實際晶片應用，使用主從式快取記憶體控制器的晶片設計者必須留一組 I/O pin 以便

利於該晶片使用者將此參數由外部輸入至晶片或在應用程式中以軟體的方式進行設定。)，這參數的意義是跳躍發生後經過多少個週期如果還未返回，就判定不會返回，而可以提前對存放原先跳躍前指令的 sub-cache 進行新指令的取代。由於每個 sub-cache 都是獨立的，有自己的 I/O port，因此不同的 sub-cache 間可以平行的作輸出入動作。例如在此情況下，作為 master 的 sub-cache 可以繼續對處理器提供指令，而作為 slave 的 sub-cache 就可以與程式記憶體進行溝通，把新的指令讀取進來。

判斷跳躍發生後經過多少個週期如果還未返回，就判定不會返回的這個參數可以利用處理器的模擬器預先執行過觀察，或者實際用處理器晶片進行測試利用 ICE 來進行觀察測量得到。這裡提供一個基本的測量方式：先統計程式執行時發生的所有的跳躍到返回之間的週期數，如果到下次發生 cache miss 前從未發生返回，則該跳躍不列入統計。將有效的跳躍返回的週期數數據取最大值，則超過該數值者必然不會發生返回，即為所須之參數值。然而這是最簡易直觀的測量方法，但是量測出的參數卻未必是最有利的。考慮一種情況如：當幾乎所有的跳躍返回週期數都集中在三百個週期以內，然而卻有一組是六百個週期，若為了那一組將參數定為六百，則大部分的情況下都少了三百個週期可作指令的預先提取，如果將參數定為三百，雖然有一組可能會因此發生 cache miss 但是其他的情況下都可以有三百個週期作預先提取的動作，而可能因此避免大量因為快取記憶體空間不足所產生的 cache miss。上述提到取捨的情形，須以統計分析的方式去作權衡的，才能得到最有利的參數值。整個過程如圖 2-1(e)。

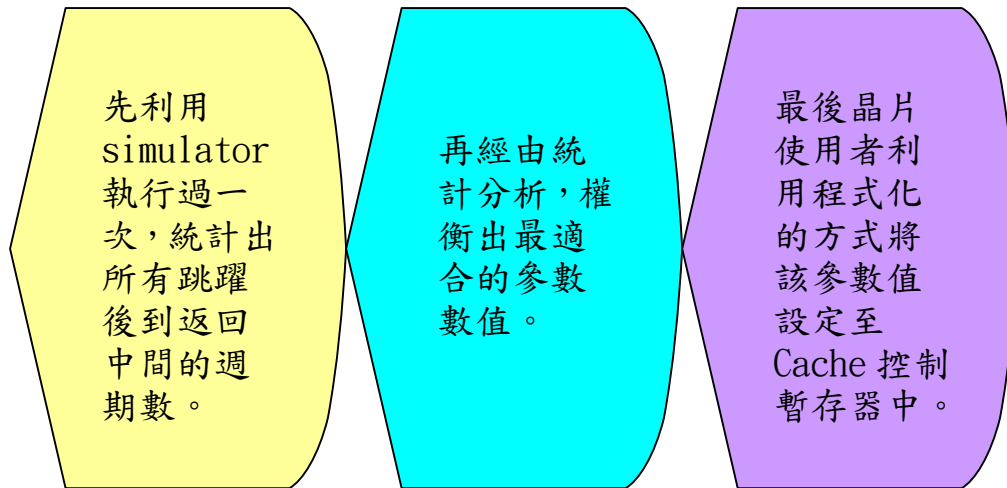


圖 2-1(e)：參數值的使用流程。

2.1.3 低功耗設計

主從式快取記憶體控制器的另一項特點就是可以用數位邏輯的方式來作功率的控制管理，以降低不必要的功率消耗，達到低功耗的設計。由於主從式快取記憶體裡每一個 sub-cache 都是獨立的，擁有自己的 I/O，因此可以對每一個 sub-cache 作個別的控制。一般快取記憶體降低功率的方法大都是類比策略，也就是在佈局圖上手動的方式來作規劃。而在主從式快取記憶體控制器裡，可以利用開關邏輯的方式來達成快取記憶體的功率控制，也就是說除了正在使用的 sub-cache 需要將它的 enable 打開，其餘的 sub-cache 可以將其 enable 關掉，用開關邏輯的方式來達成快取記憶體的功率控制。然而控制的方式也牽扯到作為快取記憶體的元件特性，如果該元件有使用時脈，也可以用 gated clock 的概念對時脈信號作控制，可以選擇降頻或關閉時脈信號(如:DRAM)，然而有些元件這些操作可能會導致儲存的資料流失，這是必須要注意的地方。相較於同等大小的 L1 快取記憶體用數位的方式幾乎很難進行整塊快取記憶體的功率管理，比較可行的方法還是在佈局圖上以 Full-Custom 的方式細步調整才能達到效果，主從式快取記憶體控制器對於簡化低功耗設計上具有優勢。

2.2 主從式快取記憶體控制器架構

本論文實作主從式快取記憶體控制器的RTL Model，記憶元件採用CIC提供的Synchronous的SRAM Model，工作頻率配合處理器為135MHZ。此控制器為三級管線結構。第一級為控制級，負責sub-cache的主從地位仲裁，sub-cache預先讀取指令的決策，cache miss時與外部記憶體的溝通控制以及處理器發生中斷時的應變處理。第二級為Tag的搜尋比較。第三級為當Hit時的資料讀取，將指令送往處理器。

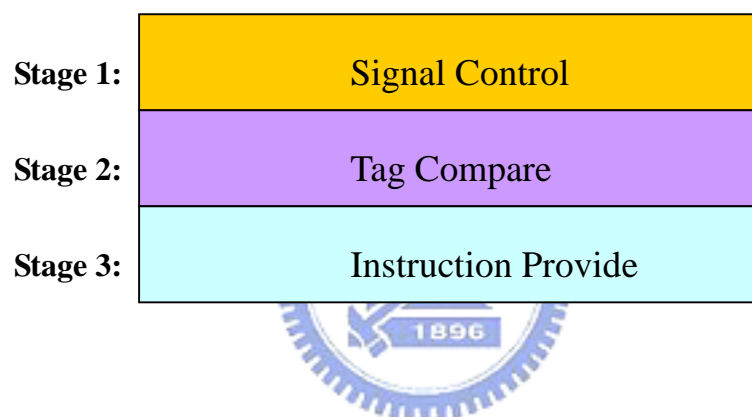


圖 2-2(a)：主從式快取記憶體控制器三級管線架構圖。

下面將針對每一級中的重要元件的功能逐一介紹。


2.2.1 程式初始指令讀取控制

當程式剛開始執行時，快取記憶體內並沒有指令，所以需要從指令記憶體中將指令讀取進來。讀取的方式是從編號第一類的 sub-cache 開始，將所有的 sub-cache 都填滿指令，因此與 L1 的快取記憶體相比較，所容納的總指令數是相同的。初始化完成後，發送 Ready 信號給處理器，處理器的程式計數器才開始計數。

2.2.2 位置比較器

當處理器的程式計數器開始計數後，計數值(Program Counter: PC)將會被送到快取記憶體控制器中，快取記憶體控制器依照PC後幾位的值（位數由快取記憶體大小所決定，如 1024 個entry則看後 10 位，為 2 的次方數。）去讀取Tag值與PC值的前幾位（位數由快取記憶體的大小和程式記憶體的大小作決定，如程式記憶體的entry數是 $65536=2^{16}$ ，快取記憶體entry數 $1024=2^{10}$ ，則 $16-10=6$ ，比較前六位。）作比較，若相同則判定則Hit，由下一級將指令輸出至處理器。若不同則將程式計數器的值與其餘的sub-cache的Tag作比較，若有相同，則將該sub-cache設定為master，由下一級將指令輸出至處理器。若所有sub-cache的Tag都沒有吻合的，則為發生cache miss。

2.2.3 發生 cache miss 時的處理



cache miss 一旦發生，主從式快取記憶體將把新的指令讀進最近最久未使用的 sub-cache(LRU sub-cache)，因此保留了原先發生跳躍前的指令。這樣的方法對於跳躍後還會返回的程式 ex: Function Call，就不會造成二次的 cache miss。而在發生跳躍後，依據 2.1.2 的使用者可調性設計，跳躍後到達使用者設定的參數值的週期數，則會執行預先存取指令的動作，將存放跳躍前指令的 sub-cache，做為新指令的取代(如圖 2-2(b))。

當 cache miss 時主從式快取記憶體控制器將發送 NOP 指令給處理器，因此不需要其他的信號線來對處理器發出中斷，維持管線的流暢性及設計上的簡化，一直到快取記憶體的指令更新完畢，才會重新由發生 Miss 的 PC 所對應的指令開始送出至處理器，同時有信號通知處理器的程式計數器可以重新開始計數。

2.2.4 處理器中斷時的處理

處理器中斷時，主從式快取記憶體控制器將把現在的 PC 值記錄起來，而輸出指令會維持在 NOP 指令，一直等待處理器的中斷解除，才會由發生中斷的 PC 位置開始重新執行快取記憶體的存取。操作流程如圖 2-2(c)。

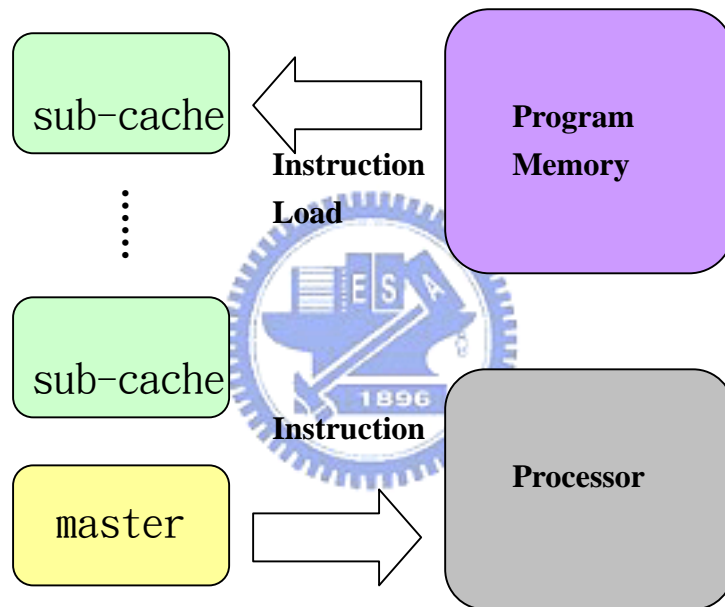


圖 2-2(b)：主從式快取記憶體預先存取流程圖。

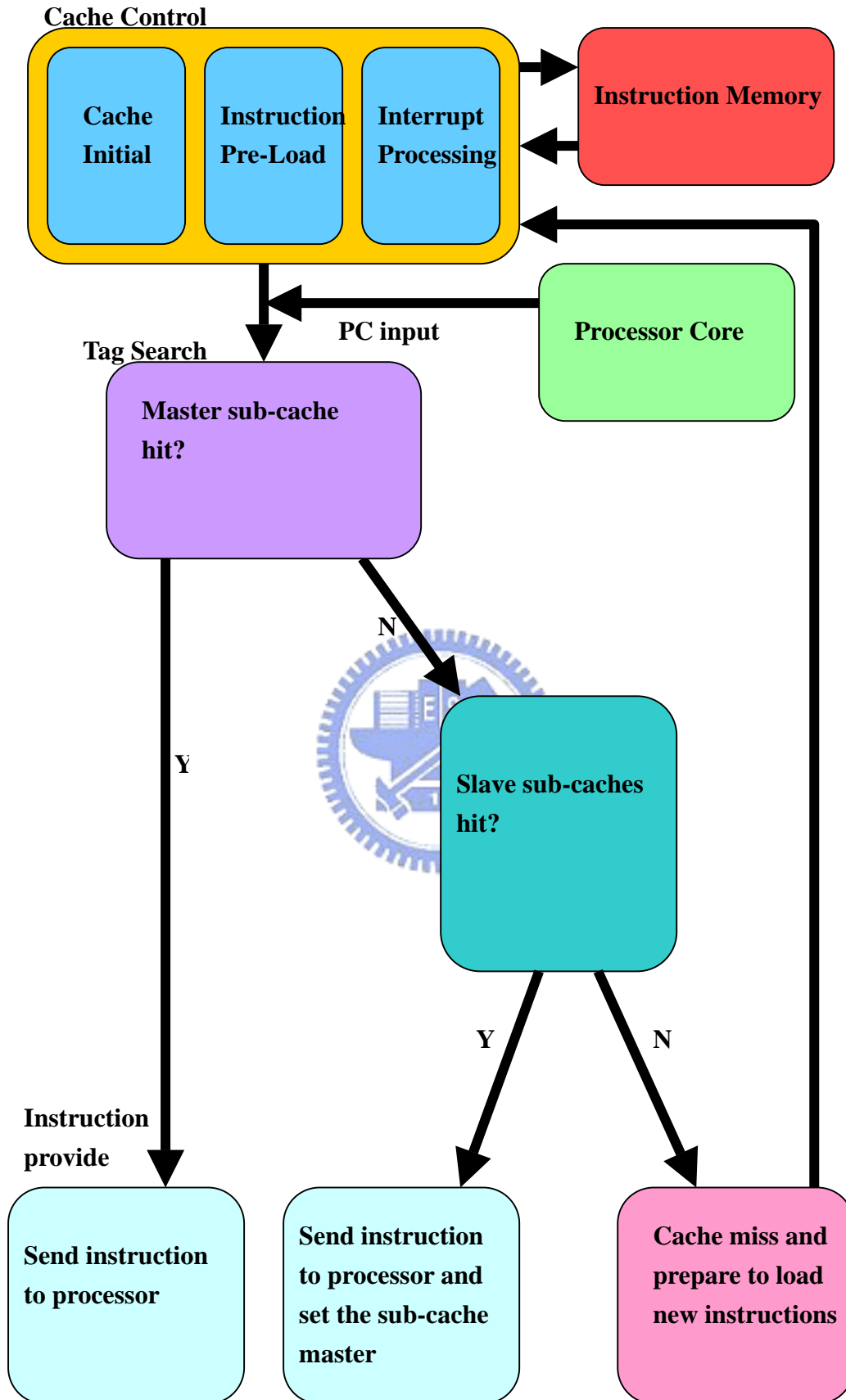


圖 2-2(c)：主從式快取記憶體操作流程圖。

2.3 主從式快取記憶體模擬器

為了驗證比較主從式快取記憶體控制器與 L1-Direct Map 控制器的效能差異，除了 RTL Model 外，本論文中也以 C++ 語言製作了兩者的模擬器，並製作了測試程式產生器(Testbench Generator)，以方便自動的大量驗證比較效能。

2.3.1 主從式快取記憶體與 L1 快取記憶體控制模擬器

為了比較兩者的效能，必須先整理出快取記憶體本身會影響效能的參數，首先針對要比較的 L1 架構的 cache 定出下列參數。

L1_Cache_Size L1 cache 的大小。

L1_Cache_Line_Size L1 cache line 的大小。

L1_Cache_Entry_Amount L1 cache 的 entry 數目。

N_Way N-Way associative 的 L1 cache。

其中 $L1_Cache_Entry_Amount = L1_Cache_Size / L1_Cache_Line_Size$ ， $N_Way=1$ 時為 Direct Map 架構，由於 L1 的架構也有可能是 N-Way associative 的架構，因此在模擬器實驗中也一併比較。

接下來針對對應的主從式快取記憶體架構定出以下參數：

MS_Cache_Amount 主從式快取記憶體架構下的 sub-cache 個數。

MS_Cache_Line_Size 主從式快取記憶體架構下每個 cache line 的大小。

MS_Cache_Entry_Amount 主從式快取記憶體架構下每個 cache 的 entry 數目。

其中在一般情形下 $MS_Cache_Line_Size = L1_Cache_Size$,
而 $L1_Cache_Entry_Amount = MS_Cache_Entry_Amount * MS_Cache_Amount$ 。

以上是跟本身快取記憶體的大小結構等有關的參數，然後測試程式本身的特性也會影響執行的效能結果，因此測試程式產生器必須能產生不同特性的測試程式，才能作準確的分析，與測試程式相關的參數將在下一節中作介紹。

2.3.2 測試程式碼產生器

本論文也以 C++ 語言開發出了一個測試程式碼產生器，以便產生出具有不同特性的測試程式。測試碼產生器將針對下列的參數作隨機種子，產生出具有各種特性的測試程式：



Code_Size 測試程式的長度。

General_Jump_Freq 平均多少個指令會發生一次跳躍。

General_Jump_Depth 平均每次跳躍的深度距離。

General_Jump_Freq_Off 為避免跳躍太頻繁的一個 offset，確保幾個指令以後
才會發生一次跳躍。

General_Jump_Freq_Upp 為避免跳躍太久未發生的一個參數，表示至少多少
個指令應該發生一次跳躍。

CR_Ratio 有返回動作的跳躍在程式跳躍指令中佔的比例。

2.3.3 效能指標

模擬器每執行一個測試程式，就會記錄與效能有關的結果參數，如下：

MS_Cache_Miss_Times master-slave cache miss 的次數。

MS_Cache_Penalty_Cycle master-slave cache miss 所造成的 penalty cycle 數。

L1_Cache_Miss_Times L1 cache miss 的次數。

L1_Cache_Penalty_Cycle L1 cache miss 所造成的 penalty cycle 數。

如前 1.1 節所述，對於不同的快取記憶體控制器，不能只以 cache miss 的次數作為效能的判斷依據，所以最終效能的判斷是用 cache miss penalty 的週期數來決定。

2.3.4 效能測試

本論文利用測試程式產生器產生大量的測試程式來進行效能測量，測量時參數跟本論文實際發展的晶片使用的參數完全相同，如下：

L1 快取記憶體大小：2k x 64 (bit)

主從式快取記憶體大小：2k x 64 (bit)

主從式快取記憶體 sub cache 個數：2

sub cache 大小：1k x 64 (bit)

測試程式個數：20000 筆

測試程式碼大小上限：32768 行

測試程式碼執行週期數上限：2000000 cycles

本論文的效能測量的觀察，如圖 2-3(a)，為在不同 CR_Ratio (有返回動作的跳躍在程式跳躍指令中佔的比例。)的測試程式中，使用主從式快取記憶體控制

器對於使用 L1-Direct Map 快取記憶體控制器的效能增進百分比，在此將主從式快取記憶體架構整體的效能提升參數定義名稱為 Eff_Improve，其計算方式如下：

$$\text{Eff_Improve} = (\text{L1_Cache_Penalty_Cycle} - \text{MS_Cache_Penalty_Cycle}) / \text{L1_Cache_Penalty_Cycle} \quad (2.1)$$

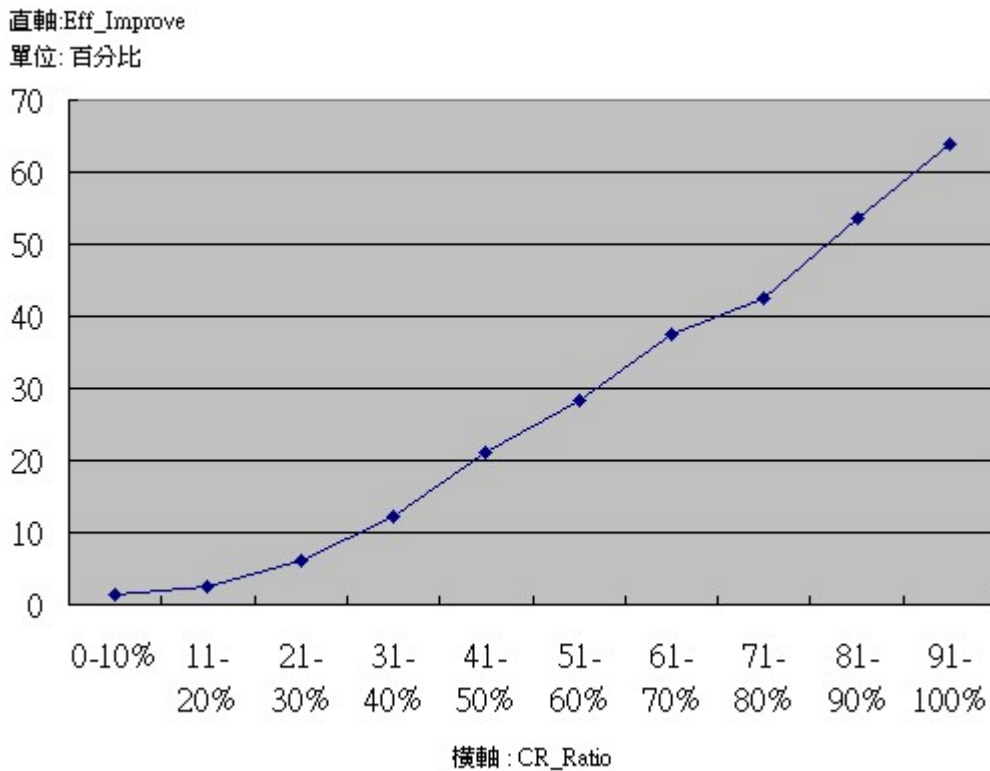


圖 2-3(a)：主從式快取記憶體的效能提升示意圖。

觀察模擬結果，可發現 CR_Ratio 與 Eff_Improve 呈現正成長的關係，當 CR_Ratio 的值越大，也就是 function call/return 的現象越明顯的測試程式，主從式快取記憶體的架構的效能提升也就越大。此外，即使 CR_Ratio 的值很低，主從式快取記憶體架構的效能仍不輸 L1-Direct Map 快取記憶體架構，換句話說主從式快取記憶體在任何的測試程式情形下，效能都還是優於或等於 L1-Direct Map 的快取記憶體架構。

深入對模擬曲線進行分析，發現 CR_Ratio 與 Eff_Improve 雖是正成長，但是卻不是 1：1 成長，也就是 CR_Ratio 成長 10%，Eff_Improve 卻不會也跟著成長 10% 這麼多，這裡與模擬前預估的情況有差異。探究原因後發現，在跳躍發生 call/return 的區間在 L1 cache 的 size 以內會使 CR_Ratio 高可是效能上卻看不出(即確實有 call/return 發生，但是因為 L1 本身不會 miss，所以效能上看不出)。這種跳躍在每個測試程式中的跳躍比例佔了越 20%~60%，自然的造成了 CR_Ratio 與 Eff_Improve 無法 1：1 成長。

本論文針對使用者輸入預先存取參數(詳見 2.1.2)定一名稱為 PLC (Pre-fetch Load Count)，針對主從式快取記憶體控制器使用 PLC 及不使用 PLC 的結果作一比較，以觀察 PLC 為主從式快取記憶體所帶來的增益。由於 PLC 當初的設計目的是為了消弭 sub-cache 的個數增加導致每個 sub-cache 的大小縮減所帶來的 trade-off(因為 $L1_Cache_Entry_Amount = MS_Cache_Entry_Amount * MS_Cache_Amount$ ，詳見 2.3.1。)，所以觀察重點放在 PLC 帶來的效能改進及 sub-cache 的大小的關係。因此使用 PLC 的改善比例參數定義名稱為 Eff_Improve_by_PLC，其中：

$$Eff_Improve_by_PLC = \frac{(MS_Cache_Penalty_Cycle_No_PLC - MS_Cache_Penalty_Cycle)}{MS_Cache_Penalty_Cycle_No_PLC} \quad (2.2)$$

MS_Cache_Penalty_Cycle_No_PLC：主從式快取記憶體未使用 PLC 時的 Miss Penalty Cycle。

圖 2-3(b)是當使用不同的 sub-cache size 時所展現出式(2.2)的結果。

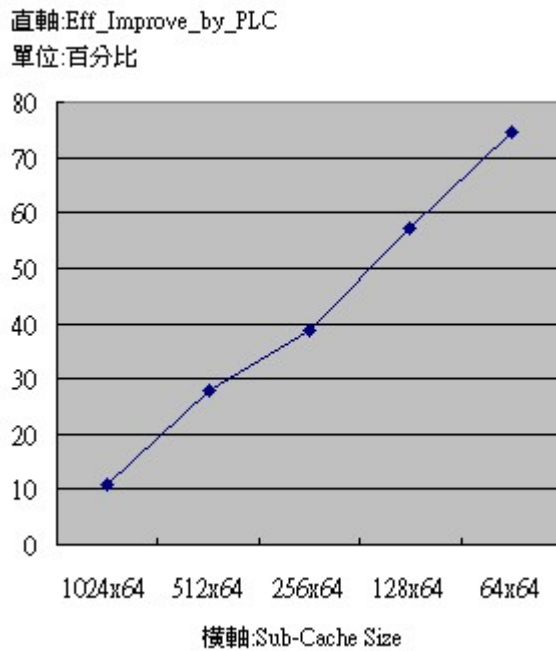


圖 2-3(b)：PLC 對主從式快取記憶體效能提升示意圖。

結果如預期，當 sub-cache 本身的 size 越小，PLC 平衡此 trade-off 的功能越顯著，然而此模擬也觀察出了一個現象，當 Eff_Improve_by_PLC 的值超過 50% 時，即使使用了 PLC，預先存取的速度還是略嫌不足(其原因為，由於此情況發生於 sub-cache 的大小較小，每次能預先存取的指令數也較少。)，整體的主從式快取記憶體架構的 Miss Penalty 會開始高於 L1-Direct Map 快取記憶體架構的 Miss Penalty，因此晶片設計者在決定主從式快取記憶體的 sub-cache 大小及個數時便可以此現象為重要的參考指標。

本論文也針對一般嵌入式處理器常使用的快取記憶體大小，在 General Case 下針對不同個數的 sub-cache 的情形作比較。實驗中快取記憶體大小分別為 1k-entry(即 cache line 總數為 1k)，2k-entry，4k-entry，8k-entry，在 20000 筆測試程式中，分別對 sub-cache 的個數為 2，4，8 的情形作測試。每筆測試程式都會跑過 sub-cache 的個數為 2，4，8 的情況，然後統計每種情形下擁有幾次最少的 cache miss Penalty 的測試程式筆數。圖 2-3(c)~圖 2-3(f)是模擬後的結果。

1k-entry :

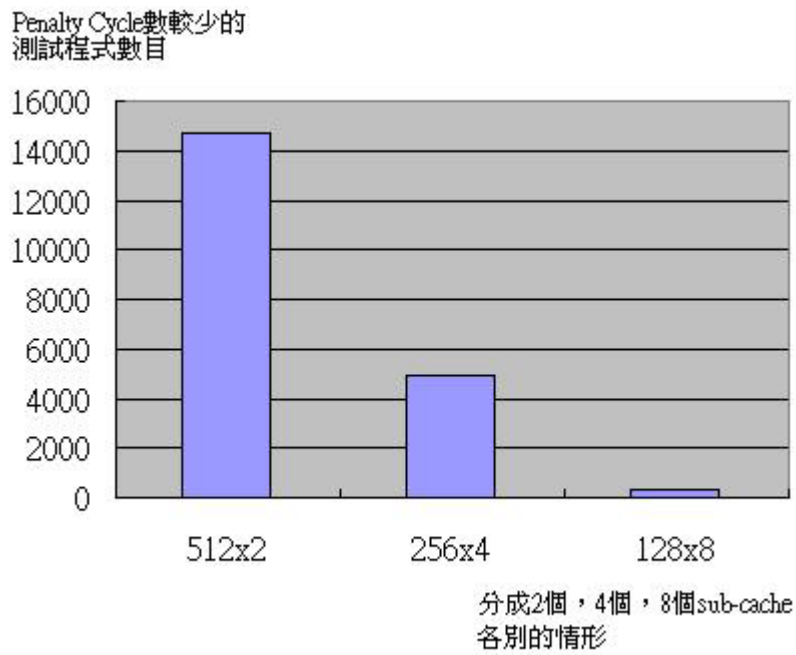


圖 2-3(c)：1k-entry 時的比較結果。

2k-entry :

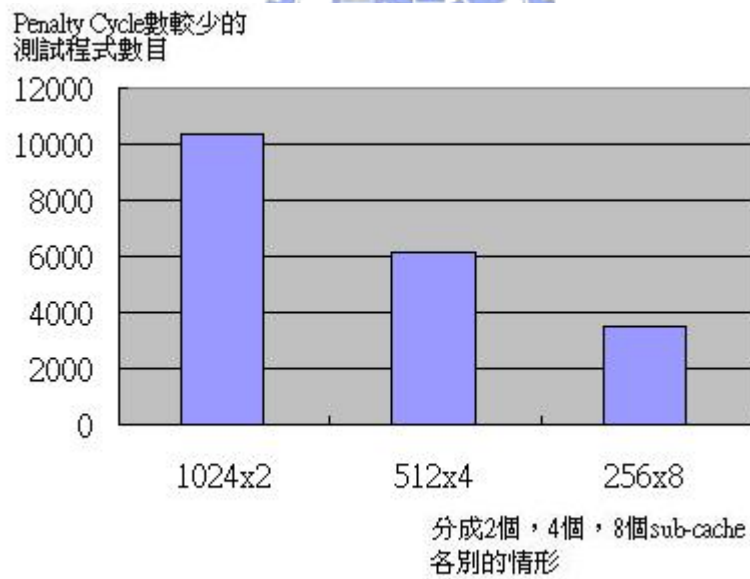


圖 2-3(d)：2k-entry 時的比較結果。

4k-entry :

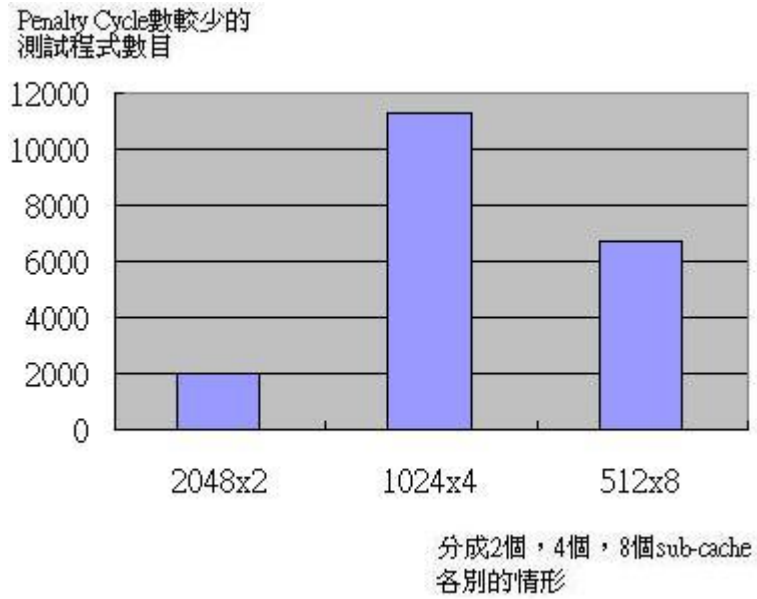


圖 2-3(e)：4k-entry 時的比較結果。

8k-entry：

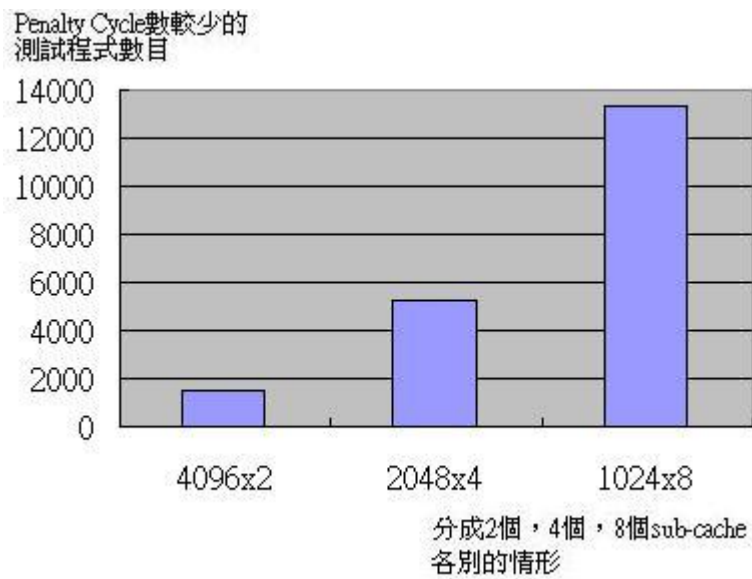


圖 2-3(f)：8k-entry 時的比較結果。

比較以上結果，可知在快取記憶體 entry 數在 1k 及 2k 的時候，以拆成兩顆 sub-cache 為在 general case 下 cache miss Penalty 最少的。在 entry 數為 4k 的時候，則是拆成 4 顆為最少的，8k 的時候則為 8 顆。在本實驗裡可發現，在 general case 下每個 sub-cache 的大小以 1k-entry 表現為最好。

接下來本論文將針對一個常見的特殊情形進行探討，以便明確的突顯主從式快取記憶體優勢，考慮下列的情形：

```
For(i=0;i<100;i++)  
{  
    ⋮  
    FFT(x,y,z)  
    ⋮  
}
```

圖 2-3(g)：可利用主從式快取記憶體架構大幅改善的特殊例子

當函式 FFT 在記憶體位置較遠，且函式內容較大，而被呼叫時無法和主程式同時被放進快取記憶體中，由於又在迴圈內部，所以會被反覆呼叫，而迴圈前後又還需要處理主程式的運算，所以會造成反覆大量的 cache miss，本論文也針對此 case 做了測試，得到結果如下：

Call Return Ratio : 0.995754
L1_Cache_Miss_Times : 974
L1_Cache_Penalty_Cycle : 997376
MS_Cache_Miss_Times : 3
MS_Cache_Miss_Penalty_Cycle : 768
Eff_Improve : 0.99923

由此結果發現，在主從式快取記憶體的架構下，cache miss 的次數及 Penalty Cycle 都大幅減少，以致於 Eff_Improve 大幅提升。此例在數位訊號處理的應用中尤其常見。而在現行大部分的處理器遇到這樣的情形，大都是仰賴高性能的編譯器對迴圈進行展開(loop unrolling)及重組指令間的相對位置來減少 cache miss 的發生，然而性能佳的編譯器研發並不是一件容易的工作，而使用主從式快取

記憶體架構就可以自然的去處理掉這種會造成重複且大量 cache miss 的情況，提供了另一種硬體上的解決方法。

2.3.5 功率管理

如 2.1.2 節所述，主從式快取記憶體架構中由於每個 sub-cache 是獨立的，有各自的 I/O，因此可用邏輯的方式對每個 sub-cache 的 Enable 及 CLK 進行控制，來達到功率的控管。然而由於在不同大小的快取記憶體，使用目前已研究出的各種數位的低功耗設計方法[13][16]，可能都會有不同的效果表現，因此本論文並未再深入對此進行探討，而是留作另一個研究主題，對各種已研發出的數位低功耗設計方法搭配主從式快取記憶體架構進行比較。

2.3.6 延伸比較

由於在嵌入式處理器中，L1 指令快取記憶體的架構幾乎都還是以 Direct-Map 的方式。因此本論文硬體實作上的對照組也是採取 Direct-Map 的架構，然而若延伸探討與 N-Way Associative 的架構作比較[33]，主從式快取記憶體仍舊具有以下 N-Way Associative 所沒有的優點：

- 1) N-Way Associative 架構的搜尋時間長，以 4-way 而言，每次要循序搜尋四個 cache line(entry)，而主從式快取記憶體架構可以平行搜尋每個 sub-cache，即使 sub-cache 個數增加也不會因此延長處理的時間。
- 2) N-Way Associative 架構相較於主從式快取記憶體架構，比較困難用數位的方式管理 power。
- 3) 因為 N-Way Associative 架構 I/O pin 只有一組(為單一顆 Cache)而主從式快取

記憶體架構每個 sub-cache 都是獨立的 pin，因此主從式快取記憶體架構可以平行的做指令的預先存取工作且不必中斷正在做指令輸送給 core 的 sub-cache[14]。

2.4 結語

本章節針對主從式快取記憶體架構以及 L1-Direct Map 快取記憶體架構比較了兩萬組 general case 的測試程式。觀察出在測試程式發生跳躍返回比例(CR Ratio)越高的情形下，主從式快取記憶體的表現越好，然而在跳躍返回比例(CR Ratio)很低的情形下，主從式快取記憶體的表現依舊優於或等於 L1-Direct Map 快取記憶體的表現。另外也對主從式快取記憶體在不同大小(1k entry~8k entry)的快取記憶體空間的情形下對兩萬組 general case 的測試程式做比較，得出在不同大小的快取記憶體空間的情形下，如何切割 sub-cache 的 size 為最佳方法。



第三章 多核嵌入式處理器

主從式快取記憶體控制器硬體地位上為一 IP，不能單獨存在運作，需搭配一個處理器。因此，本論文也製作出一個 64 位元超長指令集架構 (VLIW) 處理器核心，以呈現完整的應用。以下章節將說明此超長指令集架構處理器的特性、指令集、軟體開發環境及可搭配的其他 IP 作一介紹。

3.1 VLIW 處理器架構

本論文設計的 VLIW 訊號處理器擁有七級管線架構的設計，為一個 Multi-core 和 Multi-Ram 的處理器 IP，Multi-core 可以是 DSP 運算單元、或微控制單元。當一個指令透過程式計數器 (Program counter) 抓取進入處理器內部後，會先進入預先解碼 (Pre-decoder)，在這一級將一個 64bit 的 VLIW 長指令做拆解，成為兩個 32bit 的指令，並決定兩個 32bit 指令分別應該進入哪一個 core 中執行。當指令進入某一個核心時，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。周邊裝置的存取透過智慧型 DMA (3.3 節) 來規劃，組譯器及處理器直接支援智慧型 DMA 的設定及使用，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智慧型 DMA，將資料儲存在內部記憶體，或內部運算的結果送到周邊裝置上輸出，架構如圖 3-1(a)。

3.1.1 VLIW 處理器核心

VLIW 架構訊號處理器擁有七級管線架構，七級管線包含程式計數/跳躍判斷模組/指令抓取模組、快取記憶體模組、指令預解碼模組、指令解碼模組、暫存器模組、算數邏輯單元模組及隱含的寫回級，以下分別描述其主要的工作簡述

如下：

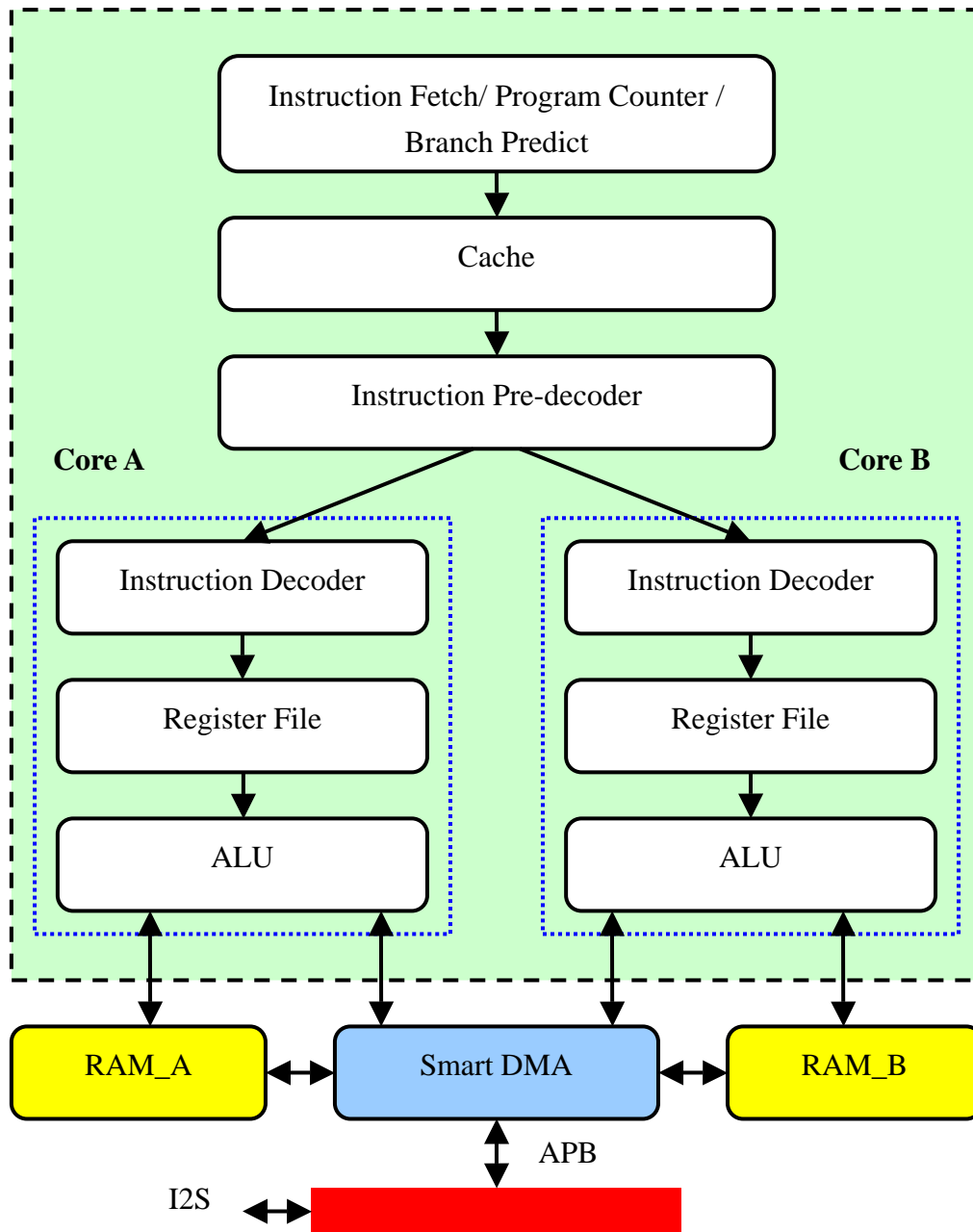


圖 3-1(a)：VLIW 架構多媒體訊號處理器的組成結構

1. **PC 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，轉成程式記憶體的字址，送進下一級的快取記憶體去抓取指令。其中，由於有些指令在 ALU 級時會需要處理中指令的程式計數器值，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。

2. **Cache**：將 PC 位址傳到快取記憶體去抓指令，其中 PC 的位址線為 16-bit，傳回 VLIW 指令為 64-bit，然而若快取記憶體模組本身有切級，則實際管線總級數必須將快取記憶體模組的級數也一併作計算，以本論文所使用的主從式快取記憶體架構為例，由於快取記憶體控制器架構為三級，因此若搭配此 VLIW 訊號處理器，則總級數為九級。
3. **Instruction Pre-decoder**：這一級的目的是把一個 64-bit 的 VLIW 長指令，切割成兩個 32-bit 的指令分別交給下面的兩個處理器核心做運算。由於整個設計都是朝 IP 化的原則，所以當增加新的 IP 處理核心，尤其是新增不同類的處理器核心時，本級電路會判斷不同的指令運算適合什麼樣的核子去處理。指令碼也在此級被抓取，判斷怎樣的指令要交給哪一個處理核心執行需要指令碼去進行判斷，指令碼的提前抓取也很方便在下一級的指令解碼器中做作解碼的動作。
4. **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。內建兩組處理核心，所以也提供了指令去從兩個處理核心中互相提取對方暫存器組的資料，方便資料的直接交換。由於此處理器整合智慧型 DMA 模組的設計，因此新增一些智慧型 DMA 控制指令。還需提供智慧型 DMA 一組存取控制暫存器的位址訊號線，以決定智慧型

DMA 的控制参数要写到哪一个智慧型 DMA 控制暂存器内。

5. **Register File**：每个处理核心有 32 个通用暂存器。主处理核心(core A) 有额外的 16 个中断暂存器来处理外部中断、内部优先权中断、与其他 IP 状态设定所使用的 register。暂存器组为 2 读 1 写的格式，负责提供解码器所解码出的来源运算元值，并将 ALU 级算出的目的运算元写回。在此模组中，有很多连接 ALU 级模组的输出信号，目的是把这些 ALU 级要用到的信号经由管线级送往 ALU 级中，利用这些信号来完成 Data forwarding 的动作。
6. **ALU**：本级功能是计算出逻辑或运算值，为了 Data forwarding 的实作，Data memory 及 Write back 这两级隐含在 ALU 级内。Data forwarding 的机制是利用前面一级一级传回的信号实作而成，目的是为了减少 RAW hazard。
7. **Write-Back**：由 ALU 或 Memory 写回 Reg File 的动作，或由 ALU 写入 DMA 或 Memory。



除了以上基本的管线模组设计外，还有一些特殊硬體设计需求被强调出来，包含以下五种说明：

1. **Bit Reverse**：针对 FFT 运算所增加的 memory 定址模式[34]，例如位址 (01101) 可被转成 (10110) 的位置储存。
2. 一个指令周期完成乘加运算。
3. **Regular Loop Prediction**：数位讯号处理运算中有很多固定次数迴圈的运算，利用一个良好的跳躍预测 (Branch prediction)，使处理器不会有 Control hazard 造成的不必要 Stall。
4. 良好的 **Data Forwarding** 机制。
5. **Condition Branch**：预测採用 Prediction-untaken 方法设计。

3.1.2 VLIW 處理器指令集

VLIW 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、其他類指令、智慧型 DMA 控制類，列表如下：

資料搬移指令：

Instruction	Opcode	Example	Mode
MOVRC	000001	MOV rd,data	Direct
MOVRR	000010	MOV rd,rs	Reg-Reg
MOVRM	000011	MOV rd,address	Direct
MOVMR	000100	MOV address,rs	Direct
MOVMRM	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
MOVREVMRM	101100	MOV @rs2,rs	Bit Reverse
MOVREVRM	101101	MOV rd,@rs	Bit Reverse

表 3-1(a)：資料搬移指令列表

VLIW 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，其中 MOVARR 可以互相提取不同處理器核心的通用暫存器的內容。

算數與邏輯運算指令：

Instruction	Opcode	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 3-1(b)：算數邏輯運算指令列表

跳躍指令：

Instruction	Opcode	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

表 3-1(c)：跳躍指令列表

Address 的部分也可以是 Label，組譯器會自動轉換成對應的位址。

其他指令：

Instruction	Opcode	Example
SET	011100	SET A,rs
INTOK	011101	INTOK
SHR	100000	SHR rs
SHL	100001	SHL rs
Instruction	Opcode	Example
LOOP	110010	LOOP 100
END_LOOP	110011	END_LOOP
ENDC	011111	ENDC

表 3-1(d)：其他指令列表

SET 指令是當 VLIW 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。LOOP 與 END_LOOP 則是迴圈控制指令，LOOP 後面的數值為迴圈重複的次數。

智慧型 DMA 控制相關指令：

Instruction	Opcode	Example
SDMAD	100100	SDMAD data
SDMAR	100101	SDMAR rs
GDMA	100110	GDMA rd
DMAOK	101001	DMAOK
GDMAR	101110	GDMAR rd,address

表 3-1(e)：智慧型 DMA 控制指令列表

SDMAD 指令是以 10 進位數值的方式直接設定智慧型 DMA 的控制暫存器。
 SDMAR 指令則是以處理器暫存器內的數值來設定智慧型 DMA 的控制暫存器。
 GDMA 指令則是以一個處理器暫存器的值來存放一個智慧型 DMA 是否已經做完的旗標。DMAOK 指令是智慧型 DMA 的所有控制暫存器都設定完以後，產生通知智慧型 DMA 可開始運作的信號。GDMAR 指令則是將智慧型 DMA 的結果暫存器的值放入處理器暫存器中。

3.1.3 子核心特色

主核心除了資料的搬移及運算外，同時可處理分支跳躍、外部中斷、內部優先權中斷、與其他 IP 的狀態設定。而副核心只拿來處理運算及資料搬移指令，可以需求自由的增加副核心數量，為 flexible n-core parallel processor[32]架構，每個 core 的 Reg 可互相存取。

3.1.4 不同子核心間的資料讀取與寫入

當運算完成後，必須將運算結果寫回 Reg File，而寫回 Reg File 的來源有兩個：本身的 core，其他的 core。在本論文裡，要將自己的 core 的運算結果寫入至其他的 core 中，必須使用指令 MOVARR，而優先權上，若發生本身的 core 及其他 core 同時發生寫入的現象，則以本身的 core 的寫入為優先，同時並發送中斷信號給其他的 core。(然而理論上不應該會有此情形發生，程式設計者應避免邏輯上出現此一情況，而開發工具發現程式設計者寫出這樣的程式也應會發出警示訊息。)

3.2 軟體開發環境

3.2.1 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。

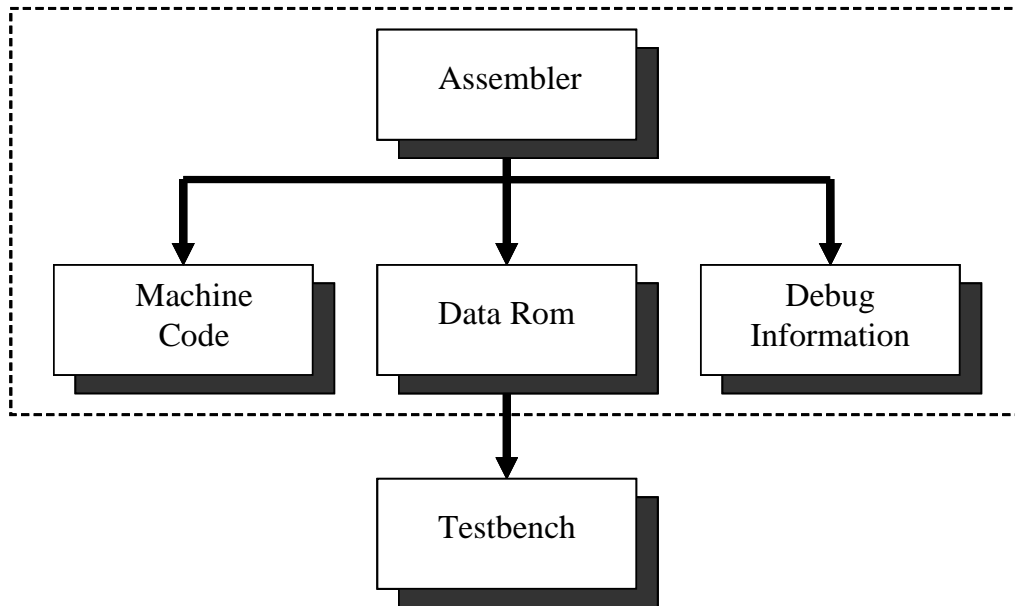


圖 3-2(a)：組譯器 (Assembler) 的用途

圖型化介面組譯器如下圖，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：產生程式記憶體體的資料，提供模擬及後續測試必備的資料。

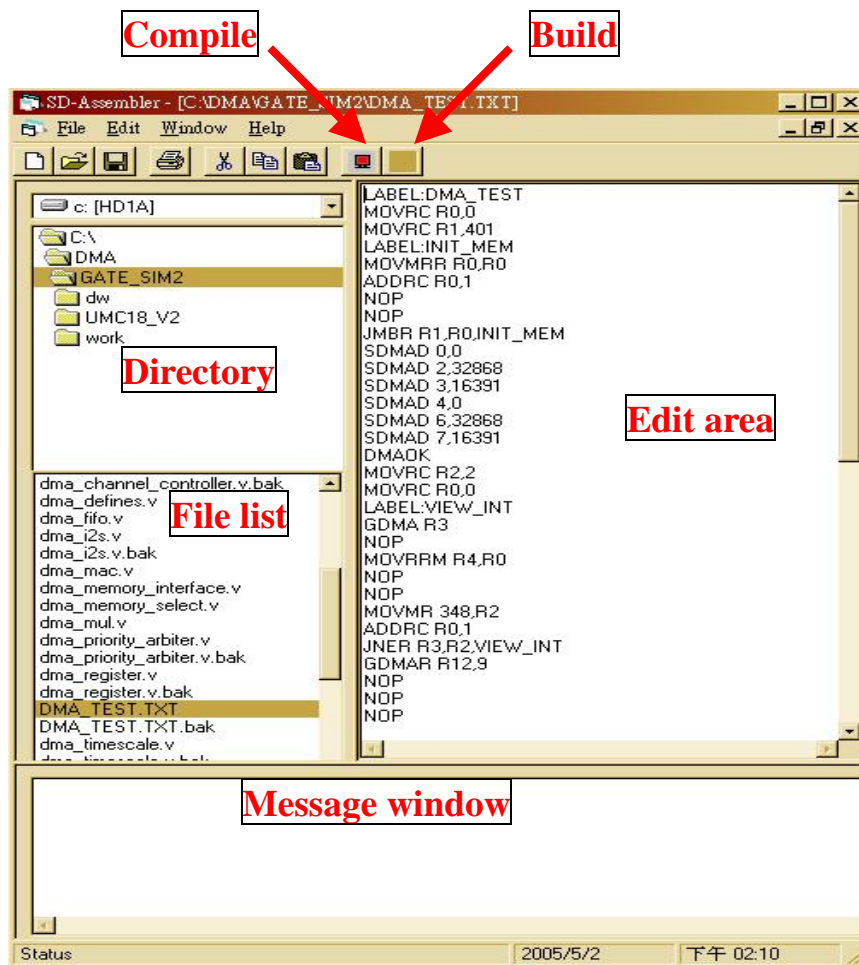


圖 3-2(b)：圖形化介面組譯器 (Assembler)

VLIW 處理器使用超長指令集的設計核心，在編寫程式及組譯時須注意：

- 平行處理：平時以處理核心 A 為主，若需使用處理核心 B，則需在兩指令間加上分隔符號『||』，若不需有處理核心 A，則須指令 B 前加上『NOP』指令，如下圖：

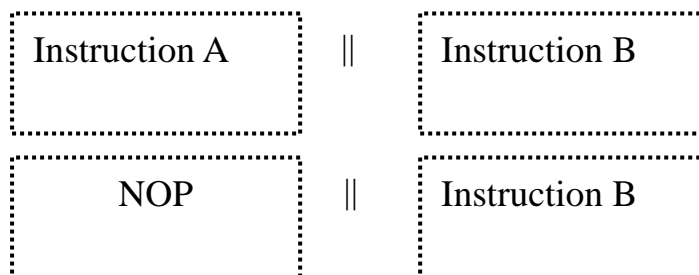


圖 3-2(c)：平行處理指令的使用

3.2.2 計次性迴圈的處理

在數位訊號處理的應用中，for迴圈是非常常見的，for迴圈的特性是只要是在迴圈次數內，則每次都會跳躍至迴圈開始處，因此不同於一般的conditional branch每次要去判斷是否要跳躍，像for迴圈這種regular loop，可以不經判斷而在每次的迴圈尾就直接返回迴圈頭，而減少管線在跳躍處理上造成的Penalty Cycle。

本論文開發出的組譯器遇到LOOP指令時，會去計算LOOP到END_LOOP間的指令數，然後將此數值與迴圈要重複運行的數值都編入機械碼中，處理器內部會有兩個counter分別存入這兩個數值。存放LOOP到END_LOOP間的指令數的counter此時會控制程式計數器，以避免抓取到END_LOOP以後的指令而影響效能，而存放迴圈重複次數的counter則提供處理器跳離迴圈的依據。本論文就是以軟體聯合硬體的方式來解決計次性迴圈[10]，這樣的方法與常見的Hardware Loop的解決方式也是不相同的，然而都可以達成目的。



3.2.3 模擬器

本論文也針對該處理器以高階語言(C++)設計了一個 simulator，以便能快速驗證各種測試程式。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，

如下圖所示：

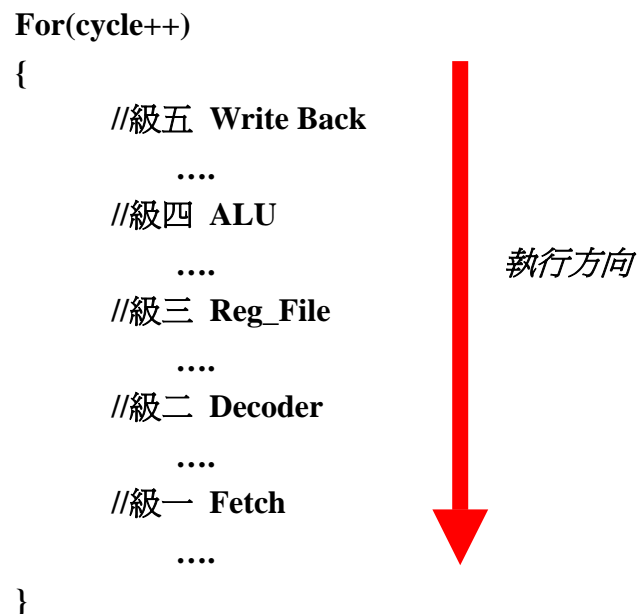


圖 3-2(d)：利用倒寫管線以使用高階語言的寫法描述管線。

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。

3.3 智慧型 DMA 控制器

本處理器搭配有本實驗室研發的智慧型 DMA 控制器，下面將針對該控制器的功能，架構，使用流程，作一簡介，詳細內容請參考本實驗室畢業之蘇育緯學長於 2005 年發表的碩士論文：智慧型直接記憶體存取器設計。

3.3.1 智慧型DMA控制器功能

傳統 DMA 僅有連續資料傳輸及透過 LLI (Linked List Item) 不連續資料傳輸，並且支援四種傳輸模式，記憶體到記憶體、記憶體到周邊、周邊到記憶體、周邊到周邊。針對目前 DMA 控制器的研究[17]，僅著重在傳輸效率的提升和通訊應用層面。為了使得記憶體內資料的提取及運算更有效率[29]，因此，本處理器搭配了實驗室開發的智慧型 DMA 控制器(於 2005 年由蘇育緯製作並發表碩士論文：智慧型直接記憶體存取器設計[38]。)利用 DMA 有控管大量資料的特性及結合 DMA 資料傳輸與數位訊號處理能力的 IP[15]，有效率地輔助一般訊號處理器做解決大量資料排序與大量乘加 (MAC) 運算功能[37]，經測試後發現，本處理器和智慧型 DMA 的結合，可達成具有 DSP 的工作能力。智慧型 DMA 的主要設計特點如下：

(a)應用上支援廣泛的 I/O 系統，並有效率地處理資料流：

多媒體訊號處理需要大量的資料流做為輸入輸出，因此必須設計一個 DMA 模組，支援匯流排結構，並對 I/O 進行大量資料移動、緩衝及控管的工作。一般的 DMA 僅支援連續資料傳輸，因此在處理數位訊號資料時顯得沒有效率，若只需要移動資料的一半 (如: down sampling 動作)，仍必須將所有資料搬入記憶體中來處理，既沒有效率又佔用匯流排的頻寬。本論文提出的智慧型 DMA 改善原始 DMA 傳輸設計，並提供四種定址方式有效處理資料搬移問題：

遞增/遞減定址法 (Increasing/Decreasing Addressing)

環狀定址法 (Circular Addressing)

鏡射定址法 (Mirror Addressing)

索引定址法 (Index-based Addressing)

透過以上四種定址法的組合，達到選取有效資料，排除無效的資料，降低頻寬的使用，並提高資料傳輸的效率及降低處理器的負擔。

(b)輔助 DSP 運算的 Co-processor：

智慧型 DMA 控制器內建了一組乘加運算器 (multiply-and-accumulate, MAC)，搭配上上述四種定址法，使得只能處理資料傳輸的 DMA 提升到運算的層次，如：DCT、FIR、DWT…等運算[7]。

DCT：Mirror + Index-based + Increasing Addressing

DWT：Increasing + Decreasing Addressing

FIR：Increasing + Decreasing Addressing

(c)增加少量硬體成本達到 DSP 效能：

智慧型 DMA 支援雙通道資料記憶體快速向量運算[21]，擁有直接存取兩個記憶體的能力，且在 I/O 匯流排上支援 APB 匯流排的標準[8][31]，使用者能夠在周邊匯流排上加掛相容 APB 介面的周邊裝置，擁有記憶體到記憶體、記憶體到周邊、周邊到記憶體、周邊到周邊四種傳輸模式。如表 3-3(a)，加上 MAC 運算單元後，僅增加 10%的 Gate Count，成本相當低。

ITEM	Gate count
Smart DMA	31.5k
DMA	28.5k
MAC	3.0k

表 3-3(a)：智慧型 DMA 與 DMA 面積比較

3.3.2 智慧型 DMA 控制器架構

一般的 DMA 與智慧型 DMA 架構的差別在於後者在通道控制器對定址法的支援、內建 MAC 運算器、因應不同功能的暫存器組、不同匯流排的支援和直接支援雙記憶體的介面等[36]。智慧型 DMA 整體架構設計如圖 3-3(a)，包含通道控制器、優先權仲裁器、暫存器組、乘加運算器、中斷控制器及記憶體介面[23]，其設計方法參閱智慧型直接記憶體存取器設計[39]。

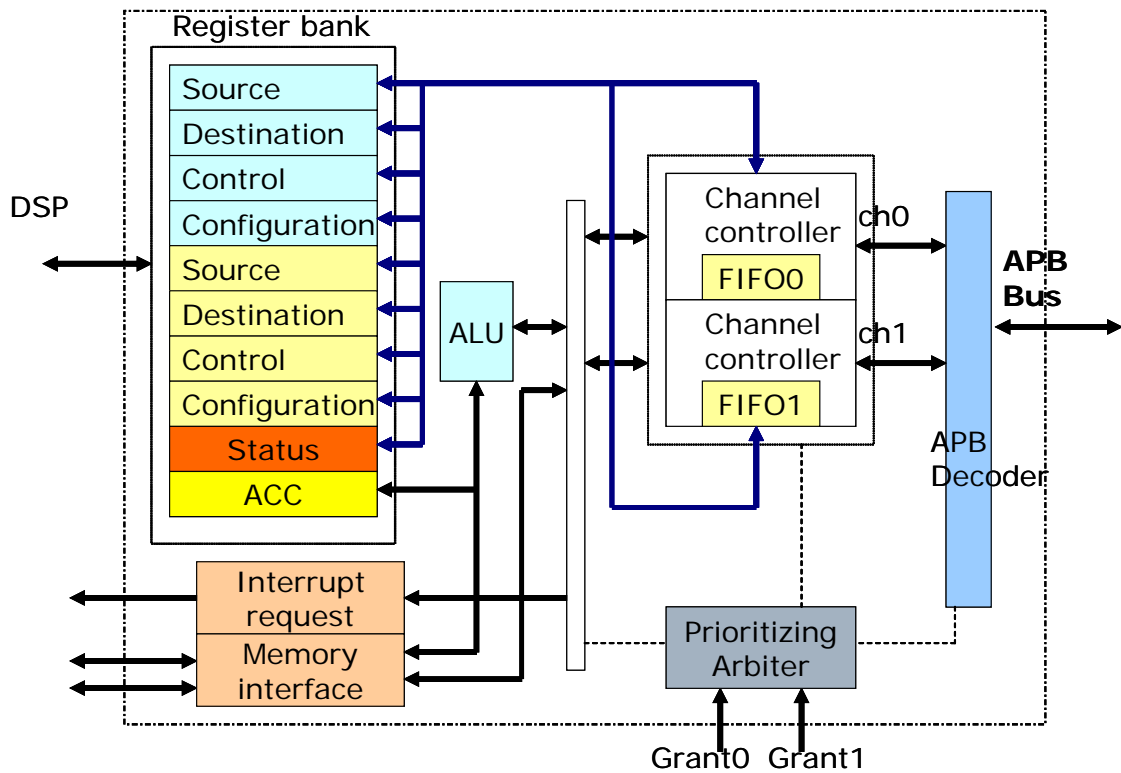


圖 3-3(a)：智慧型 DMA 的架構

3.3.3 智慧型 DMA 使用流程

智慧型 DMA 的設定方式如圖 3-3(b)，將來源暫存器、目的暫存器及控制暫存器設定後，配置暫存器必須最後一步設定，用以將通道致能。此時可去處理其他工作，待中斷發生後，智慧型 DMA 處理的工作跟著結束。在設定暫存器後，其值多會被保留下來，因此，若每次設定僅需選擇有改變的暫存器設定即可。

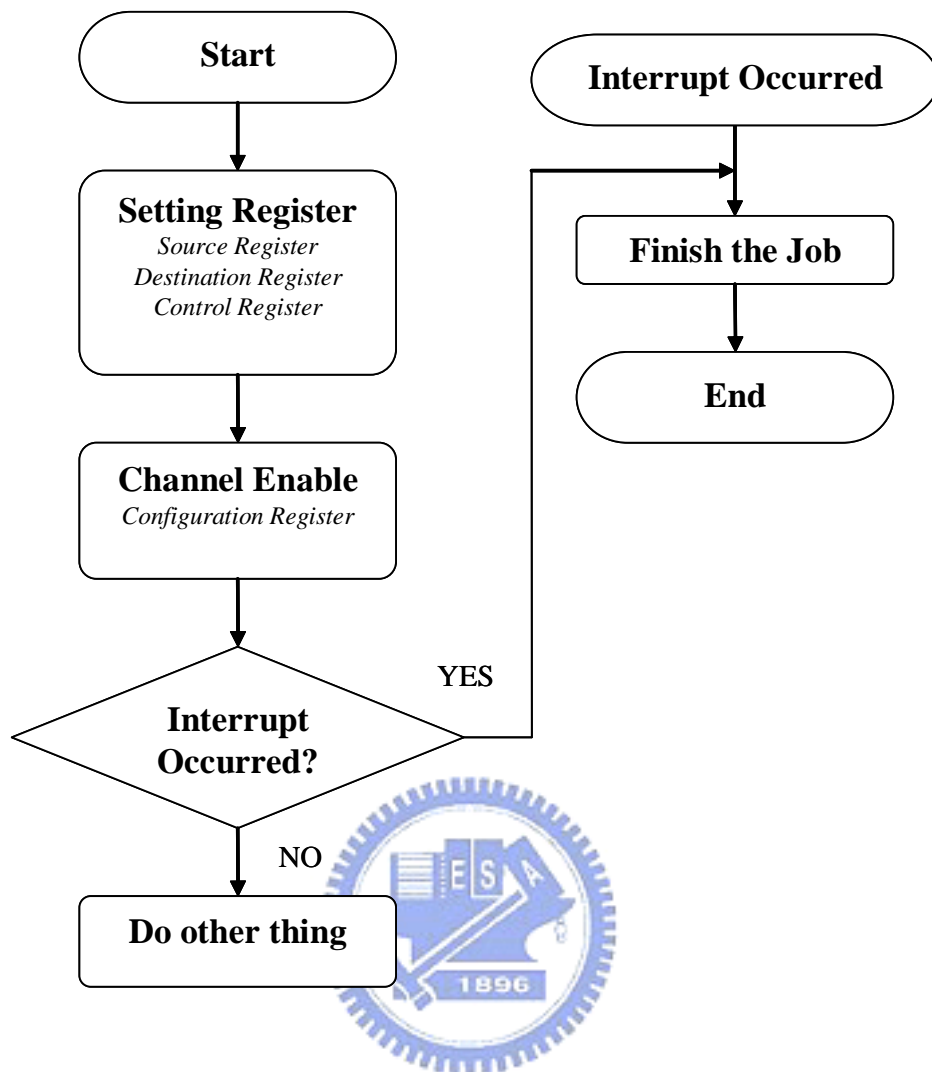


圖 3-3(b)：智慧型 DMA 的使用流程

3.4 整合

本論文將智慧型 DMA 和 VLIW 處理器做一個整合，以作為驗證主從式快取記憶體正確性及效能的平台。如圖 3-3(c)，VLIW 處理器整合智慧型 DMA 成為一個系統，擁有多個共用匯流排。程式記憶體為外接方式，內部兩個記憶體與智慧型 DMA 共用兩個資料匯流排。智慧型 DMA 並支援周邊匯流排，增加整體的擴充性。

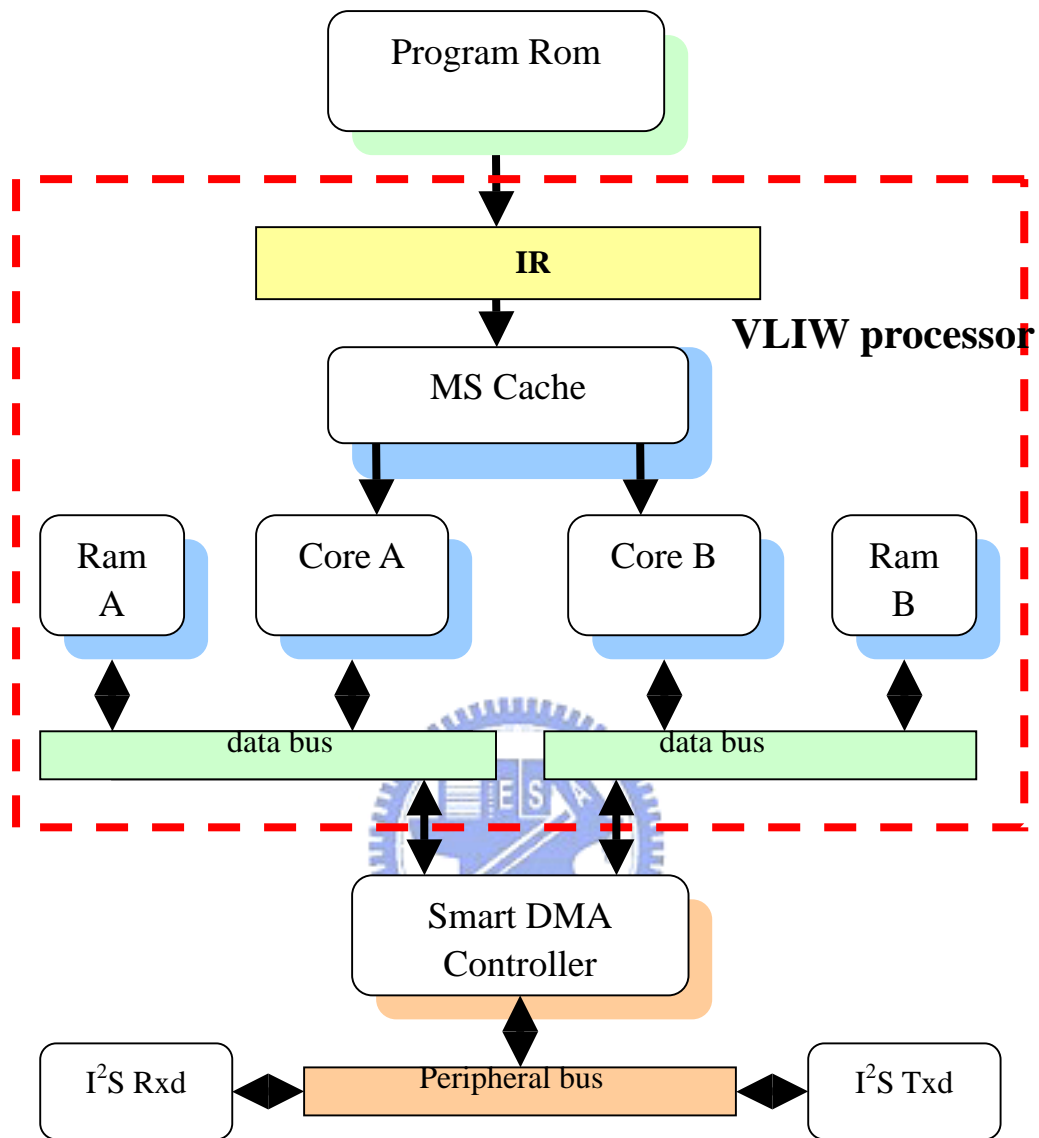


圖 3-3(c)：VLIW 處理器與智慧型 DMA 的整合架構圖

整個系統共有四條分開的匯流排，智慧型 DMA 與處理器共用二條資料匯流排，透過資料匯流排讓處理器與智慧型 DMA 共用兩個資料記憶體。資料匯流排的管制與衝突，由 VLIW 處理器解決，當沒有使用到記憶體相關的指令，記憶體的主控權會釋放給智慧型 DMA，以解決衝突的情況。

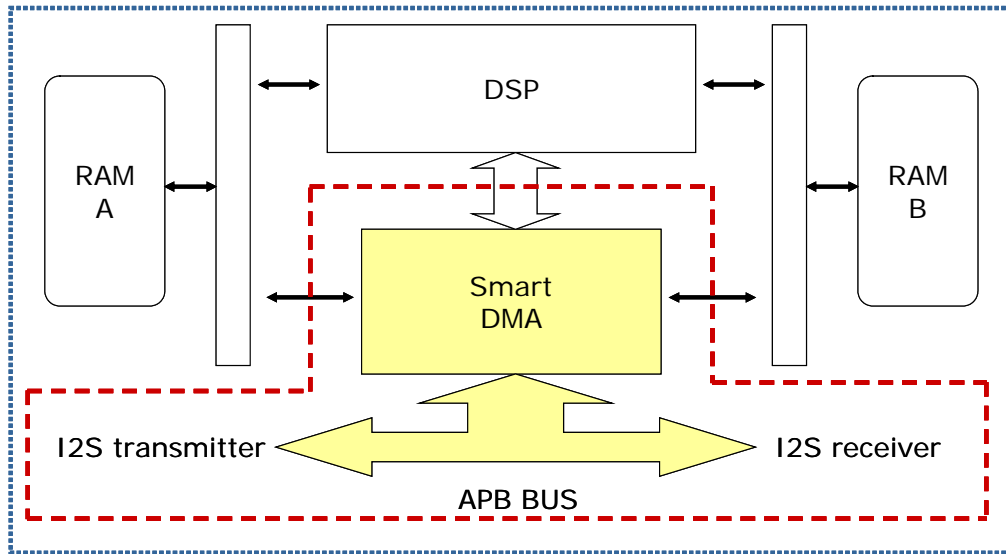


圖 3-3(d)：智慧型 DMA 與處理器及週邊的對應圖

除了共用匯流排外，為了設定智慧型 DMA 的動作，處理器必須存取智慧型 DMA 的暫存器。整合兩個 IP 的方式有兩種，第一，是從硬體電路上直接支援，有對應的指令解碼，存取智慧型 DMA 暫存器。第二，可以採用記憶體對映的方式，更動電路較小，也不需增加額外的指令集，如下圖：

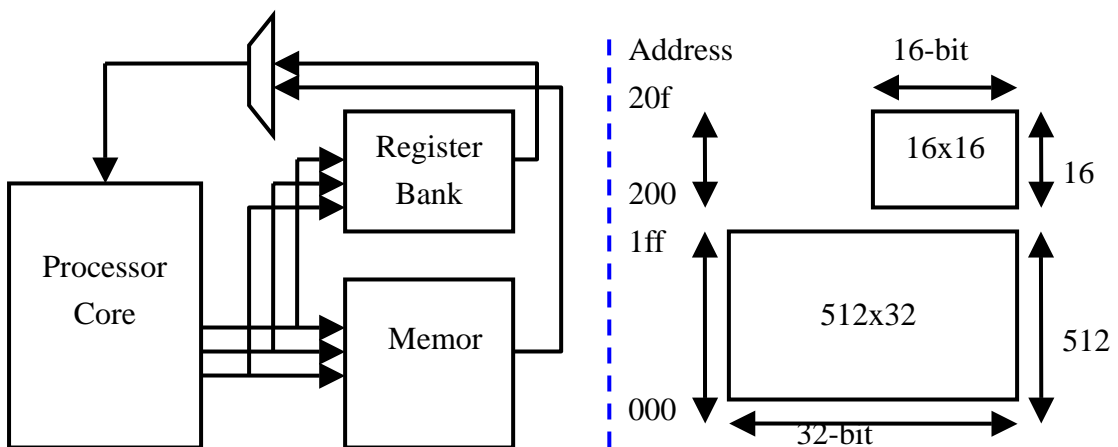


圖 3-3(e)：VLIW 處理器存取智慧型 DMA 暫存器-記憶體對映

3.5 結語

本章節介紹了主從式快取記憶體控制器的驗證平台：一個 VLIW 的多核嵌入式處理器的架構及一個可增進訊號處理運算能力及輔助記憶體及 I/O 溝通效能的 IP：智慧型直接記憶體存取控制器。VLIW 多核數位訊號處理器搭配上智慧型直接記憶體存取控制器可達到數位訊號處理器(DSP)的效能，而主從式快取記憶體將與 VLIW 多核數位訊號處理器核心及智慧型直接記憶體存取控制器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[35]，設計流程如下：

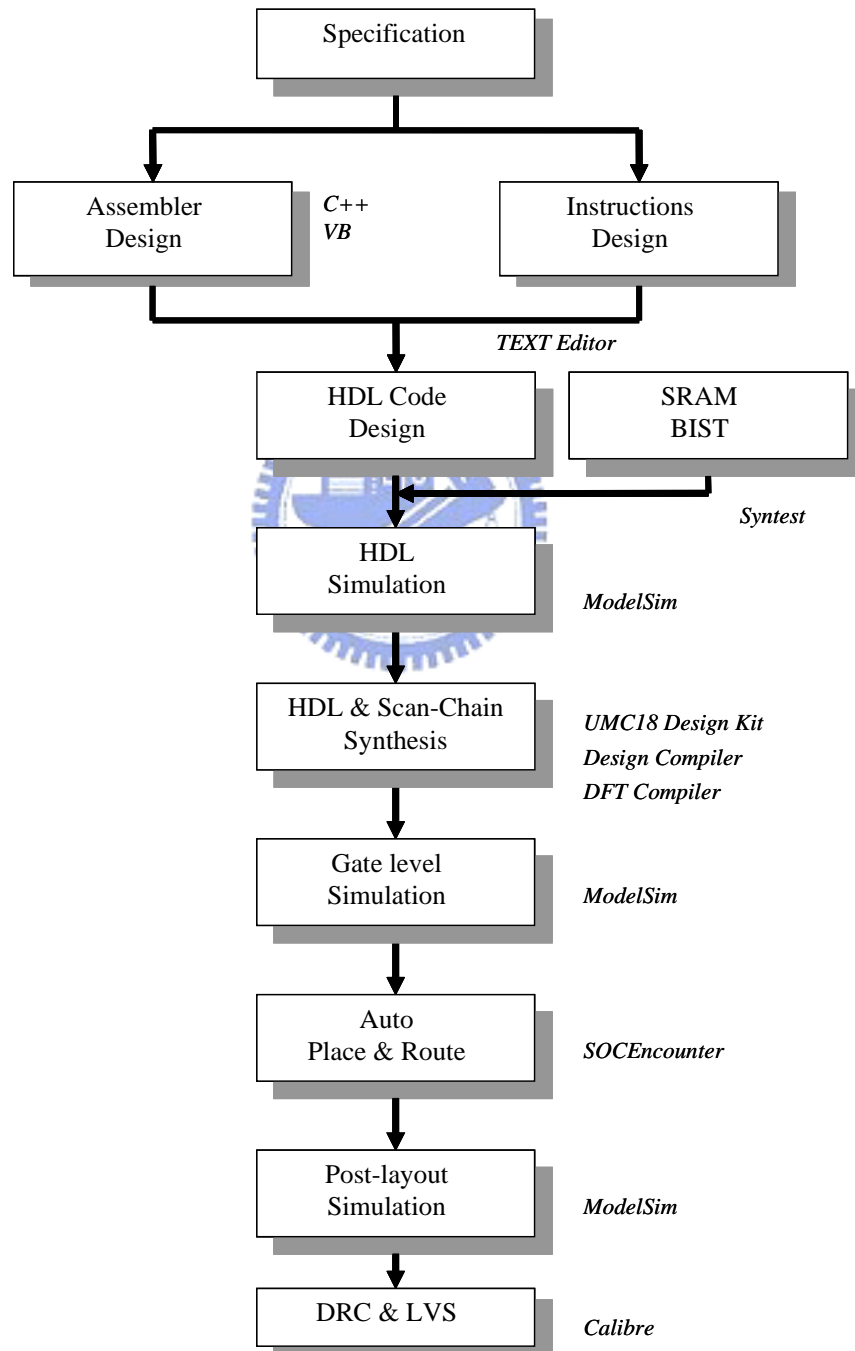


圖 4-1(a)：晶片設計流程

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 UMC 0.18um 製程。經由合成軟體 Design Compiler 合成後，其結果如下：

ITEM	Area (mm ²)	Timing	Fault coverage
Multi-Core with MS-Cache and SDMA	1173973	7.15 ns	98.65 %

表 4-1(a)：合成的結果



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 : SD297_V2

Technology : UMC 0.18um 1P6M CMOS

Package : 128 CQFP

Chip Size : 3.126× 3.126 mm²

Gate Count : 118K gate count

Power Dissipation : ~400mW

Max. Frequency : 135MHz (7.40 ns)

使用Prime Power量測功率，跑驗證功能的測試程式，得到平均消耗功率約400mW。圖4-1(b)為佈局圖，圖4-1(c)為打線圖，圖4-1(d)為對應的腳位圖，以128-CQFP的方式包裝。

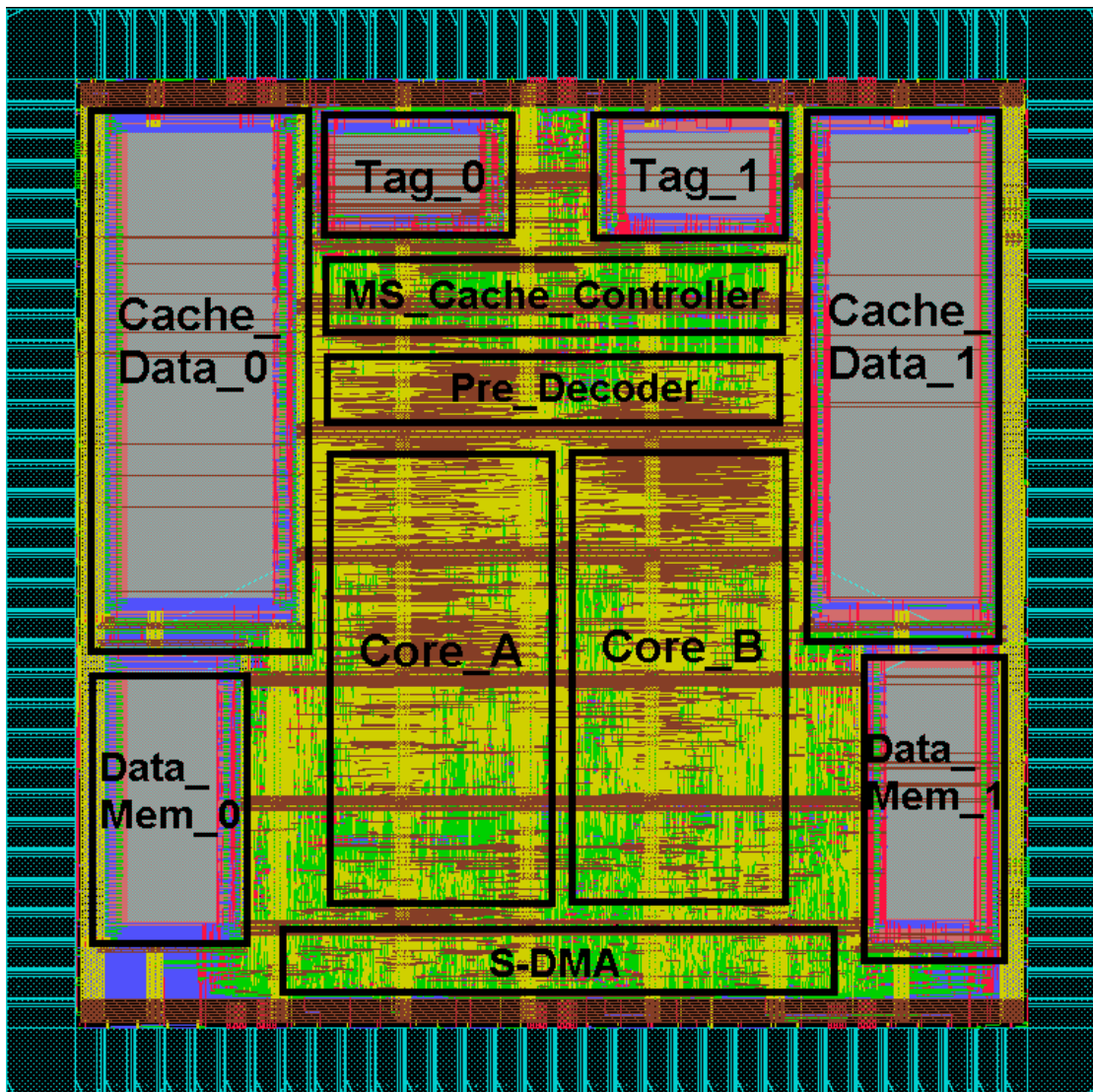


圖 4-1(b)：晶片佈局圖

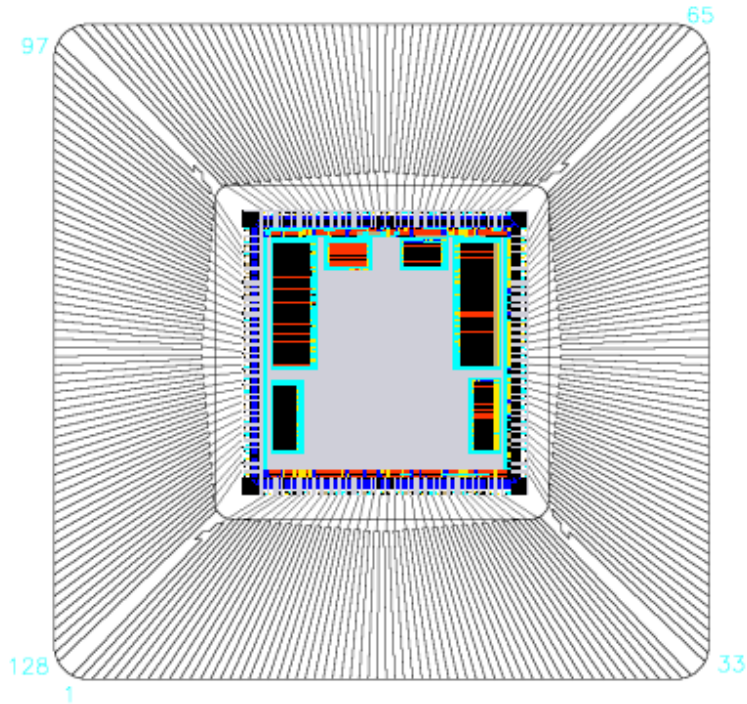


圖 4-1(c)：打線圖

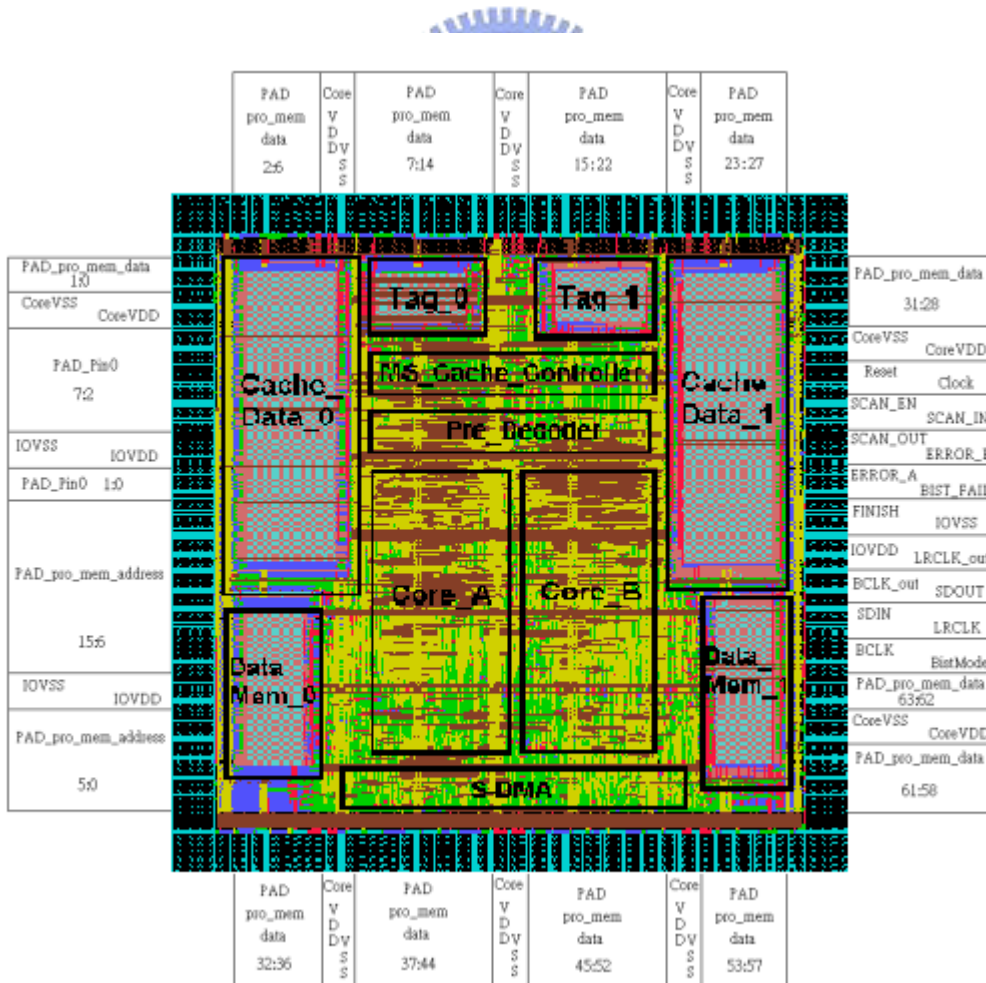


圖 4-1(d)：腳位圖

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。

晶片設計規格如表 4-1(b)。

Technology	Description
Process	UMC 0.18 μ m 1P6M Mixed Signal
Architecture	VLIW 9-stage pipeline
Synthesis	Synopsys Design Compiler
Gate Count	118K
Embedded Memory	RAM(512x32)x2 RAM(1024x64)x2 RAM(1024x6)x2
Die size	3.1 \times 3.1 mm ²
Supply	1.8V/3.3V \pm 10%
Input Delay Time	Max 0.714ns/ Min 0.543ns
Power consumption	400mW
Operating Frequency	125MHz
Post-Sim Speed	135MHz

表 4-1(b)：晶片設計規格

4.2 測試驗證

4.2.1 餘弦轉換 (Discrete Cosine Transform)

採用定點數平移 13-bit 做 36 點 DCT-2 運算，比較使用主從式快取記憶體控制器及未使用的結果是否一致以驗證該快取記憶體控制器的可靠性。結果如圖 4-2(a)，共有 36 筆資料輸出。其數學式如下：

1-D 36-point DCT-2

$$X[k] = \sqrt{\frac{2}{N}} \beta[k] \sum_{n=0}^{N-1} x[n] \cos\left(\frac{\pi k(2n+1)}{2N}\right), 0 \leq k \leq N-1 \quad \beta[k] = \begin{cases} 1/\sqrt{2} & \dots\dots\dots k=0 \\ 1 & \dots\dots\dots 1 \leq k \leq N-1 \end{cases}$$

輸入 X :



Columns 1 through 18

1 1 1 1 2 2 2 2 3 3 3 3 4 4 4 4 5 5

Columns 19 through 36

5 5 6 6 6 6 7 7 7 7 18 18 18 18 -100 -100 100 100

[71]	71	5917
[70]	70	-1704
[69]	69	7092
[68]	68	-14485
[67]	67	33029
[66]	66	-61415
[65]	65	103246
[64]	64	-154031
[63]	63	209000
[62]	62	-260075
[61]	61	298022
[60]	60	-317656
[59]	59	312576
[58]	58	-285519
[57]	57	234471
[56]	56	-170299
[55]	55	83082
[54]	54	0
[53]	53	-90493
[52]	52	202875
[51]	51	-305511
[50]	50	407942
[49]	49	-490476
[48]	48	550088
[47]	47	-572744
[46]	46	557738
[45]	45	-504600
[44]	44	422795
[43]	43	-327352
[42]	42	229330
[41]	41	-149119
[40]	40	82295
[39]	39	-54265
[38]	38	19501
[37]	37	-135562
[36]	36	251160

圖 4-2(a)：DCT-2 Post-layout Simulation 的運算結果

4.2.2 快速傅立葉轉換(FFT)

底下驗證了 512 點 FFT 結果，列出前 32 點，分別為定點化 C 程式模擬與 Post-sim 結果，如圖 4-2(b)，驗證無誤。

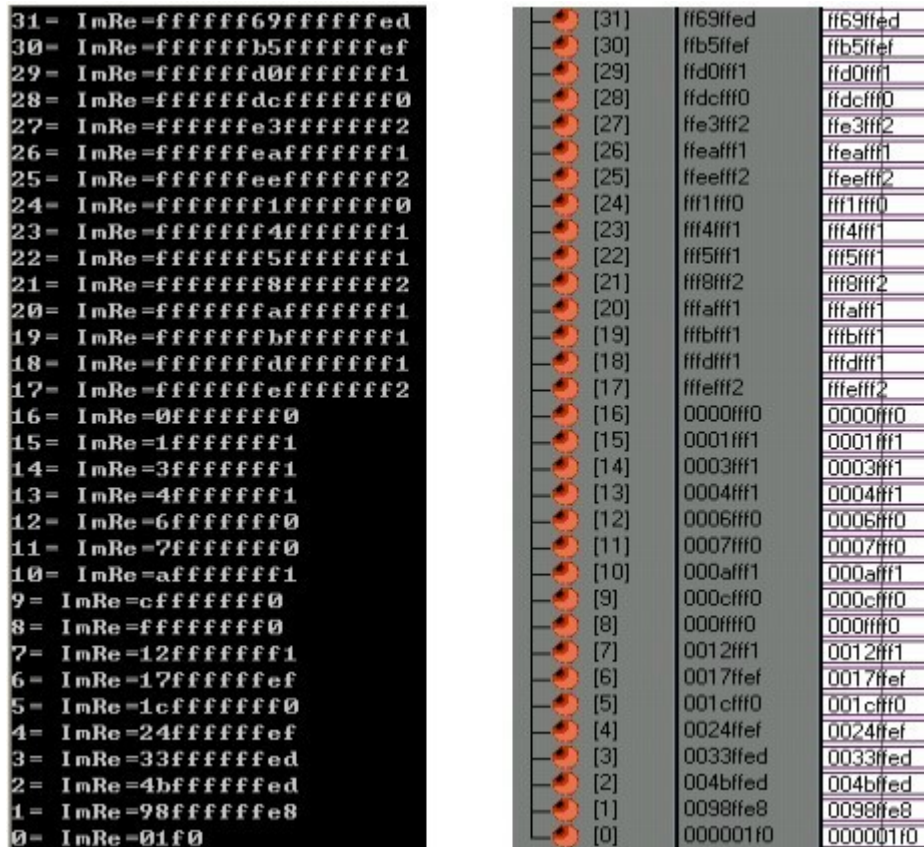


圖 4-2(b)：512 點 FFT 前 32 點的運算結果

4.2.3 量化線性預估係數(LPC)

驗證量化 Linear prediction 係數採用 Multi-stage Vector Quantization 化簡編碼簿的大小，並使用 M-L 搜尋法加強 MSVQ 的精確度。圖 4-2(c) 為定點化 C 程式的模擬結果，與圖 4-2(d)相比，驗證無誤。

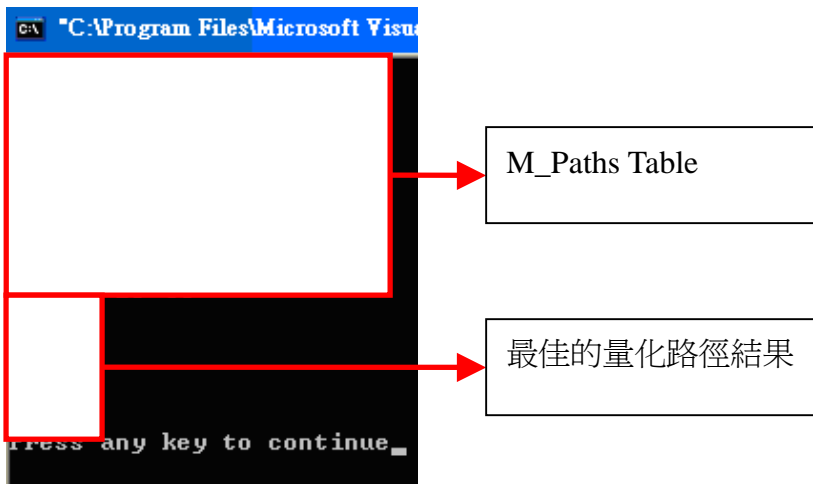


圖 4-2(c)：多級向量量化 C 程式執行結果

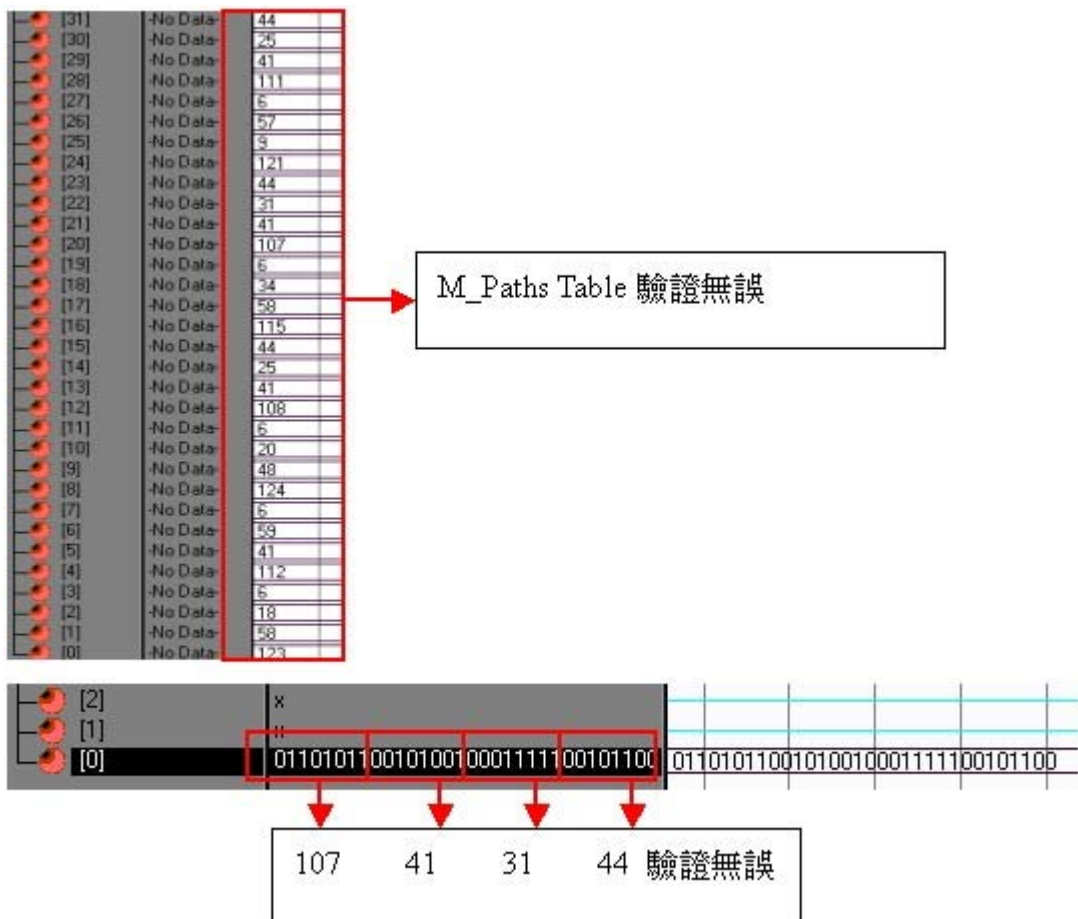


圖 4-2(d)：多級向量量化 Post-sim 結果

4.3 效能比較

如表 4-3(b)，以本論文設計的 VLIW 處理器，比起其他研究的數位訊號處理器 (DSP)，複雜度、成本都相當低。

Processor	Oloffsson ISSCC2002	Agarwala ISSCC2002	TI C55X	Proposed VLIW Design
S-P –Com MAC	1/4 Cycle	1/4 Cycle	2 Cycle	1 Cycle
S-P –Real MAC	1/8 Cycle	1/8 Cycle	1/2 Cycle	1/2 Cycle
Tech	.13 8M	.13 6M	-	.18 6M
Area(mm ²)	10 x 10	8.5 x 8.5	-	3.1 x 3.1
Power(mw)	1000	718	-	400

表 4-3(a)：與其他 DSP 的規格比較表

VLIW 處理器搭配智慧型 DMA 及主從式快取記憶體控制器後，其效能已追上一般的數位訊號處理器，而成本遠低於數位訊號處理器。如表 4-3(b)。

Item	50-taps FIR, 100 samples	50-taps complex FIR, 100 samples	Die size(mm ²)
Proposed VLIW Design	11200 (cycles)	24000 (cycles)	3.1x3.1
A 32-b RISC/DSP Microprocessor	12200 (cycles)	22000 (cycles)	3.4x3.4
TI-s C60	5000 (cycles)	20000 (cycles)	3.9x3.9

表 4-3(b)：與 DSP 運算效能比較表



第五章 結論

利用智慧型 DMA+VLIW 核心達到一個具有數位訊號處理器能力的核心，在搭配針對多媒體應用的函式及迴圈運算所造成的 cache miss 具有明顯改善效果的主從式快取記憶體控制器，此 SOC 晶片核心非常適合搭配上各種多媒體應用的介面使用。

本論文將主從式快取記憶體控制器使用在指令快取記憶體控制上，一個延伸主題是將主從式快取記憶體控制器使用在資料快取記憶體上對處理器效能的提升性。由於程式運行時，函式的呼叫返回也會對不同函式裡的區域變數在資料快取記憶體中發生取代現象，因此發生 cache miss，此現象與指令快取記憶體中的情形十分類似，因此推斷將主從式快取記憶體控制器應用在資料快取記憶體的 control 上應能對處理器的效能作有效的提升。

本論文實作的處理器已在國家晶片中心下線成功，並驗證無誤，未來此顆晶片可用做低成本的數位訊號處理器來使用，也可以 IP 的方式，將系統整合起來，成為一個 SOC 的系統。

參考文獻

- [1] Serene Banerjee, Humid R. Sheikh, Lizy K. John, Brian L. Evans, and Alan C. Bovik, “VLIW DSP vs. Superscalar Implementation of a Baseline H.263 Video Encoder,” *IEEE Press*, pp. 318-420, 2000.
- [2] Alberto Ferreira de Souza, Peter Rounce, “On the Scheduling Algorithm of the Dynamically Trace Scheduled VLIW Architecture”, *IPDPS MORGAN*, pp. 390-482, May, 2000.
- [3] T. Nakra, R. Gupta, and SOFFA, M. L. Soffa "Value Prediction in VLIW Machines", *ISCA-26*, May, 1999.
- [4] Sung-Hyun Jee, K. Palaniappan, Dynamically scheduling VLIW instructions with dependency informationn, *IEEE 6th Workshop on Interaction Between Compilers and Computer Architectures*, 15-23, February 2002.
- [5] Monica D. Lam , Edward E. Rothberg , Michael E. Wolf, “The cache performance and optimizations of blocked algorithms”, *Proceedings of the Fourth international conference on Architectural support for programming languages and operating systems*, p.63-74, April 08-11,1991.
- [6] Vijay K. Madiseti,“VLSI Digital Signal Processors: An Introduction to Rapid Prototyping and Design Synthesis”, *IEEE Press*, 1995.
- [7] Luca Breveglieri and Luigi Dadda, “A VLSI inner product macrocell,” *IEEE Trans. on VLSI*, vol. 6, no. 2, pp. 292-298, 1998.
- [8] ARM Ltd, AMBA Specification, rev. 2.0, <http://www.arm.com>, 1999.
- [9] John L. Hennessy and David A. Patterson, *Computer Architecture*, 3rd Edition, Morgan Kaufmann, 2003.
- [10] John L. Hennessy and David A. Patterson, *Computer Organization & Design : The Hardware / Software Interface*, 2nd Edition, Morgan Kaufmann

Publishers, pp 272-430, 1998.

- [11] Andrea Bona, Mariagiovanna Sami, Donatella Sciuto, Vittorio Zaccaria, Cristina Silvano, Roberto Zafalon: “Energy estimation and optimization of embedded VLIW processors based on instruction clustering”, *DAC*, 2002.
- [12] Domenico Barretta, William Fornaciari, Mariagiovanna Sami, Danilo Pau: “SIMD Extension to VLIW Multicluster Processors for Embedded Applications”, *ICCD*, 2002.
- [13] Domenico Barretta, Gianluca Palermo, Mariagiovanna Sami, Roberto Zafalon: “Energy/Performance Evaluation of the Multithreaded Extension of a Multicluster VLIW Processor”, *CAMP*, 2005.
- [14] Bill S.-H.Kwan, Bruce F.Cockburn, and Duncan G. Elliott, “Implementation of DSP-RAM: architecture for parallel digital signal processing in memory” Proceedings of *IEEE Canadian Conference on Electrical and Computer Engineering*, pp. 341-346, 2001.
- [15] Dave Comiskey, Sanjive Agarwala, and Charles Fuoco, “A scalable high-performance DMA architecture for DSP application” Proceedings of the *IEEE International Conference on Computer Design (ICCD)*, pp. 414-417, 2000.
- [16] Matteo Monchiero, Gianluca Palermo, Mariagiovanna Sami, Cristina Silvano, Vittorio Zaccaria, Roberto Zafalon: “Low-power branch prediction techniques for VLIW architectures: a compiler-hints based approach” Proceedings of the *IEEE Integration 38(3)*, pp 515-524, 2005.
- [17] C. M. Yuen, K. F. Tsang, and W. H. Chan, “Direct memory access frequency synthesizer for channel efficiency improvement in frequency hopping communication” Proceedings of *IEEE International Symposium on Circuits and Systems (ISCAS 2000)*, pp.485-488, 2002.

- [18] D. Stiliadi and A. Varma, "Selective Victim Caching: A Method to Improve the Performance of Direct Mapped Caches" Proceeding of the 27th Hawaii International Symposium on System Science, January, 1994.
- [19] Rui Min, Wen-Ben Jone, Yiming Hu: Location cache: a low-power L2 cache system, *ISLPED*, 2004.
- [20] D. Stiliadis and A. Varma, "Selective Victim Caching: A. Method to Improve the Performance of Direct-Mapped. Caches" Proceeding of *IEEE Transactions on Computers*, vol.46, no.5, pp. 603–610, May, 1997.
- [21] Kenneth M. Wilson , Kunle Olukotun, "High Bandwidth On-Chip Cache Design", Proceeding of *IEEE Transactions on Computers*, v.50 n.4, p.292-307, April, 2001.
- [22] E. Speight, et al., "Adaptive Mechanisms and Policies for Managing Cache Hierarchies in Chip Multiprocessors", *ISCA*, 2005.
- [23] Mattias O’Nils and Axel Jantsch, "Synthesis of DMA controllers from architecture independent descriptions of HW/SW communication protocols" Proceedings of *IEEE International Conference on VLSI Design (ICVD)*, 1999.
- [24] Alan V. Oppenheim, Ronald W. Schafer, and John R. Buck, "Discrete Time Signal Processing" 2nd Edition, Prentice Hall, pp 479-513, 1999.
- [25] John L. Hennessy and David A. Patterson, *Computer Architecture*, 3rd Edition, Morgan Kaufmann, pp 105-168, 2003.
- [26] CY Chang, JP Sheu, and SJ Chen, "Reducing Cache Conflicts by Multi-Level Cache Partitioning and Array Elements Mapping" The *IEEE Seventh International Conference on Parallel and Distributed Systems, ICPADS*, 2000.
- [27] MD Powell, S.-H. Yang, B. Falsafi, K. Roy, and TN Vijaykumar, "Reducing leakage in a high-performance deep-submicron instruction cache" Proceedings of *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 9(1):77

- 89, February, 2001.

- [28] John L. Hennessy and David A. Patterson, *Computer Organization & Design : The Hardware / Software Interface*, 2nd Edition, Morgan Kaufmann Publishers, pp 615-641, 1998.
- [29] Bill S.-H.Kwan, Bruce F.Cockburn, and Duncan G. Elliott, "Implementation of DSP-RAM: architecture for parallel digital signal processing in memory" *Proceedings of IEEE Canadian Conference on Electrical and Computer Engineering*, pp. 341-346, 2001.
- [30] Prateek Pujara, Aneesh Aggarwal: "Restrictive Compression Techniques to Increase Level 1 Cache Capacity" *Proceedings of ICCD*, 2005.
- [31] Steve Fuber, *ARM System-on-Chip Architecture*, 2nd edition, Addison-Wesley Professional, pp 257-418, 2000.
- [32] Hans-Joachim Stolberg, Mladen Berekovic, Lars Friebe, Sören Moch, Mark Bernd Kulaczewski and Peter Pirsch, "HiBRID-SoC: A Multi-Core System-on-Chip Architecture for Multimedia Signal Processing Applications," *Proceedings of International Conference on Very Large Scale Integration of System-on-Chip*, pp. 155-160, 2003.
- [33] HC Chen and JS Chiang, "Design of An. Adjustable-Way Set-Associative Cache" *Proceedings of IEEE Pacific Rim Conference on Communication, Computers and Signal Processing*, vol. 1, Aug, 2001.
- [34] Jae Sung Lee and 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.
- [35] Nian Shyang Chang, *Cell-Based IC Physical Design and Verification*, Chip Implementation Center, July, 2004.
- [36] Faraday, "Direct Memory Access Controller: Faraday/UMC FTDMA020" rev.

- 1.2, <http://www.faraday.com.tw>, 2003.
- [37] Yuan-Hao Huang, Hsi-Pin Ma, Ming-Luen Liou, and Tzi-Dar Chiueh, "A 1.1 G MAC/s Sub-Word-Parallel digital signal processor for wireless communication applications" *IEEE Journal of Solid-State Circuits*, vol. 39, no. 1, 2004.
- [38] L. D. Van, H. F. Luo, C. M. Wu, W. S. Hu, C. M. Huang, and W. C. Tsai, "A high-performance area-aware DSP processor architecture for video codecs" *Proceedings of IEEE Int. Conf. Multimedia and Expo (ICME)*, vol. 3, pp. 1499-1502, Jun, 2004.
- [39] 蘇育緯, "智慧型直接記憶體存取器設計"碩士論文, Jun, 2004.



附錄

A. 測試程式

Memory A to Memory A	Memory A to Memory B
<i>SDMAB 0,0_00000001_0000000</i>	<i>SDMAB 0,1_00000001_0000000</i>
<i>SDMAB 1,0_000000111111111</i>	<i>SDMAB 1,0_000000111111111</i>
<i>SDMAB 2,0_00000001_0000000</i>	<i>SDMAB 2,0_00000001_0000000</i>
<i>SDMAB 3,0_000000000000000</i>	<i>SDMAB 3,1_000000000000000</i>
<i>SDMAB 4,01_10_00_0100000000</i>	<i>SDMAB 14,00000000_00000000</i>
<i>SDMAB</i>	<i>SDMAB 4,01_10_00_1000000000</i>
<i>5,0_1_000_000_00_0_000_1_1</i>	<i>SDMAB 5,0_1_000_000_00_0_000_1_1</i>
<i>DMAOK</i>	<i>DMAOK</i>
<i>MOVRC R2,2</i>	<i>MOVRC R2,2</i>
<i>LABEL:A2A</i>	<i>LABEL:RAM_COPY</i>
<i>GDMA R3</i>	<i>GDMA R3</i>
<i>JNER R3,R2,A2A</i>	<i>JNER R3,R2,RAM_COPY</i>

Memory A to Memory B (Circular)	Memory A to Memory B (Mirror)
<i>SDMAB 0,0_00000111_0000000</i>	<i>SDMAB 0,1_00000111_0000000</i>
<i>SDMAB 1,0_000000000000000</i>	<i>SDMAB 1,0_000000000000000</i>
<i>SDMAB 2,0_00000001_0000000</i>	<i>SDMAB 2,0_00000001_0000000</i>
<i>SDMAB 3,1_000000000000000</i>	<i>SDMAB 3,1_000000111111111</i>
<i>SDMAB 14,00100000_00000000</i>	<i>SDMAB 14,00100000_00000000</i>
<i>SDMAB 4,10_10_00_1000000000</i>	<i>SDMAB 4,10_01_00_1000000000</i>

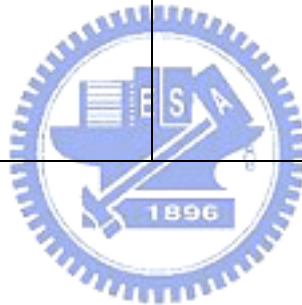
<i>SDMAB</i>	<i>SDMAB 5,0_1_000_000_00_0_000_1_1</i>
<i>5,0_1_000_000_00_0_000_1_1</i>	<i>DMAOK</i>
<i>DMAOK</i>	<i>MOVRC R2,2</i>
<i>MOVRC R2,2</i>	<i>LABEL:A2B_mirror</i>
<i>LABEL:A2B_circular</i>	<i>GDMA R3</i>
<i>GDMA R3</i>	<i>JNER R3,R2,A2B_mirror</i>
<i>JNER R3,R2,A2B_circular</i>	

Inner product	Convolution
<i>SDMAB 0,0_00000001_0000000</i>	<i>SDMAB 0,0_00000001_0000000</i>
<i>SDMAB 1,0_0000000000000000</i>	<i>SDMAB 1,0_0000000000000000</i>
<i>SDMAB 6,0_00000001_0000000</i>	<i>SDMAB 6,0_00000001_0000000</i>
<i>SDMAB 7,1_0000000000000000</i>	<i>SDMAB 7,1_000000111111111</i>
<i>SDMAB 4,10_00_00_1000000000</i>	<i>SDMAB 4,10_00_00_1000000000</i>
<i>SDMAB 10,10_00_00_1000000000</i>	<i>SDMAB 10,01_00_00_1000000000</i>
<i>SDMAB 14,00000000_00000000</i>	<i>SDMAB 5,0_1_000_000_00_0_001_1_1</i>
<i>SDMAB 15,00000000_00000000</i>	<i>SDMAB</i>
<i>SDMAB</i>	<i>11,0_1_000_000_00_0_001_1_1</i>
<i>5,0_1_000_000_00_0_001_1_1</i>	<i>DMAOK</i>
<i>SDMAB</i>	<i>MOVRC R2,0</i>
<i>11,0_1_000_000_00_0_001_1_1</i>	<i>LABEL:CONVOLUTION</i>
<i>DMAOK</i>	<i>GDMA R3</i>
<i>MOVRC R2,0</i>	<i>JNER R3,R2,CONVOLUTION</i>
<i>LABEL:Inner_product</i>	

<i>GDMA R3</i> <i>JNER R3,R2,Inner_product</i>	
DCT0	DCT4
<i>SDMAB 0,0_00000001_0000000</i>	<i>SDMAB 0,0_00000001_0000000</i>
<i>SDMAB 1,0_0000000000000000</i>	<i>SDMAB 1,0_0000000000000000</i>
<i>SDMAB 6,1_00000000_0000000</i>	<i>SDMAB 6,1_00001000_0000100</i>
<i>SDMAB 7,1_0000000000000000</i>	<i>SDMAB 7,1_0000000000000100</i>
<i>SDMAB 4,10_00_00_0000100100</i>	<i>SDMAB 4,10_00_00_0000100100</i>
<i>SDMAB 10,10_00_00_0000100100</i>	<i>SDMAB 10,10_00_00_0000100100</i>
<i>SDMAB 14,00000000_00000000</i>	<i>SDMAB 15,01001001_00000000</i>
<i>SDMAB</i>	<i>SDMAB 5,0_1_000_000_00_0_001_1_1</i>
<i>5,0_1_000_000_00_0_001_1_1</i>	<i>SDMAB 11,0_1_000_000_00_0_001_1_1</i>
<i>SDMAB</i>	<i>DMAOK</i>
<i>11,0_1_000_000_00_0_001_1_1</i>	<i>MOVRC R2,0</i>
<i>DMAOK</i>	<i>LABEL:DCT4</i>
<i>MOVRC R2,0</i>	<i>GDMA R3</i>
<i>LABEL:DCT0</i>	<i>JNER R3,R2,DCT4</i>
<i>GDMA R3</i>	
<i>JNER R3,R2,DCT0</i>	

DWT_A0	DWT_A4
<i>SDMAB 15,00000101_00000000</i>	<i>SDMAB 1,0_0000000000000000</i>
<i>SDMAB 0,0_00000001_0000000</i>	<i>SDMAB 7,1_000000010000000</i>
<i>SDMAB 1,0_000000111111000</i>	<i>SDMAB 4,10_00_00_0000001001</i>

<i>SDMAB 6,1_00000001_0000000</i>	<i>SDMAB 10,10_00_00_0000001001</i>
<i>SDMAB 7,1_000000010000000</i>	<i>SDMAB 5,0_1_000_000_00_0_001_1_1</i>
<i>SDMAB 4,10_00_00_0000001001</i>	<i>SDMAB</i>
<i>SDMAB 10,10_00_00_0000001001</i>	<i>11,0_1_000_000_00_0_001_1_1</i>
<i>SDMAB</i>	<i>DMAOK</i>
<i>5,0_1_000_000_00_0_001_1_1</i>	<i>MOVRC R2,0</i>
<i>SDMAB</i>	<i>LABEL:DWT_A4</i>
<i>11,0_1_000_000_00_0_001_1_1</i>	<i>GDMA R3</i>
<i>DMAOK</i>	<i>JNER R3,R2,DWT_A4</i>
<i>MOVRC R2,0</i>	
<i>LABEL:DWT_A0</i>	
<i>GDMA R3</i>	
<i>JNER R3,R2,DWT_A0</i>	



<i>DWT_D0</i>	<i>DWT_D6</i>
<i>SDMAB 15,00000100_00000000</i>	<i>SDMAB 1,0_000000000000111</i>
<i>SDMAB 1,0_000000111111011</i>	<i>SDMAB 7,1_000000011000000</i>
<i>SDMAB 7,1_000000011000000</i>	<i>SDMAB 4,10_00_00_0000000111</i>
<i>SDMAB 4,10_00_00_0000000111</i>	<i>SDMAB 10,10_00_00_0000000111</i>
<i>SDMAB 10,10_00_00_0000000111</i>	<i>SDMAB 5,0_1_000_000_00_0_001_1_1</i>
<i>SDMAB</i>	<i>SDMAB</i>
<i>5,0_1_000_000_00_0_001_1_1</i>	<i>11,0_1_000_000_00_0_001_1_1</i>
<i>SDMAB</i>	<i>DMAOK</i>
<i>11,0_1_000_000_00_0_001_1_1</i>	<i>MOVRC R2,0</i>

<i>DMAOK</i>	<i>LABEL:DWT_D6</i>
<i>MOVRC R2,0</i>	<i>GDMA R3</i>
<i>LABEL:DWT_D0</i>	<i>JNER R3,R2,DWT_D6</i>
<i>GDMA R3</i>	
<i>JNER R3,R2,DWT_D0</i>	



B. 佈局驗證結果說明

1. DRC

```
-----  
--- RULECHECK RESULTS STATISTICS (BY CELL)  
---  
-----  
--- SUMMARY  
---  
TOTAL CPU Time:           382  
TOTAL REAL Time:         392  
TOTAL Original Layer Geometries: 307776 (2422821)  
TOTAL DRC RuleChecks Executed:  386  
TOTAL DRC Results Generated:   0 (0)
```

DRC 驗證無誤

2. LVS



OVERALL COMPARISON RESULTS

```
      #          #####  
      #          #          #          *   *  
#     #          #          #          |  
#     #          #          #          \___/  
      #          #####
```

LVS 驗證無誤

C. CIC 下線報告書(A)

1. 晶片概述：

- 1-1. 專題名稱：具有Smart -DMA之VLIW架構多媒體訊號處理器
- 1-2. Top Cell 名稱：SD297
- 1-3. 使用 library 名稱：
 CIC_CBDK35
 CIC_CBDK25
 v CIC_CBDK18
版本： v2
- 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：400mW
- 1-7. 晶片面積：2375 X 2372

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？ No
- 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 (手動修過)
- 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？ Yes

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 2 , ROM 0
Memory大小：512x32 (Word x bit)
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 No，BIST controller數量 2

3-4. Scan Chain Information

Flip-Flop 共有多少個？ 4970

Scan chain 的數量共有多少條？ 1

Scan chain length (Max.) ? 137781.929

3-5. Uncollapsed fault coverage是否超過 90% ? Yes，為多少？ 98.45%

ATPG pattern的數目為多少？ 983

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、 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. 是否上加 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？ IMS 測試機台

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: 蘇育緯/周經翔/鍾仁峰

指導教授簽名: 林進燈



D. CIC 下線報告書(B)

1. 晶片概述：

專題名稱：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

1-1. Top Cell 名稱：SD297_V2

1-2. 使用 library 名稱：

CIC_CBDK35

CIC_CBDK25

CIC_CBDK18

版本：v2

1-3. 是否使用CIC提供之Memory？ Yes

1-4. 工作頻率：135 MHz

1-5. 功率消耗：400mW

1-6. 晶片面積：3126 X 3126

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？ No

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？ Yes

3. 可測試性設計(前瞻性晶片必填)：

3-1. 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 6 , ROM 0

Memory大小: 512x32 x2 1024x64x2 1024x6x2

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 No , BIST controller數量 6

3-4. Scan Chain Information

Flip-Flop 共有多少個? 5165

Scan chain 的數量共有多少條? 1

Scan chain length (Max.) ? 143286.337

3-5. Uncollapsed fault coverage是否超過 90% ? Yes , 為多少? 98.45%

ATPG pattern的數目為多少? 1183

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. 是否上加 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? Yes

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

6-2. 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？ IMS 測試機台

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: 周經翔/鍾仁峰

指導教授簽名: 林進燈



E. Module I/O

Chip I/O :

Pin Name	I/O	Width	Function
Global			
CLK	I	1	System Clock
Reset	I	1	Synchronous Reset
Scan Chain			
Scan in	I	1	Scan Chain Input
Scan en	I	1	Scan Chain Enable
Scan out	O	1	Scan Chain Output
Memory BIST			
BistMode	I	1	BIST Mode Select
BistFail	O	1	BIST Failure
Finish	O	1	BIST Finish
ErrMap A	O	1	Error Map of Ram A
ErrMap B	O	1	Error Map of Ram B
Program Rom			
pro mem address w	O	16	Program Rom Address
pro mem data w	I	64	Program Rom Data
I2S interface			
BCLK	I	1	I2S Bit Clock
LRCLK	I	1	I2S Word Select Clock
SDIN	I	1	I2S Data Input
BCLK out	O	1	I2S Bit Clock Output
LRCLK out	O	1	I2S Word Select Clock Output
SDOUT	O	1	I2S Data Output

Processor Modules

PC Counter / Branch Protect Stage/Instruction Fetch Stage:

I/O Type	Pin Name	Width
Input	CLK	1

Input	Reset	1
Input	pc	16
Input	j bit	1
Input	jump done	1
Output	pc predec	16
Output	pro mem address	16

Cache Stage:

MS_Cache_Control :

I/O Type	Pin Name	Width
Input	CLK	1
Input	ICLK	1
Input	Reset	1
Input	PC	16
Input	Pro Mem Data	64
Input	BistMode	1
Output	Pro Mem Addr	16
Output	Instruction	64
Output	Hit	1
Output	Ready	1
Output	Ready Tag O	1
Output	Load Finish	1
Output	Miss PC	16
Output	BistFail	1
Output	Finish	1

MS_Tag_Control :

I/O Type	Pin Name	Width
Input	CLK	1
Input	ICLK	1
Input	Reset	1
Input	PC Tag	16
Input	Tag D 0	64

Input	Tag D 1	64
Input	Tag A 0	6
Input	Tag A 1	6
Input	Tag WEN 0	1
Input	Tag WEN 1	1
Input	Ready Tag	1
Output	Cache WEN 0	1
Output	Cache WEN 1	1
Output	Cache D 0	64
Output	Cache D 1	64
Output	Cache A 0	10
Output	Cache A 1	10
Output	Hit	1
Output	Miss PC	16
Output	Sel	1
Output	Ready Tag O	1
Output	BistFail	1
Output	Finish	1



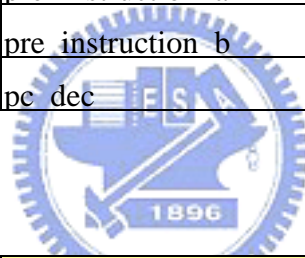
MS_Data_Control :

I/O Type	Pin Name	Width
Input	CLK	1
Input	ICLK	1
Input	Reset	1
Input	Cache_D_0	64
Input	Cache_D_1	64
Input	Cache_A_0	10
Input	Cache_A_1	10
Input	Cache_WEN_0	1
Input	Cache_WEN_1	1
Input	Hit	1
Input	Sel	1
Input	BistMode	1
Output	Instruction	64

Output	BistFail	1
Output	Finish	1

Instruction Predecoder Stage:

I/O Type	Pin Name	Width
Input	CLK	1
Input	Reset	1
Input	pc predec	16
Input	instruction	64
Input	j bit	1
Input	jump done	1
Output	op a	6
Output	op b	6
Output	pre instruction a	32
Output	pre instruction b	32
Output	pc dec	16



Instruction Decoder Stage:

I/O Type	Pin Name	Width
Input	CLK	1
Input	Reset	1
Input	instruction	32
Input	op	6
Input	pc dec	16
Input	j bit	1
Input	jump done	1
Output	rd	6
Output	direct	20
Output	op reg	6
Output	pc reg	16
Output	flag	2
Output	dec to reg	6
Output	address1	6

Output	address2	6
Output	DMA address	4

Register File Stage:

I/O Type	Pin Name	Width
Input	CLK	1
Input	Reset	1
Input	pc reg	16
Input	op	6
Input	rd	6
Input	flag	2
Input	direct	20
Input	address1	6
Input	address2	6
Input	address3	6
Input	reg w data	32
Input	Device Int	4
Input	int done	1
Input	dec to reg	6
Input	DMA address	4
Output	pc alu	16
Output	op alu	6
Output	rd alu	6
Output	addr1 alu	6
Output	addr2 alu	6
Output	flag alu	2
Output	direct alu	20
Output	reg r data1	32
Output	reg r data2	32
Output	reg to alu	31:0
Output	DMA address alu	3:0

ALU Stage:

I/O Type	Pin Name	Width
Input	CLK	1
Input	Reset	1
Input	pc	16
Input	op	6
Input	rd	6
Input	flag	2
Input	direct	20
Input	A	32
Input	B	32
Input	jump done	1
Input	mem output data	32
Input	alu work	1
Input	addr1 alu	6
Input	addr2 alu	6
Input	reg to alu	32
Input	Intn	2
Input	RegOut	32
Input	DMA address	4
Output	Cout	32
Output	C address	6
Output	i bit	1
Output	jump addr	32
Output	mem w address	9
Output	mem input data	32
Output	wren	1
Output	Pin0	16
Output	Pin1	16
Output	mem r address	9
Output	int done	1
Output	mgrant a	1
Output	mgrant b	1
Output	Rcen	1

Output	Rwen	1
Output	Raddr	4
Output	RegIn	32

Smart DMA Modules

Pin Name	I/O	Width	Function
Global			
CLK	I	1	System Clock
rstn	I	1	Synchronous Reset
Setting DMA			
mgrant0	I	1	Memory A Select
mgrant1	I	1	Memory B Select
rcen	I	1	Register Chip Enable
rwen	I	1	Register Write Enable
raddr	I	4	Register Address
RegIn	I	32	Register Data Input
RegOut	O	32	Register Data Output
Intn	O	2	Channel Controller Interrupt
Memory Interface			
cen0	O	1	Chip Enable of Memory A
oen0	O	1	Output Enable of Memory A
wen0	O	1	Write Enable of Memory A
a0	O	9	Address of Memory A
d0	O	32	Data Input of Memory A
q0	I	32	Data Output of Memory A
cen1	O	1	Chip Enable of Memory B
oen1	O	1	Output Enable of Memory B
wen1	O	1	Write Enable of Memory B
a1	O	9	Address of Memory B
d1	O	32	Data Input of Memory B
q1	I	32	Data Output of Memory B
I2S interface			
BCLK	I	1	I2S Bit Clock
LRCLK	I	1	I2S Word Select Clock

SDIN	I	1	I2S Data Input
BCLK out	O	1	I2S Bit Clock Output
LRCLK out	O	1	I2S Word Select Clock Output
SDOUT	O	1	I2S Data Output

