國 立 交 通 大 學

電機資訊學院電子與光電學程

碩 士 論 文

適用於異質性平台之低功率可程式化資料流設計

On the Design of a Low Power Programmable Datapath for Heterogeneous Computing Platform

研 究 生： 劉 建 良

指導教授： 劉 志 尉 教授

中華民國 九十四 年 一 月

上網授權書

受權書

適用於異質性平台之低功率可程式化資料流設計

On the Design of a Low Power Programmable Datapath for Heterogeneous Computing Platform

研 究 生: 劉 建 良　　Student : Chien-Liang Liu

指導教授: 劉 志 尉　　Advisor : Chih-Wei Liu

國 立 交 通 大 學

電機資訊學院電子與光電學程

碩 士 論 文

A Thesis

Submitted to Degree Program of Electrical Engineering Computer Science

College of Electrical Engineering and Computer Science

National Chiao Tung University

in Partial Fulfillment of the Requirements

for the Degree of

Master of Science

in

Electronics and Electro-Optical Engineering

June 2001

Hsinchu, Taiwan, Republic of China

中華民國 九十四 年　一 月

審定書

適用於異質性平台之低功率可程式化資料流設計

研究生： 劉建良　　指導教授： 劉志尉博士

國立交通大學

電機資訊學院 電子與光電學程

論文摘要

在硬體的使用效率和功率的考量上,在多媒體應用上的運算需求並非單一處理器系統所能負擔。因此，在多媒體 SOC 的設計中，常會外掛其他專門輔助處理數位信號的電路，如：加速器或資料流，來處理信號轉換的運算。在針對加速信號處理運算的問題上，本實驗室曾提出一個新式的加速器(DSP-lite)。它是一個以軟體控制來改變硬體資料流型態的計算引擎,利用微程式碼表格來控制資料的流向，用以降低硬體的複雜度。模擬的結果顯示，以工作時脈(execution cycles)來比較的話，使用幾乎同樣運算資源,DSP-lite 的資料流經最佳化過的運算元排程後，其運算的效率約為 ADSP-21xx 效能的 3 倍。而 DSP-lite 實作晶片的面積為 1.7x1.7mm^2、功率消耗：52mW。比起使用 DSP 微處理器來做加速硬體時，在運算效率與功率消耗方面都有大幅改善。

在電路的控制方面，DSP-lite 所使用的微程式碼未經編碼，因此所使用的微程式碼表格容量很大，且在重新更新資料路徑(reconfigure datapath)時必需做額外的工夫更新表格。因此，以 DSP-lite 的資料流觀念為藍本，本論文改變其中的控制機制以指令控制取代原來的微程式碼表格,重新規劃一個適於處理線性轉換運算問題的處理器：定義一組指令集架構,能把資料流向的控制和運算單元的運算分開,使運算單元能夠專注於讀入和產生資料數值，而不用理會該數值來源和去向；並採用分散式的暫存器檔，相較於傳統集中式暫存器檔的設計，在運算速度方面：做信號處理時，它必需多花 30%個時脈做資料交換的動作，但可使晶片面積縮小 75%，功率消耗減少 68%，大幅降低了運算單元的平行度對傳統暫存器檔在電路複雜度上的衝擊。以指令集寫成的程式比 DSP-lite 使用微程式表格可以減少 50%的記憶體用量，同時保有原本的效能。本論文利用此處理器核心完成了一連串 DSP 線性轉換的模擬。以實作的比較結果顯示，在加速 DSP 運算方面，它只使用相當於 TI C55x 一半的硬體資源，但卻有更好的運算效率。

最後，以 UMC 1P6M 0.18um CMOS 製程，實作此一處理器的晶片，其最高工作頻率可達 268MHz, 晶片面積為 0.6x0.6 mm^2，平均消耗功率為 37mW。

誌　　謝

在在職研究中，受到實驗室許多人的幫助才能順利取得學位，在此獻上無限的感激。

能夠完成本論文，首先要感謝指導教授任建葳老師和劉志尉老師。跟著老師學習不僅在專業知識方面獲益良多，還有更多論文寫作的方法及學術研究方面應有的嚴謹態度，首先致上最深的感謝。同時，感謝口試委員：鐘太郎教授、張錫嘉教授及闕河鳴教授。謝謝你們於百忙之中，撥冗參與論文口試及提供寶貴的論文指導意見。

感謝林泰吉學長，三年多來，在我研究工作方面不論在研究方向和研究工作中的諸多建議與指導協助。及 VLSI 實驗室的學弟們，佳憲、晉宏、丕承、至敏及士豪，謝謝你們在 CAD、工作站的使用上及面臨許多問題時的討論和各方面的支援。

最後，我要由衷的感謝親愛的父母及弟妹們，一路上的支持及關心，沒有你們的關懷就沒有今日的我。

謹將此篇論文獻給所有曾經支持我，協助我的人，衷心感謝、祝福你們。

建良

謹誌於 新竹

2005 年 1 月

目錄：

表目錄：

圖目錄：

第1章　簡介

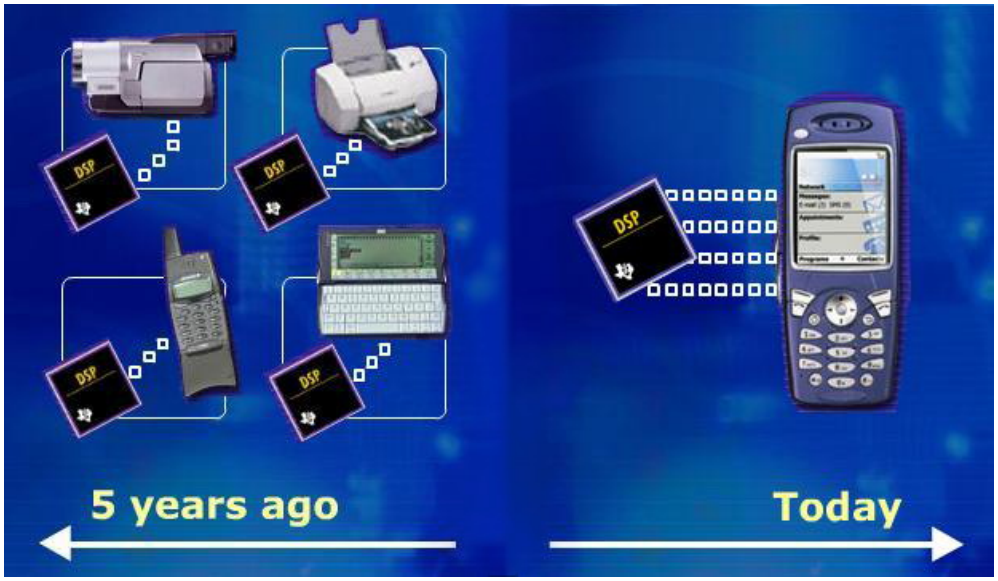


圖 1-1 DSP 於電子產品的應用驅勢

過去的電子產品，通常是各別的單一功能的產品，而現今個人消費性電子產品的發展趨勢，如圖 1-1 所示，朝向多樣化的功能整合的方向發展[5]。以目前處理器的發展而言，一般功能的 RISC 並無法負擔數位信號處理(Digital signal processing；簡稱 DSP)這麼大運算量的需求的應用[6]。因此，在產品的設計上，通常會使用外加的 ASIC 來處理這些 DSP 的運算。但是，ASIC 只能支援固定的運算，在硬體的利用上缺乏彈性。用於支援的功能需要多樣化整合的產品時，必需將多個 ASIC 整合於系統中，不但硬體驗証困難，對產品的開發時間和後續產品的開發都有不良影響。因此，在現今的 SOC 設計中，常使用可調整(configurable)或可程式化(programmable)的 DSP 的資料流 (DSP datapath)來加速 DSP 運算。如 TI(Texas Instruments Incorporated)所提出的 OMAP 處理器，就是使用一個 RISC 和另一個 DSP 微

處理器兩種不同特性的核心(kernel)來組成整個系統。利用 DSP 處理器可程式化的特性(programmable)來達到功能多樣化，減少硬體更新及驗証的風險，達到：加速開發時間(time to market；簡稱 TTM)、延長市場時間(time in market；簡稱 TIM)，以及減少後續產品的開發困難度。

1.1. 多核心系統設計

在多媒體或通訊系統上，依處理工作的性質，大致可以被劃分為兩種：以處理程式控制流程為主的任務(control oriented task)、和另一種需要快速處理資料轉換運算的任務(data-intensive task)。前者以處理計算機中一般性應用的工作為主，它必需能適時地回應由週邊設備(甚至使用者)所產生的中斷要求。此類工作常需要根據系統中硬體資源即時的使用情形和中斷要求情況，做動態的判斷與回應，如：決定處理中斷的優先順序、安排硬體資源去處理中斷要求…等等。這類任務資料的來源來自四面八方種類繁多，要做的回應動作也是五花八門，因此處理器必需掌握硬體資源的使用量，做出最佳的判斷。另一種任務則是於幕後提供足夠速度的資料轉換(Data transform)，使資料可以即時地展現或傳送。在實際應用上，信號處理都有一定的規格，如：通信的傳輸速率，影片的播放速度…等等。因此，這類任務的重點為：必需達到足夠的運算能力，否則來不及處理的信號將瞬間堆積如山。相對的，需要做這類處理的資料來源、種類和運算方式則相當固定，什麼樣來源的資料應做什麼樣的處理都可在事先預測及規劃。因此處理這類任務時並不需要太複雜的判斷，而必需專注於計算速度的展現，在規格的時限內處理完資料。

正因現今系統上，對處理這兩類的任務的效能的需求都同時上昇。因此，在系統的設計上無可避免的必需考慮如何整合這兩類工作的問題。在單一微處理器的整合方法中，可大概分成三種：(1)具有強化 DSP 運算的 RISC (DSP-enhanced RISC CPU)，(2)強化控制機能的 DSP 微處理器(control-enhanced DSP)，和(3)新式的混合型架構(blended architecture)[9]。

以強化 DSP 運算的 RISC 而言，如 MIPS 曾在其 RISC 上加入 DSP 運算中常用的乘加器(MAC)運算單元。雖然 MIPS 宣稱這樣的做法可以讓其 RISC 在信號處理方面加快 30%，但是比起相近等級的 DSP 處理器，它的速度還是慢了數倍之多。因此這樣的架構只能適用在 DSP 運算負荷不高的系統，如單一功能的印表機，影印機…等等，當 DSP 的需求增加時，為了達到規格的運算速度，這類處理器就只能以再加快操作頻率來加速，徒增大量的功率消耗。

第二種是強化控制機能的 DSP 處理器，DSP 處理器都具有寬位元的運算單元，當使用這些運算單元用來做單位元(bit-wide)或位元組(byte-wide)的處理和判斷時，明顯可以看出硬體上的浪費，而且即使強化了判斷的功能，這類 DSP 仍然缺乏高階作業系統的支援。因此這樣的作法也只能適用於一些不需高階 RTOS(real time operating system)的應用，如數位馬達控制(digital motion control system)。再者以目前 SOC 平台設計研究而言，還沒能找到一個架構能把這兩種任務有效率地整合在同一個系統(同一記憶體、週邊…等架構配置)內運作[8]，因此除了處理器核心本身的問題

外，整合型的系統的設計也是困難重重。所以，混合使用控制與 DSP 兩種處理器核心的異質性平台是目前較可以雙方面兼顧應用和效能的做法。

在異質性平台的架構中，圖 1-2 總合了一般 SOC 異質性平台架構的設計的做法：以 RISC 的子系統處理一般性控制的工作，以外掛的硬體加速器做信號處理，中間再以背景記憶體或少數溝通的網路連結起來。而在外掛的加速器方面，依其可程式化的程度由弱到強可概略分為：固定功能的 ASIC、可調整型的資料路徑、和可程式化的 DSP 處理器等等。

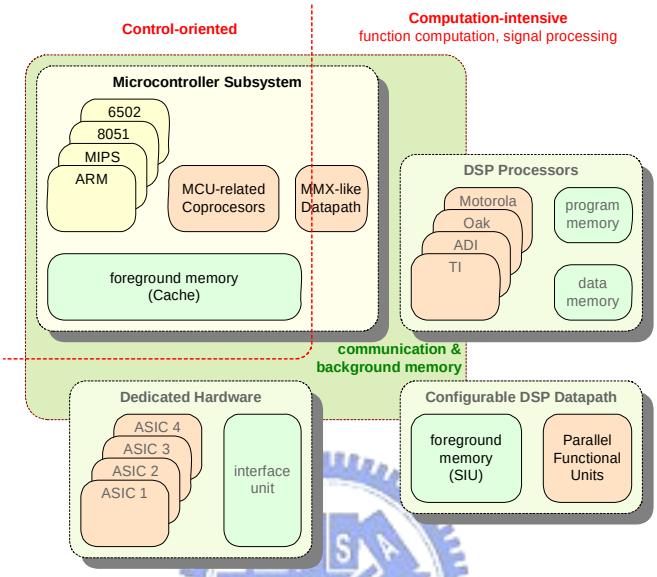


圖 1-2 異質性運算平台

當使用 ASIC 做加速時，無庸置疑的，ASIC 因為功能固定，因此對專屬功能而言可以做到硬體最省、效率最佳、功率最小的好處。但是其致命傷亦為功能固定，只要信號處理的資料流有任何更新或增減就要變更硬體，當系統支援多種信號處理時，還要把多個 ASICs 整合到系統上，這樣的做法，在每次產品更新時就必承受硬體更新的風險和重做硬體驗証的工夫。以需要大量 DSP 與功能多變的電子產品的開發來說，使用 ASIC 已經很難達到 TTM & TIM 的要求。

而另一個方法為使用 DSP 微處理器來加速。如圖 1-3 即為 TI 提出的雙處理器架構－使用另一個 DSP 處理器做信號處理的加速。圖中由 ARM 組成一個 MCU 子系統(subsystem)，連結週邊設備與使用者介面，負責整個系統中硬體資源的控管；而 DSP 微處理器所組成的 DSP 子系統則連接和處理來自 codec 和外部 DSP 記憶體的資料。使用另一個 DSP 處理器最大的好處為：可程式化。而且，所使用的 DSP 處理器亦為已通過獨立驗証的產品，因此可以縮短產品開發驗証的時間，也可使後續產品的開發容易，往往只需做軟體更新或少數硬體修改就可以讓同一個硬體延用至多個世代的產品。以硬體利用率的角度來看時，MCU 子系統負責所有系統中控制方面的任務，因此在 DSP 的子系統中，只需負責做信號處理的運算，並不需要處理額外控制方面的工作。但是，C54x 在原本的設計上是一個可以獨立工作(stand alone)的微處理器，當用在 DSP 的子系統時，原本為了獨立工作所設計的控制方面的電路就反而變成了多餘且重覆的設計了。因此使用外掛 DSP 處理器最直覺的缺點就是多餘硬體設計造成硬體和功率的浪費。

因此在這兩種方法中我們希望能找到一個較好妥協點－一個可以使用軟體控制的硬體資料流(software controlled hardware datapath)。軟體控制：可以達到可程式化的好處，減輕硬體更新的風險，增加 IP 的適用性；硬體資料流：可以減少直接使用 DSP 處理器時和 RISC 重複且多餘設計。

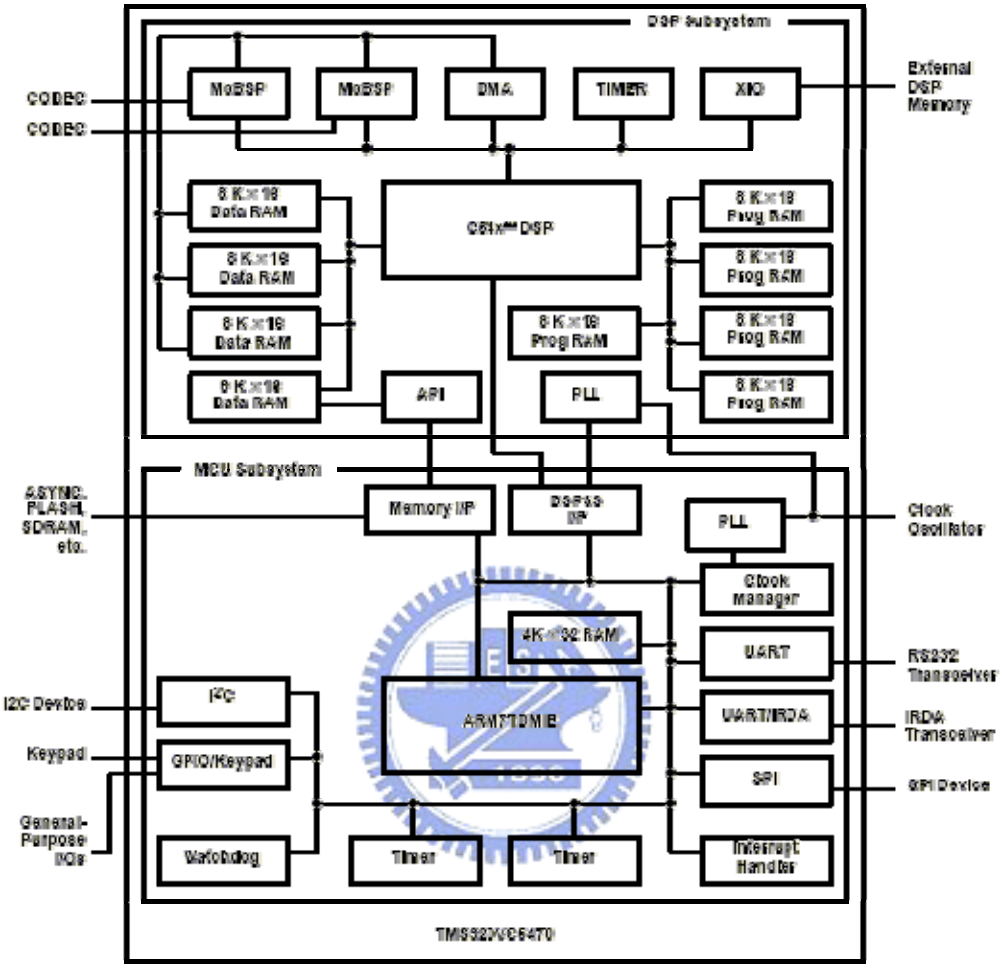


圖 1-3 TI TMS320 VC5470 雙處理器架構

1.2. 相關研究

在雙處理器的平台中，控制方面的工作可由 RISC 子系統統一控管，因此在 DSP 子系統中主要的工作就只剩信號處理的運算，需要控制的部分已經很少了。而我們實驗室小組其中的一個研究的方向即為針對，如 DSP 子系統這樣的系統特性下，對資料流做改善－減少重複的控制方面的硬體(redundancy removal)，讓硬體能更有效的被利用於信號資料的運算。

在這個研究方向的成果上：DSP-lite 為我們提出的第一個適用於 DSP 子系統的硬體加速器雛形。在資料流向的控制方面，它使用一種新的名為 SIU(stream interface unit)的資料暫存和流向控制的機制以其 DSG(data stream generator)全權掌控資料的流向，讓運算單元(function unit)能專注於資料的運算。以嚴格的分工使資料能更順暢的運算。在實驗模擬的結果顯示，SIU 的機制能更有效的利用運算單元，讓使用與 ADSP-21xx 同

樣運算單元資源的 SIU 資料路徑達到其約三倍的運算效能。此外在數值運算中，DSP-lite 引用一個新的輕量型算術－靜態浮點數(SFP)運算，以極小部分的運算單元修改，使 16 位元的運算達到 38.1dB 的 SNR，高出 16 位元的整數運算約 5dB。DSP-lite 以 UMC0.18 制程實作驗証結果為：晶片面積 1.7x1.7 mm^2 功率消耗為 52mW，最高工作頻率為 314MHz 用來做 DSP 加速器時會比直接使用 DSP 處理器在減少功率消耗上有數倍的改善[15]。

在資料路徑的控制部分，DSP-lite 以微程式碼表格(micro-code table)控制資料的流向，以每個時脈 128 位元的微程式碼發出(issue)四種運算單元的動作和控制 IO 的流向，因此 DSP-lite 在微程式部分的記憶體使用量較大。另外，在切換不同的運算路徑時，DSP-lite 必需花費額外的工夫做更新微程式表格的動作(將微指令從外部記憶體搬至內部程式記憶體)，才可以開始下一個不同的運算。因此，在使用上程式記憶體的使用量較大和切換時的更新表格的動作是其缺點。本論文引用 DSP-lite 中 SIU 資料路徑的觀念並使用支援 SFP 運算的運算單元，重新改寫一個 RISC-like 的處理器以做為信號處理加速器的核心。提出一組 VLIW(very long instruction word) RISC 的指令集，同樣可以把處理器中暫存器檔的架構規劃成 SIU DSG，並將四路(4-way) VLIW 指令長度縮減至 64 位元，減少約 50%的程式記憶體用量。以處理器指令分叉判斷和同步解碼的方式可以自然免除重新更新控制表格的工作。

另外，在處理器中使用之暫存器組織的選擇上，若以傳統的單一集中式的暫存器檔(centralized register file；簡稱 CRF)來支援多個運算單元平行運算時，會需要多個平行的埠(parallel IO port)來連結每個運算單元和暫存器檔。因此，隨著運算單元的個數呈線性比例增加時，暫存器檔的輸入輸出埠也會跟著增加，使得存取延遲(access delay)、面積(silicon area)，和功率消耗(power dissipation) 會隨著呈 1.5 次方、三次方、三次方的比例增加。在這樣的成長速度下，暫存器檔電路的複雜度(包括：延遲、面積、和功率)會很快的隨著運算單元的增加主宰整個晶片[11]。然而，這樣的複雜度可以靠著犧牲存取的便利性而達到若干改善。換言之，若限制某部分的運算單元都只有直接存取某特定部分暫存器的權利時，暫存器檔硬體的複雜度即可隨埠的數目或每個埠所能存取的暫存器的數目減少而下降。

本論文以結合分散式的暫存器檔和 SIU 資料流設計一個精簡型的 DSP 核心和適用於 SIU 靜態排程的指令集。這是一個試驗性質的架構，但在實作分析中，我們發現這樣的架構有效的簡化 DSP 子系統的硬體複雜度和提高計算效率。

1.3. 論文架構

在本論文中，我們以 SIU 的觀念，設計一組 VLIW 的指令集架構，嚴格地把資料流向和運算單元獨立開。在運作時，運算單元只需不停的接受並運算被送過來的資料，幾乎沒有需要做動態決定的運算(dynamic decision)，而資料的來源和流向會在指令中被送到正確的地方。另外本論文也將提出一個新的以資料流向為主及運算排程的程式寫法，不同於一般以流程或迴圈為主的程式寫法。經由實際執行的結果，我們的方法十分適

用於解決 DSP 的運算。

在第二章，將介紹 SIU 資料流和幾種現今常用的暫存器檔的組織(register organization)，並以實作分析的數據比較以集中式暫存器組織和分散式暫存器組織實現 SIU 資料流時，硬體複雜度和計算效能方面的比較。另外也將介紹本 DSP 所支援的輕量型算數方法，包函：整數，非條件式的區域型浮點數和靜態浮點數。

在第三章，將介紹控制 SIU(stream interface unit)資料產生器的方法、指令集架構以及用 SIU 資料產生器的方式產生程式方法。把平行處理的運算單元(function units)經由程式排程，可變成，類似 ASIC，有前後運算因果關係的資料流，提高硬體資源的使用率。

第四章會簡介硬體的實現方法、結果、和軟體的模擬效能，並和市面上的 DSP 做比較。

最後在第五章總結論文結果和未來的方向。

第2章　資料流設計

在現今通訊或多媒體系統的需求而言，所處理訊號的頻寬愈來愈大。相對地，運算量的需求也愈來愈多。單方面的增加處理器的時脈的效益並不足以應符這些應用的運算需求。因此，在 DSP 資料流的設計上常會使用多個運算加速器來做平行處理。

圖 2-1(a) 是傳統 ASIP(application specific instruction-set processor)處理 DSP 資料路徑的做法。大部分應用於信號處理的 VLSI 也大多以這樣的 ASIP 架構為藍圖來處理信號。圖中，每一個運算模組都有自己的 IO 規格，當運算的需求增加時，以目前 VLSI 技術可以輕易地在晶片中增加運算模組滿足計算能力。然而，相對要，把這麼多 IO 的規格全部交由匯流排和記憶體來安排時，複雜資料順序及重複的資料進出就會變成運算的瓶頸。當資料無法即時送達時，再強的計算能力 DSP 資料路徑都將面臨無值可算的窘境。因此，如圖 2-1 (b)所示，在匯流排和 DSP 資料流中間放一個規劃好的 SIU(Stream Interface Unit)，讓不同 IO 規格中改變資料排列方式的重排動作及 DSP 資料流中會被重複計算的資料做暫存動作都放在 SIU DSG(Data stream generator)中來完成[20]，這樣一方面可以讓資料在記憶體進出的順序變得簡單，這樣就不需要使用複雜的定址方式，另一方面可以將會被反覆運算的資料保留在 SIU 與運算模組之中，做完一定的程度之後才寫回記憶體，以減少產本資料每進出運算模組一次就必需來回一次記憶體的流量。

因此把 SIU 的觀念應用在微處理器資料流的設計上時，運算模組和 SIU 之間的關係就如同運算單元和暫存器檔一樣。在本章將介紹 SIU 的運作原理、幾種暫存器組織的分類及利用實做的方法，試用集中式和分散式暫存器檔的結果分析。

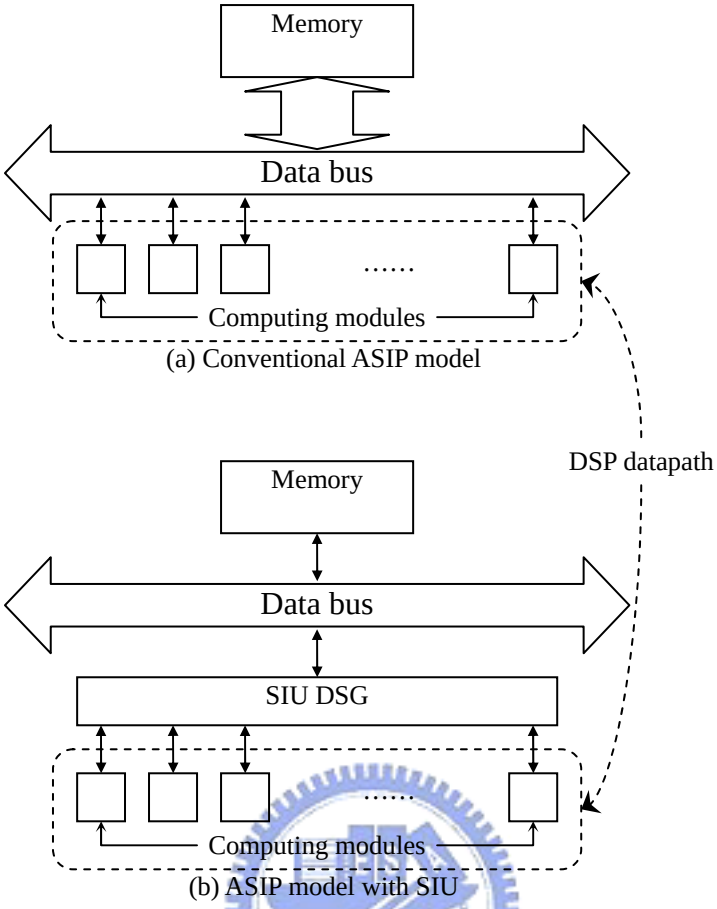


圖 2-1 SIU decouples accelerating DSP datapath from system bus

另外，在利用數位計算處理類比信號的算術方法中，可大致分為動態比率(dynamic scaling)和靜態比率(static scaling)兩種。其中浮點數運算是常見的動態比率的作法，以浮點數算數做計算能得到最大的精確度，但是因為浮點數在計算上必需動態更改指數項的大小，使得計算上速度較慢而其硬體複雜度也相當高。而另一種，靜態比率的計算中以整數運算為最常見，相對於浮點數的特性，整數運算速度快、硬體簡單、且沒有指數項對齊或更新的問題，但在運算前必需保留一定大小的保留位元(guard bits)以免溢位(overflow)，這樣會使位元的利用率變差。因此，本章中將介紹幾種常用於 DSP 處理的數學運算，並定義一個新的靜態浮點運算元，用於處理靜態比率中可能發生溢位的問題，使用這個運算元能在精確度和運算速度中取得較佳的平衡點。

2.1. SIU DSG

如同前言所述 DSG 主要的工作一方面要暫存馬上就用得到的暫時性的資料，另一方面要重排資料進出的順序，產生不同的資料流向(data stream)讓不同 I/O spec 之間能順利的做轉換。因此一個標準的 SIU DSG 的設計流程可以分成兩大步驟：

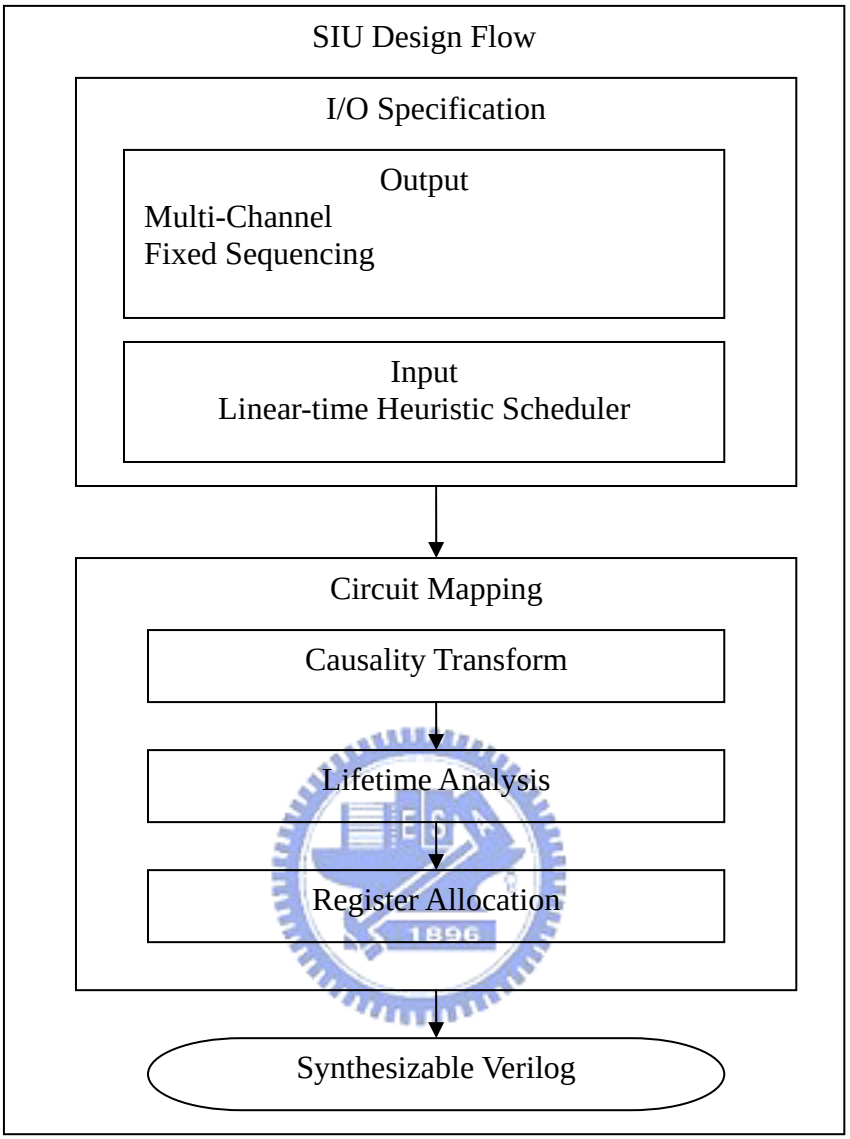


圖 2-2 SIU generation

(1)IO 規格(I/O specification)：清楚定義 I/O 通道中資料進出的順序的規格。

(2)電路對應(circuit mapping) ： 解決資料形式的轉換(data format conversion；簡稱 DFC)問題。

每個步驟的所需完成的工作詳列於圖 2-2 中。

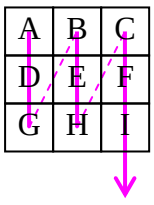


圖 2-3 Matrix transpose output

舉例而言：設一個九宮格的矩陣的資料是以循序的方式由記憶體、codec，或其他 I/O 所讀出，但是卻必需以轉置(matrix transpose)的方式送到計算模組再處理。以 SIU 來處理時，首先要定義 IO 的規格。因此如圖 2-3 所示，距陣轉置的 IO 的順序規格定義如下：

input sequence: A B C D E F G H I
output sequence: A D G B E H C F I

第二步，以此 IO 順序，做資料生命週期的分析(lifetime analysis)。生命週期分析的結果顯示於表 2-1 和圖 2-4。表 2-1 中 T_{input} 為資料輸入的順序也就是資料產生的時間(data birth time)，T_{zlout} 為(無延遲時)資料輸出的順序。前兩者數值的差額即為順序誤差值 T_{diff}。T_{diff} 最負的值也就是解決此問題至少必需延遲的時間。本例中，此值為 -4，意即要完成這個工作，輸入和輸出至少必需有四個時脈(cycle)的延遲。因此 T_{output} 的值為 T_{zlout} 加 4，表示真正資料輸出的時間(資料死亡；data death time) 。因此從資料「出生」到「死亡」的時間即為其生命週期(life period)。將生命週期的結果繪成工作表的型式於圖 2-4，由圖 2-4 中就可以估算出同一時間暫存器的使用量最大量為四個。

Sample	T_{input}	T_{zlout}	T_{diff}	T_{output}	Life Period
A	0	0	0	4	0 ～ 4
B	1	3	2	7	1 ～ 7
C	2	6	4	10	2 ～ 10
D	3	1	-2	5	3 ～ 5
E	4	4	0	8	4 ～ 8
F	5	7	2	11	5 ～ 11
G	6	2	-4	6	6 ～ 6
H	7	5	-2	9	7 ～ 9
I	8	8	0	12	8 ～ 12

表 2-1 life time analysis

完成生命週期分析後再使用前推和後推指派暫存器(forward-backward register allocation)。因為暫存器的用量為四個，因此在表 2-2(a)的最上方先填上 R1 ~ R4 四個暫存器。使用前推的方法把資料依「出生」的循序推入暫存器中。因此可以得到資料 A D G E H I 的輸出時的暫存器。再使用後推的方法把生命週期尚未結束，但卻已經被推到 R4 的資料向前推回前面被空出的暫存器。如表 2-2(b)中，在時脈 5 和 9 時 R4 必需把資料推回給 R3。時脈 6 時 R4 必需把資料推給 R1。另外在考慮使用迴圈連續處理的情形下，時脈 9~12 其實也就是下一個矩陣操作的 0~3 的時脈。因此對應到實際上硬體電路上的結果就如圖 2-5 所示。

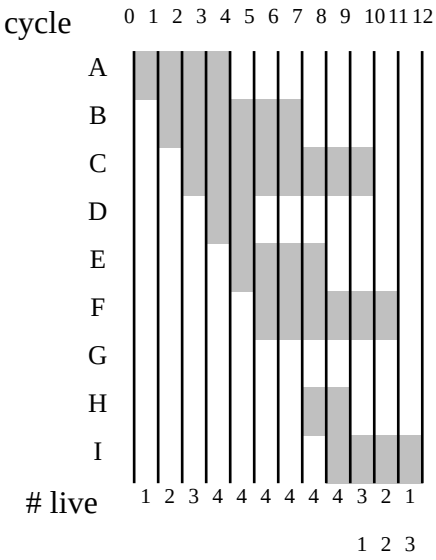


圖 2-4 register allocation

cycle	input	R1	R2	R3	R4	output
0	A					
1	B	A				
2	C	B	A			
3	D	C	B	A		
4	E	D	C	B	A	A
5	F	E	D	C	B	D
6	G	F	E		C	G
7	H		F	E		
8	I	H		F	E	E
9		I	H		F	H
10			I			
11				I		
12					I	I

(a)

cycle	input	R1	R2	R3	R4	output
0	A					
1	B	A				
2	C	B	A			
3	D	C	B	A		
4	E	D	C	B	A	A
5	F	E	D	C	B	D
6	G	F	E	B	C	G
7	H	C	F	E	B	B
8	I	H	C	F	E	E
9		I	H	C	F	H
10			I	F	C	C
11				I	F	F
12					I	I

(b)

表 2-2 forward-backward register allocation

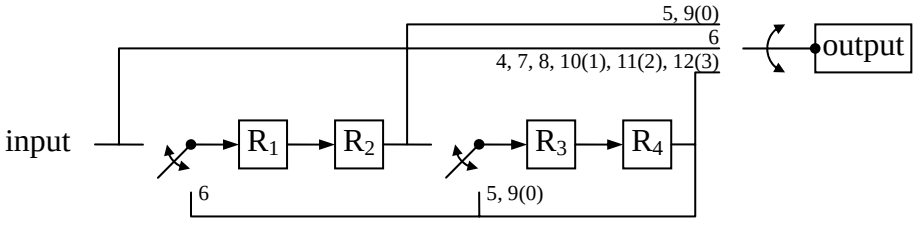


圖 2-5 HW implementation of SIU DSG

以上是一個單輸入，單輸出的例子。當然，以同樣的方法，必要時 SUI 也可規劃成單輸入，多輸出或多輸入，多輸出的架構。如要將一個 4x4 矩陣要同時做圖 2-6 三種順序的輸出則以同樣的方法將三個輸出一起做生命週期分析。表 2-3 顯示三個輸出生命週期分析的結果，需要 13 個暫存器和 9 個時脈的延遲。另外，若改變一下輸入的順序，可以將輸入的順序以對應的 T_{zlout} 的最小值當作優先順序排序，這樣可以得到較少的延遲和暫存器用量的 SIU。如將表 2-3 的 T_{input} 以該資料對應的 T_{zlout} 的最小值(Min(T_{zl}))重新排序，並重做生命週期分析時就可以得到表 2-4 的結果。延遲減短為 4 個時脈，暫存器用量也降為 8 個。以這樣的暫存器做前推後推的暫存器分配後可以得到表 2-5 的結果及硬體電路[20]。

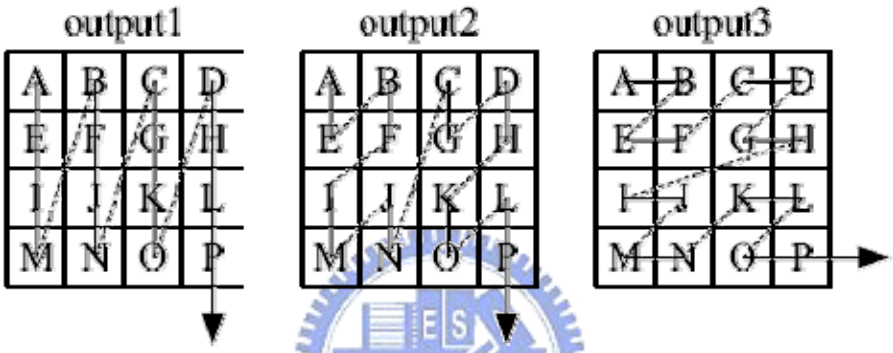


圖 2-6 Parallel execution of SIU

	Tinput	Tzl1	Tdiff1	Tout1	Tzl2	Tdiff2	Tout2	Tzl3	Tdiff3	Tout3	Life	Period
A	0	0	0	9	0	0	9	0	0	9	0	9
B	1	4	3	13	2	1	11	1	0	10	1	13
C	2	8	6	17	8	6	17	4	2	13	2	17
D	3	12	9	21	10	7	19	5	2	14	3	21
E	4	1	-3	10	1	-3	10	2	-2	11	4	11
F	5	5	0	14	3	-2	12	3	-2	12	5	14
G	6	9	3	18	9	3	18	6	0	15	6	18
H	7	13	6	22	11	4	20	7	0	16	7	22
I	8	2	-6	11	4	-4	13	8	0	17	8	17
J	9	6	-3	15	6	-3	15	9	0	18	9	18
K	10	10	0	19	12	2	21	12	2	21	10	21
L	11	14	3	23	14	3	23	13	2	22	11	23
M	12	3	-9	12	5	-7	14	10	-2	19	12	19
N	13	7	-6	16	7	-6	16	11	-2	20	13	20
O	14	11	-3	20	13	-1	22	14	0	23	14	23
P	15	15	0	24	15	0	24	15	0	24	15	24

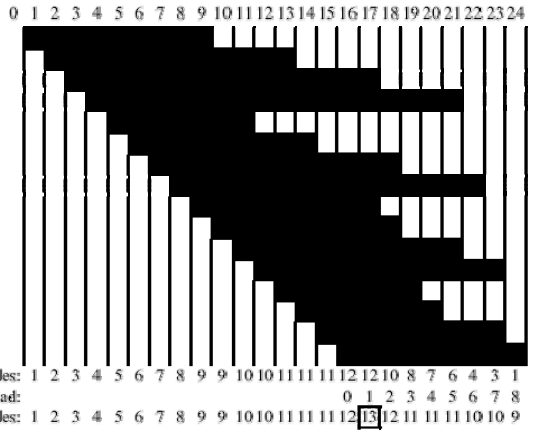


表 2-3 lifetime analysis with raster-scan input

	Tinput	Tzl1	Tdiff1	Tout1	Tzl2	Tdiff2	Tout2	Tzl3	Tdiff3	Tout3	Life	Period
A	0	0	0	4	0	0	4	0	0	4	0	4
B	2	4	2	8	2	0	6	1	-1	5	2	8
C	6	8	2	12	8	2	12	4	-2	8	6	12
D	7	12	5	16	10	3	14	5	-2	9	7	16
E	1	1	0	5	1	0	5	2	1	6	1	6
F	4	5	1	9	3	-1	7	3	-1	7	4	9
G	8	9	1	13	9	1	13	6	-2	10	8	13
H	11	13	2	17	11	0	15	7	-4	11	11	17
I	3	2	-1	6	4	1	8	8	5	12	3	12
J	9	6	-3	10	6	-3	10	9	0	13	9	13
K	12	10	-2	14	12	0	16	12	0	16	12	16
L	14	14	0	18	14	0	18	13	-1	17	14	18
M	5	3	-2	7	5	0	9	10	5	14	5	14
N	10	7	-3	11	7	-3	11	11	1	15	10	15
O	13	11	-2	15	13	0	17	14	1	18	13	18
P	15	15	0	19	15	0	19	15	0	19	15	19

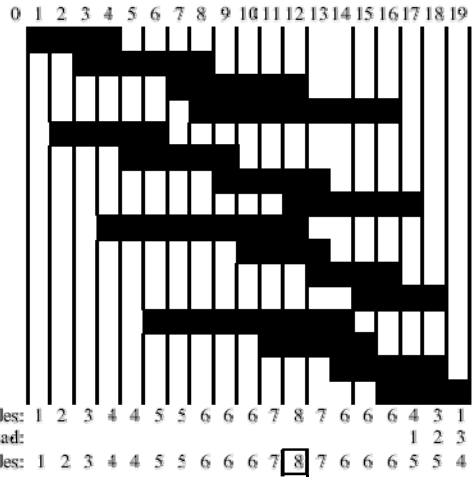


表 2-4 lifetime analysis with small Min(T_{zl}) first input

T	input	reg 1	reg 2	reg 3	reg 4	reg 5	reg 6	reg 7	reg 8	output 1	output 2	output 3
0	A											
1	E	A										
2	B	E	A									
3	I	B	E	A								
4	F	I	B	E	A(1,2,3)					A	A	A
5	M	F	I	B(3)	E(1,2)					E	E	B
6	C	M	F	I(1)	B(2)	B(3)				I	B	E
7	D	C	M(1)	F(2,3)	I	B				M	F	F
8	G	D	C(3)	M	F	I(2)	B(1)			B	I	C
9	J	G	D(3)	C	M(2)	F(1)	I			F	M	D
10	N	J(1,2)	G(3)	D	C	M		I		J	J	G
11	H(3)	N(1,2)	J	G	D	C	M		I	N	N	H
12	K	H	N	J	G	D	C(1,2)	M	I(3)	C	C	I
13	O	K	H	N	J(3)	G(1,2)	D		M	G	G	J
14	L	O	K(1)	H	N			D(2)	M(3)	K	D	M
15	P	L	O(1)	K	H(2)	N(3)			D	O	H	N
16		P	L	O	K(2,3)	H			D(1)	D	K	K
17			P	L(3)	O(2)		H(1)			H	O	L
18				P	L(1,2)	O(3)				L	L	O
19					P(1,2,3)					P	P	P

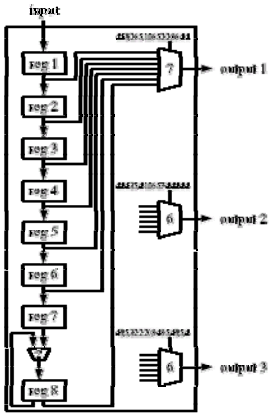


表 2-5 forward backward register allocation

由以上 SIU DSG 的分析方法可以看出：SIU 的硬體架構主要由佇列(queue)和多工器(mux)所組成的電路。主要的技巧則為資料生命週期的分析和暫存器的分配。上面的兩個例子利用前推後推來分配暫存器都是將先將輸入資料放到佇列中，當需要輸出時再用多工器把資料送出去。

因此當 SIU 的設計能有效掌管整個資料流向的運送時，計算模組只要專心處理由 SIU 送來的資料，處理完的資料再送回 SIU 即可。這樣的分工方式經實做數據的分析比較顯示，能讓資料流向更順暢，增加運算效率。

2.2. 用於多媒體運算的暫存器組織

把 SIU 的觀念應用於微處理器內，其資料流產生器(DSG)和運算模組(computing modules)的關係就如暫存器檔與運算單元一般。因此在本節中，將對一些暫存器組織的方法做分類和其複雜度成長的分析。並於下一節(2.3 節)以實做分析集中式及分散式暫存檔實現 SIU 的試用結果。

圖 2-7(a)顯示一個基本的暫存器的硬體結構，它經由兩個反閘(NOT gate)組成儲存單元，再經由電晶體連接到每個埠(IO port)，實際運作時，將位址解碼後，由 word line 選擇電晶體開關，再由 bit line 讀出或寫入資料的數值(0 或 1)。而在布局(layout)的實現上(圖 2-7(b))，一位元的儲存單元需要固定大小的面積 w x h，而每個埠的 word line 和 bit line 的金屬線也必需保持一定的間距。設每個埠的金屬線間距為 p 的，基本單元布局區域的長、寬會隨著 IO 埠的數目呈線性比例成長。所以一個基本暫存器的結構的面積為(w+p)＊(h+p)。因此整個暫存器檔的面積正比於暫存器的位元數和暫存器總數，並正比於可直接存取的 IO 埠數之平方[11]。以數學式表示時，設暫存器總數為 n、IO 埠數為 P，則：

暫存器檔面積 $\propto (n \cdot P^2)$

在一般 RISC 運算單元的平行度不高的情形下，IO 埠通常會被設計成共用的方式，因此整體而言 IO 埠的數量並不需要很多，因此在 IO 埠有限的情形下，使用集中式的暫存器的面積只隨著暫存器的總數線性成長，並不會造成複雜度過度膨脹的問題。但是在需要高平行度運算的 DSP datapath 中，運算單元通常必需有專屬的 IO 埠才能順利達成平行運作。在這樣的情形下，設運算單元的個數為 N時，暫存器的總數和 IO 埠的數量都會和運算單元的個數成正比，因此會使暫存器檔的面積變成三次方倍的成長：

暫存器檔面積 $\propto (n \cdot P^2) \propto (N^3)$

而延遲(access delay)則決定於位址由 word line 進入到資料經 bit line 送出時的整個傳輸延遲(propagation delay)。因此在，即使在最佳佈局(layout)的情形下(正方型的佈局)，word line 和 bit line 的總長度正比於整個暫存器檔的邊長。如圖 2-8 正方形的邊長有：$\sqrt{n}$ 個 cell，而每個 cell 的邊長正比於 P:

暫存器檔延遲 $\propto \sqrt{n} \cdot P \propto (N^{3/2})$

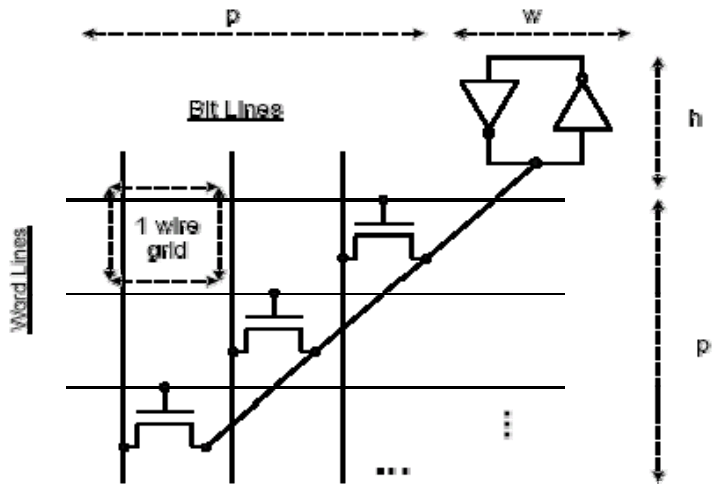
暫存器檔功率的消耗決定於電容，而電容則大部分取決於 word line 和 bit line 的面積。word line 和 bit line 的總寬度和埠數 P 成正比，而其長度不論是(p+w)或(p+h)也都隨埠的個數成長，因此單一 cell 的電容量正比於 P^2，故整個暫存器檔的功率消耗的成長速度可表示為：

暫存器檔功率消耗 $\propto (n \cdot P^2) \propto (N^3)$

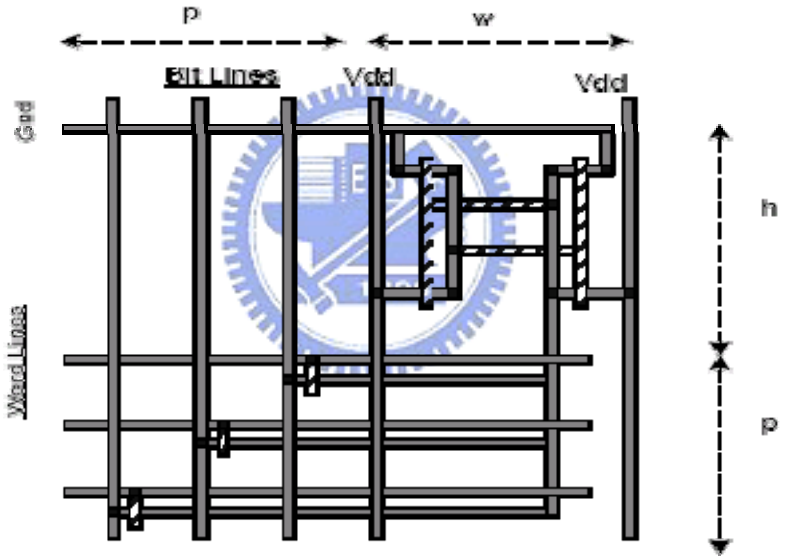
由此可知，集中式暫存器的複雜度在高度平行處理的情形下，會隨著運算單元的個數增加呈指數倍的成長，但其用在一般 RISC 下並不會產生問題。以一般性的經驗而言，其成長的速度如下：當使用七個以上的平行運算單元時，暫存器檔的面積就會主宰整個晶片；而使用八個以上時，暫存器檔的耗電量就會超過一個可運算浮點數乘法的乘法器[11]。

因此應用在 DSP 上的暫存器檔常會被做相當程度的切割，其目的無非在

於減少直接存取暫存器的 I0 埠的使用量。以限制暫存器存取的自由度的代價換取電路複雜度的改善。



(a)



(b)

圖 2-7 register cell

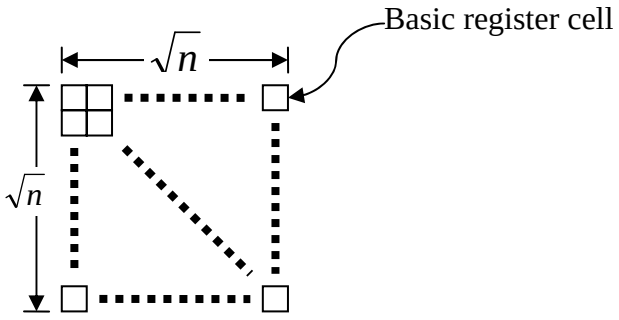


圖 2-8 register file layout placement with ***n*** registers

2.2.1. 暫存器的分類

以暫存器切割的方式劃分，大致可分為 banking 和 clustering 兩種方法。Banking 的方法是把真實的埠和邏輯上的埠做動態對應，其限制在於同時間在同一個 bank 上存取的自由度。而 Clustering 是在運算單元上做切割，限制運算單元存取的自由度。

- Banking

  Banking 使用多個實體埠和邏輯埠做動態對應的方法來減少在每個 bank 上 IO 埠的使用量。每一個實體埠只能存取所屬的 bank 的暫存器，而透過對應的方法可以把這些實體埠對應到每個運算單元。以軟體的角度而言，每個運算單元都可以存取任何一個暫存器，就存取權限而言和集中式的暫存器是一樣的。但就硬體的角度來看，在同一時間內存取同一個 bank 的頻寬不能超過該 bank 的埠的數量。否則就會發生埠數不足的情形。因此在暫存器的配置上 Banking 的暫存器不如完全集中式的暫存器檔自由，組譯器在分配暫存器(register dispatch)時就必需避開在同時間內多個運算單元同時間搶同一個 bank 的資源的情形。

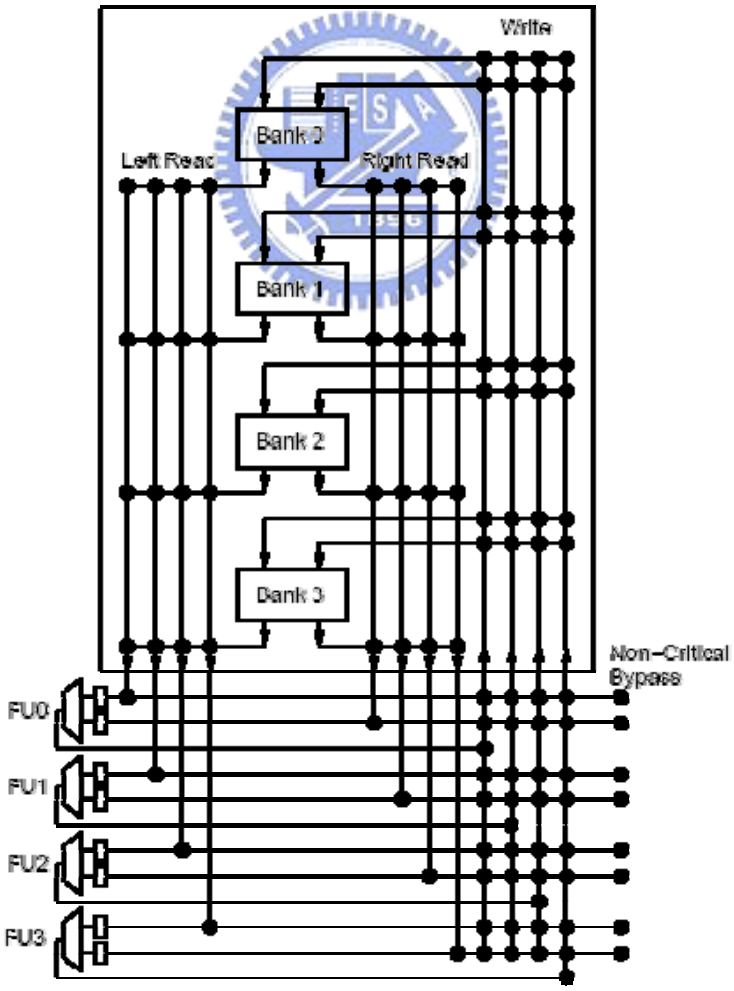


圖 2-9 Banked register organization

圖 2-9 為一使用四個二讀二寫(四個埠)的暫存器 bank 所組成的八讀四寫(十二埠)集中式暫存器檔的情形。在這樣的設計下每個 bank 的埠數變成三分之一，暫存器的個數也變成四分之一。因此整體的暫存器檔的架構就由：四個 bank (面積總合就約為原來集中式暫存檔的九分之一)和四組四對四的交換電路(crossbar router)所組成[17]。

- Clustering

另一種切割的方法是依照運算單元來切割暫存器。先把運算單元劃分成幾個小群組(cluster)，令每個群組只能存取特定範圍的暫存器。這樣就能把一個大暫存器檔和多個運算單元區分成幾個獨立的小群組，在群組內的運算單元存取自己的群組內暫存器時仍保有相當的頻寬和自由度，而且因為小群組中 IO 埠和暫存器的數量可以控制在固定的範圍內，所以每個小組中局部的暫存器檔的大小能固定下來(圖 2-10)。因此可以看出暫存器檔的電路只會隨著群組數量的增加約呈線性比例成長。另外，再額外加上一些群組間交換資料的機制就可以讓不同群組間做資料的溝通和交換[18]。

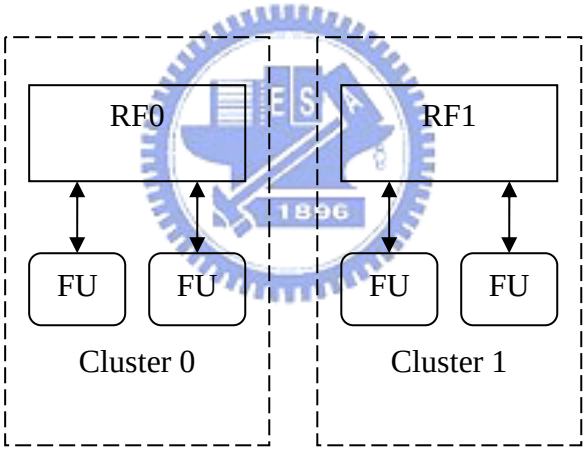


圖 2-10 Clustered RF organization

在 Clustering 中如果把每個運算單元都個自獨立為一組則這樣的組織則另稱為分散式的暫存器(distributed register file；簡稱 DRF)。在這樣的分組下，每個 DRF 只需支援自己的運算單元，所以在暫存器檔的硬體上可以做得很精簡。但相對的必需多付出一些代價做資料交換。

2.2.2. 暫存器檔的資料交換機制

切割暫存器檔雖然有減少硬體複雜度的好處，但卻必需付出其他如需要額外做資料交換的代價。由軟體方面來看就是暫存器使用上的自由度；由硬體的角度而言就是必需增加額外的電路在不同暫存器檔之間做資料交換。在本節中，我們綜觀一些常用的交換機制並大致區分為兩類：延伸存

取，和共用空間。

- 延伸存取(extended access)

  由硬體的角度來看，延伸存取是經由額外的交換繞線(crossbar router) 使功能單元(function unit) 在一些特定的情況下可以直接讀出或寫入非專屬的暫存器檔的方式。其中這個功能單元並不一定指運算單元(ALU) ，它可以是匯流排(BUS)，IO 單元(load-store unit)，或是其他可用做資料交換的電路(圖 2-11)。以軟體的角度來看時，延伸存取還可以再細分成三種類形：資料搬移、延伸寫入和延伸讀取。

  - 資料搬移(copy operations)

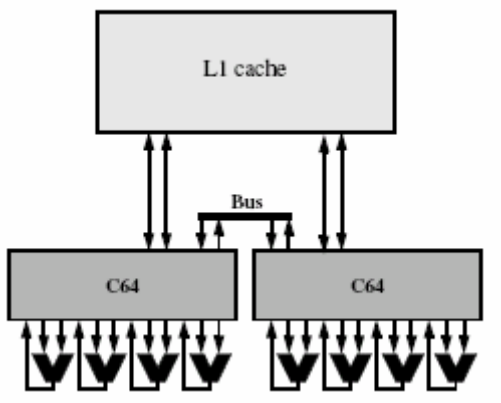
    在指令中，運算單元要用到外面的資料時，必需先透過專門的搬移的指令把資料複製到專屬的暫存器檔內才能使用[13]。這些搬移資料的指令通常會經由特定的發出格(issue slot)來發出(issue)並執行，而硬體的交換電路會出現在用來執行搬移指令的運算單元中。如 ADSP-21xx 必需透過特定的資料搬移的指令，它利用 R bus 把資料般到專屬的暫存器檔才能被下一個運算單元讀取(圖 2-11 (a))。

  - 延伸寫入(extended write)

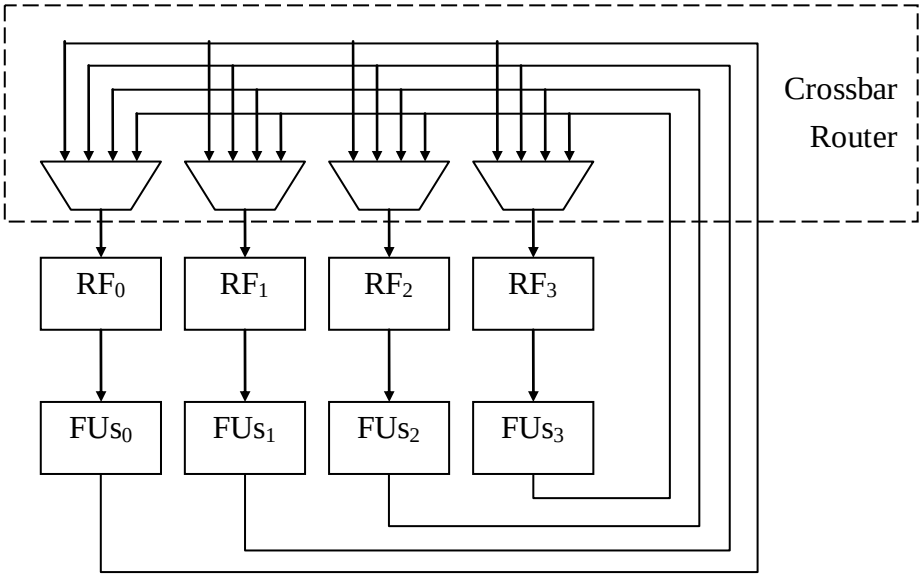
    在原本限制存取的機制中開放寫入的權限給運算單元。讓運算單元可以直接寫入其他的暫存器檔。在指令中的運算的結果，可以不受存取範圍的限制，利用廣播或運送…等等各種方法送到非專屬的暫存器檔諸存。在這樣的機制下硬體的交換電路會出現在運算單元的輸出端。(圖 2-11 (b))

  - 延伸讀取(extended read)

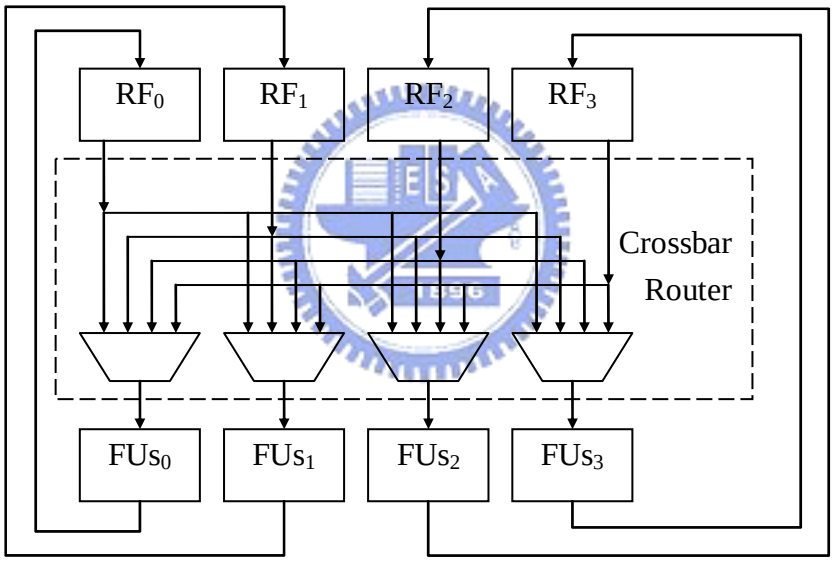
    和偷寫相反的概念，交換電路被置於運算單元的輸入端。這樣運算單元的輸出結果雖然只能存入專屬的暫存器內，在需要時可被讀取並運送到其他的運算單元繼續被運算 (圖 2-11 (c))[19]。



(a) copy operations (thru bus)



(b) extended write



(c) extended read

圖 2-11 extended access model

■ 共用空間(shared storage)

相對於延伸存取利用額外的交換電路做資料溝通，而共用空間則利用的是分級(Hierarchical)的概念以增加額外的暫存器空間來達成資料交換。在這樣的交換機制中主要有基本型的共用暫存器和增強型的 Ring structure 兩種做法(圖 2-12)。

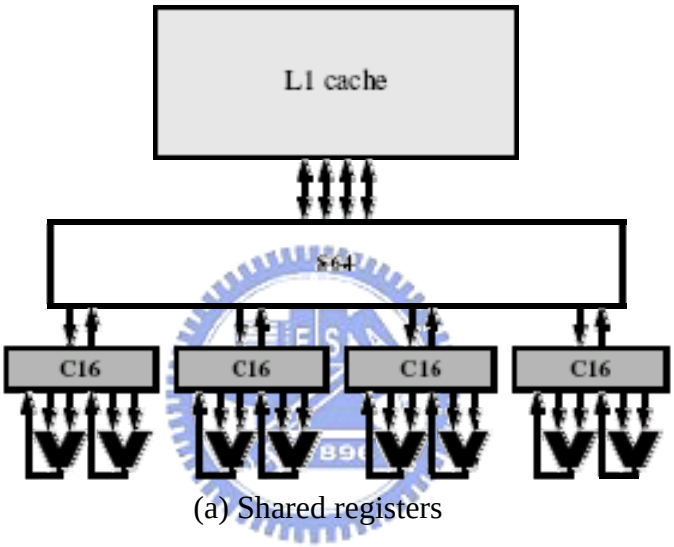
◆ 共用暫存器(shared registers)

這是共用空間最基本的蓋念，在專屬的暫存器檔之下再加一層共用的暫存器檔，資料可以透過共用暫存器做交換。這個共用的暫存器檔的功能就如同集中式的暫存器一般可以和每

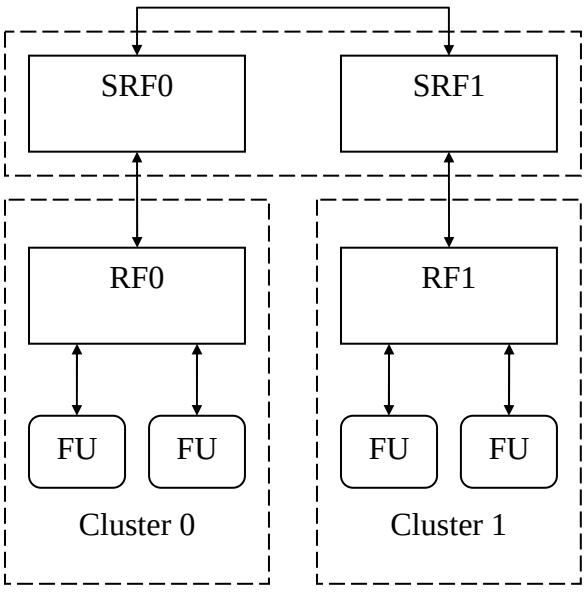
個專屬的暫存器做資料的交換。雖然功能上像集中式的暫存器，但是因為它只支援必要的資料交換的動作，所以暫存器的數量和 IO 埠的需求比起真正的集中式的暫存器檔已經少了很多[18]。

◆ Ring Structure

Ring Structure 是把 banking 的概念再套用在上面的共用暫存器上面，如圖 2-12 (b)共用的暫存檔一樣可以分成不同的 bank 一次對應到一組專屬的暫存檔。當需要交換資料時，可以經由 bank 的動態對應，可以把不同的 bank 對應到需要資料的地方。



(a) Shared registers



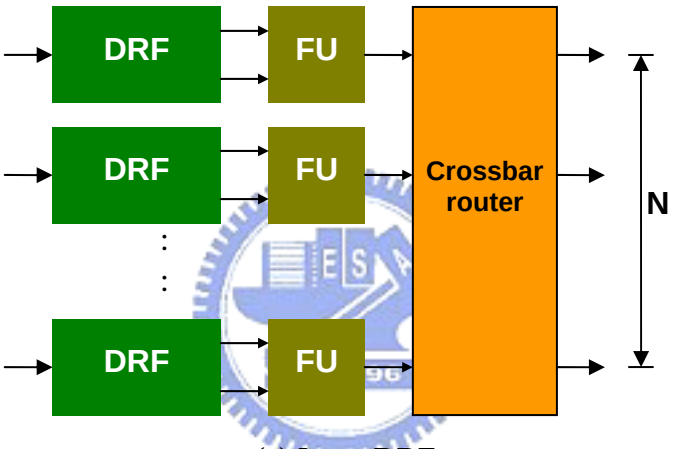
(b) Ring structure

圖 2-12 Shared storage

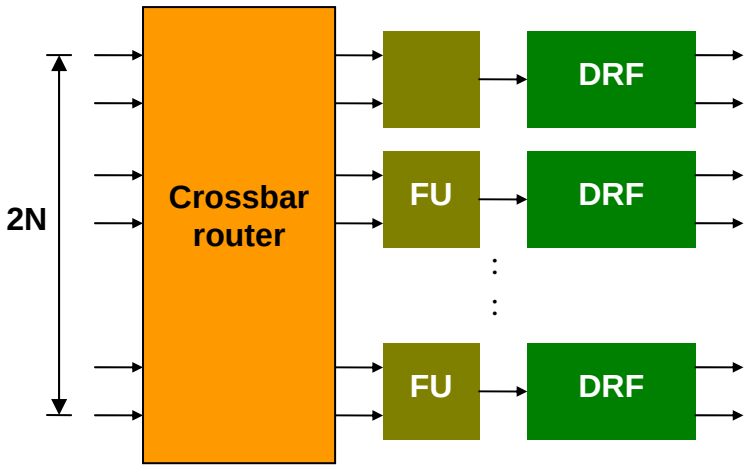
2.3. 集中式與分散式暫存器組織之比較

當使用集中式暫存器來實現 SIU 時，因為所有的運算單元都能存取任何 CRF 中的暫存器，因此不論在資料暫存、資料交換和順序重排上的工作都很容易。但是理論上它必需占用很大的矽面積和消耗大量的功率。

而使用 DRF 來實現 SIU 時，依資料交換的機制使用延伸寫入和延伸讀取的不同還可以分成輸入暫存器(input DRF)和輸出暫存器(output DRF)兩種型式(圖 2-13)。而使用 DRF 可以簡化硬體的複雜度，但卻必需額外付出延伸存取的代價做資料交換的動作。在 DSP-lite 中，曾經分析比較過 input DRF 及 output DRF 兩種組織的硬體實現結果。以一般三個運算子(2 個輸入運算子，1 個結果運算子)的運算單元為例，在圖 2-13 中用於 input DRF 的交換電路是一個 N x N 的切換(switch)電路。(N 為運算單元的個數。)而用於 output DRF 中的交換電路為 $2N$x$2N$ 的切換電路。以實作數據的比較結果 output DRF 的複雜度比 input DRF 的複雜度更高[22]。



(a) Input DRF



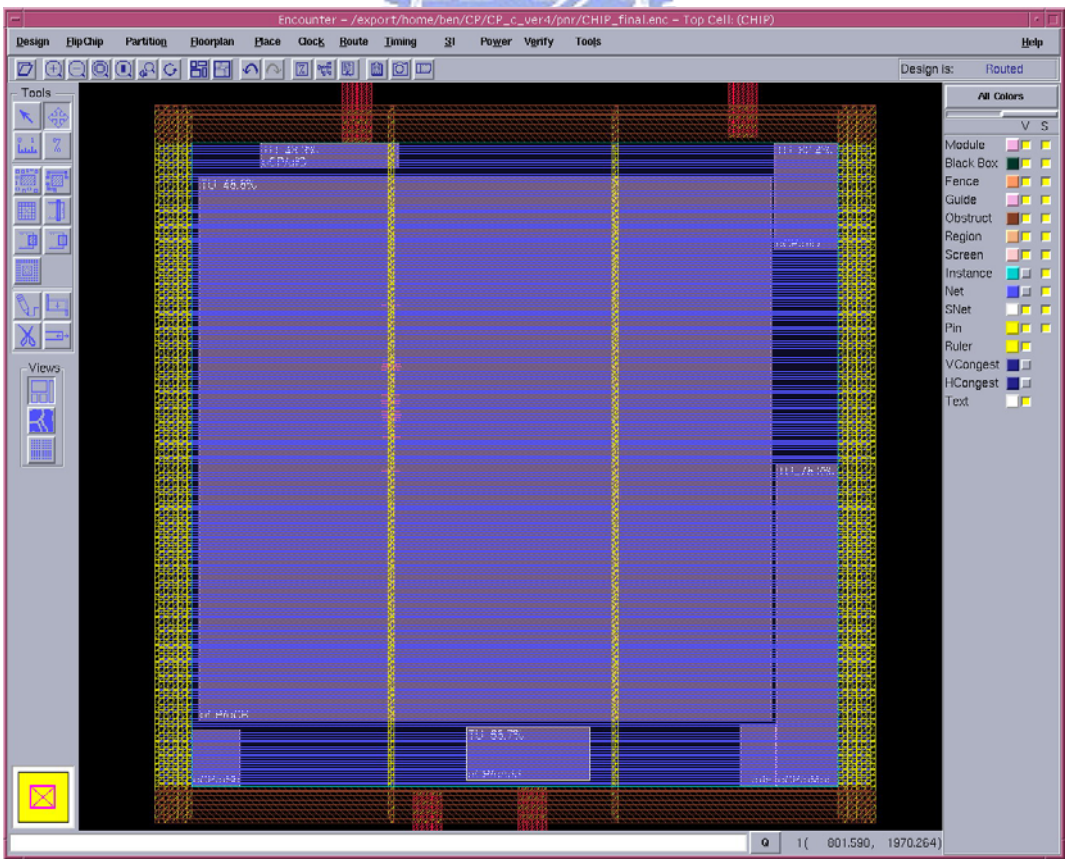
(b) Output DRF

圖 2-13 SIU implementation with DRF

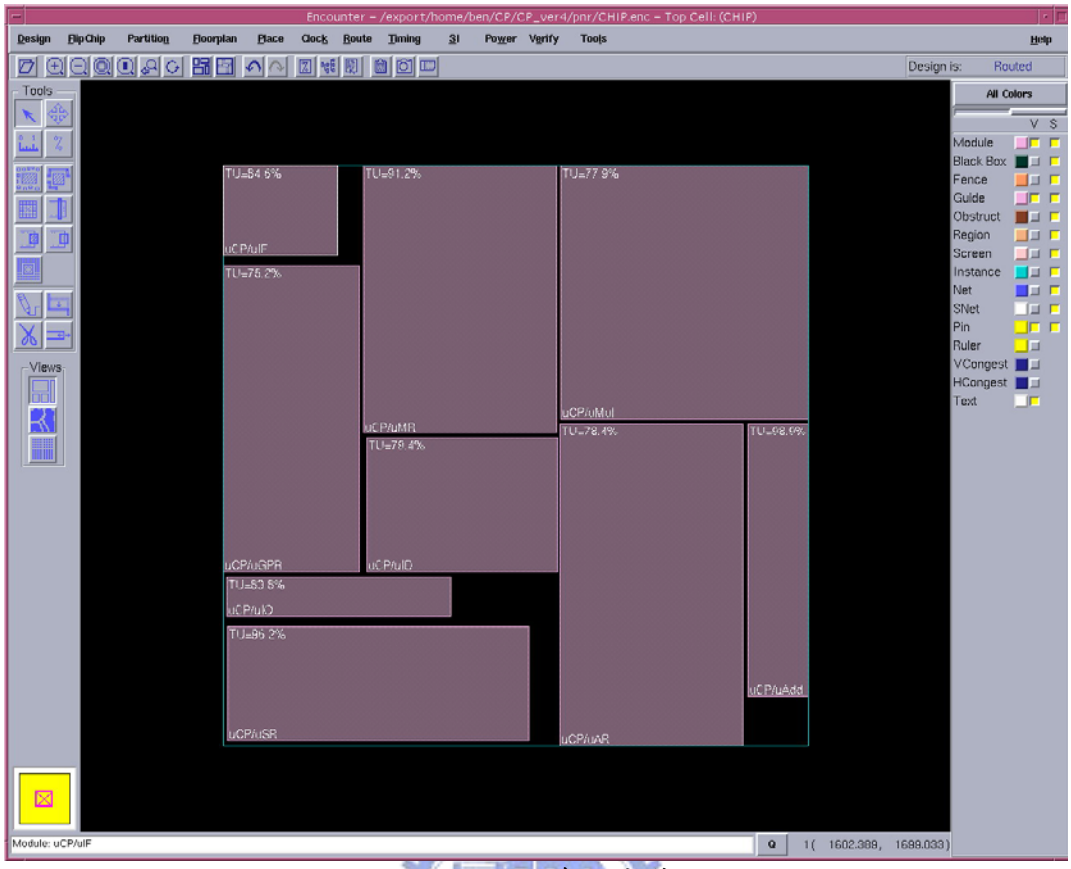
為了解集中式和分散式的暫存器檔在實際應用上對電路影響的程度，我利用 UMC 0.18 製程實做兩種暫存器組織。實驗的運算單元為一個 IO 單元

(load/store unit)，一個 ALU，一個乘法器及一組位移器，平均每個運算單元配置 16 個暫存器，每個暫存器 16 位元。集中式的暫存器檔的結構為一組包函 64 個 16 位元的暫存器，11 個埠(七讀四寫)。而分散式的暫存器檔的結構為三組暫存器檔為：16 個 16 位元的暫存器，3 個埠(二讀一寫)和一組 16 個 16 位元，2 個埠(一讀一寫)的暫存器檔所組成，使用的資料交換的機制為延伸寫入，因此交換電路為 4x4 的 crossbar router。用於比較的演算法為 16 點複數 FFT。

表 2-6 中總結實作的結果。在計算效率上 DRF 比 CRF 多花約 31%的時脈做資料交換的工作。而在電路複雜度方面：圖 2-14 比較兩者利用 standard cell APR 的結果，在 DRF 中 crossbar router 在中央，連接四個暫存器檔，最角落才是運算單元。在繞線複雜度上，線路由 crossbar router 連接四個暫存器檔，而四個暫存器檔到四個運算單元的繞線都是局部的繞線，所以除了電晶體用量變少之外，面積的利用率也高達 82% ；而圖 2-14(a)為 CRF 的佈局圖，CRF 主宰了晶片的面積，其他的運算單元的電路在四週。由於 CRF 電路的複雜度使得繞線難度增加，相對地降低面積的利用率。所以在節省面積和耗電方面 DRF 都有三倍以上的改善。相對效率的部分，在低功率消耗的考量上，DRF 可以輕易用增加運算單元來改善效能的缺陷，但是使用 CRF 每增加一個運算單元所要再付出的代價就遠大於前者了。因此我選擇以 DRF 做為資料流設計的暫存器組織，進一步控制和資料交換的機制將於下一章中介紹。



(a) CRF 的布局分佈



(b) DRF 的布局分佈

圖 2-14 CRF 與 DRF 布局分佈圖比較

Organization	集中式(CRF)	分散式(DRF)
# of Registers	1x64x16bits	4x16x16bits
Gate count	33536	13161
Utilization	43%	82%
silicon area	1.15x1.15mm^2	0.6x0.6mm^2
Access delay	2.80ns	1.87ns
Power	116mW	37mW
Performance(FFT)	210 cycle	277 cycle

表 2-6 集中式和分散式暫存檔比較表

2.4. 輕量型算數介紹

當 SIU 掌管資料的流向之後，運算單元便只需專注於資料運算方面的工作。在這樣的分工下，選擇何種算數運算的運算單元和 SIU 無關，只要運算單元間彼此協調好就行了。在本節中將介紹一些算數運算及其分類。

在 DSP 計算中，必需要先將類比的信號量化才能做數位的處理。在量化的過程中用來表示信號的有效位數的多少直接影響到運算品質的誤差。因此在計算的過程中數值比例的選擇即為 DSP 運算誤差的主因，選的太小計算時會造成溢位(overflow)，選得太大會造成有效位數不足或過小(underflow)。在比率的選擇的方法上可大致區分成兩類：動態比率

(dynamic scaling)和靜態比率(static scaling) 。動態比率顧名思義就是在計算過程中依照數字實際的大小決定其指數項的值。因此，一般而言動態比率的有效位數會比較高。然而動態比率在計算上必需一邊計算，一邊控制位移量還要同時考慮溢位，借位的問題。因此它計算量的負擔相當重。然而，在 DSP 的應用上誤差值的高底並非決定 DSP 好壞的因素，當誤差值滿足一個規定值以下時，更精確的運算其實並非必要的。因此在極度要求計算速度的 DSP 設計上，設計者往往會考慮用其他計算量較輕的靜態比率的方法來做 DSP 運算。靜態比率的優點在於：數值的指數項，在實際運算前就已經使用統計、分析或其他數學方法規劃固定。在運算中只需計算對數值，所有該位移的值也都由事先預測和安排。因此在計算中靜態比率的速度要比動態比率快得多，但相對的，使用靜態規劃的算數在位元的利用率上當然比不上動態的判斷，因此付出的代價就是計算的精確度。

■ 動態比率

在動態比率的算數上目前主要有兩種方法:浮點數和區域浮點數(Block floating-point；簡稱 BFP)

◆ 浮點數

浮點數(floating-point；簡稱 FP)是最直覺，也是目前公認精確度可以達到最高的算數。以 32 位元的浮點數而言，在 IEEE754 單精確度浮點數的格式如下：(圖 2-15)

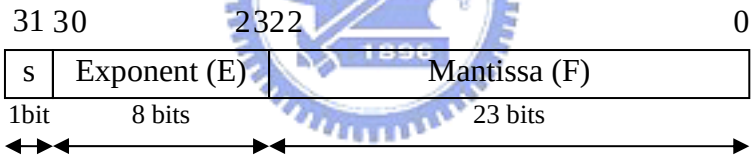


圖 2-15 IEEE754 single precision floating point format

用來表示 $(-1)^s * F * 2^E$ 的數值。

當兩個浮點數做加法時，至少必需有以下步驟：

- 先把兩個指數項相減
- 以相減所得到的數值將指數項較小數值的對數項向右位移
- 把位移後的對數項(含正負號)相加
- 辨斷結果是否需要正規化(normalize)並把結果正規化

就以上的步驟而言，兩個浮點數的加法就至少需要做二或三次的加法 (指數項、對數項和正規化時的對數項)和一或二次的位移(計算前的小數點對齊時和正規化)。而做浮點數的乘法時必需要有：

- 指數相加
- 對數項有號數相乘
- 正規化及取概數(rounding)

因此就必需要有二次加法、一次乘法及一次正規化。

由此可知不論加法和乘法的複雜度都相當高，而且前後步

驟因果的相依性高。所以一般在支援浮點的運算中大多使用浮點運算加速器分成多級管線化來處理。浮點數的數值方法可以直接套用在 DSP 理論的運算中實數的計算，不必做任何修改，且運算所得的值直接就是運算的結果。但，就如一般所認知的，處理浮點數需要額外花費的硬體成本相當高，速度也慢。唯一的好處是不論任何情形下，都可達到 23 位元的精確度，誤差的比值始終保持在 2^{-24} 的範圍內。

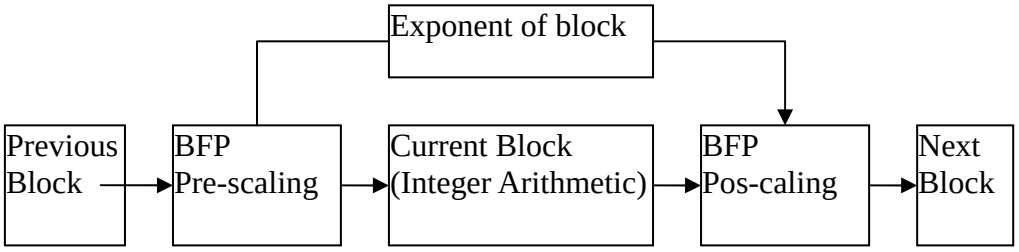


圖 2-16 BFP Arithmetic

◆ 區域浮點數

區域浮點數的有別於浮點數之處在於指數項的共用。區域浮點數將整個 DSP 運算劃分為數個區域(block)，在區域內的數值的指數項控制成相等且共用。如此在區域內的運算就只有對數項的運算。只有當整個區域運算完要將數值傳到下個區域之前才會對這些數值中的最大值做正規化，並更新指數項。BFP 的流程如圖 2-16 所示，BFP 會根據區域中的出現的極值，動態地位移並更新區域指數項。相較於 FP 計算的複雜度，BFP 以一定的比率減少指數項更新的次數。

■ 靜態比率

靜態比率中最經典的為整數比率(integer scaling)，而其他方法大多以其為出發點向動態比率方向修正如：非條件式區域浮點數和靜態浮點數。

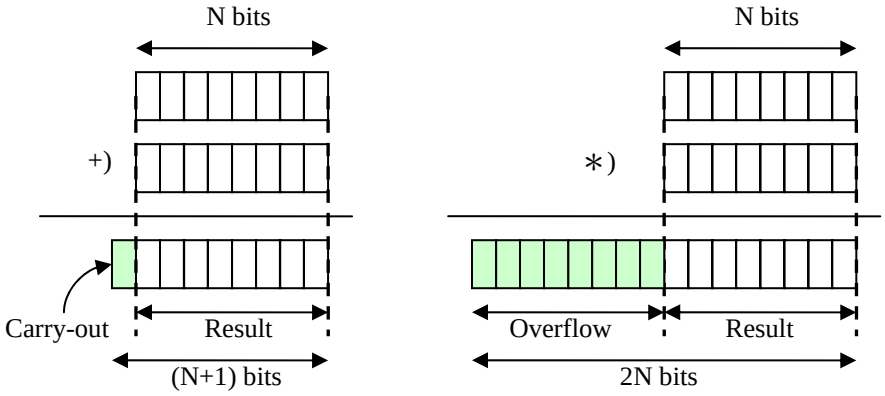


圖 2-17 整數運算的溢位情形

◆ 整數比率

整數比率在運算的過程中全部使用整數的算數運算。當資料的數值在進入運算前就必需預留一些保護位元(guard bits)給運算時可能發生的溢位之用(圖 2-17)。因此整個整數的位元就只有一部分能用來表示信號的有效位元。舉例來說，以 24 位元的整數比率算術來實現將 256 灰階的圖片通過以參數為[-0.0645, -0.0407, 0.4181, 0.7885]的 7-tap symmetric FIR filter 時，必需考慮輸入的最大值為 255(即使用 8bits)，假設預留 3bit 為保護位元。 則還剩下的 13bit 即為此整數運算指數項的準位(即 2^{13})。因此原來的參數在取樣後的數值即為[-528, -333, 3425, 6459](即乘以 2^{13}後再取概數)。 如此運算的雜訊則會固定在 2^{-13}以內的數值。對於運算路徑中最大的數值而言(2^{11})，雜訊比的比值為 2^{-24}，但是對輸入的最大值而言(2^{8})雜訊的比值為 2^{-19}，而對輸入的最小值而言(2^{0})雜訊的比值就只有 2^{-13}。整數的數值方法會因為要預留保護位元而使得誤差值被限制在整個 DFG 中會可能出現的最大值的 2^{-24}。 對於數值較小的數字而言，差訊的比值就會因為位元數中有效的位數變少而相對增加。

當實際運算並沒有溢位的情形發生時，那麼先前預留的保護位元就會因為沒有利用到而浪費了。另一方面，如果輸入的信號最大值和最小值相差很多，則對於數值較小的信號有效的位數也會相對的減少，甚至產生過小(under flow)無法表示的情形。這些是靜態比率的無法避免的缺點。因此就產生由整數比率向動態比率方向改進的其他輕量型算數，其主要的目的都是希望能在計算的過程中加入適當的位移，以減少需要預留保護位元的位元數，增加有效位元的使用率。

◆ 非條件式的區域浮點數

其運作模式是將 BFP 中需要以動態更新的指數項部分改為靜態的規化。先用靜態的分析求出區域間位移的關係，當資料由一個區域進入另一個區域時再利用位移器(shifter)調整成目標區域的小數點準位。舉例而言：在 16 點的 FFT 運算中，運算路徑中最大值為輸入信號的 32 倍，所以如果用整數運算時就必需先預留 5 個保護位元。然而這 5 個保護位元並非一次計算就會成長到 32 倍，整個 FFT 運算可以先劃分成四層 butterfly 的運算，每一層 butterfly 運算都當成一個區域。經由分析，butterfly 運算的輸入到輸出的數值，最大為 2.4 倍的成長 (Rabiner and Gold,1975)[13] 。因此使用非條件式 BFP 時，一開始只需預留 2 個保護位元即可，當運算完一層 butterfly 後再把所有的數值向右位移出一個保護位元，再進入下一層 butterfly(圖 2-18) 。如此在運算中，一開始保護位元的個數為 2 位元。比起整數比率，BFP 計算精確度上可以多出三個有效位元。

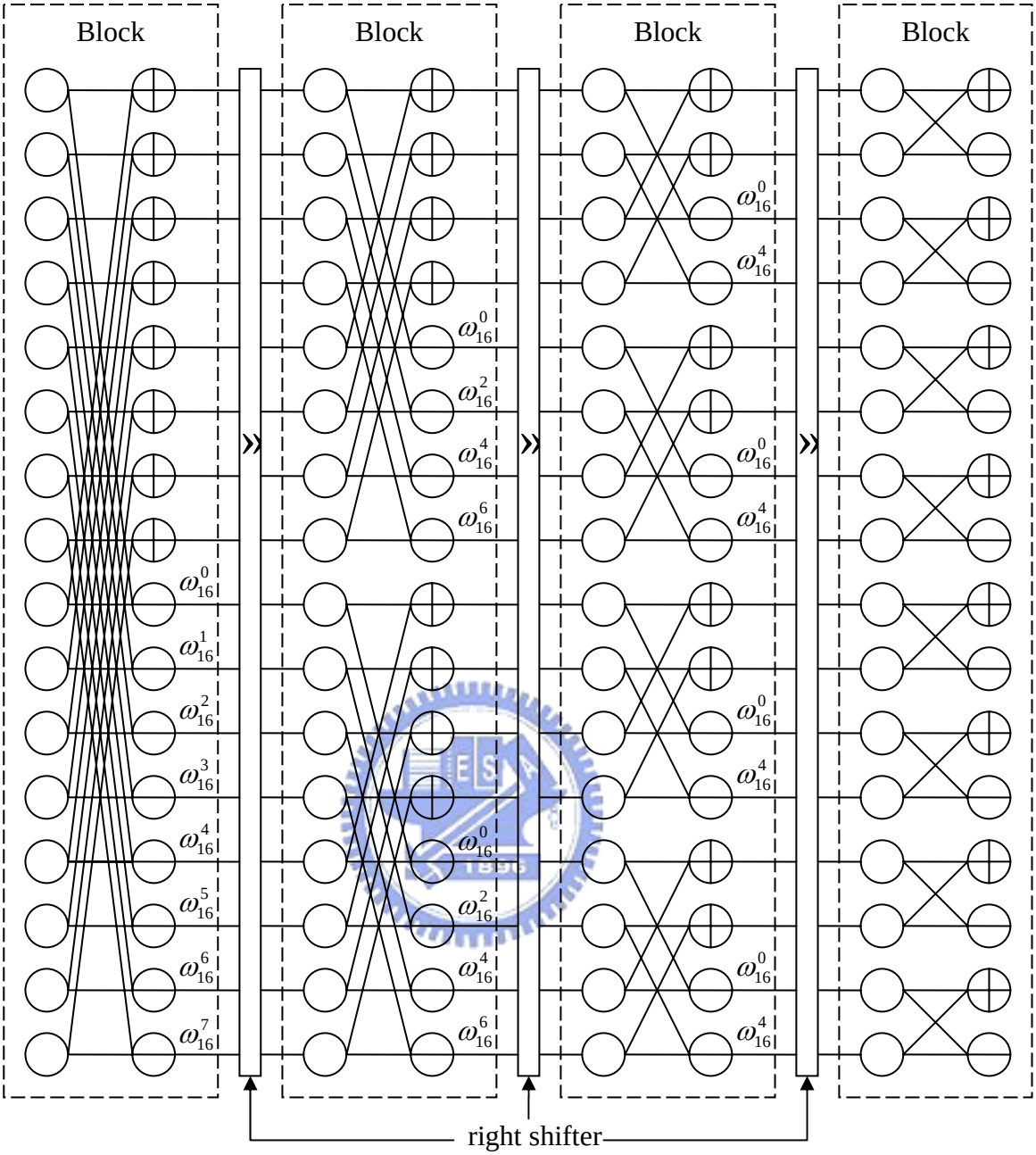


圖 2-18 FFT using unconditional BFP arithmetic

◆ 靜態浮點數[20]

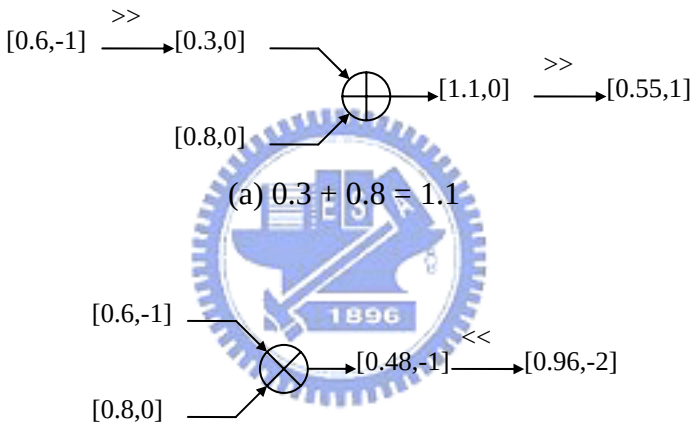
靜態浮點數(static floating-point;簡稱 SFP)運算是以估算計算過程中，每個節點中可能出現的最大值做為指數項的基準，而對數項部分以純小數(fractional number)表示。因此對數項的運算為純小數的運算，且指數項變更需要的位移也直接定義在運算元中。

首先，定義 SFP 數字的表示法。SFP 中數字分成純小數和指數，而純小數的數值則介於 ±(1 0.5] 的區間內。 舉例而

言：0.8 表示為[0.8,0]，而 0.3 則表示成[0.6,-1](即 $0.6*2^{-1}$ 之意)。圖 2-11 顯示 0.8 和 0.3 使用純小數運算做加法和乘法的過程。在做加法運算前要先做位移讓指數項相同，在做完運算後必需再位移，調整指數項的大小。在此，定義新的純小數運算的運算子如下：

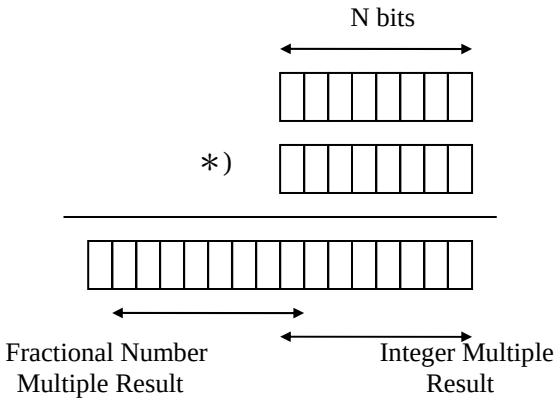
0.6 (+,101) 0.8 = 0.55 和
0.6 (*,1) 0.8 = 0.96

其中(+,101)中參數 101 的意義為第一個運算子和結果運算子都向左位移一個位元。而(*,1)中的 1 表示乘法的結果運算子要向右位移一個位元。因此，從硬體的角度來看，如圖 2-19 所示，要使用這樣具有位移參數的計算必需要在傳統的加法器的前後和乘法器後面都加上位移器。



(a) 0.3 + 0.8 = 1.1

(b) 0.3 * 0.8 = 0.24



(c) Fractional number multiplier

圖 2-19 純小數運算

在此以圖 2-20 的實例解說 SFP 計算在 DSP 中的運作，並估算在硬體上需要加多少位元的位移器才能滿足實際應用的

需要。圖 2-20 為一個 8 點 DCT 的 DFG (Data flow graph)。令輸入信號大小在 ±1 之間，則可以用(1,0)來表示輸入最大值。經由線性組合的向量分析可以得到 DFG 中每個節點(node)可能出現的最大值，把這些值用 SFP 表示法表示後即為圖中每個節點上面的數值。接著，檢視每個運算元所連接的運算子的指數項就可以決定 SFP 做在該節點的運算時需要使用的位移參數了。統計圖中的結果：在 DCT 運算 29 個加法中兩個輸入運算子的指數項關係為：

相同的情況有 20 次佔 69%，

指數項差 1 的情況有 7 次，佔 24%，

相差 2 的有 2 次，佔 7%，

相差 3 以上 0 次。

可見在實際應用上，DSP 加法運算中兩個輸入運算元的峰值指數項會相差很大的機會很少，大多數是相同或相差 1。因此在簡化硬體的考量上加在運算元的位移器只需加上一位元的移位器即可。處理相差 2 個位元以上的運算時，則利用其他的位移器幫忙處理即可。

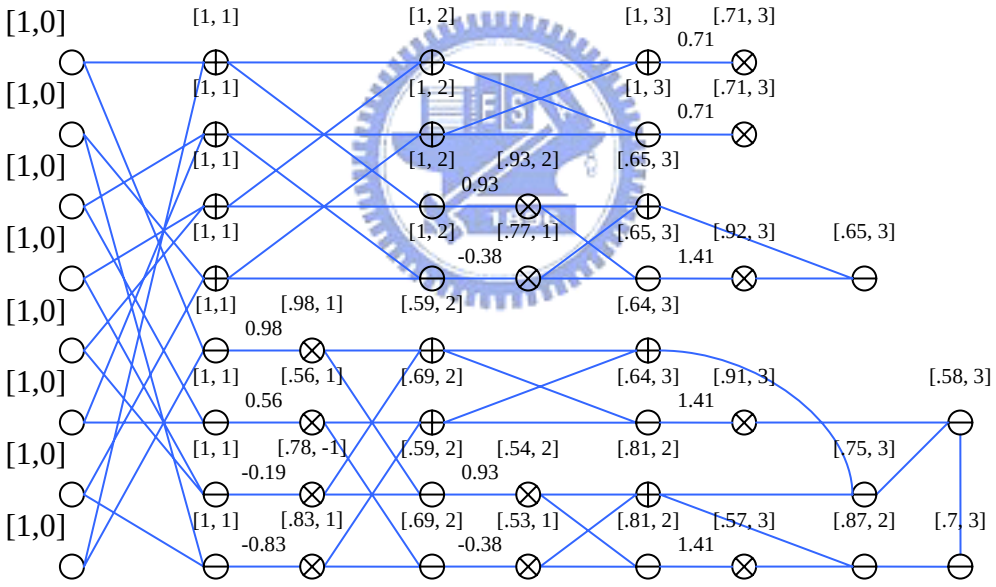


圖 2-20 Peak-value estimation result of 8x1 DCT data flow

在硬體成本和計算效率的考量下，靜態比率的運算因為能以靜態安排計算排程。比動態比率的運算更適合整合於 SIU 的資料路徑中。

最後，必需附帶一提的是 SIU 是資料流向的控制，和資料的形態並無關係。因此不論任何數字形態，只要使用對應的運算單元能支援該算數運算即可。在本論文的資料流設計中選用 SFP 的運算單元。因為它可以利用設定運算參數或少部分的程式修改而直接支援整數和非條件式 BFP 等輕量型的運算。

第3章　指令集規劃

在本章將介紹本論文所提出的對 SIU 資料流向的控制機制和指令集的規劃。最後以實際的例子介紹利用指令集合成 SIU 運算路徑的方法。

3.1.　SIU 資料流硬體架構

SIU 資料流的硬體主要可以分成三個部分分別控制：運算單元、暫存器檔和交換電路(crossbar；簡稱 Xbar)。在每個時脈中資料的流向為：暫存器檔把運算子的資料送給運算單元；運算單元把結果運算子送到交換電路；交換電路還擇運算的結果送回暫存器檔儲存。因此指令解碼器(instruction decoder)在每個時脈對都必需發出如圖 3-1 的三種控制信號。對運算單元：選擇運算子；對暫存器檔：選擇要送出的運算子資料；對交換電路：選擇要存回暫存器檔的運算結果。所以在指令集的設計上每種指令都必需包函這三種資訊。

資料流中包函四種功能的運算單元,每個運算元的延遲均設定為 2 個時脈(圖 3-2)：

- 控制單元：包函程式流程的控制和記憶體存取的功能。
- ALU 單元：可處理加減法和位元邏輯(bit logic)的運算。
- 乘法器單元：可做整數和純小數兩種乘法
- 位移器單元：處理位元位移(bit shift)的工作。

因此在指令中使用 VLIW(very long instruction word)的方式，指令長度為 64 位元，依運算單元分成四個欄位(field)，每個時脈發出四種指令平行處理。而在程式流程的控制方面使用延遲分叉(delayed branch)的方式控制，因此在分叉指令後的兩個指令還是會在程式分叉時被執行。

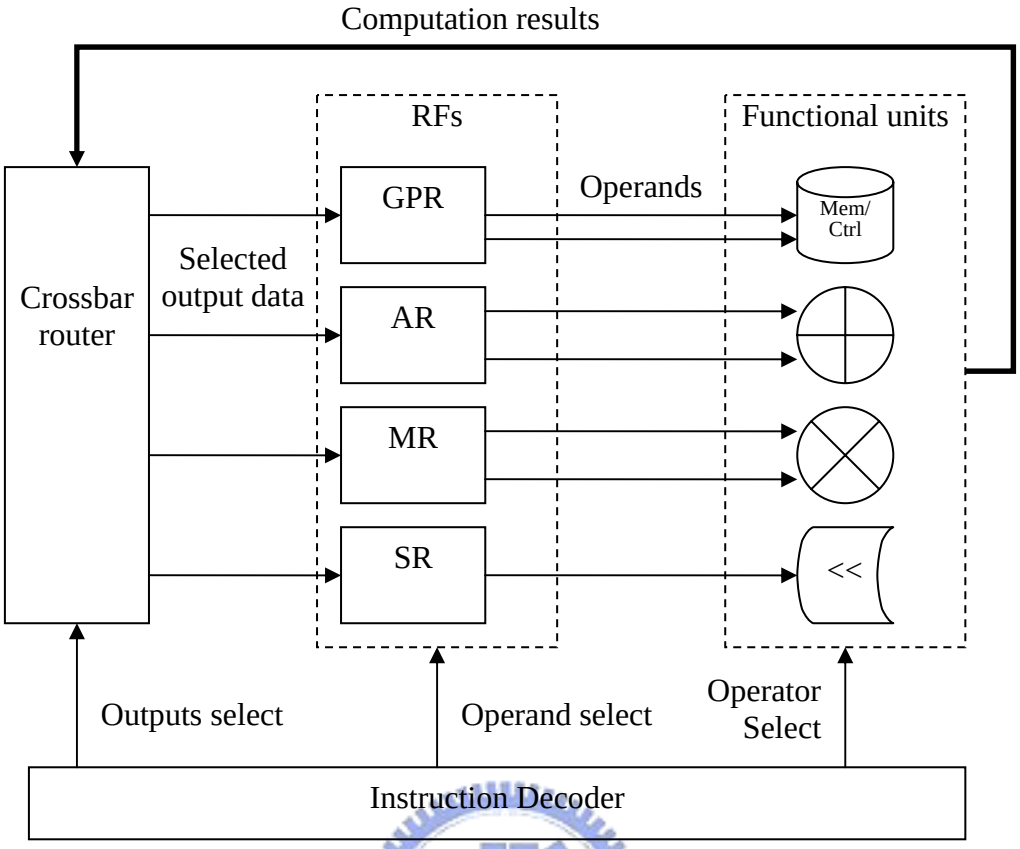


圖 3-1 Proposed control scheme of SIU datapath

在 DRF 的分配上，分配於前三個運算單元的是二讀一寫的 DRF，而位移器的專屬 DRF 為一讀一寫。其中暫存器的設置如下：

控制單元的暫存器稱為 GPR，可用於一般暫存或定址之用。其中有 10 個 16 位元之實體暫存器 R1 ~ R10 可供 IO 定址或暫存之用。另外 7 個為特殊用途或特殊定義的暫存器，和一個 4 位元的計數器和 2 位元的計數器位移暫存器。其定義和作用詳列如下：

- R0=0
- R11：定義為 ALU 單元的運算輸出 ALU unit output (Ao)
- R12：定義為乘法器單元的運算輸出 Multiplier unit output (Mo)
- R13：定義為位移單元的運算輸出 Shifter unit output(So)
- R14：定義為控制單元的運算輸出 Control unit output(Io)
- R15：程式計數器 Program counter (PC) 指向指令抓取 (instruction fetch)的位址
- counter：計數器。計數器的值會在索引定址(index addressing)或自動索引定址(auto index addressing)時與 R9 和 R10 相加，產生實體位址。計數器為 4 位元，只能在控制指令中重置(reset)或是索引定址時才能改變計數器的值，不能任意賦與數值也不能直接讀取。
- shifter register：位移暫存器有 2 位元，其作用在於指定計數器暫存器的位移量。換言之，就是計數器暫存器每次累加時，實體位址的增加量。當位移暫存器為 0 時，計數器每加一則表示指向下一

個 16-bit word 的位址，為 1 時則表每次跳 2 個 word，以此類推，最大值 3 時則一次跳 8 個 word 的距離。

而在其他三個運算單元的專屬暫存器檔中都各為 16 個 16 位元的暫存器，功能均作為暫存之用。其名稱各別為 AR0～AR15、MR0～MR15、SR0～SR15。

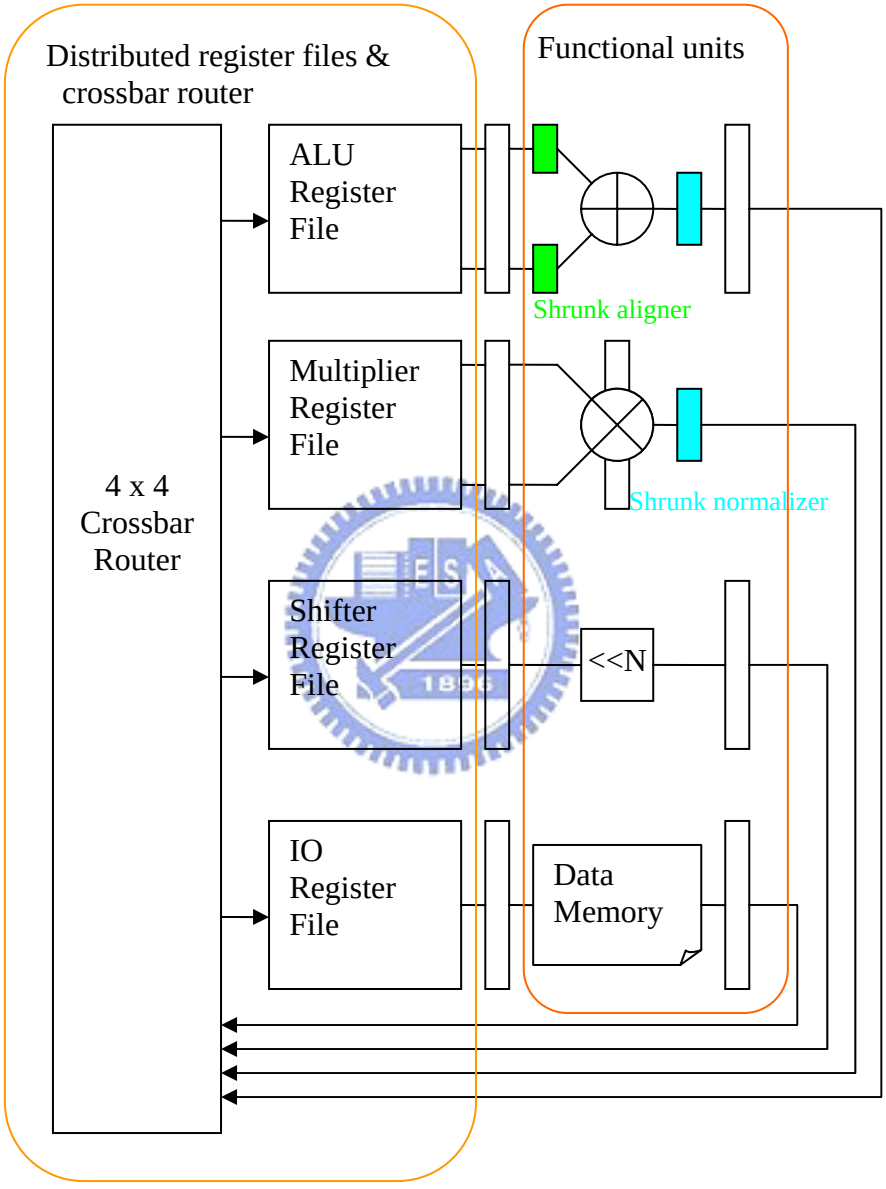


圖 3-2 functions block of the basic system platform of proposed ISA

3.2. 指令集架構(ISA)

在本節，只簡略介紹指令集的所有指令及其支援的運算功能，至於指令詳細的格式及編碼(opcode)將於附錄中補述。指令結構為四路 VLIW (4 way VLIW)，每個時脈都會發出以下四種運算單元的指令：

3.2.1. 控制單元：

控制單元欄位的長度為15位元,同時控制load/store單元和GPR更新。在Load/Store單元中可以使用的功能有：

- LW:load word

  指令格式為：LW R_s [Z][S shift]*

  - R_s：

    當 R_s 為 R1~R8 或 R11~R14 時為間接定址(register indirect addressing)，指令會把data_memory(R_s)的值送到Xbar。

    當R_s為 R9 或R10 時為索引定址或自動索引定址，指令會把data_memory(R_s+ (counter << shift))的值送到Xbar。當使用自動索引定址時counter在定址後會自動增加。

    當R_s為R0時不動作。
  - Z：重置計數器，計數器會在下個時脈被重置為0。
  - S shift:當使用S參數時，會把位移暫存器的值設定為為shift的值。
- SW：store word

  指令格式為：SW R_s R_t

  - R_s R_t：

    R_s的定址模式和LW相同，定址時會把R_t的值寫入定址的位址，同時會把R_t的值送到Xbar。

和控制單元同一欄位的還有控制其專屬暫存器更新的指令及流程控制的指令：

- register update:

  指令格式為：Os R_s

  指示Xbar選擇一個運算結果更新(update) R_s的內容。
- branch：

  指令格式為：Os [J | JZ | JNZ | JC]

  - J, JZ, JNZ, JC: 為程式流程的控制，自前至後分別表是branch, branch if zero, branch if NOT zero, branch if carry。這裡所使用的zero或carry的條件判斷均使用ALU單元的運算結果來做依據。會把R15(PC)的值更新為Xbar選擇的結果。

3.2.2. ALU單元：

本欄位長度為22位元，且支援SFP運算。當運算元中的參數為設為0時，(如：(+,000))就和一般整數的運算的加減法相同。因此在指令的設計中運算元並不特別指定是整數運算或純小數運算,只由運算元的參數值來做決定。功能上，ALU可使用加減法及位元邏輯運算：

- 加、減法：把兩個運算子相加(減)送到xbar，指令格式為：

  [ADI|SUI] Imm ARx_2 parameter

  [ADD|SUB] ARx_1 ARx_2 parameter

  [ADL|SUL] Os AR ARx_1 ARx_2 parameter

*[ ] 表示可省略的參數或指令。

[|] 表可選擇性的參數或指令，可選用括號內其中一種指令。

- ◆ ARx_1 ARx_2：
  可選用 AR0~AR15、及 R0，Ao，Mo，So，Io 等 21 種數值。
- ◆ Imm：immediate value
- ◆ parameter：有 3 個位元，為 2.4 節中為 SFP 定義的加法運算元參數。
- ◆ [ADL|SUL]：在做加(減)法同時更新暫存器檔的值
  指示 Xbar 選擇一個運算結果更新 AR 內容，其中 AR 可指定 AR0~AR15 等 16 個暫存器。

3.2.3. 乘法器單元：

本欄位長度為 12 位元，可控制乘法器及乘法器的 DRF，但是因為欄位長度的限制，在此欄位中只能控制運算單元或 DRF 其中一個的動作。即在同一個時脈中，只能做一次乘法或更新一次乘法暫存器，兩者無法並行。指令格式為：

- ■ 乘法：支援整數與純小數的有號數乘法，把兩個運算子相乘的結果送到 Xbar。指令格式為：
  [M|fM] MRx_1 parameter MRx_2
  - ◆ [M|fM] parameter：M 為整數乘法；fM 為純小數乘法。1 位元的 parameter，為 2.4 節中為 SFP 定義的乘法運算元參數。
  - ◆ MRx_1 MRx_2：
    可選用 MR0~MR15、及 R0，Ao，Mo，So，Io 等 21 種數值。
- ■ 暫存器更新的指令格式為：
  [LM Os|uM] MR
  - ◆ LM 指令為更新暫存器，指令會用 Os 選擇的結果更新 MR，其中 MR 可指定 MR0~MR15 等 16 個暫存器。
  - ◆ uM：本指令不做乘法亦不做暫存器更新，只會把乘法暫存器 MR 的值直接送到 Xbar。

3.2.4. 位移器單元：

本欄位長度為 15 位元，功能為被動式的位移的功能。

- ■ 位移器的運算子一個為暫存器，另一個為一個立即值(immediate value)，會把暫存器的值依立即值做位移的動作，指令格式為：
  [S|SLA|SLM|SLI] SRx_1 Imm SR
  - ◆ [S|SLA|SLM|SLI] S 為只做位移，不更新暫存器；SLA，SLM，SLI 除位移外，還會把 Ao，Mo 和 Io 的值更新到 SR 的暫存器。
  - ◆ SRx_1：
    可選用 SR0~SR15、及 R0，Ao，Mo，So，Io 等 21 種數值。

全部的運算單元的功能整理如下表 3-1。整體來看運算單位的功能並無對 DSP 運算做任何特別的設計，但是此 DSP 處理器適合於 DSP 計算的重點在於 DSP 運算路徑的合成，而非運算單元的功能本身。在下一節中，將以實例來解說怎麼將 SIU 的觀念和這些運算單元整合成 DSP 的計算路徑。

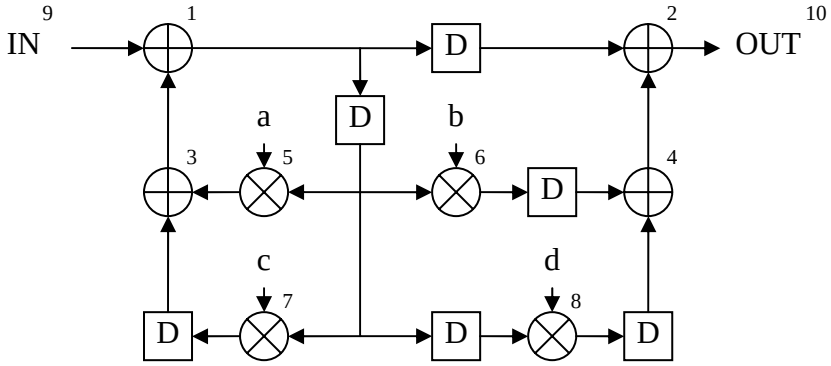
Contol unit	15bits
Memory access	LW, LW Z, LW S, LW ZS LW R9, LW R10
	SW, SW R0 SW R9, SW R10
Register update	Ao, Mo, So, Io
Program control	J
	JZ, JNZ, JC
ALU	22bits
Arithmetic	ADD, SUB, ADL, SUL
	ADI, SUI
Logic	AND, OR
	ANL, ORL
Multiplier	12bits
Arithmetic	M
	fM
Data pass	UM
Register update	LM
Shifter	15bits
Shift	SLA, SLM, SLI
	S

表 3-1 instruction summary

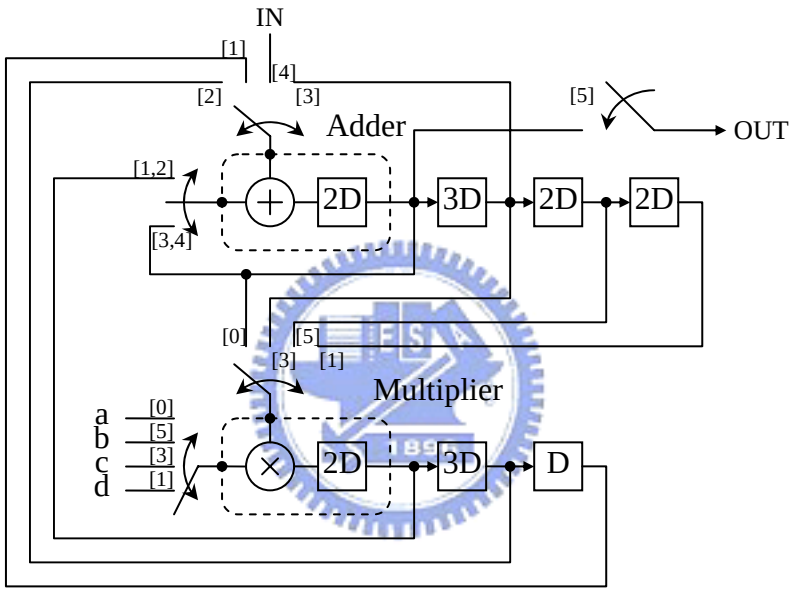
3.3. 以資料流向為主的程式化方法

要快速達成 DSP 的運算，最有效的做法莫過於使用類似於 ASIC 的做法，針對特定的 DSP 運算，直接依照 DFG (Data Flow Graph)設計一個專門的硬體資料路徑[21]。但這樣的設計只能做特定的 DSP 運算，要使用的運算單元也多。而在資料路徑的設計中，常使用 folding transformation 的技巧：利用切割時間軸，使原本需要很多運算元的 DFG 的運算只需要用單一的運算元，和若干的時間間隔(time slot)就能達成。

如：圖 3-3(a)為一個二階 biquad filter 的 DFG。要將此濾波器寫成資料流的控制程式時可把該 DFG 先做類似 folding 的轉換，因此 folding 轉換後可得圖 3-3 (b)的 SIU 硬體電路和在每個時脈時多工器的選項。而其中 folding 的過程為：



(a) biquard filter after retiming (folding factor=4)



(b) Folded biquard filter

圖 3-3 Folding a biquard filter

- 排程運算單元的佔用時間(operation scheduling & allocation)

  把所有出現的運算元編號，並把每個執行佔用的時間設為變數。如圖中把四個運算元從 1～10 編號，因此每個運算元的執行時間則為 T[1..10]。

- 延遲級數的計算(delay calculation)

  以延遲時間設定方程式如 $D_f(1,2)$即為把運算元 2 執行的時間和運算元 1 執行後所產生的結果運算子的時間相減所得的值。在每個結果運算子產生資料的時間就是該資料的出生時間，而當資料最後被送到該送的地方運算時就是資料的死亡時間。因此式子 $D_f(1,2)$所代表的物理意義就是由 1 號運算元所產生用來當 2 號運算元的運算子的這筆資料的生命週期。

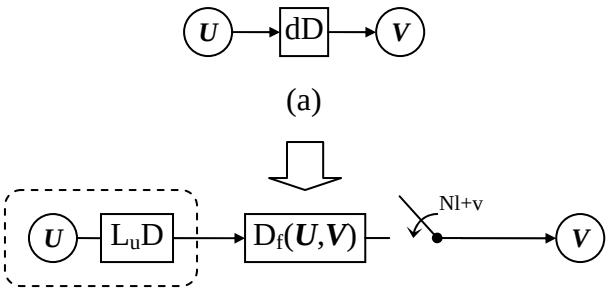


圖 3-4 Delay calculation

以數學式表示時：如圖 3-4，設 DFG 中運算元 *U, V* 的執行時間為 u, v，運算元延遲(latency)為 L_u, L_v，延遲數為 d。

當圖 3-4(b)等效於圖 3-4(a)時：

iff: $D_f(U, V) = [N(1+d)+v] - [Nl+L_u+u]$

$D_f(U, V) = Nd - L_u + v - u$

如上例中 $D_f(1,2) = 4*1-1+T[2]-T[1]=3+T[2]-T[1]$

- 解前後相依性方程組(causality transformation)

在此分析生命週期的方法與 3.1 節中靜態的分析方法有所不同。因為每個運算元的執行時間都代表著其輸入運算子的死亡時間和其結果運算子的出生時間。出生和死亡的順序會隨著運算元排程的不同而改變，並非如 3.1 中是固定的值。因此必需列出所有延遲方程式的相依性方程式，並解出此聯立不等式方程組的解，使得所有的 $D_f(U,V) \geq 0$。(意即：資料使用的時間必需在其產生的時間之後)

如上例中相依性方程組即為：

$$\begin{cases} D_f(1,2) = N-2+T_2-T_1 \geq 0 \\ D_f(1,5) = N-2+T_5-T_1 \geq 0 \\ D_f(1,6) = N-2+T_6-T_1 \geq 0 \\ D_f(1,7) = N-2+T_7-T_1 \geq 0 \\ D_f(1,8) = 2N-2+T_8-T_1 \geq 0 \\ D_f(3,1) = 0-2+T_1-T_3 \geq 0 \\ D_f(4,2) = 0-2+T_2-T_4 \geq 0 \\ D_f(5,3) = 0-2+T_3-T_5 \geq 0 \\ D_f(7,3) = N-2+T_3-T_7 \geq 0 \\ D_f(6,4) = N-2+T_4-T_6 \geq 0 \\ D_f(8,4) = N-2+T_4-T_8 \geq 0 \\ D_f(9,1) = 0-2+T_1-T_9 \geq 0 \\ D_f(2,10) = 0-2+T_{10}-T_2 \geq 0 \end{cases}$$

表 3-2 相依性不等式方程組

- 產生 SIU (SIU generation)

以求得的解即可產生 SIU 的資料路徑以及計算暫存器的使用量，暫存器的生命週期(register life time)及分派暫存器。

經由以上 folding 方法可以找到一組運算單元的排程[4]。因此，理論上只要把運算單元按照 folding 的結果安排的話就可以把所有能夠表成 DFG 的運算映射到 SIU 的資料流。

3.3.1. 檢查 ISA 的限制

在把求得表 3-2 的解後檢查以下 ISA 和硬體的限制才能確定所求得之解是否可以改寫成程式：

(1) 暫存器的使用量：硬體架構中暫存器的數量是有限的，而 SIU DSG 中暫存器的用量是經由生命週期分析而來取決於原始 DFG 的大小及資料的相依性。因此 16 個暫存器的硬體架構只能實現一定大小以內的 DFG，對於太大的 DFG 則必需先用人工區分成幾個較小的 DFG 再分別執行。

(2) 存取埠(IO port)的限制：在 DRF 中每一個 RF 都只有一個寫入埠，所以在 SIU 中如果有兩個以上的運算單元要在同一個 time slot 寫入同一個 DRF 時，這樣就會出現埠數不足的情形。

(3) 乘法器和乘法器的專屬暫存器不能同時工作：這並非是硬體架構的限制，而是在設計指令集時為了將指令長度簡化到 64 位元以內而做的取捨。正如 3.2.3 所言，在指令中使用 M 或 fM 的指令時 MR 的資料就無法被更新，反之亦然。

因此在繼續改成程式碼前必需先考慮硬體架構或 ISA 這些限制的部分。

繼續之前的例子，求解表 3-2，當 N=6 時可得一組如下的解滿足表 3-2 的不等式方程組：

T[1..4]=[4 3 2 1],
T[5..8]=[0 5 3 1]
T[9 10]=[2 5]

把求得之解再代回表 3-2 驗証可得表 3-3(a)的聯立方程組，在表 3-3(a)中，最後結果等於 0 的式子在指令中都可以用 bypass 的方式從輸出結果直接 bypass 到下一個運算單元當輸入運算子，只有結果不等於 0 的式子所代表的資料才需要存入暫存器，因此只需要檢查這些不等於 0 的式子是否有以上的限制的情形即可。以表 3-3(a)的生命週期的式子和 T[1..10]的解，可以得到表 3-3(b)的生命週期分析圖。在表 3-3(b)把表 3-3(a)中不等於 0 的式子在資料的出生時間標上"I"；當資料要送到運算單元時標上"O"。

由表 3-3(b)可以直接看出 AR 的暫存器需求為 3，MR 為 2。並未超出 16 個暫存器的限制。

接著檢查每個 time slot 中在每個 DRF 中標示"I"的數量，不能超過一個。在表 3-3(b)中，AR 在 time slot=0,3,5 時都必需用到 DRF 的輸入埠，MR 在 time slot=0 時會用到輸入埠。且都只有 1 個，所以第(2)項限制也沒問題。

最後檢查 MR 的部分，"I"和"O"不能在同一個 slot。檢查的結果發現：在 time slot=0 時，"I"和"O"會同時出現，因此第(3)項限制就發生了。

如果沒有發生限制的情形就可以接著把 SIU 的解寫成指令集的程式。在

下節將繼續介紹消除這些限制和改寫成指令的方法。

$$
\begin{cases}
D_f(1,2) = 6-2+T_2-T_1 = 3 \\
D_f(1,5) = 6-2+T_5-T_1 = 0 \\
D_f(1,6) = 6-2+T_6-T_1 = 5 \\
D_f(1,7) = 6-2+T_7-T_1 = 3 \\
D_f(1,8) = 12-2+T_8-T_1 = 7 \\
D_f(3,1) = 0-2+T_1-T_3 = 0 \\
D_f(4,2) = 0-2+T_2-T_4 = 0 \\
D_f(5,3) = 0-2+T_3-T_5 = 0 \\
D_f(7,3) = 6-2+T_3-T_7 = 3 \\
D_f(6,4) = 6-2+T_4-T_6 = 0 \\
D_f(8,4) = 6-2+T_4-T_8 = 4 \\
D_f(9,1) = 0-2+T_1-T_9 = 0 \\
D_f(2,10) = 0-2+T_{10}-T_2 = 0
\end{cases}
$$

(a) 運算子生命週期

Cycle		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
Time slot		0	1	2	3	4	5	0	1	2	3	4	5	0	1	2	3	4	5	0	1	2	3	4	5
AR	AR0	I			O			I			O			I			O			I			O		
	AR1			O			I			O			I			O			I			O			I
	AR2		O		I				O		I				O		I				O		I		
GP																									
MR	MR6	IO			O		O		O					IO			O		O		O				
	MR5		O					IO			O		O		O					IO			O		O

(b) 生命週期分析表與暫存器分配

表 3-3 暫存器生命週期分析

3.3.2. 排除限制的情形

當發生限制時最直覺得解決方法就是：

(1) 增加 N 的數值，尋求其他解：當驗証限制發生時最直接的方法就是回到不等式方程組(表 3-2)，當 N 的值增加時，方程組的解也會隨之增加，直到找到一組沒有發生限制的解為止。如本例中，當 N=7 時可以找到一組解：

T[1..10]=[5 4 3 2 1 6 4 2 3 6]

這組解在檢查後就沒有限制發生。可以直接以這個排程改寫成程式。但是這樣會讓降低計算速率。因為以 SIU 的生命周期分析本來可以 6 個時脈完成一次濾波器的計算，現在卻必需以 7 個時脈才

能完成。

(2) 尋找可替代的路徑：(1)的方法必可以找到一組解，但是會降低效率。如果可以找到替代的路徑(datapath)可以在資料"出生"時幫忙做暫存的動作，當資料運算時再由 bypass 的方法傳送給運算單元，如此就可以在不增加 N 的情形下解決問題。可以當成替代路徑的條件如下：

- 在發生限制時的 time slot，必需要有空閒輸入埠可以替代接收發生限制的資料。
- 在替代的資料需要運算的前 2 個時脈，該替代的運算單元必需是空閒的。

滿足以上兩個條件時，才可以借用該運算單元的路徑。所以並不是每次都能找到替用的路徑。但是，一旦可以找到替用的路徑時，那麼就不必靠延長時脈來求解，會是較有效率的方法。

繼續上例，在表 3-3(b)中依暫存器使用量做暫存器分派(register allocation)。分派完暫存器後由表 3-3(b)中雙線的中間部分，可以映射成以下 unroll 兩層迴圈的運算單元排程：

```
loop:
                |A   AO AR0 R0  R0      |M  MR1 R11 AO  MR5        ; //slot=0 cycle=0
                |A          R12 AR2     |M  MR4 MR6                ; //slot=1 cycle=1
LW              |A          R12 AR1     |                          ; //slot=2 cycle=2
                |A   MO AR2 R11 AR0     |M  MR3 MR5                ; //slot=3 cycle=3
                |A          R11 R14     |                          ; //slot=4 cycle=4
SW R11          |A   MO AR1 R0  R0      |M  MR2 MR5                ; //slot=5 cycle=5
                |A   AO AR0 R0  R0      |M  MR1 R11 AO  MR6        ; //slot=0 cycle=6
                |A          R12 AR2     |M  MR4 MR5                ; //slot=1 cycle=7
LW              |A          R12 AR1     |                          ; //slot=2 cycle=8
                |A   MO AR2 R11 AR0     |M  MR3 MR6                ; //slot=3 cycle=9
                |A          R11 R14     |                          ; //slot=4 cycle=10
SW R11          |A   MO AR1 R0  R0      |M  MR2 MR6                ; //slot=5 cycle=11
```

若使用(1)的方法時則 N=7 的排程會如下所示：

```
loop:
                |A   AO AR0 R0  R0      |LM AO  MR5                ; //slot=0 cycle=0
                |A   MO AR3 R0  R0      |M  MR1 MR5                ; //slot=1 cycle=1
                |A          AR3 AR2     |M  MR4 MR6                ; //slot=2 cycle=2
LW              |A          R12 AR1     |                          ; //slot=3 cycle=3
                |A   MO AR2 R11 AR0     |M  MR3 MR5                ; //slot=4 cycle=4
                |A          R11 R14     |                          ; //slot=5 cycle=5
SW R11          |A   MO AR1 R0  R0      |M  MR2 MR5                ; //slot=6 cycle=6
                |A   AO AR0 R0  R0      |LM AO  MR6                ; //slot=0 cycle=7
                |A   MO AR3 R0  R0      |M  MR1 MR6                ; //slot=1 cycle=8
                |A          AR3 AR2     |M  MR4 MR5                ; //slot=2 cycle=9
LW              |A          R12 AR1     |                          ; //slot=3 cycle=10
                |A   MO AR2 R11 AR0     |M  MR3 MR6                ; //slot=4 cycle=11
                |A          R11 R14     |                          ; //slot=5 cycle=12
SW R11          |A   MO AR1 R0  R0      |M  MR2 MR6                ; //slot=6 cycle=13
```

若要使用(2)的方法時，檢查(2)中的兩個條件。當 time slot=0 時，MR 的輸入埠受到限制，但是 SR 的輸入埠在 time slot=0 時是閒置的，另外

當 time slot=1,3,5 時，乘法器需要該筆資料，因此位移器必需在 time slot=5,1,3 時把資料送出來才能經由 xbar bypass 給乘法器。檢查位移器在這 3 個 time slot 亦為閒置的，因此可以使用位移器做替代的路徑，因此把原來程式中，MR5 和 MR6 的值以 SR0 和 SR1 替代。再把運算單元的排程寫成指令的格式。因此原程式就會變成以下主要迴圈的程式碼：

```
//assign: R9= input address pointer; R10= output address pointer
//change: MR5 -> SR0; SR6 -> SR1

loop:
                |ADL AO AR0 R0  R0   000|M  MR1 R11 0|SLA SR0       ; //slot=0 cycle=0
                |ADD        R12 AR2  000|M  MR4 R13 0|S       SR0 0 ; //slot=1 cycle=1
LW R9           |ADD        R12 AR1  000|            |              ; //slot=2 cycle=2
                |ADL MO AR2 R11 AR0  000|M  MR3 R13 0|S       SR0 0 ; //slot=3 cycle=3
                |ADD        R11 R14  000|            |              ; //slot=4 cycle=4
SW R10 R11      |ADL MO AR1 R0  R0   000|M  MR2 R13 0|S       SR0 0 ; //slot=5 cycle=5
                |ADL AO AR0 R0  R0   000|M  MR1 R11 0|SLA SR1       ; //slot=0 cycle=6
                |ADD        R12 AR2  000|M  MR4 R13 0|        SR1 0 ; //slot=1 cycle=7
LW R9           |ADD        R12 AR1  000|            |              ; //slot=2 cycle=8
                |ADL MO AR2 R11 AR0  000|M  MR3 R13 0|        SR1 0 ; //slot=3 cycle=9
                |ADD        R11 R14  000|            |              ; //slot=4 cycle=10
SW R10 R11      |ADL MO AR1 R0  R0   000|M  MR2 R13 0|        SR1 0 ; //slot=5 cycle=11
```

在原本乘法器的暫存器中必需在 time slot=0 時取得 Ao 的值，所以改在 cycle=0 時由 SR0 來承接，當乘法器在第 3,5 和 7 個時脈要用到這個值時，位移器必需在第 1,3,5 個時脈把值送出來，再經由 crossbar router 的 R13 直接 bypass 到乘法器當運算子。而此迴圈所表示的實際硬體資料流即如圖 3-5。

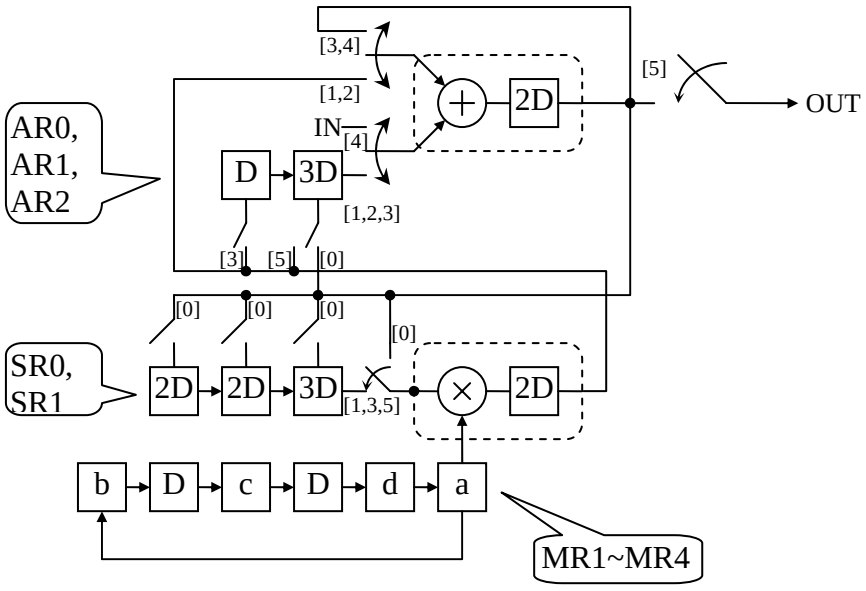


圖 3-5 SIU with input queue style

在主要的迴圈寫好後，最後加上迴圈控制的程式碼：加入 IO 所需要實體的記憶體位址的運算和一些迴圈的所需要的指令就算完成，在此假設輸入資料的位址已經存於 R9，要輸出資料的位址存於 R10，而迴圈 loop:的位址存於 R1，以及每次迴圈位址的增加量存於 AR15。因此上面的程式就可寫成：

```
//assign: R9= input address pointer; R10= output address pointer
//change: MR5 -> SR0; SR6 -> SR1
assign: R1=loop; AR15=4(address increment per loop)
loop:
                |ADL        R0  R0   000|fM MR1 MR5 0|              ; //slot=0 cycle=0
                |ADD        AR3 AR2  000|fM MR4 MR6 0|              ; //slot=1 cycle=1
LW R9           |ADD        R12 AR1  000|            |              ; //slot=2 cycle=2
                |ADL MO AR2 R11 AR0  000|fM MR3 MR5 0|              ; //slot=3 cycle=3
                |ADD        R11 R14  000|            |              ; //slot=4 cycle=4
SW R10 R11      |ADL MO AR1 R0  R0   000|fM MR2 MR5 0|              ; //slot=5 cycle=5
                |ADL AO AR0 R0  R0   000|fM MR1 R11 0|SLA SR1       ; //slot=0 cycle=6
                |ADD        R12 AR2  000|fM MR4 MR5 0|        SR1 0 ; //slot=1 cycle=7
LW R9           |ADD        R12 AR1  000|            |              ; //slot=2 cycle=8
SW R0  R9       |ADL MO AR2 R11 AR0  000|fM MR3 R13 0|        SR1 0 ; //slot=3 cycle=9
SW R0  R10      |ADD        R11 R14  000|            |              ; //slot=4 cycle=10
SW R10 R11      |ADL MO AR1 R14 AR15 000|fM MR2 R13 0|        SR1 0 ; //slot=5 cycle=11
                |ADD AO AR0 R14 AR15 000|LM AO  MR5  ;
SW R0  R1  AO R9 |ADD MO AR3 R0  R0   000|LM SO  MR6  ;
LW R0  Z   AO R10;
LW R0      IO J  ;
```

在加入迴圈的控制後使得原本 12 個時脈的指令變成了 16 個時脈，換句話說，平均 8 個時脈可完成一次 biquard filter 的運算，比起 folding 方法的 6 個 time slot 多了兩個時脈。這是 unroll 兩層迴圈的情形，由於計數器暫存器最多可以支援 16 個 word 的資料循序讀寫的容量，所以在此 ISA 中最多可以 unroll 到 16 層迴圈。當 unroll 到最大 16 層的迴圈時，加上迴圈控制所花費的 4 個時脈被平分下來就只多了 0.25 個時脈。這樣就可以幾乎接近 folding 的最佳解的效率。

總結本節的程式化過程可以整理以下的流程圖(圖 3-6)。經過反覆的求解，檢查即可得到一個較有效率的排程。

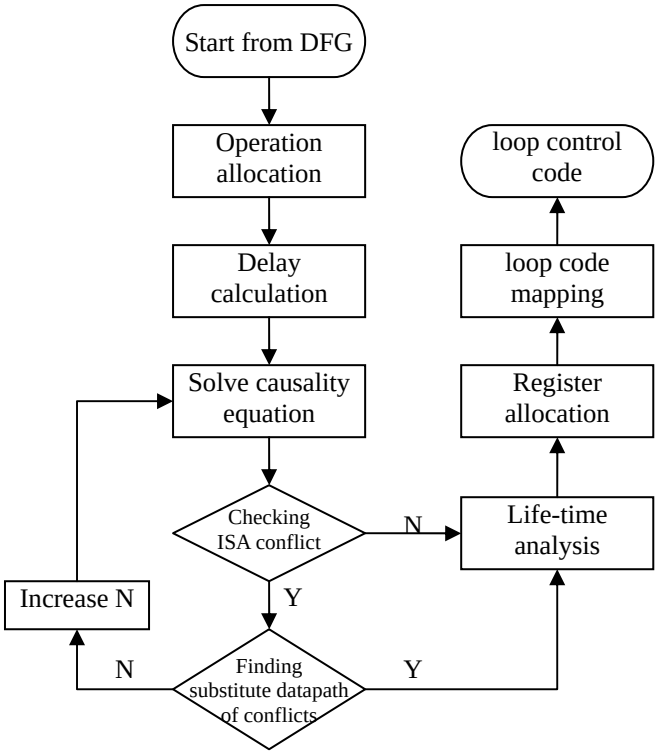


圖 3-6 Coding method flow

3.4. 結論

我們提出的 ISA 把暫存器和交換電路規化成 SIU DSG，讓運算單元在經過排程後可以全力投入 DSP 運算路徑的計算工作。這樣的規劃方法應用於靜態排程下會讓運算單元的利用變得十分有效率，即使運算單元增加，使用解方程組的方法一樣能解出平行度相當高的解，讓硬體的利用率提高，但 SIU DSG 並無法處理需要動態判斷決定運算方式的情形，遇到這樣的情形還是得靠程式流程或條件式的執行來處理。因此，在分類上此 ISA 架構十分適合處理資料量大且運算方式固定的 DSP 問題，對於以控制為主的一般 RISC 應用並無法得到良好的加速的效果。

第4章 硬體實現與效率比較

於上一章引述了 SIU DSG 的想法，並提出一組 VLIW ISA 和 DRF 的架構。在本章中，將以這樣的想法實作一個 VLIW DSP 處理器核心。由 3.1 的硬體架構中，每個欄位的指令解碼成微指令後必需有三種信號控制三種電路(圖 3-1)。

4.1. 微處理器的架構設計

4.1.1. 軟體效能驗証

首先，為了驗証指令集的完整性和效能，首先，我們以 C++寫成一個指令集模擬器(instruction set simulator；簡稱 ISS)。在這個 ISS 上我們試著將一些 DSP 的核心，如 DCT, FFT, filter 等等，以手動排程寫成程式碼。之後再將程式的計算結果和 Matlab 運算結果比較誤差值，確定 ISA 能確實合成 DSP 核心，且程式碼的功能也正確無誤。。

4.1.2. 硬體管線化的規劃

接著是指令管線化的分級。在運算單元分成兩級執行。並在後級的電路與交換電路整合成輸出選擇(output select)級。因此以一個完整的運算而言五級架構就如圖 4-1 所示。其中，每級所負責的工作簡述如下：

- IF: instruction fetch，程式碼在這一級由程式匯流排中讀入
- ID: instruction decode，在這一級將程式碼解碼微程式，並處理程式碼中的立即數值(immediate value)和計算索引定址的位址。

另外直接指示暫存器檔和交換電路把資料送到每個執行單元的輸入端中。

- EX: execution stage 1, 執行階段的第一級，在這個階段四個運算單元和四個暫存器都會同時做計算和更新資料的動作。只是在這一級所處理的資料更新所更新的資料為前兩個指令的運算結果。
- SEL: execution stage 2 and outputs selection, 這是運算執行的第二階段，同時會將運算結果由交換電路送到每個暫存器檔的輸入埠等待下一級存入暫存器中。
- WB: write back result, 將運算結果寫回輸入佇列的目的地。

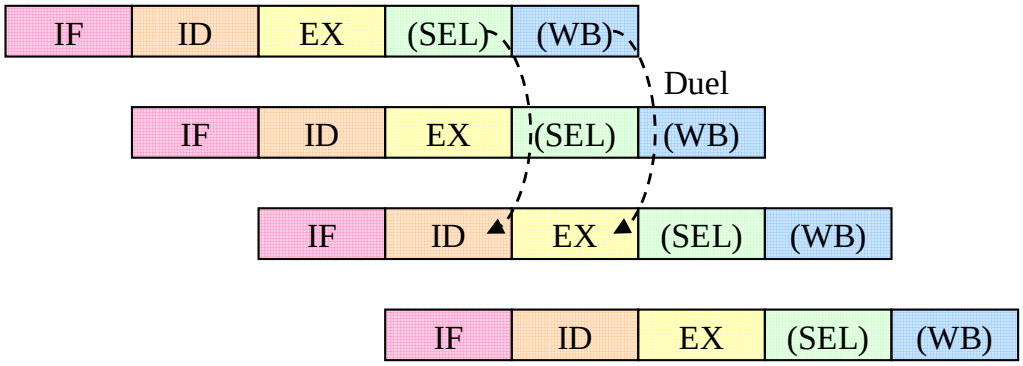
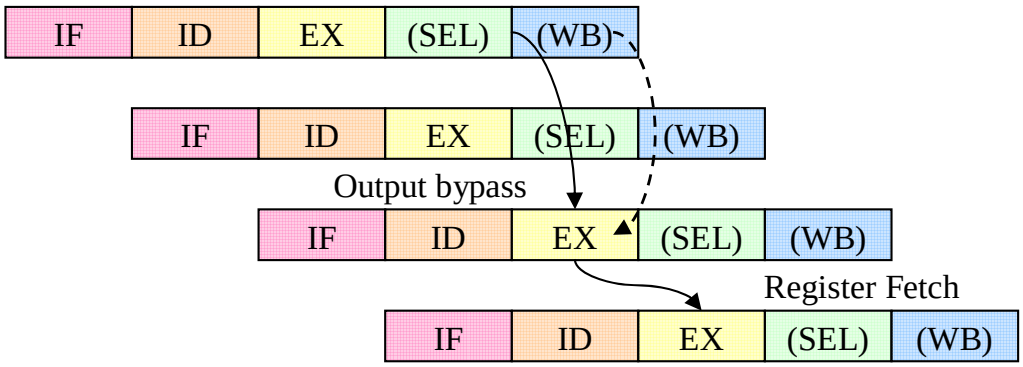
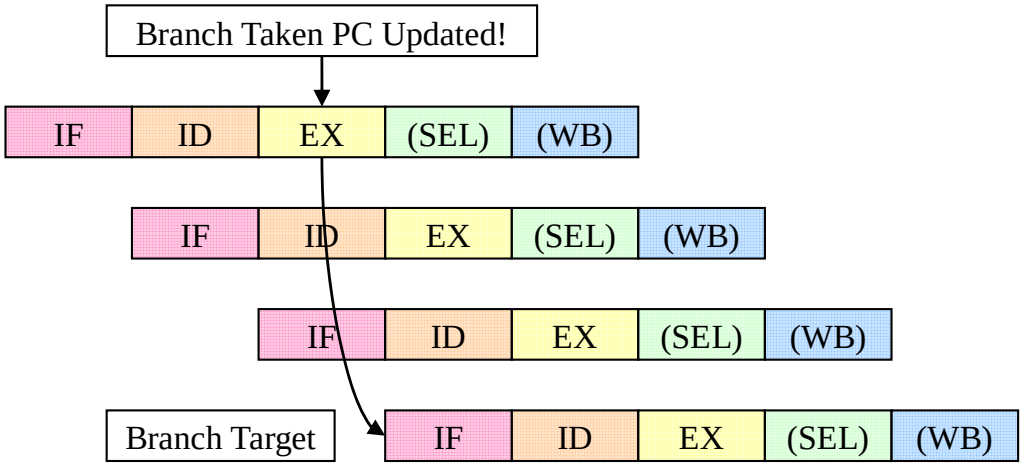


圖 4-1 Pipeline Execution Parallelism

以這樣的管線化分級後，原本的 ISS 就必需更改，SIU DSG 輸入端取得資料的時間就必需向後延遲 2 個時脈，因此在每次更新暫存檔時，所用來更新的資料實際上為兩個時脈以前運算元的運算結果。如圖 4-2(a)所示，SIU 要把輸入端的資料直接繞到(bypass)輸出端也需要在相對的運算後兩個時脈才能執行。但若已經寫入佇列的資料就不需要，可以在下一個時脈直接取用。因此以指令的方向來看，5 級的分析在每個指令中：IF、ID 和 EX 都是處理指令本身的資料，但 SEL 和 WB 所處理的資料是兩個時脈以前的計算結果。



(a) Data dependence in Pipeline stage



(b) Branch control in pipeline stage

圖 4-2 pipeline stage design

另外，有關程式流程的控制，因為 PC 即 R15，是專屬控制單元暫存器檔中的一員，因此在執行分叉(branch)的指令時，同樣的也會在兩個時脈後分叉的新位址才會送到暫存器中，因此在分叉指令後的兩個時脈的指令還是會被執行，這樣的方法稱為延遲分叉(delayed branch)的做法。總之，在分叉指令後的兩個指令都還是算在迴圈內的指令。IF 要在第三個時脈後才能取得分叉後新位址的程式碼。(圖 4-2(b))

在加入管線化後，程式必需再重新修正，因為在 SIU 中資料的輸入與輸出順序的變動都必需重新安排排程與驗証。

4.1.3. 微處理器架構

重新驗証完因管線分級造成輸入輸出順序的變動後的 SIU，之後就每一級電路所負責的工作定義輸入和輸出的規格，分別合成組合邏輯。

- IF:輸入為 16 位元 PC 值，輸出為 64 位元程式碼
- ID:ID 的工作分兩個部分：一為當時時脈的解碼器(ID)，和兩個時脈前的運算輸出的交換電路(SEL)
- 就解碼器的部分：輸入為程式碼，輸出為運算單元的微程式碼
- 就交換電路的部分：輸入為四個運算單元的運算結果、PC 值。輸出為所有運算單元的運算子、所有該送入暫存器檔的資料、和下一個 PC 值。
- EX:EX 的工作就運算單元和暫存器檔也有兩種工作：一為運算單元的運算(EX)，和把前兩個時脈的運算結果送入暫存器更新(WB)
  - 就運算單元部分：輸入為運算子，運算元種類，運算參數。輸出為運算結果
  - 就暫存器檔更新部分：輸入為要更新的資料，及存放的目的地，無輸出。

所有單元的 IO 規格詳繪圖 4-3。再整合所有的單元加上管線化暫存器(pipeline register)就可以得到完整的微處理器架構如圖 4-4。

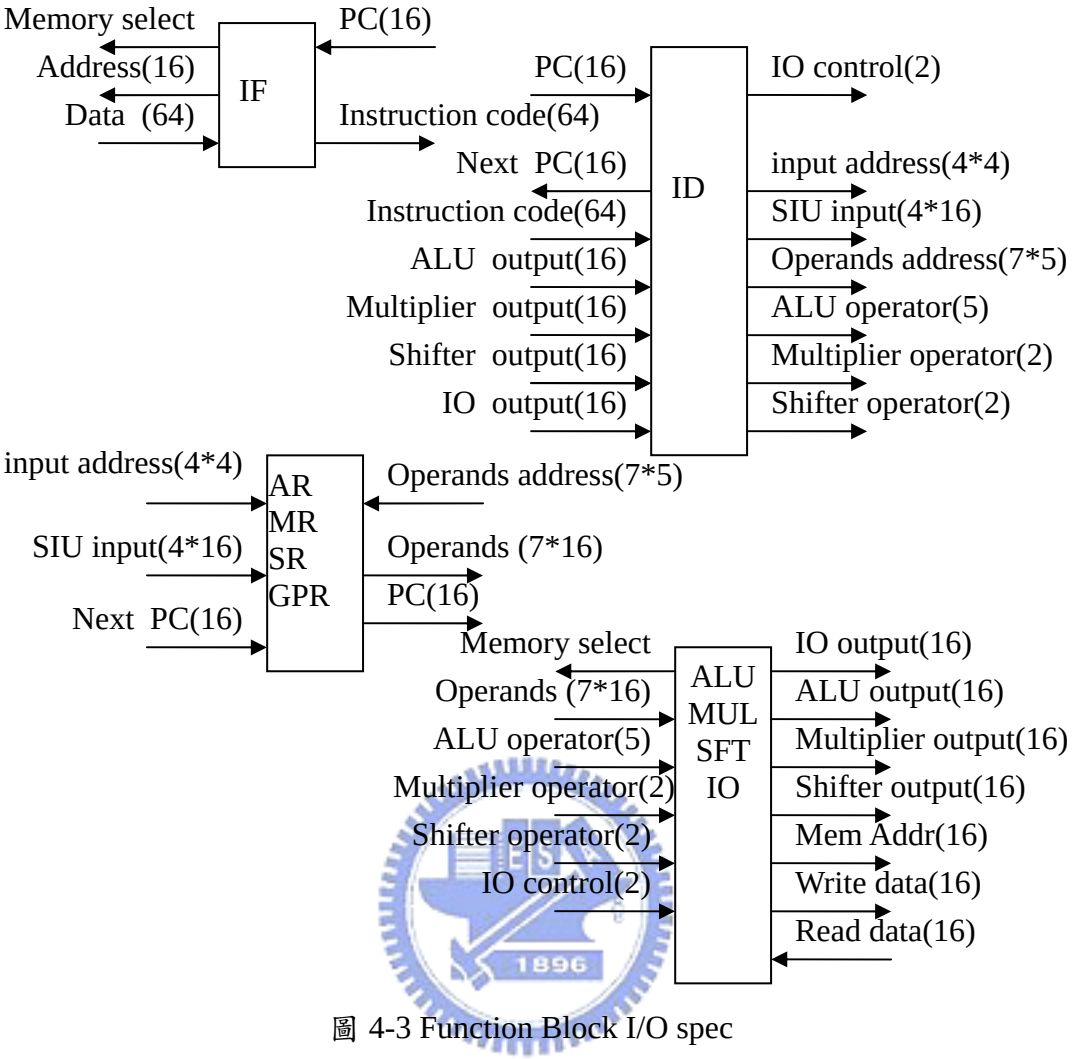


圖 4-3 Function Block I/O spec

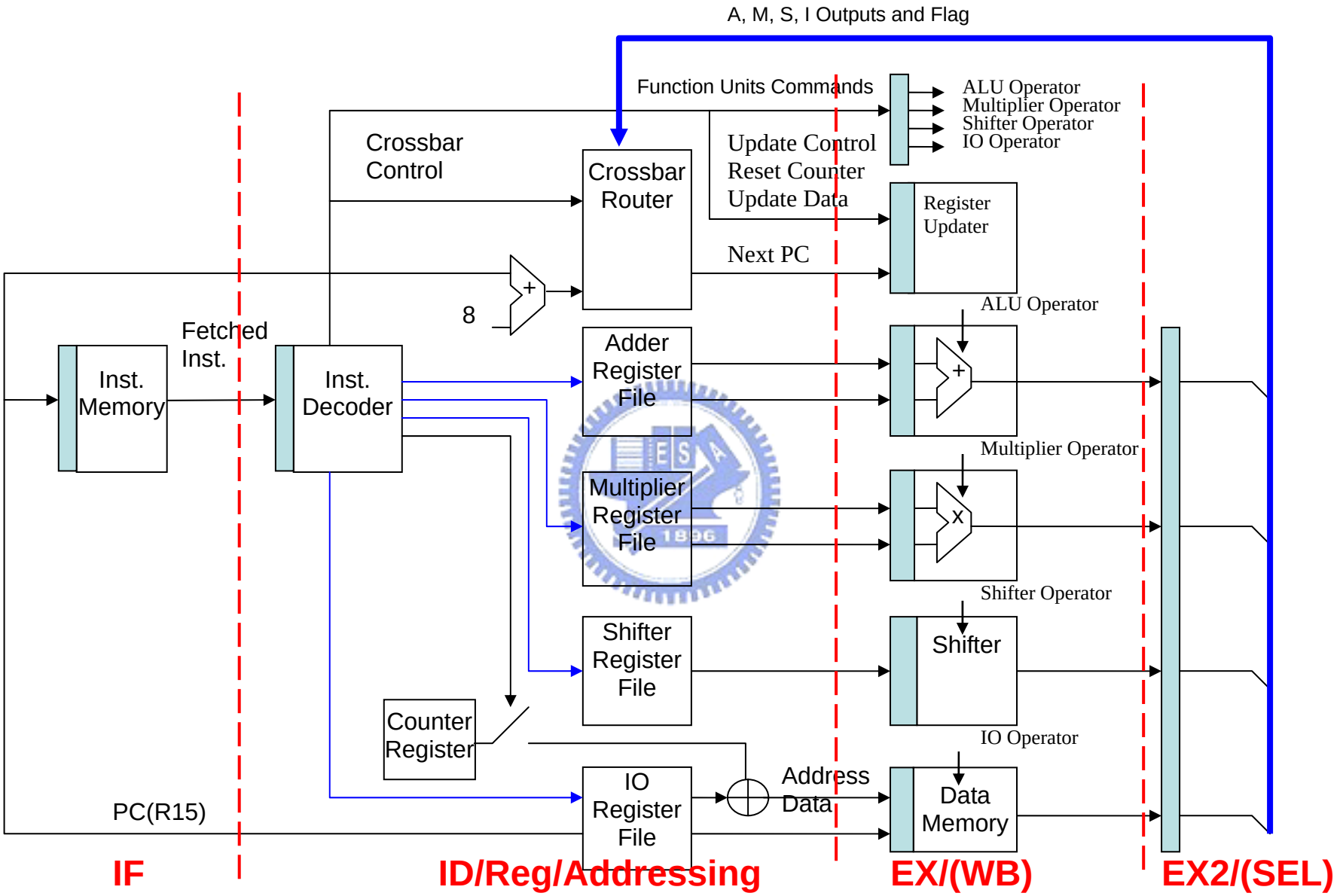


圖 4-4 Architecture Design

4.2. 硬體實現結果

本節將簡述以 UMC 0.18 cell-based library 和圖 4-5 的流程實做前一小節所規劃的架構的結果。在 4.1 由完成的架構設計，使用 ISS 做系統層次的驗証。驗証結果顯示，此 ISA 可以有效地把資料路徑對映到 SIU DSG。接著以 RTL code 去合成每個運算單元的組合電路和暫存器。並以 NC-Verilog 做 RTL 層次模擬執行所有程式的二進位碼。在 RTL 驗証無誤後使用 Synopsys design compiler 為合成工具把 RTL code 合成邏輯閘層次的電路，並再次以 Verilog 驗証邏輯閘電路的正確性。在邏輯閘的電路產生後，就可以概略估算電路時脈的速度。當邏輯閘電路驗証正確後，進入 Place & route 階段。使用 Cadence SoC encounter 為 APR 工具，最後再通過 post layout simulation 的驗証，DRC(design rule checking)和 LVS(layout vs. schematic)的檢查。最後 APR 的結果顯示於圖 4-6，其 IO 和晶片特性列於表 4-1。

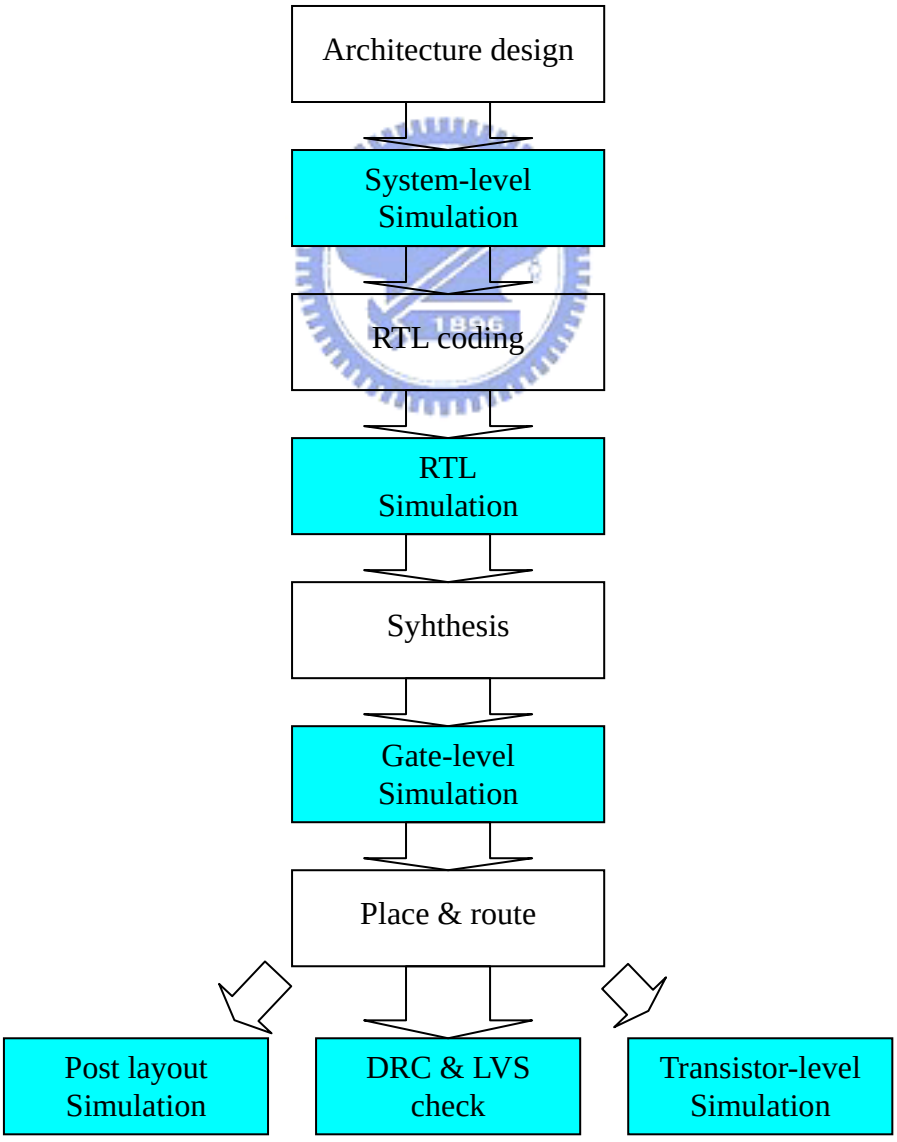


圖 4-5 Silicon Implementation flow

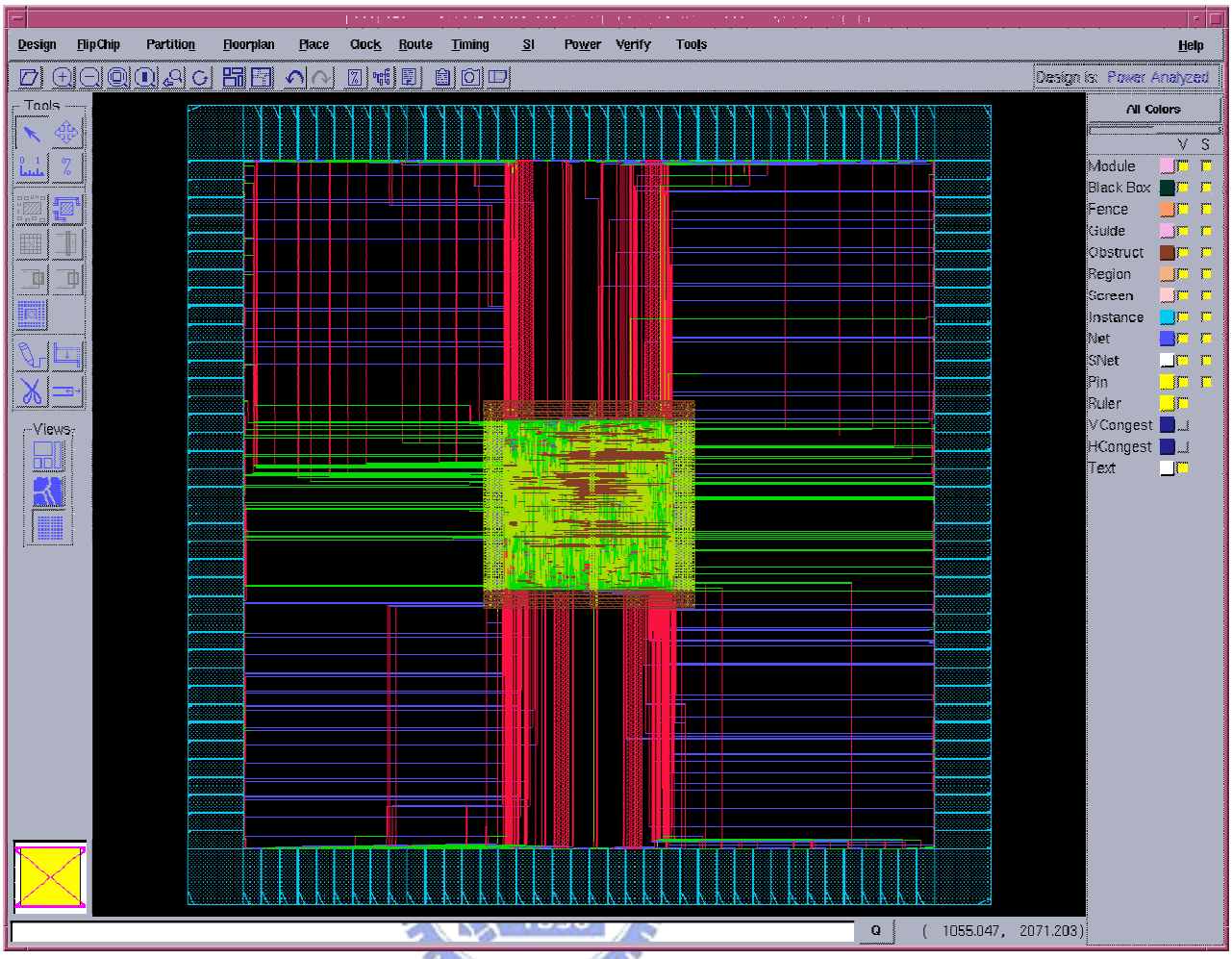


圖 4-6 Chip Layout

Name	Direction	Width(bits)	Description
CLK	Input	1	Clock input
RESET_	Input	1	Active low to reset RISC
BUSY_	Input	1	Active low to freeze RISC
Instruction memory interface			
Imem_select	Output	1	Instruction memory select
Imem_read	Output	1	Instruction memory read
Imem_addr	Output	16	Memory address
Imem_data	Input	64	Fetched instruction from memory
Data memory interface			
Dmem_select	Output	1	Data memory select
Dmem_read	Output	1	Memory read
Dmem_addr	Output	16	Memory address
Dmem_write_data	Output	16	Writing data to memory
Dmem_read_data	Input	16	Reading data from memory

(a) chip IO interface

Technology	UMC .18um CMOS
Core Size	0.6x0.6 mm^2
Chip Size	2.8x2.8 mm^2
Gate Count	25710
Power dissipation	37mW
Max frequency	268MHz

(b) Silicon Spec

表 4-1 Proposed DSP datapath silicon spec

4.3. 效率比較

為了比較 SIU 資料路徑對運算單元的利用效率，我們也勘查了一些市面上使用的運算單元與此架構相近的 DSP 處理器：

- ADSP-21xx 系列：使用一個加法器，一個乘加器，一個位移器均 16 位元，一個位址產生器，和一個雙向資料匯流排。
- TI C'55：使用兩個加法器(一個 16 位元，一個 40 位元)，兩個乘加器(17 位元)，一個位移器(40 位元)，一個位址產生器，和 5 個單向資料匯流排。

在運算資源方面，我們的原始架構和 ADSP-21xx 系列相近。而在 C'55 中，它的 40 位元加法器和位移器都可以調整成兩個 16 位元的運算單元來做 SIMD。因此總合而言，C'55 的計算的硬體資源約為兩倍。

在此我們以 DCT 和 FFT 這兩個轉換來比較運算的效率：

- DCT:

  Forward discrete cosine transform, DCT 是實數對實數間的轉換，為影像處理(如:JPEG,MPEG)的運算核心。數學式如下所示：

$$y(k) = h(k)\sum_{n=0}^{N-1} x(n)\cos(\frac{(2n+1)k\pi}{2N}), 0 \le k \le (N-1)$$

，其中 $h(0) = \sqrt{\frac{1}{N}}$ $h(k) = \sqrt{\frac{2}{N}}, 1 \le k \le (N-1)$

雖然以數學式來看 DCT 的複雜度為 N^2 但是因為乘法係數 cos() 的對稱性，可以簡化成 $N*\log_2 N$ 的複雜度。圖 2-20 即為簡化後 DCT 的運算路徑，在實際安排運算排程後可以得到以下的組合語言程式。程式為一個 8 點一維的 DCT 轉換，使用的算術方法為 SFP；其輸入信號為 16 位元的純小數、指數項為 0，以 SFP 表示其輸入的範圍即為±[1,0](相當於實數的±1)。輸出信號為 16 位元純小數，指數項為 3，以 SFP 表其範圍為±[1,3](相當於實數±8)。程式處理一次 8 點 DCT 需要 50 個時脈。在數學上，二維的 DCT 可以簡化為兩組橫向和縱向的一維 DCT。以此 ISA，可以在 705 個時脈內完成一次 8x8 二維 DCT 的轉換。

```
NOP                |OR           R0   1004     |NOP            |NOP                ;
NOP                |OR           R0   1014     |NOP            |NOP                ;
LW R0   ZS0 AO R10 |OR           R0   1024     |NOP            |NOP                ;
LW R10      AO R0  |NOP                         |NOP            |SLA  R0   0  SR15;
LW R10      AO R0  |NOP                         |NOP            |SLA  R0   0  SR14;
LW R10      AO R0  |NOP                         |LM IO   MR0    |NOP                ;
LW R10      AO R0  |NOP                         |LM IO   MR1    |NOP                ;
LW R10      AO R0  |NOP                         |LM IO   MR2    |S    SR15 0        ;
LW R10      AO R0  |NOP                         |LM IO   MR3    |NOP                ;
LW R10 Z    SO R10 |NOP                         |LM IO   MR4    |NOP                ;
LW R10      AO R0  |NOP                         |LM IO   MR5    |NOP                ;
LW R10      AO R0  |NOP                         |LM IO   MR6    |NOP                ;
LW R10      AO R0  |ADL IO AR0   R0    R0   000|NOP            |NOP                ;
LW R10      AO R0  |ADL IO AR3   R0    R0   000|NOP            |NOP                ;
LW R10      AO R0  |ADL IO AR1   R0    R0   000|NOP            |NOP                ;
LW R10      AO R0  |ADL IO AR2   R0    R0   000|NOP            |S    SR14 0        ;
LW R10      AO R0  |SBL IO AR6   R14   AR2  001|NOP            |NOP                ;
LW R10      SO R10 |SBL IO AR5   AR1   R14  001|NOP            |NOP                ;
NOP                |SBL IO AR7   R14   AR3  001|fM   R11 MR2  0|NOP                ;
NOP                |SBL IO AR4   AR0   R14  001|fM   R11 MR1  0|NOP                ;
NOP                |ADL MO AR6   AR2   AR6  001|LM AO    MR7   |NOP                ;
NOP                |ADL MO AR5   AR1   AR5  001|LM AO    MR8   |NOP                ;
NOP                |ADL AO AR2   AR0   AR4  001|fM MR8   MR0  0|NOP                ;
NOP                |ADL AO AR1   AR3   AR7  001|fM MR8   MR0  0|NOP                ;
NOP                |ADL AO AR0   R12   AR6  011|fM MR7   MR3  0|NOP                ;
NOP                |SUL AO AR3   R12   AR6  011|fM MR7   MR3  0|NOP                ;
NOP                |ADL AO AR4   R12   AR5  001|NOP            |NOP                ;
NOP                |SUB          AR5   R12  001|fM   R11 MR4  0|NOP                ;
NOP                |ADL AO AR5   AR0   AR2  001|NOP            |NOP                ;
NOP                |SUL MO AR6   AR0   AR2  001|fM   R11 MR5  0|NOP                ;
NOP                |ADL AO AR0   AR1   AR3  001|NOP            |NOP                ;
NOP                |SUL MO AR7   AR1   AR3  001|fM   R11 MR4  0|NOP                ;
NOP                |ADL AO AR1   AR4   AR5  001|NOP            |NOP                ;
NOP                |SUL MO AR2   AR4   AR5  001|fM   R11 MR5  0|NOP                ;
NOP                |ADL AO AR4   AR6   AR7  010|NOP            |SLA  R0   0  SR1   ;
NOP                |SUL MO AR3   AR6   AR7  010|fM   R11 MR6  1|NOP                ;
NOP                |ADL AO AR6   AR0   AR1  001|NOP            |NOP                ;
NOP                |SUL MO AR5   AR0   AR1  001|fM   R11 MR6  0|NOP                ;
NOP                |ADD          AR2   AR3  011|fM   R11 MR6  0|NOP                ;
NOP                |SUL MO AR7   AR2   AR3  011|fM   R11 MR6  0|S    SR1  0        ;
SW R10 R12 AO R2   |SUL AO AR2   AR6   AR4  000|NOP            |NOP                ;
SW R10 R13 MO R4   |SUB          AR7   AR6  010|fM   R11 MR6  1|NOP                ;
SW R10 R2  AO R3   |SUB          AR5    R11 000|NOP            |NOP                ;
SW R10 R3  AO R0   |SUB           R12  AR2  000|NOP            |S    R11 1         ;
SW R10 R4  AO R5   |ADL AO AR5    R0    R0   000|NOP            |NOP                ;
SW R10 R5  AO R6   |SUB           R13  AR5  000|NOP            |NOP                ;
SW R10 R6  AO R0   |NOP                        |NOP            |NOP                ;
SW R10 R11 AO R0   |NOP                        |NOP            |NOP                ;
NOP                |NOP                        |NOP            |NOP                ;
NOP                |NOP                        |NOP            |NOP                ;
END of Code;
```

■ FFT:

Fast Fourier Transform, FFT 為複數對複數間的轉換，常用於通信系統基頻信號的處理(如：802.11a)。下式為使用 Cooley-Turkey radix-2, DIF 的方法表示長度為 N 的 FFT 的數學式：

$$A_{2k} = \sum_{n=0}^{\frac{N}{2}-1} (X_n + X_{N/2}) \bullet \omega_{N/2}^{nk} \quad , \quad A_{2k} = \sum_{n=0}^{\frac{N}{2}-1} (X_n - X_{N/2}) \bullet \omega_{N/2}^{nk}$$

，其中 $\omega_N^{nk} = e^{\frac{-j2\pi \cdot nk}{N}} = \cos(\frac{2\pi \cdot nk}{N}) - j \cdot \sin(\frac{2\pi \cdot nk}{N})$

圖 2-18 顯示一個 16 點複數 FFT 轉換的 DFG。以本 ISA 在映射這個 DFG 時就會發生 DFG 太大，暫存器用量太多，無法一次完成映射的情形，這時就必需把整個 DFG 拆開成兩個小部分－在算完第一層 butterfly 的運算後必需把一半的資料存回記憶體中，然後把 DFG 分成兩個 8 點的 FFT 依序運算。最後完成一次 16 點複數 FFT 轉換運算的結果需要 277 個時脈。

表 4-2 總合以上勘查及模擬的結果。由結果可以大致看出 SIU 的排程可以很有效的安排和利用運算資源。以較少的時脈完成同樣的計算工作。在表 4-2 中 DSP-lite 和 Proposed DSP kernel 是使用 SIU 的排程，比起具有同樣硬體計算資源的 ADSP-21xx 在計算的時脈上有約 3~4 倍的改善，比起具有兩倍硬體資源的 TI C55 系列效率更好。而本文所提出的 DSP 核心在時脈上比 DSP-lite 多了一些時脈，這是程式化後程式必需先做一些初始化的工作所必需付出的時脈。但是以同樣的運算而言，DSP-lite 以微程式碼來控制，因此其程式記憶體的使用量是時脈數乘以 128 位元，而本文所改良的核心在記憶體的使用量上只有時脈數乘以 64 位元。因此本架構所使用的程式碼只有 DSP-lite 一半的大小。

Execution kernel	ADSP-21xx	TI C55x	DSP-lite	Proposed DSP
Working frequency	160MHz	200MHz	314MHz	268MHz
Instruction length	24bits (single way)	8~48bits (variable length)	128bits (4-way)	64bits (4-way)
16 points complex FFT	874 cycles	356 cycles	268 cycles	277 cycles
8 points 1-D DCT	154 cycles	--	43 cycles	50 cycles
8x8 points 2-D DCT	2452 cycles	1078 cycles	688 cycles	705 cycles
2^{nd}-order biquad filter	13 cycle	5 cycle	16 cycles	7 cycle
Numerical method	BFP	BFP	SFP	SFP
Power dispassion	120 mW	321 mW	52 mW	37 mW

表 4-2 效率比較表[10][13][22][24][26][27]

第5章　總結

本論文提出一個以程式控制硬體資料流的可程式化的 DSP 加速器核心。其一，以長度精簡的指令集的程式化模式改善其前身 DSP-lite 架構中微程式碼過大的缺點；其二，以程式記憶體取代微程式碼表格，以同步解碼和分叉指令免除原 DSP-lite 需要更新微程式表格的工作。

在使用 SIU DSG 的排程下，可把微處理器的運算單元規畫成類似 ASIC 的資料流。在設計上，採用 SFP 的輕量型運算和分散式記憶體等節能的手段。在運算速度上，以表 4-2 的結果，在同樣的硬體資源下，SIU 可以更有效的運用運算單元，達到較好的效能。而在功率消耗方面，以目前市面上的硬體規格而言，一個用於 802.11b 中 FFT 運算的 ASIC IP 功率消耗為 16mW[25]。使用 DSP 處理器的加速器，其消耗功率平均在 100mW 以上[26][27]，而以 SIU DSG 為主的資料路徑能把功率消耗控制在 50mW 以下。因此在功率的比較下，以 SIU 的方法可以有效減少功率消耗，但是距離要達到模擬 ASIC 資料流的目標還是有改善的空間。在程式碼的產生方面，我們最大的缺撼為必需從 SDFG 的方式來產生程式碼，在一般 DSP 處理器上，大都能提供如 C compiler 之類的高階語言的組譯器，但是此 ISA 在合成高階語言方面的確有執行上的困難。

在未來的工作中，本實驗室目前亦有希望能把程式以自動化產生這方向的研究在進行中。在高階語言的支援上，雖然難以產生組譯器，但是仍可以應用程式界面(API)的方式提供物件連結。在節能方面，本架構中仍可再使用一般節能的方法來改善功率消耗如使用低功率的加法器，使用 gated clock…等等。另外，在多處理器的平台設計中，未來使用更多可平行操作的運算單元(concurrent function unit)來平行處理已是無可避免的趨勢之一，如何劃分(partition)這些運算單元才能讓平行處理更有效率亦為一個研究的課題。以我們的設計而言，擁有 load/store，加法器，乘法器，和位移器四個運算單元，在功能上，已足以獨立完成各種信號處理的運算，但未必是分工最平均，效率最好的安排。在研究上，如果一件工作需要 12 個

運算單元同時運作才能符合計算需求，是用 3 個四運算單元一組的資料流好還是 2 個六運算單元為一組的好？因此，我們還希望在 ISA 上多做一些未來可 scalable 的規化，如定義出只要加多少位元及解碼的電路就可以讓硬體架構順利多一個運算單元而不需要將整個資料流的架構重新設計。這樣可以幫助在硬體劃分時更方便得到效能驗証。

Reference:

[1] Alan V. Oppenheim, Ronald W. Schafer, “Discrete-Time Signal Processing”, 2nd Edition, Prentice Hall, Upper Saddle River, NJ, 1998

[2] David A. Patterson, John L. Hennessy, “Computer Organization & Design The Hardware/Software Interface”, 2nd Edition, Morgan Kaufmann, San Francisco, CA, 1997

[3] David A. Patterson, John L. Hennessy, “Computer Architecture A Quantitative Approach”, 2nd Edition, Morgan Kaufmann, San Francisco, CA, 1995

[4] Keshab K. Parhi, “VLSI Digital Signal Processing Systems: Design and Implementation”, John Wiley&Sons, 1999

[5] Website: DSP village, http://dspvillage.ti.com/

[6] Gatherer, et al, "DSP-based architectures for mobile communications: past, present and future", IEEE Communications, vol. 38, Jan. 2000

[7] OMAP5910 Dual Core Processor – Technical Reference Manual, Texas Instruments, Jan. 2003

[8] TriCore 2-32-bit Unified Processor Core v.2.0 Architecture, Architecture Manual, Infineon Technology, June 2003

[9] R. A. Quinnell, "Logical combination? Convergence products need both RISC and DSP processors, but merging them may not be the answer", EDN, 2003

[10] "A Block Floating Point Implementation for an N-Point FFT on the TMS320C55x DSP", TI Application Report, 2003

[11] S. Rixner, W. J. Dally, B. Khailany, P. Mattson, U. J. Kapasi, and J. D. Owens, "Register organization for media processing", in Proc. HPCA-6, 2000

[12] IEEE Standard for Binary Floating-Point Arithmetic, IEEE Standard 754, 1985

[13] Digital Signal Processing – Using the ADSP-2100 Family, Analog Device Inc., 1990

[14] A Block Floating Point Implementation for an N-Point FFT on the TMS320C55x DSP, Texas Instruments, 2003

[15] Tay-Jyi Lin, Hung-Yueh Lin, Chie-Min Chao, Chih-Wei Liu, Chein-Wei Jen, "A compact DSP core with static floating-point unit & its microcode generation" Proceedings of the 14th ACM Great Lakes symposium on VLSI, 2004

[16] Serene Banerjee, Hamid R. Sheikh, Lizy K. John, Brian L. Evans, and Alan C. Bovik, “VLIW DSP vs. Superscalar Implementation of a Baseline H.263 Video Encoder” Dept. of Electrical & Computer Engineering , University of Texas, 2000

[17] Jessica H. Tseng, Krste Asanovic, "Banked multiported register files for high-frequency superscalar microprocessors", ISCA-30, 2003

[18] Javier Zalamea, Josep Llosa, Eduard Ayguade', Meteo Valero, "Hierarchial clustered register file organization for VLIW Processors", Universitat Polite`cnica de Catalunya, 2003

[19] TMS320C600 CPU and Instruction set reference guide, TI, 2000

[20] T.J. Lin, Chein-Wei Jen, "Data stream generation for concurrent computation in VLSI signal processors", International Conference on Signal Processing, 2000

[21] Y.M. Chang, “Design and Implementation of DSP Datapath for Baseband Processing”, Master Thesis, National Chiao Tung University, Taiwan, 2003
[22] H.Y. Lin, “Lightweight DSP Arithmetic and its Application on a programmable DSP core”, Master Thesis, National Chiao Tung University, Taiwan, 2004
[23] C.C. Lee, “An Embedded Digital Signal Processor Design with Hierarchical Register File & Packed Instructions”, National Chiao Tung University, Taiwan, 2004
[24] "TMS320C55x DSP Library Programmer's Reference", TI, 2003
[25] Web site: DSP core, http://www.dspcore.com/cn/Products/SIFT.htm
[26] "TMS320VC5509A Power Consumption Summary", TI C5000 Hardware Application Report, 6.2004
[27] Web site: ANALOG DEVICES' EXTENDS ADSP-218X DSP FAMILY WITH OVER 50 PERCENT POWER CONSUMPTION SAVINGS, http://www.analog.com/en/content/0%2C2886%2C431%255F%255F8954%2C00.html
http://www.analog.com/IST/SelectionTableProcessors/?selection_table_id=6

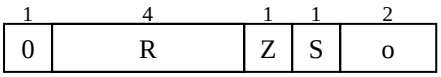
Appendix: Instruction Encoding

1. 控制單元指令格式：

控制單元的指令長度為 15bits，前 9 個 bit 控制 IO bus,後 6 個 bits 控制程式流程和暫存器的更新。

1.1. LW: Load 16bits word

Opcode:



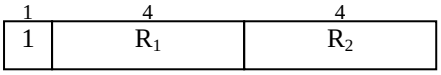
Operands: R

說明：

- 由 IO BUS 讀取一個 16 位元的值，放到 IO output
  Io = memory(R)
- 當使用 R10(R9)當成位址的數值時位址會自動被修正為 R10(R9)+counter register
  Io = memory(R10+counter register)
- 當使用 R0 當成位址時，設為 null，IO bus 不會有動作。
- 當 Z bit 被設為 1 時則 counter register 將在下一個 cycle 被重置為 0
- 當 S bit 被設為 1 時則後面的 o 則會被設定為 counter register 的向左位移量。counter register 為 4bit。在每執行一次索引定址後 counter register 會自動指向下一個 word，當位移量設為 1 時，則 counter register 每累加一次就會跳 2 個 word，同理設成 2 時會跳 4 個 word…利用此法可對記憶體做固定間隔的連續存取。

1.2. SW: Store 16-bits word

Opcode:



Operands: R1 R2

說明：

- 把 R2 的值存入 R1 所指定的記憶體位址，同時把 R2 的值放到 output 去。
  memory(R1)=R2, Io=R2
- 當使用 R10 當成位址的數值時位址會自動被修正為 R10(R9)+counter register
  memory(R10+counter register)=R2, Io=R2
- 當使用 R0 為位址時，IO bus 不會有動作。R2 的值只會被放到 output 不會被存到記憶體中
  Io=R2

1.3. Branch and I/O registers update

Instruction: J

Opcode:

2	4
O	1 1 1 1

Operands: O

說明：程式無條件 branch 至 O 所選擇的 output 的值

Instruction: JC

Opcode:

2	4
O	1 1 1 0

Operands: O

說明：當 ALU 上一個 cycle 的運算有產生 carry-out 或 burrow-in 的情形時，程式會 branch 至 O 所選擇的 output 的值

Instruction: JNZ

Opcode:

2	4
O	1 1 0 1

Operands: O

說明：當 ALU 上一個 cycle 的運算的結果不是 0 時，程式會 branch 至 O 所選擇的 output 的值

Instruction: JZ

Opcode:

2	4
O	1 1 0 0

Operands: O

說明：當 ALU 上一個 cycle 的運算的結果為 0 時，程式會 branch 至 O 所選擇的 output 的值

Instruction: Update registers

Opcode:

2	4
O	R_1

Operands: O R1

說明：會把所選的 output 的值存到指定的暫存器

R1=selected output

2. ALU 單元指令格式：

ALU 單元的指令長度為 22bits，主要提供加法，減法，AND 和 OR 的運算

2.1. ADD 加法(不做暫存器更新)

Opcode:

2	1	1	5	5	5	3
0 0	0	0	Don't care	ARx1	ARx2	S

Operands: ARx1, ARx2, S

說明：

- ARx 的值可為

  10000 ~ 11111: 表示取用 AR0 ~ AR15 的值

  00000 使用 0 為運算元

  01011 使用 Ao 為運算元

  01100 使用 Mo 為運算元

  01101 使用 So 為運算元

  01110 使用 Io 為運算元

  Ao = ARx1 + ARx2 with shift parameter (S)

2.2. ADI 函有立即數值(immediate value)的加法 (不做暫存器更新)

Opcode:

2	1	1	10	5	3
0 0	0	1	K	ARx1	S

Operands: K, ARx1, S

說明：

Ao = K + ARx2 with shift parameter (S)

其中 K 的值會自動 sign extended 至 16bits

2.3. ADL 加法並做暫存器更新

Opcode:

2	1	2	4	5	5	3
0 0	1	O	AR	ARx1	ARx2	S

Operands: O, AR, ARx1, ARx2, S

說明：

Ao = ARx1 + ARx2 with shift parameter (S)

AR = selected output result.

2.4. SUB 減法(不做暫存器更新)

Opcode:

2	1	1	5	5	5	3
0 1	0	0	Don't care	ARx1	ARx2	S

Operands: ARx1, ARx2, S

說明：

Ao = ARx1 - ARx2 with shift parameter (S)

2.5. SUI 函有立即數值(immediate value)的減法(不做暫存器更新)

Opcode:

2	1	1	10	5	3
0 1	0	1	K	ARx1	S

Operands: K, ARx1, S

說明：

Ao = K - ARx2 with shift parameter (S)

其中 K 的值會自動 sign extended 至 16bits

2.6. SUL 減法並做暫存器更新

Opcode:

2	1	2	4	5	5	3
0 1	1	O	AR	ARx1	ARx2	S

Operands: O, AR, ARx1, ARx2, S

說明：

Ao = ARx1 - ARx2 with shift parameter (S)

AR = selected output result.

2.7. AND

Opcode:

3	5	11	3
1 0 0	ARx	K	e

Operands: ARx, K

說明：

Ao = ARx & (K << e) 其中：

& 為 bit-and 的運算

K 的值會自動 sign extended 至 16bits

e 為 K 的指數項，範圍為 000 ~ 101

2.8. ANL 做 AND 運算，同時更新暫存器

Opcode:

3	2	4	5	5	3
1 0 1	O	AR	ARx1	ARx2	Don’t care

Operands: AR, ARx1, ARx2

說明：

Ao = ARx1 & ARx2

AR = selected output result

2.9. OR

Opcode:

3			5	11	3
1	1	0	ARx	K	e

Operands: ARx, K

說明：

Ao = ARx | (K << e)　其中：

| 為 bit-or 的運算

K 的值會自動 sign extended 至 16bits

e 為 K 的指數項，範圍為 000 ~ 101

2.10. ORL 做 OR 運算，同時更新暫存器

Opcode:

3			2	4	5	5	3
1	1	1	O	AR	ARx1	ARx2	Don’t care

Operands: AR, ARx1, ARx2

說明：

Ao = ARx1 | ARx2

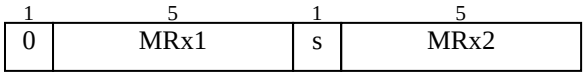
AR = selected output result

3. 乘法器單元指令格式：

乘法器指令的長度為 12bits，提供整數和純小數的乘法。

3.1. M 整數乘法

Opcode:



Operands: MRx1, MRx2, s

說明：

- MRx 的值可為

  10000 ~ 11111: 表示取用 MR0 ~ MR15 的值

  00000 使用 0 為運算元

  01011 使用 Ao 為運算元

  01100 使用 Mo 為運算元

  01101 使用 So 為運算元

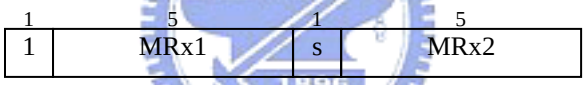
  01110 使用 Io 為運算元

  Mo = MRx1 * MRx2

  當 s 為 0 時即為整數的乘法，當 s 為 1 時乘法的結果會向右修正 1bit。

3.2. fM 純小數乘法

Opcode:



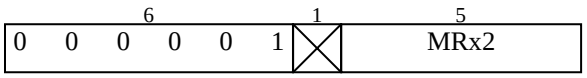
Operands: MRx1, MRx2, s

說明：

Mo = MRx1 (*,s) MRx2

3.3. UM “乘一”的乘法

Opcode:



Operand: MRx

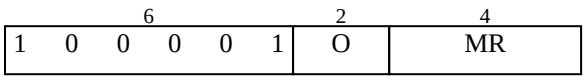
說明：

直接把 MRx 的值傳至 output 不做乘法運算

Mo = MRx

3.4. LM 只更新暫存器(不做乘法運算)

Opcode:



Operand: O, MR

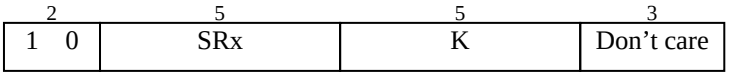
說明： MR = selected output result

4. 位移器單元指令格式：

位移器指令的長度為 15bits，提供無條件的位移指令。

4.1. S: Shift left(不做暫存器更新)

Opcode:



Operands: SRx, K (5 bit signed integer (+15~-15))

說明：

- K 為 5bit 的有號數，範圍為 -16 ~ +15
- SRx 的值可為

  10000 ~ 11111: 表示取用 SR0 ~ SR15 的值

  00000 使用 0 為運算元

  01011 使用 Ao 為運算元

  01100 使用 Mo 為運算元

  01101 使用 So 為運算元

  01110 使用 Io 為運算元

  So = SRx << K，當 K 為負值時表示向右位移

4.2. SLA,SLM,SLI: Shift 且更新暫存器

Opcode:

Operands: SRx, K (4 bit signed integer (+7~-8)), SR

說明：

- K 為 4bit 的有號數，範圍為 -8 ~ +7

  So = SRx << K，當 K 為負值時表示向右位移

  SR = selected output result

作者簡歷

劉建良，1973 年 1 月 14 日出生於台南縣。1996 年取得國立清華大學電機工程系學士學位。2001 年於國立交通大學在職專班攻讀碩士。2005 年在劉志尉教授指導下，取得碩士學位。本篇論文「適用於異質性平台之低功率可程式化資料流設計」為其碩士論文。