國立交通大學

電機學院IC設計產業研發碩士班

碩士論文

適於雙核心多媒體系統晶片之
多重執行緒協同處理器介面

Multithreaded Coprocessor Interface
for Dual-Core Multimedia SoC

研究生：卓志宏

指導教授：劉志尉 博士

中華民國九十六年六月

適於雙核心多媒體系統晶片之多重執行緒協同處理器介面
Multithreaded Coprocessor Interface for Dual-Core Multimedia SoC

研 究 生：卓志宏　　Student: Chih-Hung Cho

指導教授：劉志尉 博士　　Advisor: Dr. Chih-Wei Liu

國 立 交 通 大 學

電機學院 IC 設計產業研發碩士班

碩 士 論 文

A Thesis

Submitted to College of Electrical and Computer Engineering

National Chiao Tung University

in partial Fulfillment of the Requirements

for the Degree of

Master

in

Industrial Technology R&D Master Program on

IC Design

June 2007

Hsinchu, Taiwan, Republic of China

中 華 民 國 九 十 六 年 六 月

適於雙核心多媒體系統晶片
之多重執行緒協同處理器介面

研究生：卓志宏　　指導教授：劉志尉 博士

國立交通大學電機學院產業研發碩士班

摘　　要

在高運算需求的行動多媒體應用的嵌入式系統裡，由於任務（Task）的多樣性，適合以異質性（Heterogeneous）整合的雙核心或多核心（Dual-Core/Multi-Core）運算平台（Platform）（即整合精簡指令集處理器（RISC）以及數位訊號處理器（DSP）平台）來實現。如德州儀器公司（TI）的 OMAP 為例，它是一市面上普遍使用的雙核心運算處理平台，其中 DSP 為伺服處理器（Slave），執行主處理器（Master，即 RISC）所分派的運算任務。然而，DSP 的利用率（Utilization）通常都不高。以一典型的多媒體應用為例（如 JPEG Encoding），DSP 的利用率大約在 50~60%。造成 DSP 利用率不高的原因，可歸納為：(1)指令的延遲時間所引起的管線延遲（Pipeline Stall）；(2)有限的指令層級平行化 (Instruction-level Parallelism, ILP)；(3)處理器之間的溝通（Inter-processor Communication, IPC）以及任務/執行緒管理（Task/Thread Management）所產生的代價。

要突破單一執行緒指令層級平行度（ILP）不高的問題，現今高效能 DSP 大多為多重執行緒（Multithreading）或多核心（Multi-Core）處理器。以提升執行緒層級平行化（Thread-level Parallelism）而言，其最主要的問題在於如何有效率做到處理器同步（Synchronization）、資料搬移（Data Movement）、處理器之間的溝通（IPC）與執行緒管理（Thread Management）等，這些問題使多重執行緒或多核心處理器的系統效能降低，導致所表現出的效能與系統所能提供的最高效能（Peak Performance）之間有很大的落差。本論文提出一適用於雙核心或多核心多媒體系統晶片的多重執行緒協同處理器介面（Host Processor Interface, HPI），來解決上述問題。為了達到 Zero Instruction Latency，我們設計一可同時執行 8 個執行緒的 IMT DSP Core，其執行緒的切換方式為在每個時間週期，8 個執行緒以交錯的、依序的（Run-Robin）方式做切換，如此可避免長與短的管線延遲；此外，我們提出的 HPI 具有優先權（Priority）的執行緒分派功能，可使一個具有高優先權的執行緒不會因執行緒的切換而慢下來。我們以 JPEG Encoding 為例，由模擬結果可知，利用 HPI，可有效的提升 8 執行緒的 IMT DSP 的利用率，其利用率可以從原來的 55%增進到 93%，而 HPI 所增加的面積只有 DSP Core 的 6.25%。

Multithreaded Coprocessor Interface for Dual-Core Multimedia SoC

Student: Chih-Hung Cho Advisor: Dr. Chih-Wei Liu

Industrial Technology R & D Master Program of
Electrical and Computer Engineering College
National Chiao Tung University

ABSTRACT

Due to task divergence in most embedded systems, heterogeneous dual-core/multi-core SoC, i.e. RISC + DSP, is accepted as a cost-effective solution for the increasing computation demands in mobile media applications. TI OMAP, for example, is one popular dual-core platform, where the DSP, as the slave, performs the computation intensive task sent and requested by the host processor (i.e. RISC). However, the low DSP utilization problem may arise. For typical applications, the DSP utilization will be about 50~60%. We can generally attribute the low utilization problem to three causes: (1) pipeline stalls for instruction latency; (2) limits on instruction-level parallelism (ILP); and (3) communication overhead, including the inter-processor communication (IPC) and process/thread/task management.

For delivering high performance beyond single thread ILP, modern DSPs are multi-core or multithreaded processors. On thread-level parallelism, the problems of inefficient synchronization, data movement, and thread management between RISC and DSP definitely degrade the system performance. Consequently, the peak and the delivered performance gap increases. In this thesis, a priority-based, multithreaded coprocessor interface for dual-core/multi-core multimedia SoC is proposed to address the aforementioned problem. In order to hide both short and long stalls, the DSP core is designed by 8-thread IMT core. Threads are interleaving executed in a cycle-by-cycle fashion. With the proposed priority-based host-processor interface (HPI) to facilitate communication with the host processor and the process/thread management, a high priority thread ready to execute without stall will not be slow down. The simulation results show that for JPEG encoding example, with HPI, the 8-thread IMT DSP utilization improves from 55% to 93% with only 6.25% chip area overhead.

誌　　謝

研究生涯轉眼即逝，兩年多來受到許多人幫助及鼓勵，才能順利完成碩士學業，在此致上最深的感激。

感謝劉志尉老師的指導與照顧。老師豐富的學養與風範，使學生在專業知識及研究態度上更臻成熟。同時，特別感謝闕河鳴教授與許騰尹教授在百忙之中，撥冗參與論文口試，並對學生的研究給予寶貴的意見，讓此篇論文更加完備充實。

感謝林泰吉學長。學長廣闊的專業知識與清晰的邏輯思考引導著我，在學長的指導下，這篇論文終能開花結果。

感謝實驗室的學長姐與學弟。感謝士豪、信凱、羽庭、禮圳、朝偉、彥欽、佑昆及 Nelson，感謝學長姊們在研究生生涯中的各項協助及鼓勵。感謝電七，岳泰、Hank、電七、老板、阿德及 Van，謝謝學弟們在研究工作上的一切幫忙。

感謝與我一起打拼的炳勛、卓毅、慶至、翔升與士賢。這兩年，我們一同經歷了挑燈夜戰的努力，也共同分享研究成果的喜悅。

最後，衷心的感謝我最親愛的家人。給予我最大支持與鼓勵，包容我的一切。

謹將此篇論文獻給所有曾支持我、協助我的人，衷心的感謝並祝福你們。

志宏

謹誌於 新竹

2007 夏

目　　錄

圖 目 錄

表 目 錄

1 緒論

1.1 研究動機

隨著處理器架構與先進製程技術的不斷精進，使現今行動多媒體系統除了提供原有的通訊功能之外，還額外附加了許多各式各樣的多媒體應用。例如 2007 年 6 月在美國上市的 iPhone。iPhone 為蘋果公司（Apple）的一款智慧型手機（Smartphone），它是一支結合手機、照相機、個人數位助理（PDA）、媒體播放器（iPod）以及無線通訊的掌上型手持裝置，使用者除了原有的接聽電話功能之外，還可以上網、收發 Email、聽音樂、看電影或是玩電動遊戲。

在高運算需求的行動多媒體應用的嵌入式系統裡，由於任務（Task）的多樣性，適合以異質性（Heterogeneous）整合的雙核心或多核心（Dual-Core/Multi-Core）運算平台（Platform）（即整合精簡指令集處理器（RISC）以及數位訊號處理器（DSP）平台）來實現，以達到最佳的成本效益（Cost-effective） [1]：其中 RISC 用來執行以控制為主的任務（Control-oriented Task）;而 DSP 則執行高運算需求的任務（Computation-intensive Task)。圖 1-1 為 TI OMAP 1510 之方塊圖。OMAP 為德州儀器公司（TI）為實現多媒體

與無線通訊系統應用所提出的雙核心運算平台，它包含了一顆 ARM9 處理器、一顆 TI C55x 系列的 DSP 處理器以及多樣的周邊電路，如通用串行總線埠(USB port)、顯示控制器、通用的輸入和輸出埠(GPIO)等等。ARM 處理器被賦予處理以控制為主的任務，如作業系統(OS)、Protocol Stacks 等; 而 DSP 處理器則負責處理規則且運算密集的工作，如離散餘弦轉換(DCT)、快速傅立葉轉換(FFT)等。不同類型的任務以相對應的處理器來分別處理，既可達到高的效能，也可達到低功耗的目的[2]。

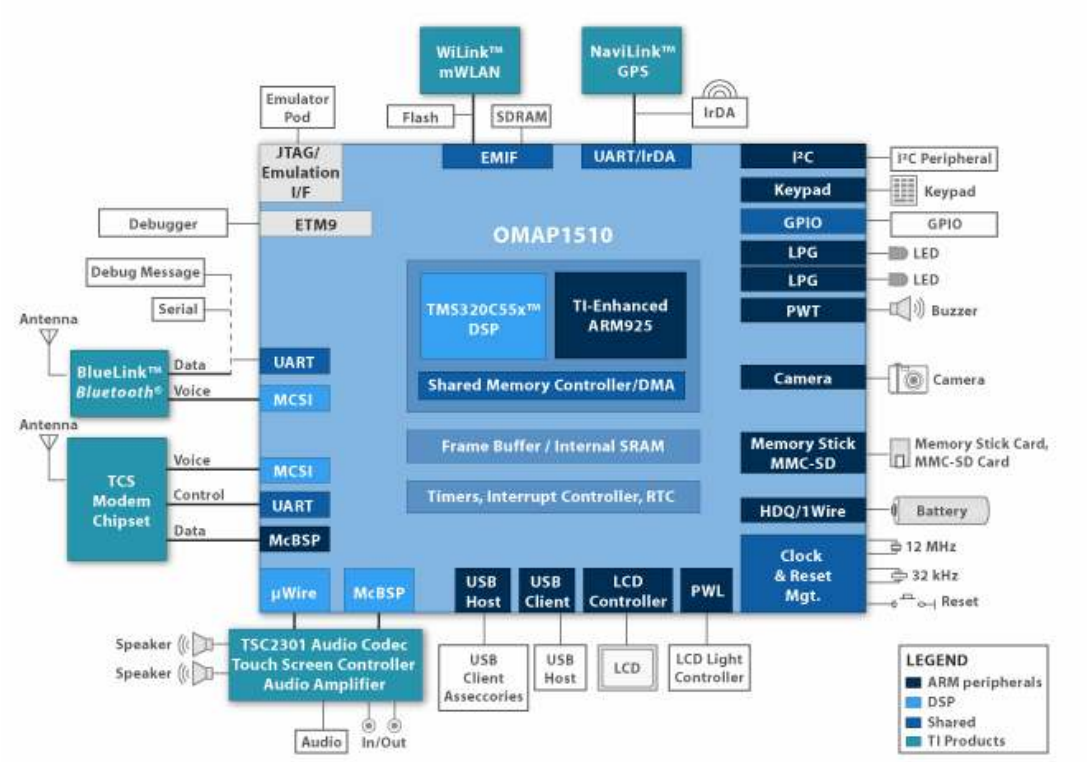


圖 1-1 TI OMAP 1510

在雙核心或多核心運算平台中，處理器之間的溝通方式，通常是經由事先定義好的共享記憶體（Share Memory）以及明確地同步機制來完成。圖 1-2 為一常用於處理器間溝通的的協同模式：基於鬆耦合並列處理架構（Loosely-coupled Framework）。簡單的說，RISC 為主處理器（Master），它為系統的控制者，負責分派任務給伺服處理器（Slave），如 DSP。當伺服處理器將任務完成時，會發出中斷指令（Interrupt）給主處理器，主處理器隨即把運算完的資料搬出，並分派下一個工作給伺服處理器來處理。以

這樣的形式反覆的執行，直到任務結束為止。

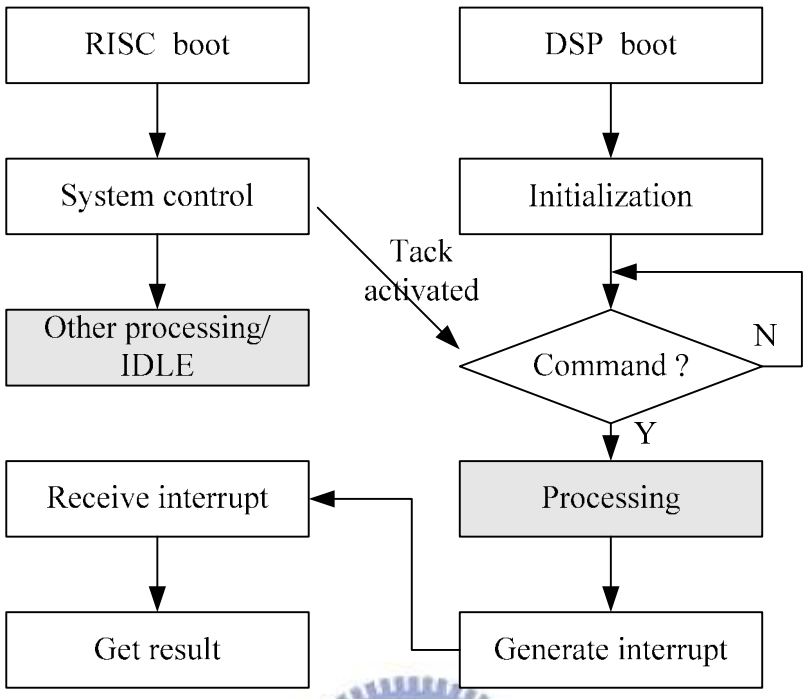


圖 1-2 鬆耦合並列處理架構

在雙核心或多核心運算平台中中，DSP 的利用率（Utilization）通常都低於 60%。探究原因，其 DSP 的利用率不高的原因如下：

- 指令的延遲時間與資料的相依性（Instruction Latency & Data Dependency）

處理器的設計通常利用管線化（Pipelining）的技巧來增加處理器的工作頻率，然而這另一方面也使指令執行需經過幾個時間週期才會得到其指令的運算結果。這對相鄰的兩個指令而言，若之間存在著資料相依性，則第二個的指令必須等待。為了維持資料流的正確性，必須在兩個相依指令之間插入數個延遲週期（Stall Cycles）。這些插入的管線延遲（Pipeline Stalls），使 DSP 無實質的運算，因而利用率降低。

- 有限的指令層級平行度（Limited Instruction-level Parallelism）

為了增加效能，透過硬體排程（Dynamic Scheduling）或是軟體排程（Software Scheduling）的方式來 Exploit 指令間平行度（ILP），可允許處理器在同一時間週期內，

發派數個指令（Multiple-issue），例如超長指令（Very Long Instruction Word, VLIW）處理器以及超純量（Superscalar）處理器。然而指令間的平行度有限，論文[3][4]中指出，在無限的硬體資源下，平均的指令間平行度約為 7。而在論文[5]的研究裡指出，現今 Multiple-issue 處理器，通常配置 8 到 16 個功能模組（Functional Units），而其平均可同時分派的指令卻只有 4~8。換句話說，DSP 的利用率都低於 50%，有些甚至更低。

- 處理器之間的溝通以及任務/執行緒分派管理（Inter-processor Communication & Process Management）

在雙核心或多核心運算平台中，某些任務適合以 DSP 來處理，例如影像處理、聲音處理等；而某些任務則適合以 RISC 處理器來處理，如使用者介面。很明顯地，這些任務的分配需要溝通協調。而任務分配與管理（Process Management），傳統上都以軟體的方式，如一般的作業系統，在 RISC 上執行；另外，也可由 DSP 來負責執行，如輕量級的微核心（Lightweight Micro Kernel）設計。

圖 1-3 為 Process Management 由 RISC 來負責處理的示意圖。為了維持系統工作的正確性，依照 Process Scheduling，RISC 把 Process 依序分派給 DSP 來處理。當 DSP 完成一個 Process 的運算後，向 RISC 發出中斷指令（Interrupt）。RISC 接收 Interrupt 後，除了接收 DSP 之運算結果外，並會分派下一個 Process 給 DSP 處理。RISC 與 DSP 之間，以訊息傳遞（Message Passing，如 Mailbox）的溝通模式相互溝通，直到系統工作完成為止。可想而知，當所需處理的 Process 越多時，RISC 與 DSP 之間的溝通將會非常繁複，且沒有效率。中斷 RISC 的動作，將使 RISC 需要一些時間來處理中斷，造成兩處理器都在等待，而沒有實質的運算。

處理器之間的溝通以及任務/執行緒分派管理這個問題在多重執行緒（Multithreading）處理器上顯得更為嚴重，因為多個執行緒的切換伴隨而來的處理器之間的溝通次數增加，將造成多重執行緒 DSP 處理器的利用率大大的降低。

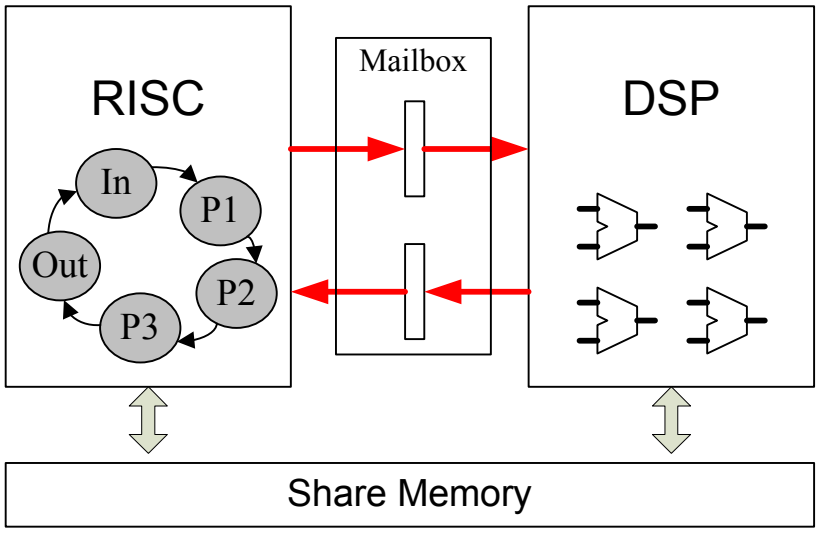


圖 1-3 Process Management 由 RISC 來負責

圖 1-4 為 Process Management 由 DSP 來負責處理的示意圖。不同於圖 1-3，RISC 只需將所需的工作分包給 DSP，DSP 在接受任務後，自行去管理 Process 的流程（Scheduling）與分派（Dispatch）。當 DSP 完成所分包的工作後，透過訊息傳遞的溝通模式（如 Mailbox）通知 RISC。如圖 1-4 所示，RISC 與 DSP 兩處理器之間的溝通次數可大幅減少，但因 Process Management 的工作由 DSP 來負責，將會佔據 DSP 內部的運算資源（如跑迴圈（Looping）或多重任務（Multitasking）分派），因此造成 DSP 的利用率的降低[6][7]。

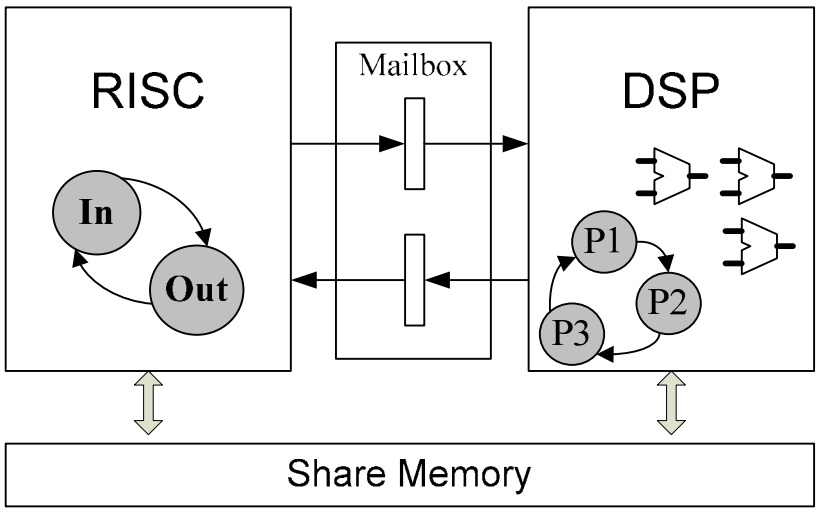


圖 1-4 Process Management 由 DSP 來負責

1.2 基於優先級多重執行緒協同處理器介面（HPI）

有鑑於管線的延遲、有限的指令層級平行度以及處理器之間的溝通以及任務/執行緒分派管理等問題，造成 DSP 利用率不高，針對此議題，我們提出基於優先級多重執行緒協同處理器介面（Host Processor Interface，HPI）來解決。

有限的指令層級平行度，使得現今高效能處理器的設計朝向具更高平行度的發展邁進：如資料層級平行度（Data-level Parallelism，DLP)以及多執行緒層級平行度（Thread-level Parallelism，TLP）。不同於 ILP 需利用軟體或硬體方式來探尋具平行化的指令，DLP 與 TLP，顯而易見的，不同的 Data Block 與 Data Block 之間、或不同的 Thread 與 Thread 之間，其本身具有相當高的平行度。多執行緒處理器，藉由執行緒之間的切換可以避掉一些延遲時間（Stall Cycles），使得管線內的功能模組都是處在有效的運算，以提升處理器的利用率。

本論文中，我們利用指令交錯多執行緒架構（Interleaved Multithreading，IMT）來實作一可執行 8-Thread 之 DSP 處理器，透過 IMT DSP 架構具有 Zero Instruction Latency 的優點，所有因指令延遲時間和資料相依性所產生的管線延遲都可以避免。此外，在處理器之間的溝通以及任務/執行緒分派管理等問題，我們提出以硬體佇列（Hardware Queue）實現之多重執行緒協同處理器介面（HPI），來協助處理器之間的溝通並負責 Process Management 的工作。如圖 1-5 所示，利用 HPI，不僅可降低處理器之間的溝通次數，並可不佔據 DSP 上強大的運算資源，使 DSP 的利用率可以提高。此外，我們設計的 HPI，具有優先權（Priority）的執行緒分派功能，可使一個具有高優先權的執行緒不會因執行緒的切換而慢下來。我們以 JPEG Encoding 為例，由模擬結果可知，利用 HPI，可有效的提升 8-Thread IMT DSP 的利用率，其利用率可以從原來的 55%增進到 93%，而 HPI 約佔 DSP 處理器晶片面積的 6.25%。

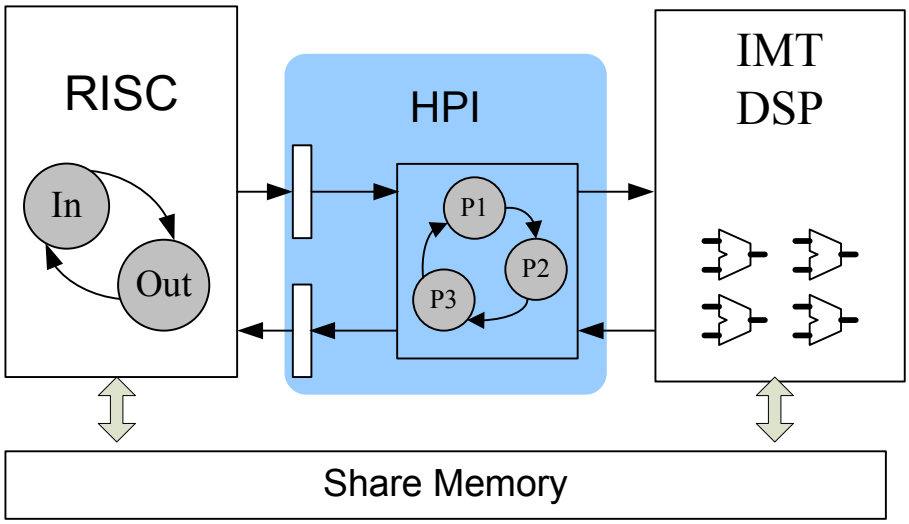


圖 1-5 基於優先級多重執行緒協同處理器介面

1.3 論文大綱

本論文在第二章將說明管線的延遲、有限的指令層級平行度以及處理器之間的溝通以及任務/執行緒分派管理等問題，與可能的解決方案。

第三章描述我們所提出的 HPI 的理論基礎及實作方法。

第四章為實驗數據與模擬結果；最後第五章為總結以及未來的工作。

2 研究背景

在這個章節，指令的延遲時間將會在 2.1 說明。有限的指令平行度以及利用執行緒層級平行度的方法所設計的處理器優缺點說明，將再 2.2 節描述；我們利用指令交錯多執行緒架構來實作我們的 DSP。最後，典型的處理器之間溝通的機制與其所應注意的議題，將在 2.3 節描述。

2.1 指令延遲時間

管線化（Pipelining）設計是提升處理器效能的常用技巧，然而這也導致了指令的執行需延遲時間（Latency Time）。以一典型的5級管線化設計的RISC來說，其指令的延遲時間為3個時間週期。若相鄰兩指令有資料的相依性，則必須在這兩指令之間插入3個管線延遲週期（Stall Cycles），以確保最後資料流（Data Flow）的正確性。基於管線化的架構，連續的兩個指令無法被依序的執行，稱之為管線危障（Pipeline Hazard）。管線危障的發生，使得處理器無法達到因使用Pipelining技巧所得到的理想效能。一般說來，

管線危障可分成三種: (1) 結構危障（Structural Hazard）：因系統的硬體資源有限，如果某些相鄰指令組合，因資源衝突而無法同時處理，稱之為結構危障。(2) 資料危障（Data Hazard）：因管線化的設計，當相鄰指令組合在管線中執行，改變了原本運算元（Operand）讀/寫的順序時，造成其讀/寫的順序與循序（Sequential）執行的非管線化處理器的指令順序不同時，稱之為資料危障。(3) 控制危障（Control Hazard）: 當條件分支（Conditional Branch）被執行時，因程式流程（Program Flow）改變，對管線化的結構所造成的影響稱之為控制危障。

解決管線危障的方式，如Harvard Architecture、Forwarding、Register Renaming、Delayed Branch、Branch Prediction、Speculation等技巧，請見計算機結構教科書[8]。

2.2 有限的指令層級平行度

數個不相關（或平行）的指令，若處理器的有足夠的功能模組，則這些平行的指令可以在同一個時間週期被執行，以提升系統效能，這種指令間可能出現的平行現象就稱為指令層級平行度（Instruction-level Parallelism，ILP）。如下圖 2-1：

```
1.  e = a + b
2.  f = c + d
3.  g = e * f
```

圖 2-1 Illustrative Code Sequence

第 3 個運算會取決於第 1 個和第 2 個運算完的結果，然而第 1 個運算與第 2 個運算並不相依於其他運算，所以第 1 個與第 2 個的運算可以同時被執行; 第 1 或 2 個與第 3 個運算之間存在了資料相依的特性，因此不可以同時被執行。

發掘（Exploit）指令間平行的方法有(1) Dynamic Scheduling: 以硬體的方式 Run-time 發掘隱藏在程式中的平行的指令，如 Tomasulo's Algorithm 用於 Superscalar Processor 中;

(2) Static Scheduling: 以軟體排程的方式，在 Compiler-time 發掘並決定可同時執行的平行指令，如 VLIW Processor。

Exploiting ILP 的技巧，一直是過去 5~10 年中處理器效能增進的主要原因。但指令間的平行度有限。在論文[3][4]中指出，於無限的硬體資源下，依照各個應用程式的不同，平均的指令間平行度約為 7。而在論文[5 的研究裡指出，現今高效能 Multiple-issue VLIW DSP 處理器，通常配置 8 到 16 個功能模組（Functional Units），而其平均可同時分派的指令卻只有 4~8。換句話說，DSP 的利用率都低於 50%，有些甚至更低。

有限的指令層級平行度，使得現今高效能處理器的設計朝向具更高平行度的發展邁進：如資料層級平行度（Data-level Parallelism，DLP)：如 MMX、SIMD 等，以及多執行緒層級平行度（Thread-level Parallelism，TLP)：執行緒為一段連續執行的程式，它可獨立於程式的其他部分而平行的去執行。多重執行緒（Multithreading）處理器在單一的管線內，可以同時執行多個不同執行緒的指令[9]。不同於 ILP 需利用軟體或硬體方式來探尋具平行化的指令，DLP 與 TLP，顯而易見的，不同的 Data Block 與 Data Block 之間、或不同的 Thread 與 Thread 之間，其本身具有相當高的平行度。

多重執行緒處理器依照執行緒切換的方式，可分成以下兩種：

1. 指令交錯多執行緒架構（Interleaved Multithreading Architecture, IMT）

指令交錯多執行緒架構，其執行緒切換為以 TDM （Time-division Multiplexing）的模式，以每一時間週期為基礎，各執行緒以依序、循環的（Round Robin）方式進行切換。在適當設計下，IMT 處理器中的 Thread 執行，可以接近類似 Sequential Processor 的 Behavior，具有 Zero Instruction Latency 的特性。處理器因不需多餘的回饋（Forwarding）路徑，可以使得處理器的 Datapath 設計更為簡單。此外，因每個執行緒以每一個時間週期為基礎做切換，若以一 8-Thread IMT DSP 為例，執行緒中的指令可以有 8-Cycle 的執行時間。因此，在設計上允許於執行緒中執行一 Complicated Instruction（如 Multiply+Shift+Accumulation 等）。

基於 IMT 架構，因 Zero Instruction Latency 的特性，可避免因指令延遲時間所產生的管線延遲（Pipeline Stalls），且因每個時間週期做不同執行緒的切換，如此可以降低程式中 Long Stalls 對處理器利用率的影響;但也因為其執行緒切換的方式為 Cycle-by-Cycle 模式，對個別執行緒而言，其效能會降低。

2. 塊狀交錯多執行緒架構（Blocked Multithreading Architecture, BMT)

塊狀交錯多執行緒架構，不同於 IMT 架構，其執行緒切換的方式為以事件驅動（Even Driven）的模式，當遇到一個很長的時間延遲，如 L2 Cache Miss 時，其延遲時間大於執行緒切換所需的代價（Cost）時，才進行執行緒的切換。BMT 架構，因以事件驅動的模式做執行緒的切換，所以仍須處理 Instruction Latency 的問題; 但因執行緒切換的頻率（相較於 IMT 架構）不高，因此單一執行緒的處理效能不會被減慢（Slow Down）。但另一方面，因執行緒的切換後，仍須付出管線的起始代價（Pipeline Start-up Cost），因此，若執行緒常做切換，系統效能會降低。另外，值得一提的是，在 BMT 架構中，為顧及各個執行緒的使用機會均等，作業程式會設定每個執行緒一個固定執行時間（可用 Timer 來設定），當 Timer Expire 時，會強制做執行緒的切換。

除了 IMT 以及 BMT 架構之外，還有結合 ILP 與 TLP 的同時多執行緒架構（Simultaneous Multithreading Architecture，SMT）。SMT 以超純量處理器（Superscalar）為基礎，利用超純量處理器內多重發派（Multiple-issue）、動態排程（Dynamic Scheduling）和暫存器重新命名（Register Renaming）的技巧，它可在同一個週期中執行多個執行緒內的指令，來增加處理器運算單元（Functional Unit）的使用率。但 SMT 架構非常複雜，同時在多個執行緒中探詢平行指令，造成設計上的困難。SMT 架構，常被運用在伺服器的處理器市場中。

我們認為，IMT 的架構可讓 DSP Processor 的 Datapath 設計簡單，如果能維持高使用率，其 Zero Instruction Latency 的功能，可使 IMT DSP 效能達到滿載。唯一需解決

的問題是，應用程式中 Thread 的產生，以及如何維持 DSP 的高使用率。

2.3 處理器之間的溝通

RISC 與 DSP 之間的同步（Synchronization）與資料搬移（Data Movement）等稱之為處理器之間的溝通（Inter-Processor Communication，IPC）。以程式撰寫者的觀點，不同階層有不同的處理器之間的溝通方式[10]。如果程式撰寫者了解其下層的程式規劃模型（Programming Models）或硬體的溝通方式，他就能直接地驅動或控制該硬體並要求其執行相關的任務，否則須藉由一個高度抽象化（High-level Abstraction，HLA）形式，如應用程式設計介面（Application Programming Interface, API），來達到相同的效果。

處理器之間的溝通可分成兩個部分：一是溝通的方式（如 Shared Memory），二是同步機制（如 Polling 或 Message Passing）。在雙核心或多核心運算平台中，處理器之間的溝通方式，通常是經由事先定義好的共享記憶體（Share Memory）以及明確地同步機制（如 Mailbox）來完成。簡單的說，RISC 為主處理器（Master），它為系統的控制者，當需與伺服處理器（Slave，即 DSP）溝通時，會傳送一 Message（或 Command）給 DSP。當伺服處理器將任務完成時，也會發出中斷指令（Interrupt）給主處理器，主處理器隨即把運算完的資料搬出，並分派下一個工作給伺服處理器來處理。

我們以 TI OMAP [11]平台中處理器間的溝通機制來當作一個說明例子。如圖 2-2 所示，在 TI OMAP 中，處理器之間的同步機制為 Mailbox，Mailbox 內有 2 個暫存器：一個給 RISC 送訊息並發出中斷指示給 DSP；另一個為 DSP 送訊息並發出中斷指示給 RISC 處理器。每一組 Mailbox 的暫存器包含了 2 個 16-bit 暫存器(cmd & data)。DSP Gateway 藉由此 Mailbox 來傳遞訊息，其溝通的步驟如下：

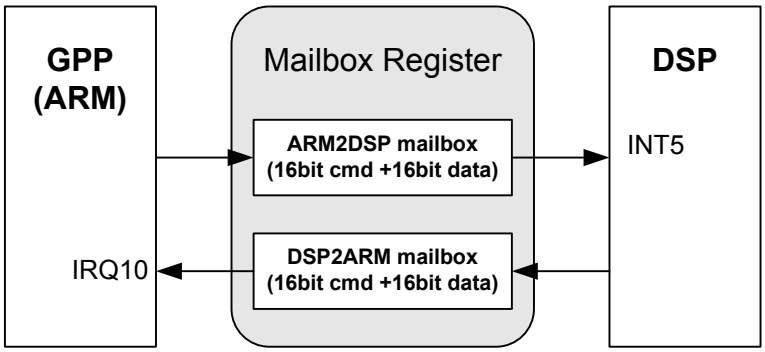


圖 2-2 OMAP 平台上的信箱

1. ARM 寫入輸入資料（Input Data）到共享記憶體（Shared Memory）
2. ARM 寫入輸入資料的位址指標與事先定義好的命令到 ARM2DSP 的信箱（Mailbox）
3. ARM2DSP 信箱會發出中斷指示經由 INT5 到 DSP 上
4. DSP 去讀 ARM2DSP 信箱上的資訊
5. DSP 開始處理輸入的資料
6. DSP 把運算完的結果寫入共享記憶體
7. DSP 寫入輸入資料的位址指標與事先定義好的命令到 DSP2ARM 信箱
8. DSP2ARM 信箱會發出中斷指示經由 IRQ10 到 ARM 上
9. ARM 進入中斷指示服務並且讀取在 DSP2ARM 信箱上的資訊
10. ARM 得到運算完的結果

這個機制簡單，但有 3 個議題需要注意：

1. 中斷指示的代價如中斷延遲時間（Interrupt Latency）：中斷指示的時間延遲定義為當中斷發生時到進入中斷指示服務的第一道指令之間所花費的時間，在

ARM 處理器這通常會需要 25 個時間週期[12]。若兩處理器之間的溝通次數繁多，將使其中斷延遲時間的大大的增加，而使兩處理器都處於閒置狀態。這個問題在以發掘 TLP 的多執行緒處理器上顯得相當重要,因為多執行緒處理器通常有多個 Process 在切換，繁複的 Process 切換所伴隨而來的處理器之間的溝通次數也將增加。

2. 信箱的容量只允許紀錄一道訊息，若有多個訊息要傳遞，而之前的訊息還未被處理，其要發送訊息的處理器必須要等待。

3. 系統匯流排的競爭：資料的傳遞，通常透過系統匯流排，然而系統匯流排同時也供其他元件使用，當兩元件都要傳資料時，有一方必須要等待另一方先將資料傳完之後，方可再傳。

3　多重執行緒協同處理器介面

Host Process Interface (HPI) 是一在 RISC 與 DSP 之間的介面，它主要負責 Process Management、記憶體管理（Memory Management）以及協助兩處理器之間的溝通。在 3.1 節中，我們首先介紹 Data Flow Process Network 的運算模型，其為我們設計所採用的 Programming Model。在 3.2 至 3.4 節中，將說明 Host Process Interface (HPI)的設計與功能，3.5 節是本章的總結。

3.1 Data Flow Process Network

Data Flow Process Network 為一種運算的模型，多個平行的 Process 可以同時的執行[13]。一個通用的 Data Flow Process Network 可以圖 3-1 來表示；P1、P2 和 P3 為 Process，而 Process 與 Process 之間的溝通則經由 FIFO Channel 來達成。當 Process 的 Input Channel 有準備好的資料時，其 Process 則可開始做運算，運算完之後將其運算的結果存放到其 Output FIFO Channel。

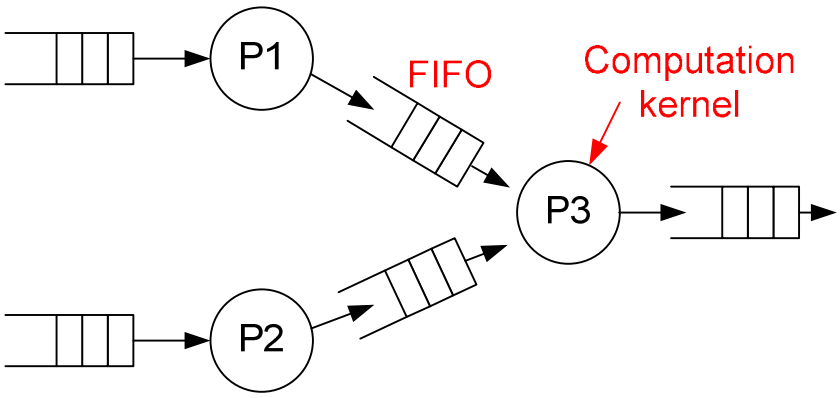


圖 3-1 Data flow process network

Streaming Media Data Application 可利用 Data Flow Process Network 來模擬，例如以一 JPEG Encoding 為例。JPEG（Joint Photographic Experts Group）是一種針對相片影像壓縮的標準[14]。JPEG 壓縮的方式是一種失真壓縮，相片影像品質會有些許損失，普遍使用在網路的影像傳輸上。JPEG Encoding 可以簡單的分成四個 Processes：

- 色彩空間轉換（Color Space Transfer, CST）

  首先，Pixel 由 RGB（紅、綠、藍）轉換為一種稱為 YCbCr 的色彩空間。Y 表示一個像素（Pixel）的亮度（Brightness）；CbCr 表示圖素的色調（Chrominance）與飽和度（Chroma）。這種編碼系統非常有用，因為人類的眼睛對 Y 成分較為敏感。利用 CST，編碼器可以被設計的更有效率地壓縮影像。

- 離散餘弦變換（Discrete Cosine Transform, DCT）

  透過 DCT，可將影像的 YcbCr 色彩空間轉換到頻率空間（Frequency Domain），以解析出影像中 Frequency Domain 的特徵。

- 量化（Quantization，Q）

  在 Frequency Domain 上做量化，簡單的作法把頻率上的值，除以一個對於該成分的常數，再對該值捨位取最接近的整數。這是整個過程的主要失真運算。以這個結果而言，經常會把很多更高頻率的成分捨位成為接近 0，且剩下的多數會變成小的正

數或負數，而利於存取。人類眼睛在一個相對大範圍區域，辨別亮度上的細微差異是優異的，但是在一個高頻率亮度變動之確切強度的辨別上，卻不是很好。高頻率部分的值經 Quantization 之後變成 0，對人類眼睛的辨識而言，影響不大。

- Entropy Coding（Zero Run Length & Variable Length Coding ,VLC）

Entropy Coding 是不失真資料壓縮的一個特別的形式。經量化後影像中的 Pixel 以 Zigzag 方式排列，把相似頻率群組在一起（註：矩陣中網左上方是較低頻率係數，往右下方是較高頻率係數），再經 Zero Run Length & Variable Length Coding 方式達到資料進一步的壓縮。

圖 3-2 以 Data Flow Process Network 來模擬 JPEG 編碼步驟的流程。其輸入是相片影像而輸出是二進位編碼，圖中圓圈部分表示 Process 運算的核心，其相依 Process 的溝通則藉由 FIFO Channel。

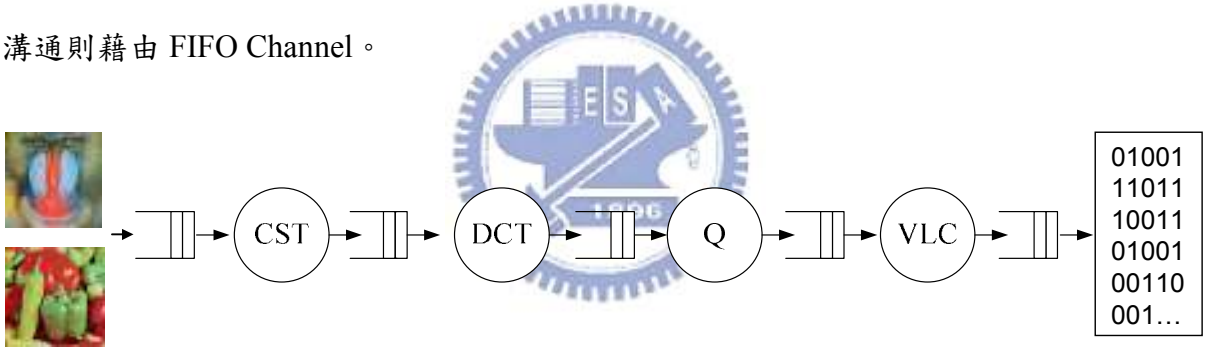


圖 3-2 JPEG Encoding 以 Data Flow Process Network 來表示

Process Netwrok 中，其每一個 Process 即為 DSP 處理器所需處理的運算工作（Task），而 FIFO Channel 則利用 DSP 的資料記憶體來做為資料的傳遞。當 Process 的 Input FIFO Channel 有準備好的資料，該 Process 就有機會被 Process Management 分派給 DSP 去做運算。而 Process 的運算有其優先順序，所以每一個 Process 會被指定一個優先權（Priority）。如果在 IMT DSP 內有一個空閒的 Thread Slot，則 Process Management 會依照所指定的優先順序來分派 Process 到 DSP 上做運算。

優先權的指定方式是由 Process Network 資料流向的後面指定到前面，圖 3-2 為例，VLC 有較高的優先權，而 CST 有較低的優先權。這樣優先權的指定方式是希望運算完

的結果能越快由 Master 搬出。若指定的方式是由前面指定到後面，則除非 CST 的 Output FIFO Channel 滿了，否則 Process Management 會一直分派 CST 的工作給 DSP，其 Process 的分派將建立在滿的 FIFO Channel 上，將使得整個 Process Flow 變的沒有效率。

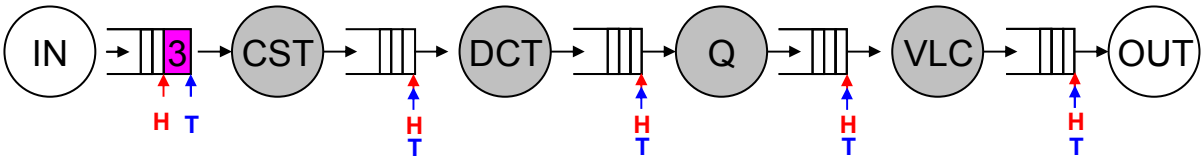
3.2 Process Management

HPI 利用三個 Tables 來進行 Process 的 Scheduling 與 Dispatch 的工作，這三個 Tables 分別是 Process List Table、Queue Table 以及 Dispatch Table，其定義與功能如以下說明：

■ Process List Table：

我們以 Process List Table 來描述 Data Flow Process Network 的行為（Behavior）。如圖 3-3 為所示，把 JPEG 編碼的 Data Flow Process Network 對應到 Process List Table 上。以下為 Table 內的各個項目的用意與解釋：

- ✓ Priority：基於 Data Flow Process Network 的行為，每一個 Process 會指定一個優先順序。其 Output Process 有最高的優先權，而 Input Process 是最低的優先權。
- ✓ I/O：在表示哪一個 Process 屬於 Input Process 或 Output Process，因為 I/O Process 有異於其他運算 Process 的分派方式，所以特別標示出來，讓 HPI 能判斷。
- ✓ Destination：它指示了目前 Process 的下一個目的 Process 是哪一個。例如 CST 的 Destination 為 DCT。
- ✓ Program Address：它記錄了此 Process 的基本位址在指令記憶體裡。



Process Name	Priority	I/O	Dest.	Program Address
OUT	0	Out	-	&OUT
VLC	1	-	OUT	&VLC
Q	2	-	VLC	&Q
DCT	3	-	Q	&DCT
CST	4	-	DCT	&CST
IN	5	In	CST	&IN

圖 3-3 Process List Table

Queue Name	Queue Pointer		Queue Element			
	H	T	0	1	2	3
OUT	0	0				
VLC	0	0				
Q	0	0				
DCT	0	0				
CST	1	0	3			

圖 3-4 Queue Table

■ Queue Table：

Queue Table 為表示該 Process 的 Input FIFO Channel。Queue Pointer 為在 Linear Memory 上實作 FIFO Queue 的 Behavior。在初始狀態時，Head 與 Tail 都指向 0 的位置；當有一筆準備好的資料要放到 Process 的 Input FIFO Channel 時，將其資料在資料記憶體的位置（Page #Number）寫入 Head 指標所指的 Queue Element，之後再將 Head 指標加

上 1;當一個 Process 運算完成時,將 Tail 指標加上 1。如圖 3-3 與圖 3-4 所示,Input Process 帶了一個 Input Data 到 CST Process 的 Input FIFO Channel，而其 Input Data 放在資料記憶體的 Page #3 位置。

- Dispatch Table：

  當一個 Process 要分派到 DSP 之前,這 Process 的一些基本資訊必須紀錄在 Dispatch Table 的內；當 DSP 的執行緒 Initialization 時，會先讀取 Dispatch Table 內部所記錄的資訊，之後才會知道要執行哪一個 Process。圖 3-5 為表示 Dispatch Table 所記錄的項目，以下為其說明：

  - ✓ Thread Enable：它紀錄 IMT DSP 內部各個 Thread Slot 的狀態，是忙碌（Busy）還是閒置（Idle）狀態。當一個 Process 被分派到閒置的 Thread Slot 時，該狀態就會由閒置轉為忙碌；當 Process 完成時，其狀態將變為閒置狀態。HPI 利用此暫存器直接對 DSP 的 Thread Slot 做 Initialization。
  - ✓ Program Address：它記錄了即將要分派的 Process 的程式位址，這個資訊可以從 Process List Table 得知。
  - ✓ Input Page：它紀錄該 Process 的輸入資料在資料記憶體的位置（以 Page #Number 來表示)，這資訊可從 Queue Table 中得到。若該 Process 已被 DSP 處理完成時，Input Page 所記錄的 Page 將會自動的被記憶體管理單元 De-allocation，使其他的 Process 就有機會再重複使用此 Page。
  - ✓ Output Page：它記錄了該 Process 所運算完的結果，可以存放到哪一個資料記憶體的 Page。當一個 Process 已準備好要分派出去，記憶體管理單元會先 Allocate 一個 Page 作為運算結果存放之用。當這個 Process 完成時，此所記錄的 Output Page 將會寫入到對應的 Queue Table。

Thread Enable	Program Address	Input Page	Output Page
Busy	&CST	3	8
Idle			
Idle			
Idle			

圖 3-5 Dispatch Table

圖 3-5 表示，現在 DSP 的 4 個 Thread Slots 中，有一個 Thread Slot 正在做 CST Process 的運算；CST Process 的程式放在程式記憶體的&CST 位址，其 Input Data 放在資料記憶體 Page #3 的位置，而運算的結果要放在 Page #8。當 CST Process 運算完成，Thread Enable 將會變為閒置狀態。

3.2.1 Process 分派的流程

一個 Process 可以被分派出去必須要滿足三個條件：(1)在 DSP 裡必須要有閒置的 Thread Slot 可以使用（Available Thread Slot）；(2)必須要有一個記憶體位置的 Page 來存放運算完的結果（Available Output Page）；(3)選擇一個有較高優先權的 Process 來分派（Process Selection），圖 3-6 表示 Process 的分派過程：

- Available Thread Slot：HPI 內有一專門的硬體在負責選擇一個閒置的 Thread Slot，選完後當 Process 要被分派時，就去更改其對應 Dispatch Table 上的 Thread Enable 狀態。
- Available Output Page：協調記憶體管理單元是否有可用的記憶體 Page，可以給要分派的 Process 使用。
- Process Selection：
  - ✓ 當每一 Process 的 Input FIFO Channel 有準備好的資料，其擁有最高優權的

Process 將優先分派，且分派出去後，它就必須執行到最後（Run to Complete Model）而不能被中斷。

✓ 對於 I/O Process，因其屬於一種不可分割的 Atomic Operation，為了避免多個 I/O Process 同時去讀取 Share Memory 而造成的誤判，在這裡我們限制 I/O Process ，只能各同時存在一個在 DSP 的 Thread Slot 裡執行，來避免額外使用複雜的測試與設定指令（Test & Set）。

■ Process 的分派：當上述的條件都滿足後，將每個步驟所決定的參數分別寫入對應的 Dispatch Table 中，然後更改 Dispatch Table 的 Thread Enable 變成忙碌的狀態，其 DSP 所對應的 Thread Slot 將開始運算，Process 的分派就算完成了。

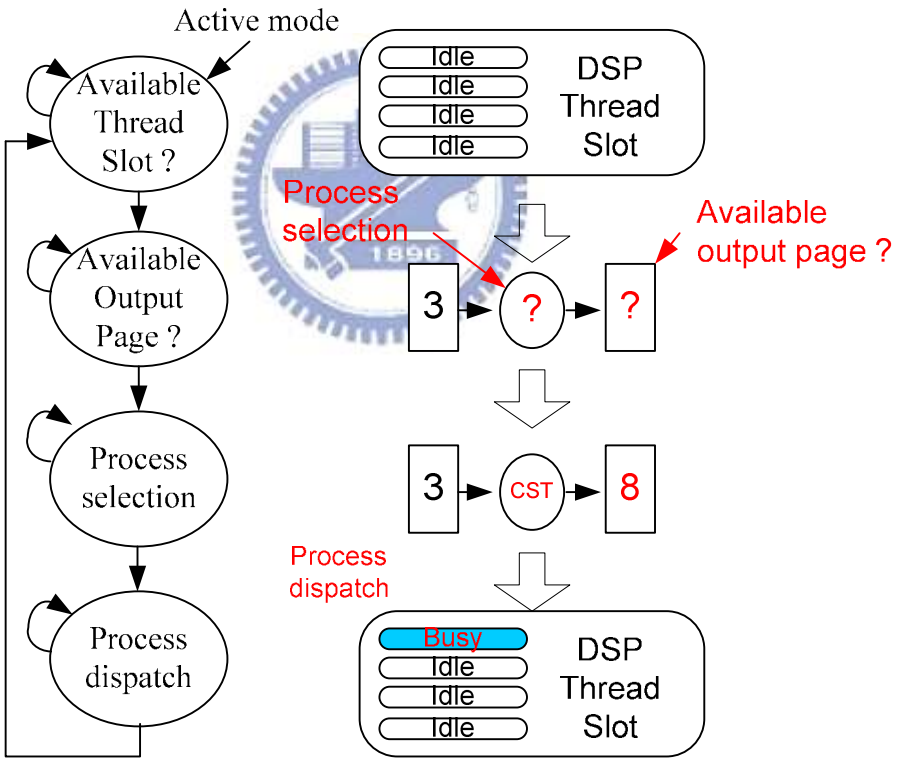


圖 3-6 Process 分派的流程

3.2.2 執行緒的 Initialization 與 Termination

■ 執行緒的 Initialization

當 Dispatch Table 的 Thread Enable 由閒置狀態轉變成忙碌狀態時，其對應的 DSP 執行緒則開始動作。執行緒的 Initialization 機制如以下說明。將指令記憶體分為兩個部分：

1. Initialization 程式的部份：這段程式必須配置在指令記憶體的初始位置。而此段程式的目的在於將 Dispatch Table 內所記錄的參數，分別的寫入其執行緒內的暫存器。如圖 3-7 所示，Initialization 程式的第一道指令為 NOP，當執行緒處於閒置狀態時，其 PC 將一直指向這道指令，而沒有動作。當此執行緒變為忙碌狀態時，PC 開始動作而執行第二道指令，將執行緒的編號寫入 r1；第三道指令再以 r1 當作基準位置到對應的 Dispatch Table 上，將 Program Address 寫到 r2；第四道指令會將 Dispatch Table 上 Input Page 寫到 r3 上等。最後再經由 JR 指令跳到運算的程式部份而開始做運算。

2. 運算程式的部份：運算程式包含 I/O Process 以及原本要運算的 Process，其可以存放於指令記憶體的任何位置（除了 Initialization 部分外）。

■ 執行緒的 Termination

當一個 Process 被 DSP 的 Thread Slot 執行完時，我們藉由一特定的指令 **SlotFree** 來終止其 Thread Slot 的動作，使其 PC 值回到初始位置（也就是指向 NOP 指令），並利用此指令來更改 HPI 的 Dispatch Table 狀態。如圖 3-7 所示，將此指令加在每一個 Process 的最後一行指令。

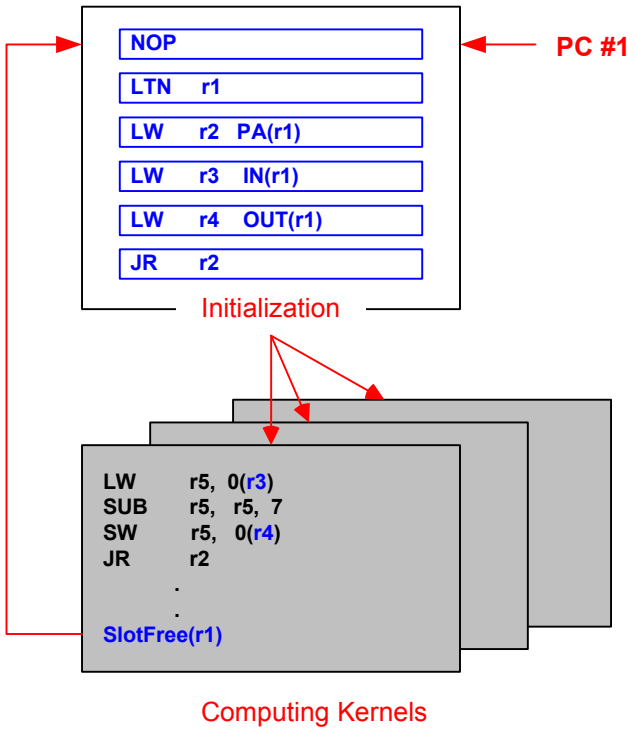


圖 3-7 指令記憶體的結構

3.3 記憶體管理單元

簡單的記憶體管理方法是運用 Page 管理，將記憶體分成固定大小的 Page。我們以此方法來管理 DSP 的資料記憶體。將 DSP 的資料記憶體分成 1Kbytes 大小的 Page，如圖 3-8 所示，其中 1K bytes 大小作為處理器之間溝通之用（FIFO Queue）。

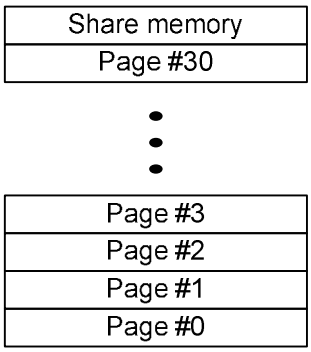


圖 3-8 資料記憶體分成固定大小的 Page

記憶體的管理功能是動態的去 Allocate 或 De-allocate 這些 Page，給一個新的 Process

或是一個終止的 Process ，如圖 3-9 所示，為記憶體管理的一個例子。在圖 3-9(a)表示，記憶體管理單元分配一個 Page #2 位置給主處理器傳 Input Data，另外記憶體管理單元也分配一個 Page #3 位置給 DSP 當作運算結果存放的地方。當此時 DSP 的 Process 運算完後，其 Input Page 將可以 De-allocate 給其他 Process 使用，所以 Page #0 就被記憶體管理單元來 De-allocate。在圖 3-9(b)表示，當另一個 Process 被分派到 DSP 運算，其 DSP 開始從 Page #1 拿資料來算，並將運算完的結果存入 Page #0，而這 Page #0，即為剛才才被 De-allocate 的 Page。另外，剛才才運算完的資料存在 Page #3，現在正由主處理器搬出。

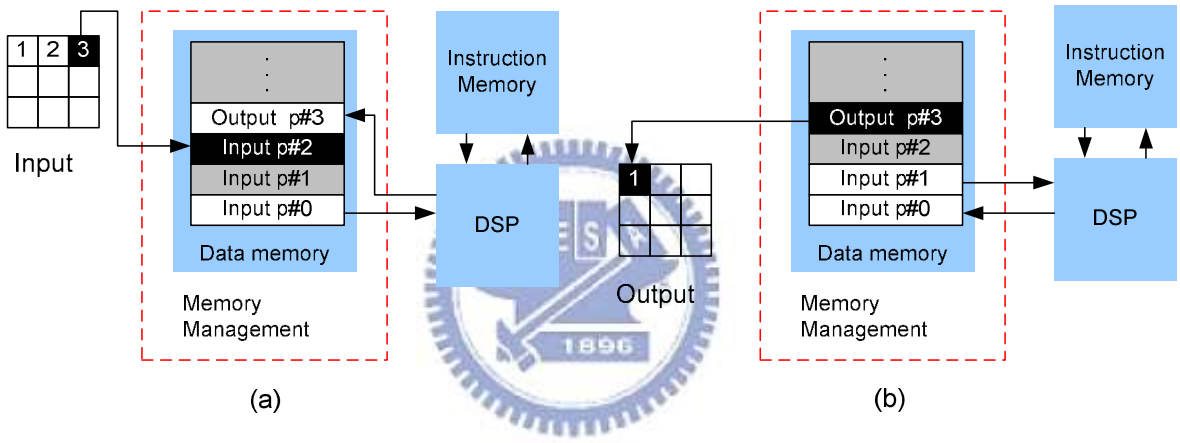


圖 3-9 動態記憶體管理

3.4 處理器之間的溝通

處理器之間的溝通可分為三個部份來說明：

1. Synchronization：

我們所使用的 Synchronization 機制是 Mailbox，當 DSP 下達特定的命令時，則中斷指示將產生給主處理器。

2. Communication：

Communication 是經由事先定義好的 Share Memory 來為之。而 Share Memory 也建構成 FIFO Queue 的形式，使 DSP 的運算與主處理器的資料搬移動作可以同時進行，緩衝了兩處理器之間工作頻率的不同。如圖 3-10 所示，使用了兩個 FIFO Queue，一個為 Data Input 使用（Input FIFO Queue），另一個為 Output Data 使用（Output FIFO Queue）。

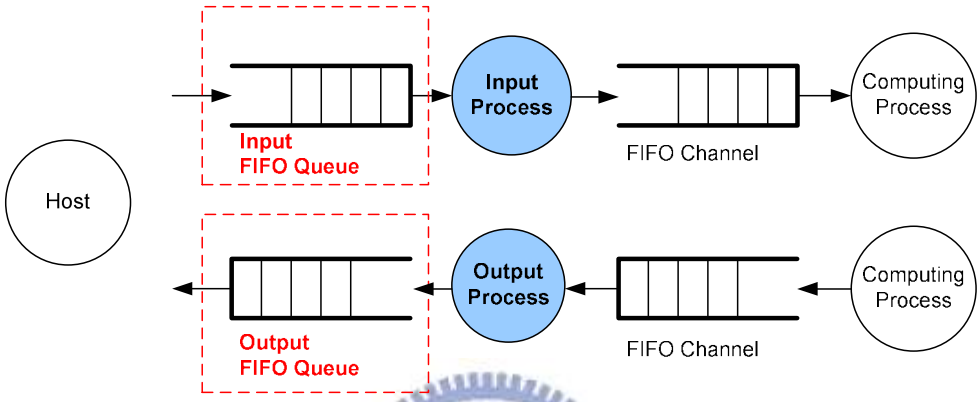


圖 3-10 I/O FIFO Queue

圖 3-11 為 FIFO Queue 資料結構（Data Structure），此 FIFO Queue 使用了 2 對指標：DSP_H/DSP_T 與 HOST_H/HOST_T。在初始狀態時，這四個指標都指向 0 的位置，藉由 I/O Process 對指標的操作來完成處理器之間的溝通。以下說明此 FIFO Queue 指標的操作方式：

- DSP_H/DSP_T：此對指標由 I/O Process 的程式來操作。當 Input Process 在執行時，將記錄在 Dispatch Table 上的 Input Page 寫入 DSP_H 所指到的 Input FIFO Queue 裡，之後將 DSP_H 加 1，即分配一個可用的 Page 給主處理器來做 Input Data 資料傳輸之用；當 Input Process 讀到 HOST_H 與 DSP_T 所指的位置之間有差值（HOST_H-DSP_T）時，表示主處理器已經有傳完的資料，此時 Input Process 會讀取 DSP_T 所指到的 Page #Number，並將此 Page #Number 寫入 CST 的 Input FIFO Channel，之後將其 DSP_T 指標加上 1，完成 En-queue 的動作。

- HOST_H/HOST_T：此對指標由主處理器的程式來控制操作。主處理器會先讀

取 DSP_H 與 HOST_T 所指的位置，若之間有差值（DSP_H-HOST_T），則表示 Input Process 已經分配一個可用的 Page 給主處理器，之後主處理再去讀取 HOST_T 所指到的 Page #Number，並將 HOST_T 加上 1；當主處理器有時間可以傳資料到 DSP 上時，就將資料傳到剛剛在 HOST_T 位置所讀到的 Page #Number，當資料傳完時，將 HOST_H 加上 1。

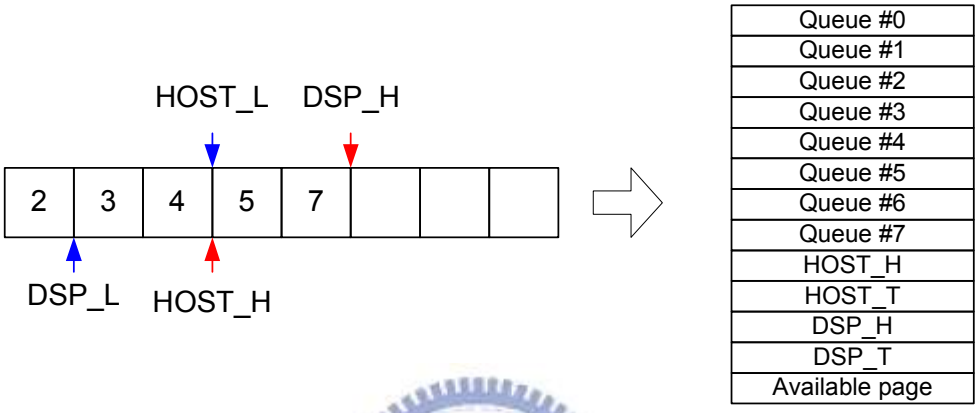


圖 3-11 FIFO Queue 的資料結構

3. I/O Process：

為在 DSP 上執行的一段程式，它配合 FIFO Queue 來完成處理器之間的溝通。

◆ Input Process 的程式流程：Input Process 主要的目的在於 Allocate 一個可用的 Page 讓主處理器將資料傳進來，當資料傳完之後，負責將資料在資料記憶體的位置（Page #Number），寫入其後 Process 的 Input FIFO Channel，之後這筆資料會經由 Process Management 分派到 DSP 做運算。其程式的流程如圖 3-12。當 Input Process 被分派到 DSP 執行時，其程式會先確定可用的 Page 是否小於 8（假設 I/O FIFO Queue 的長度(Length)為 8）。若小於 8，則可以再 Allocate 一個 Page 到 Input FIFO Queue 上；若大於等於 8，則表示可用的 Page 已經超過了 FIFO Queue 的長度。之後，藉由讀取指標的位置的值來判斷是否主處理器已經有傳完的資料；若有，則執行 En-Queue（將 Page #Number 寫入運算 Process 的 Input FIFO Channel）的工作，然後藉由 SlotFree 指令來結束 Input Process 。

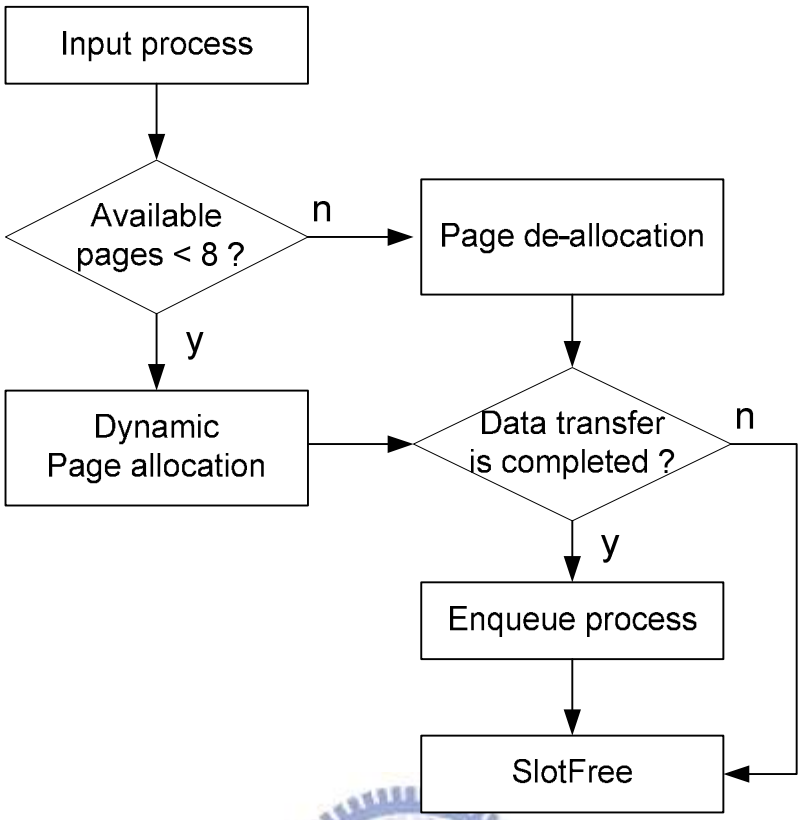


圖 3-12 Input Process 的程式流程

- Output Process 的程式流程：Output Process 的主要目的，為告知主處理器已有運算完的資料，請主處理器來搬走；另外，當主處理器搬完後，通知記憶體管理單元來 De-allocate 其 Page。Output Process 相對於其他 Process 擁有最高的優先權，所以主處理器若發現有可以搬出的資料，也需要配合著優先執行。其程式的流程如圖 3-13 所示。當 Output Process 被分派到 DSP 後，它先察看是否已經有搬出的資料，若有則通知記憶體管理來 De-allocate 此 Page；之後，再查看已分配給主處理器的 Page 數量是否小於 8，若小於 8 則分配一個 Page 到 Output FIFO Queue，並使 Mailbox 發出 Interrupt 到主處理器，之後結束 Output Process ；若分配給主處理器的 Page 數量等於 8 時，則表示，已經有 8 筆資料已運算完成，而主處理器一直沒能搬走，此時 Output Process 進入無窮迴圈，直到主處理器把資料搬走，才會結束。

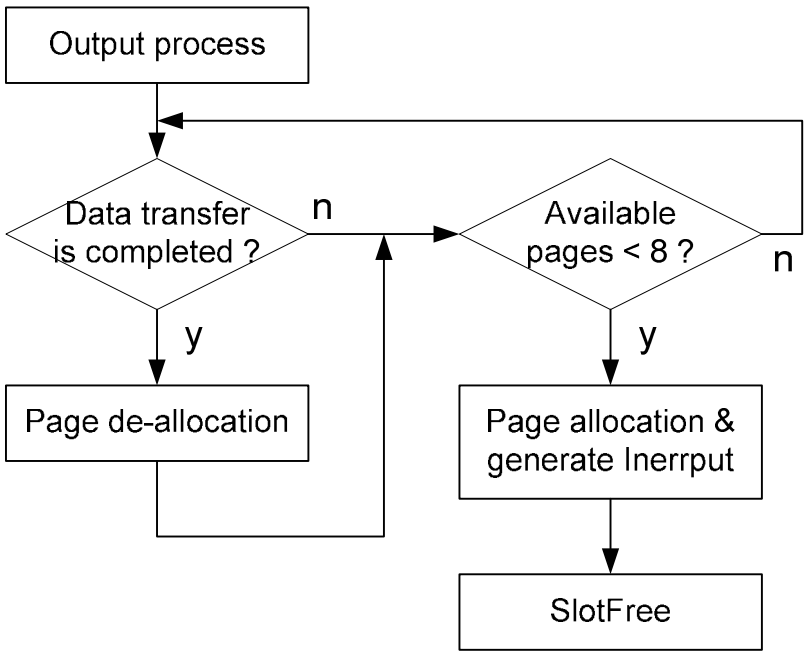


圖 3-13Output Process 的程式流程

3.5 本章總結

- Data Flow Process Network

多媒體應用可以以 Data Flow Process Network 來描述。其 Process 表示運算的核心，而相依 Process 之間的溝通則由 FIFO Channel 來傳遞。它對應到我們的 DSP 上時，DSP 負責 Process 的運算，而 FIFO Channel 則利用 DSP 的資料記憶體來實行。每一個 Process 會指定一個優先順序，Process Management 利用此優先順序進行 Process Scheduling 和 Dispatch 的動作。HPI 運用此概念來完成 Process Management 的工作。

- HPI Tables

HPI 內部三個 Tables：Process List Table、Queue Table 和 Dispatch Table。Process List Table 用來描述 Data Flow Process Network 的行為；Queue Table 為表示各個 Process 的 Input FIFO Channel；Dispatch Table 則記錄了 Process 的基本資訊如 Process 程式的位址、Input Page 和 Output Page 的位置。

- Process 分派的流程

  Process 分派必須滿足三個條件：(1) 在 DSP 裡必須要有閒置的 Thread Slot 可以使用（Available Thread Slot）；(2)必須要有一個 Page 來存放運算完的結果（Available Output Page）；(3)要選擇一個適合的 Process 來分派（Process Selection）。

- 記憶體管理單元

  將 DSP 的資料記憶體分成固定大小的 Page。記憶體管理單元可動態的 Allocation 或 De-allocation 這些 Page 給一個新的 Process 或一個終止的 Process 。

- 執行緒的 Initialization 與 Termination

  指令記憶體分成兩個部分：(1) Initialization 程式部分(2)運算程式部分。Initialization 程式部份的目的在於將 HPI 內的 Dispatch Table 所記錄的資訊寫入到其執行緒的暫存器內，再經由 JR 指令將跳到運算程式的部份。執行緒的 Initialization 是藉由 Dispatch Table 的 Thread Enable 由閒置轉變為忙碌狀態時，其對應執行緒的 PC 開始動作而完成初始動作。執行緒的 Termination 則藉由加在運算程式的最後一行特定的指令 **SlotFree** 來達成，當執行到這行指令時，其執行緒的 PC 值回覆到初始位置，並且 HPI 的 Dispatch Table 之 Thread Enable 會轉變為閒置狀態。

- 處理器之間的溝通

  Synchronization 的機制是 Mailbox；溝通經由事先定義好的 Share Memory 來為之，且 Share Memory 建構成 FIFO Queue 的形式，使運算與資料傳輸可以重疊在一起執行；I/O Process 為 DSP 上的一段程式，主要在協助主處理器與 DSP 處理器之間的溝通。

4 實現與模擬結果

我們實現了一顆 8 個執行緒的 IMT DSP 處理器，並結合我們所提出的 HPI 實作在 ARM Versatile 上面，以 JPEG 編碼為其應用程式，模擬 3 種情況：(1)Process Management 的工作由 ARM 來處理；(2)Process Management 的工作由 DSP 來處理；(3)Process Management 的工作由 HPI 來處理。分別計算各情況下 DSP 的利用率來評估 HPI 的利弊。

4.1 DSP 處理器架構

圖 4-1 表示所提出的 HPI 和 IMT DSP 核心。在 HPI 方面：HPI 對主處理器的接腳與一般的記憶體相同，猶如主處理器的附屬記憶體，藉由特定的位址和資料，可對 HPI 下 Command。其內部包含了 6 個功能模組（Function Units）：解碼器、記憶體管理單元、FSM 以及三個 HPI Tables。在運算引擎方面包含：8 個執行緒的 IMT DSP 核心、指令記憶體和資料記憶體。指令記憶體為 32-bits 的 Width 共有 32K Bytes 提供運算程式的存取。資料記憶體提供 8-bits、16-bits 以及 32-bits 的存取，共有 32K Bytes。

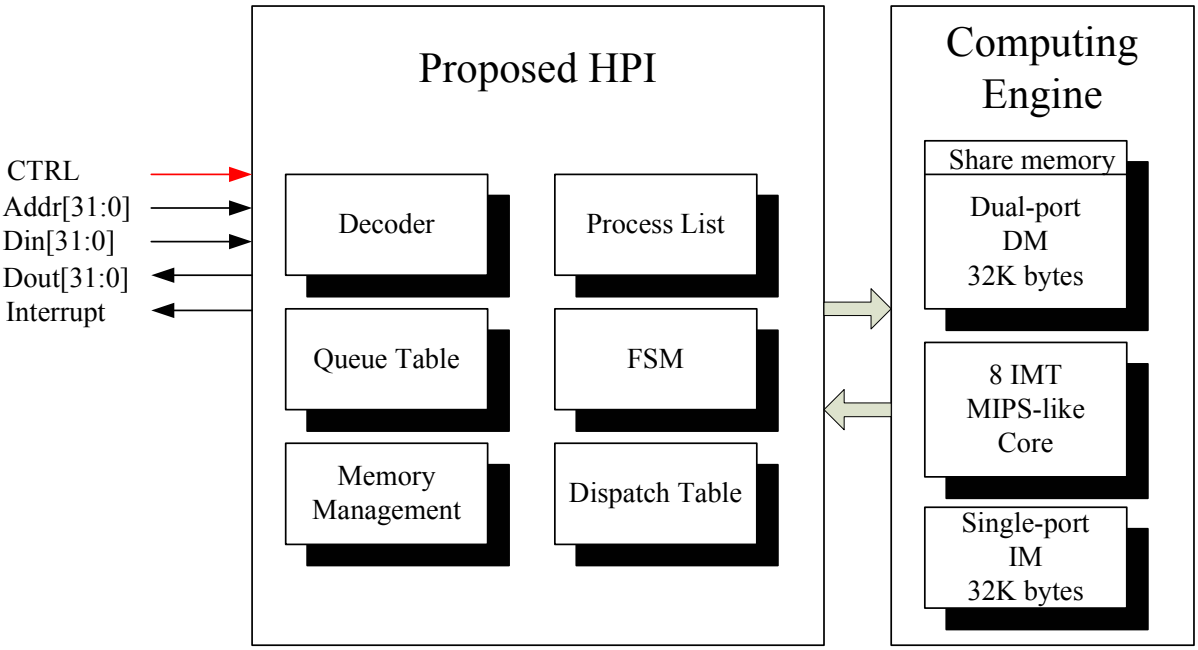


圖 4-1 HPI 與 DSP 之區塊圖

圖 4-2 為表示記憶體所對應的位址，主處理器可藉由事先定義好的記憶體位置對 DSP 下 Command 或對資料記憶體和指令記憶體作存取的動作。另外 HPI 的狀態暫存器並不提供主處理器去做更改。

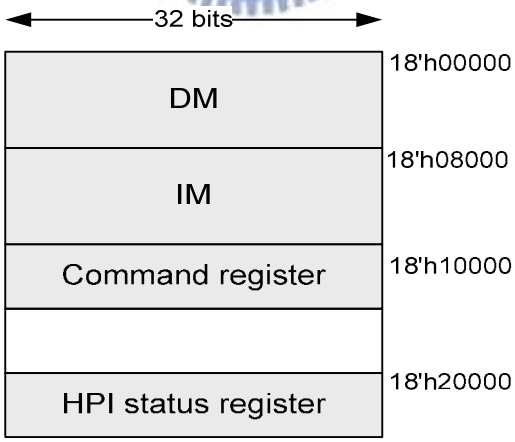


圖 4-2 記憶體位置規劃

4.1.1 HPI 的模式

HPI 主要有兩種模式如以下說明：

- 組態設定模式（Configuration Mode）

  為 HPI 的初始模式，在此模式下主處理器可以將運算的程式寫入指令記憶體中，並且也將 Data Flow Process Network 的行為寫入到 HPI 的 Process List Table 中。經由特定的 Command（HOST_INITIAL）將使 HPI 進入主動模式。

- 主動模式（Active Mode）

  當 HPI 進入主動模式時，即開始分派工作給 IMT DSP 做運算。此時指令記憶體以及 HPI 的 Process List Table 都不能被更改。

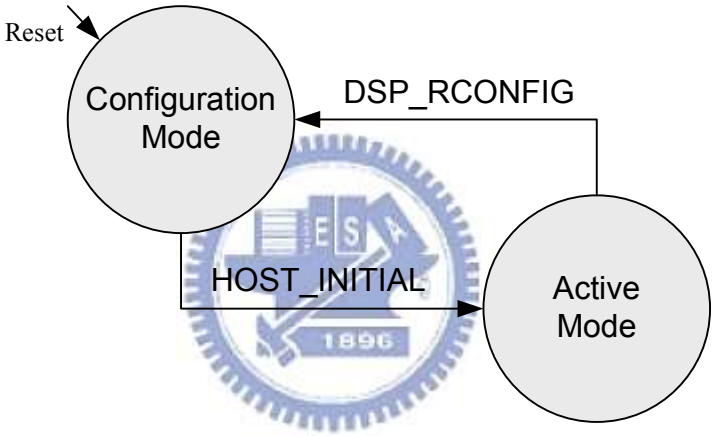


圖 4-3 HPI 的狀態

圖 4-3 表示 HPI 的狀態圖。HPI 的狀態基本上由主處理器來經由特定的 Command 來控制。其主處理器所可以下達的 Command 如表 4-1 所示：當主處理器下達 HOST_INITIAL 將使 HPI 進入到主動模式；若下達 DSP_RCONFIG 將使 HPI 由主動模式回到組態設定模式，此時主處理器可以重新寫入指令記憶體及 Process List Table；當下達 HPI_MODE 時，HPI 將回覆 HPI 現在的狀態。若主處理器讀到 0 時，則 HPI 處於組態設定模式；若讀到 1 時，則 HPI 處於主動模式。

表 4-1 主處理器的 Command

Command	Descriptions
HOST_INITIAL	Switch to active mode
DSP_RCONFIG	Switch to configuration mode
HPI_MODE	Return “0” for configuration mode Return “1” for active mode

4.1.2 指令交錯多線程之 DSP 處理器

圖 4-4 表示 5 級管線階段之 8 個執行緒的 IMT 架構。從圖中可知，有 8 套獨立的 PC 以及 8 套通用暫存器(General Purpose Register)分別給 8 個執行緒來使用，每套暫存器有包含 32 個暫存器且每個暫存器是 32bits；執行緒的切換以 Round Robin 的方式 Cycle-by-Cycle 切換。

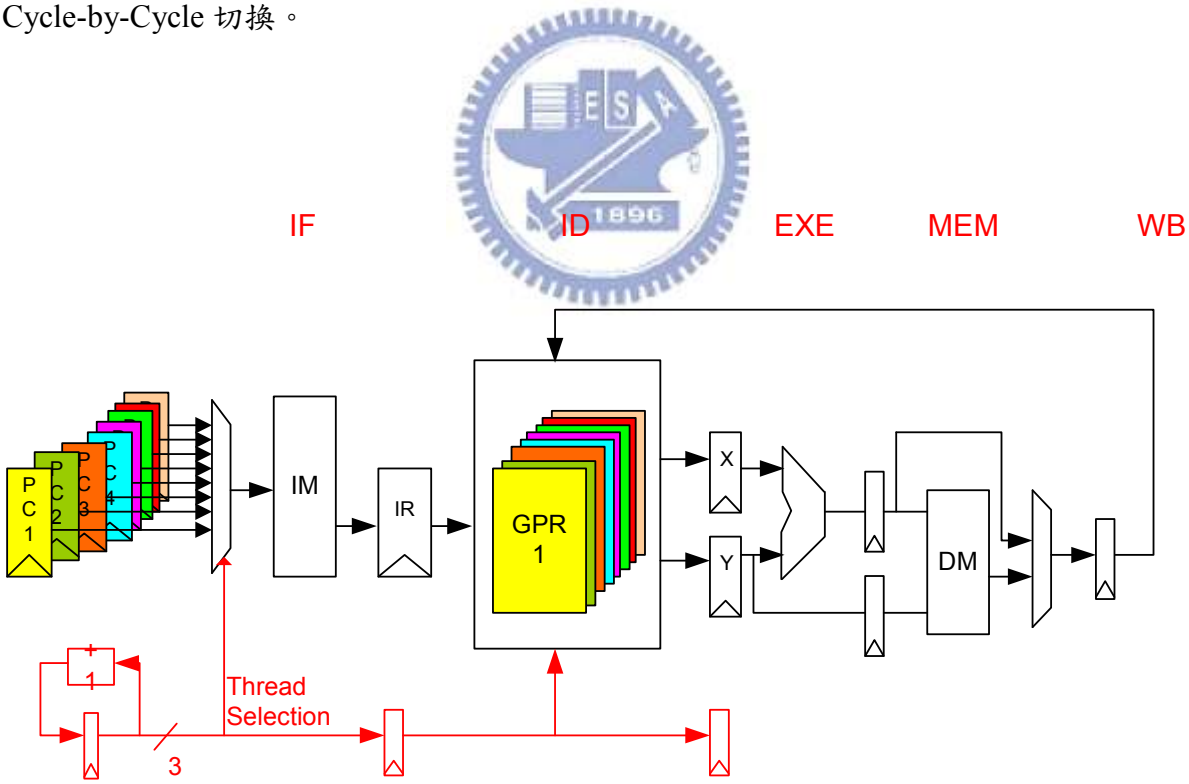


圖 4-4 指令交錯多線程 DSP 核

4.1.3 指令集

DSP 的指令集可以分成三個部分來說明：

- 算術運算指令：執行基本的運算如加、乘、移位等。如表 4-2 所示
- 資料存取指令：Load/Store 指令；提供 Byte、Half Word 和 Word 的存取。如表 4-3 所示
- 程式流程指令：如 Jump，Branch Equal 等。如表 4-4 所示

圖 4-5 表示指令編碼的格式：

- R-type 定義出需要存取 3 運算元的指令
- I-type 定義出需要存取 2 運算元的指令以及 16-bit 的立即值運算
- J-type 定義出程式流程指令

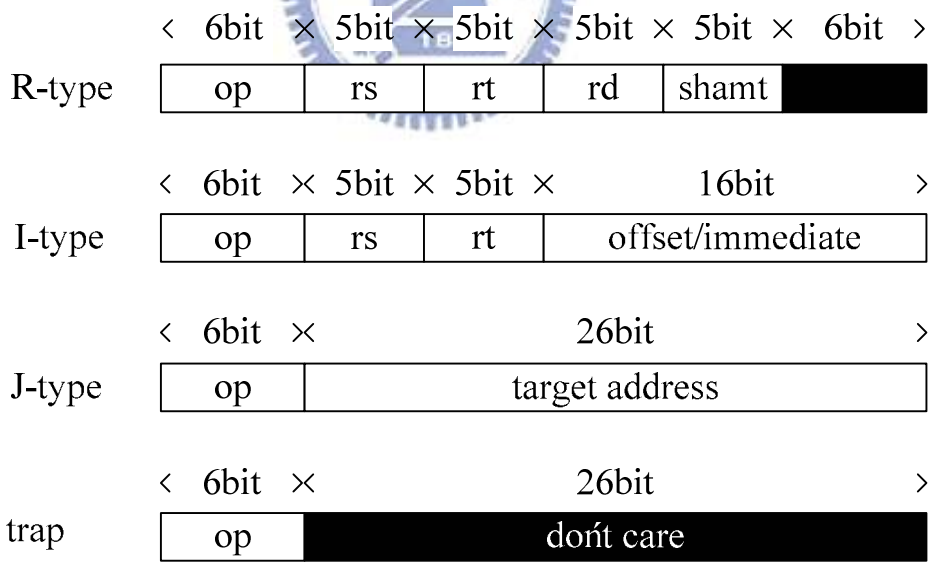


圖 4-5 指令格式

表 4-2 算術運算指令

Instruction (arithmetic)	Syntax	Semantics
addu	addu rd, rs, rt	rd = rs + rt
subu	subu rd, rs, rt	rd = rs – rt
addiu	addiu rt, rs, imm	rt = rs + imm
and	and rd, rs, rt	rd = rs & rt
or	or rd, rs, rt	rd = rs | rt
xor	xor rd, rs, rt	rd = rs ^ rt
nor	nor rd, rs, rt	rd = ~ (rs | rt)
andi	andi rt, rs, imm	r t= rs & imm
ori	ori rt, rs, imm	rt = rs | imm
xori	xori rt, rs, imm	rt = rs ^ imm
sll	sll rd, rt, shamt	rd = rt<< shamt
srl	srl rd, rt, shamt	rd = rt >> shamt(logically)
sra	sra rd, rt, shamt	rd = rt >> shamt (arithmetically)
sllv	sllv rd, rs, rt	rd = rt << rs
srlv	srlv rd, rs, rt	rd = rt >> rs (logically)
srav	srav rd, rs, rt	rd = rt >> rs (arithmetically)
slt	slt rd, rs, rt	rd = (rs < rt)? 1:0
sltu	sltu rd, rs, rt	rd = (rs < rt (unsigned))? 1:0
slti	slti rt, rs, imm	rt = (rs < imm)? 1:0
rsbiu	rsbiu rt, rs, imm	rt = imm – rs
sle	sle rd, rs, rt	rd = (rs <= rt)? 1:0
sleu	sleu rd, rs, rt	rd = (rs <= rt (unsigned))? 1:0
slei	slei rt, rs, imm	rt = (rs <= imm)? 1:0
xmpy	xmpy rd, rs, rt	rd = rs * rt
la	la rt, imm	rt = imm (address pointer)
li	li rt, imm	rt = imm

表 4-3 資料存取指令

Instructions (data access)	Syntax	Semantics
lb	lb rt, offset(rs)	rt = (sign_extend) MEM [rs + offset]
lbu	lbu rt, offset(rs)	rt = (zero_padding) MEM [rs + offset]
lh	lh rt, offset(rs)	rt = (sign_extend) MEM [rs + offset]
lhu	lhu rt, offset(rs)	rt = (zero_padding) MEM [rs + offset]
lw	lw rt, offset(rs)	rt = MEM [rs + offset]
sb	sb rt, offset(rs)	MEM [rs + offset] = rt [7:0]
sh	sh rt, offset(rs)	MEM [rs + offset]= rt[16:0]
sw	sw rt, offset(rs)	MEM [rs + offset] = rt

表 4-4 程式流程指令

Instructions (program flow)	Syntax	Semantics
beq	beq rs, rt, offset	If (rs == rt) branch "offset" instructions
bne	bne rs, rt, offset	If (rs != rt) branch "offset" instructions
j	j target	jump to the "instruction" at target
jr	jr rs	jump to the instruction whose "address" is in register
jal	jal target	jump to the "instruction" at target and save "address" of the next in $ra
trap	trap	trap
nop	nop	nop
ltn	ltn rt	store the thread number in the register rt

4.2 FPGA Prototyping

我們將所提出的 HPI 與 DSP 實現在 ARM Versatile 的平台上[15]。 圖 4-6 表示 ARM Versatile 整個系統的架構平台，其中包含了 ARM926EJ-S 核心和 128MB SDRAM，還有一些周邊電路如 LCD Controller、聲音解碼與編碼器和系統匯流排（AMBA AHB Bus）等等。我們將所設計的 DSP 將包覆在 AHB 匯流排上後，再將整個電路燒入 Xilinx XC2V6000 FPGA Logic Tile Daughter Card 上。這樣就可以利用 ARM926EJ-S 經由 AHB 匯流排來控制燒入在 Xilinx FPGA 上的 DSP 處理器。

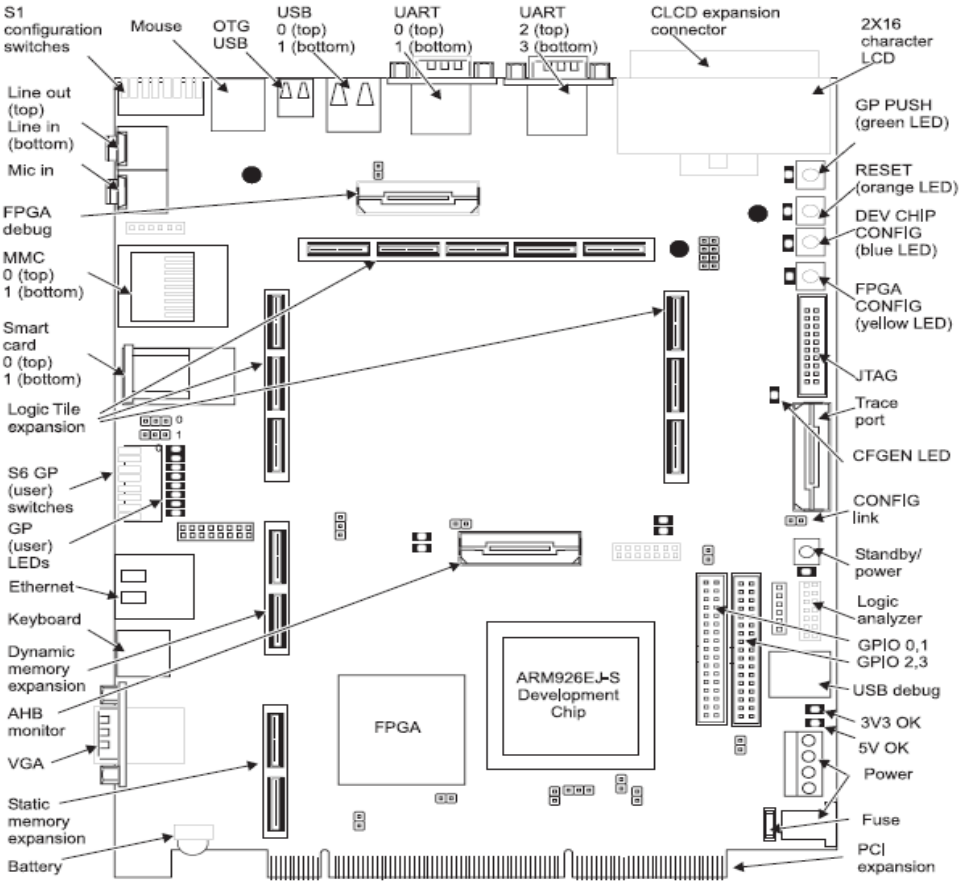


圖 4-6 ARM Versatile 的佈局

圖 4-7 表示 AHB M1 匯流排的介面[16]。DSP 核對 ARM 來說，為從（Slave）處理器，經由事先的定義，ARM 可經由 HADDR、HWRITE、HWDATA 和 HRDATA 來存取 DSP 的狀態。

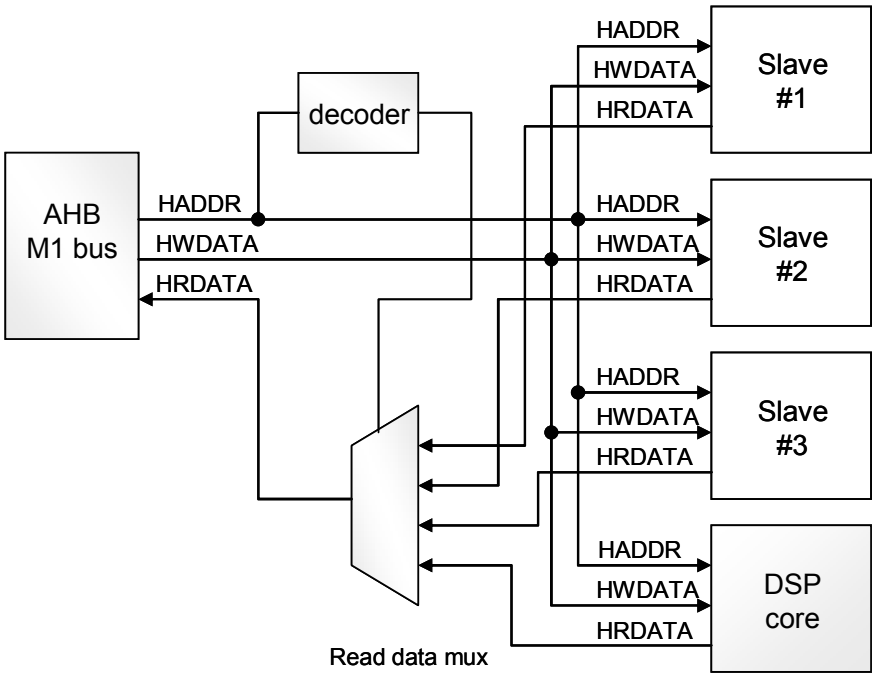


圖 4-7 AHB M1 匯流排介面

圖 4-8 表示 ARM 與 DSP 之間對應的操作流程，ARM 為主處理器而 DSP 為從處理器。在主處理器方面，在剛開始時，ARM 先將多媒體應用程式寫入 DSP 的指令記憶體裡，並將 Data Flow Process Network 的行為寫入 HPI 的 Process List Table，之後 ARM 再寫入 HOST_INITIAL 來使 DSP 從組態設定模式進入主動模式。現在 ARM 就負責讀取輸入 FIFO Queue 的指標位置來判斷可以把輸入的資料放到哪一個資料記憶體的 Page #Number，資料傳完後並更改其指標的相對位置。若此時 ARM 收到 Interrupt，則先將 Interrupt 訊號給清除，然後讀取輸出 FIFO Queue 的指標來判斷，該去哪一個 Page #Number 來將資料取出來。這樣反覆的運作直到整個任務結束。

對於 DSP 部分，當收到 HOST_INITIAL 的 Command 時，DSP 則進入到主動模式。在主動模式下，主處理器不能再對指令記憶體與 Process List Table 做寫入的動作；DSP 在主動模式下則開始做運算，當 Process 運算完時，藉由 Mailbox 對 ARM 發出 Interrupt，要求 ARM 來把運算完的資料搬出。當 DSP 收到 DSP_RECONFIG 的 Command 時，則回到組態設定模式，等待下一個任務被分派。

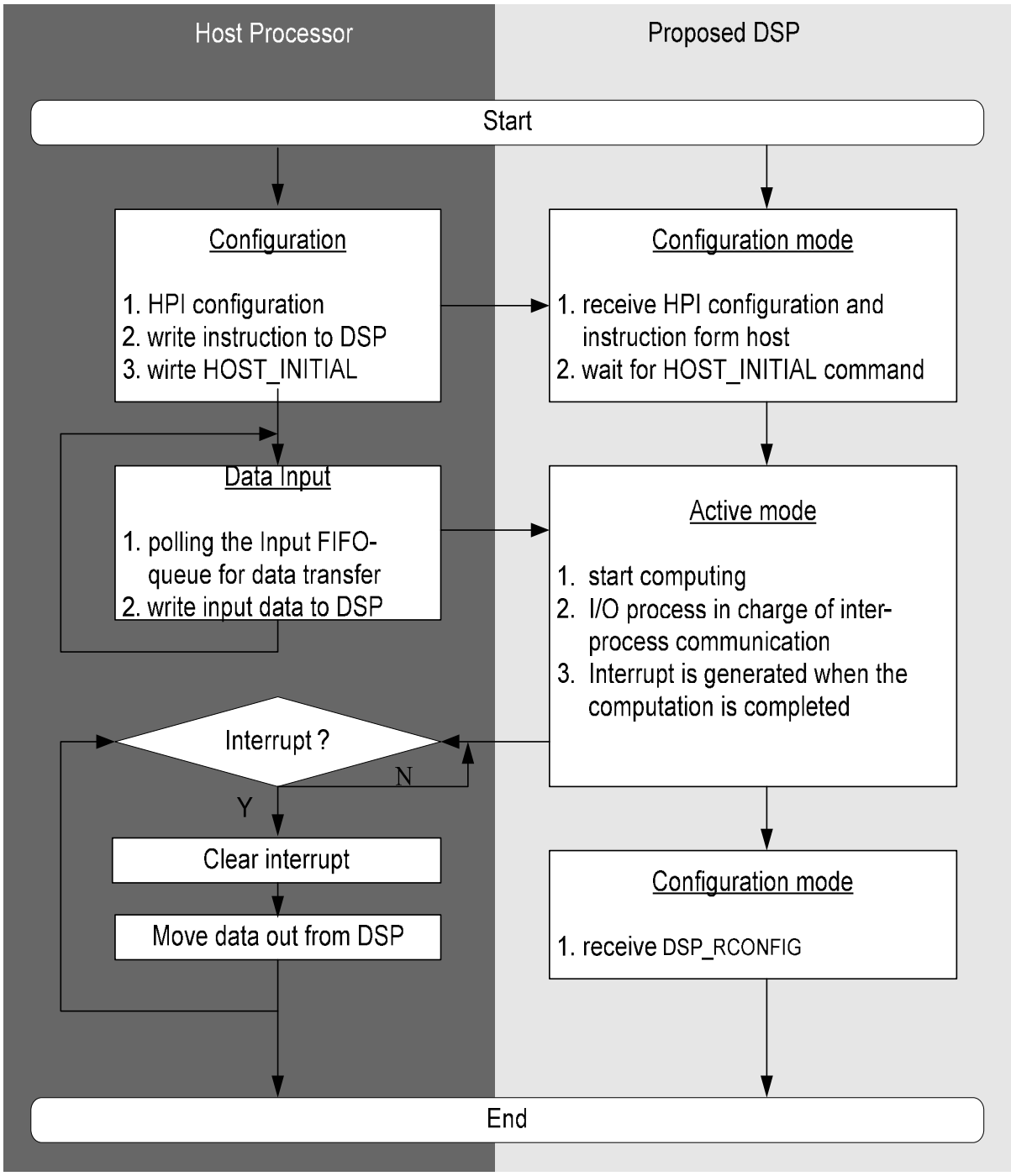


圖 4-8 操作流程

圖 4-9 表示 FPGA 的實現流程。將可合成的 RTL 電路用 Core Generator 來合成；之後用 Xilinx ISE 來實現(i.e. Translate, Map, Place & Route)整個電路；最後以 ModelSim 來驗證模擬所合成出來的電路正確性。表 4-5 為其合成出來的結果。

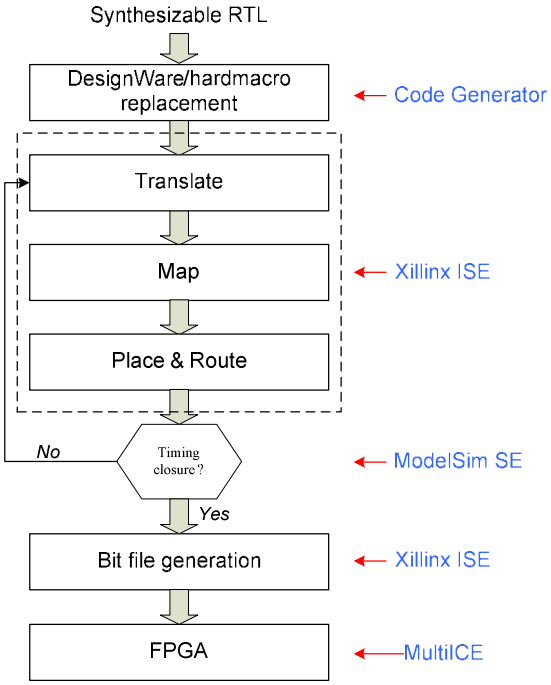


圖 4-9 FPGA 實現流程

表 4-5 FPGA 合成的結果

Device	XC2V6000	
Package	FF1517	
Speed grade	-5	
Cycle time	10.67 ns (93 MHz)	
Utilization Summary	Slices	10,679/33,792 (31%)
	Slice Flip Flops	10,154/67,584 (15%)
	4-input LUTs	12,208/67,584 (18%)
	Bounded IOBs	397/1,104 (35%)
	BRAMs	31/144 (21%)
	GCLKs	2/16 (12%)

4.3 實驗設計

我們在 ARM Versatile 設計了三個實驗，如下所示：

- Case I：以軟體來實現 Process Management 的工作，並以 ARM 來執行之

- Case II：以軟體來實現 Process Management 的工作，以 DSP 來執行
- Case III：以硬體的 HPI 來實現 Process Management 的工作

我們以 JPEG 編碼為整個實驗平台的多媒體應用程式，其相片影像為 320*240 像素 Lana 圖片。實驗目的在於比較這三種情況下其 DSP 的利用率分別為多少。表 4-6 表示 DSP 處理器等在 ARM Versatile 上的工作頻率。

表 4-6 操作頻率

Devices	Operating Frequency
Multithreaded DSP core on Xilinx Virtex II-6000	35 MHz
Host Processor- ARM926	210 MHz
AMBA AHB	35 MHz

4.3.1 IMT DSP 對 JPEG encoding 的效能

我們將 JPEG encoding 分成四個步驟，如 3.1.1 的說明。以 IMT DSP 處理器來執行 JPEG encoding，其效能如表 4-7 所示：

表 4-7 IMT DSP 執行 JPEG 的效能

Kernel	Performance (cycles)
Color space transformation	211146
Discrete cosine transform	567478
Quantization & zig-zag scan	175432
Huffman coding	1287389
Total	2241445

4.3.2 Case I-Process Management 的工作由 ARM 來執行

我們以 C 程式來描寫 Process Management 的工作並且交付給 ARM 來執行，ARM 來負責 Process 的 Scheduling、Dispatch 和資料的搬進與搬出。圖 4-10 表示 Case I 的介面，每一個 Thread Slot 有其特定的 Share Memory 來紀錄其 Process 的基本資訊(如同 HPI 的 Dispatch Table)。在剛開始時，ARM 把 Process 的基本資訊先寫入到對應的 Share Memory 裡，之後由 ARM 寫入特定的 Command 讓執行緒開始動作；當其 Process 運算完時，由 Mailbox 對 ARM 發出 Interrupt；ARM 處理 Interrupt 後將資料搬出，並且重新分配新的 Process 給 DSP 的執行緒來執行。每個 Process 的分派都由 ARM 來處理，將使處理器之間的溝通次數會增加，而沒有效率。

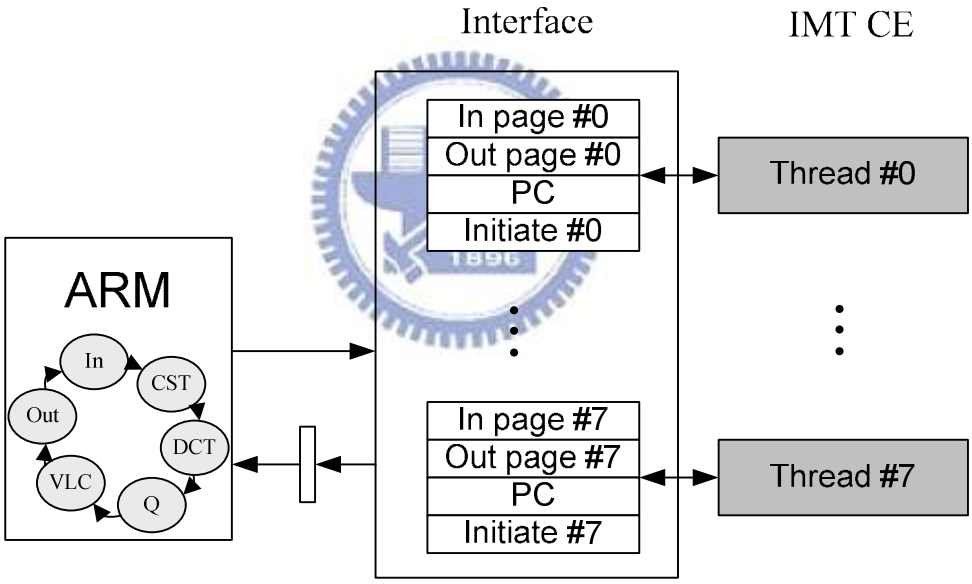


圖 4-10 Process Management 由 ARM 來處理

4.3.3 Case II-Process Management 的工作由 DSP 來執行

圖 4-11 表示其 Case II 的介面以及運算資源的分配。DSP 的執行緒分成兩種：Supervisor Thread 和 Computation Thread；DSP 的 8 個執行緒中的一個指定為 Supervisor Thread，其餘 7 個都為 Computation Thread；而其 Process Management 的工作就交由 Supervisor Thread 來負責。Supervisor Thread 還需要與主處理器做溝通，將主處理器所傳

完的資料交付給 Computation Thread 來執行；當運算完成時，負責通知 Mailbox 對 ARM 發出 Interrupt，ARM 把其運算完的資料搬出。Supervisor Thread 與 Computation Thread 之間的介面也與 Case I 相似，主要是將 Process 的基本資料寫到事先定義好的 Share Memory，之後 Supervisor 寫 Command 使 Computation Thread 開始運算，執行緒的終止也藉由 SlotFree 指令來為之。原本處理器之間的繁雜的溝通，將部分轉變成在 Supervisor Thread 與 Computation Thread 之間的溝通，使處理器之間的溝通次數可以減少；但是它佔據了 DSP 內的一個執行緒，使 DSP 的利用率也下降。

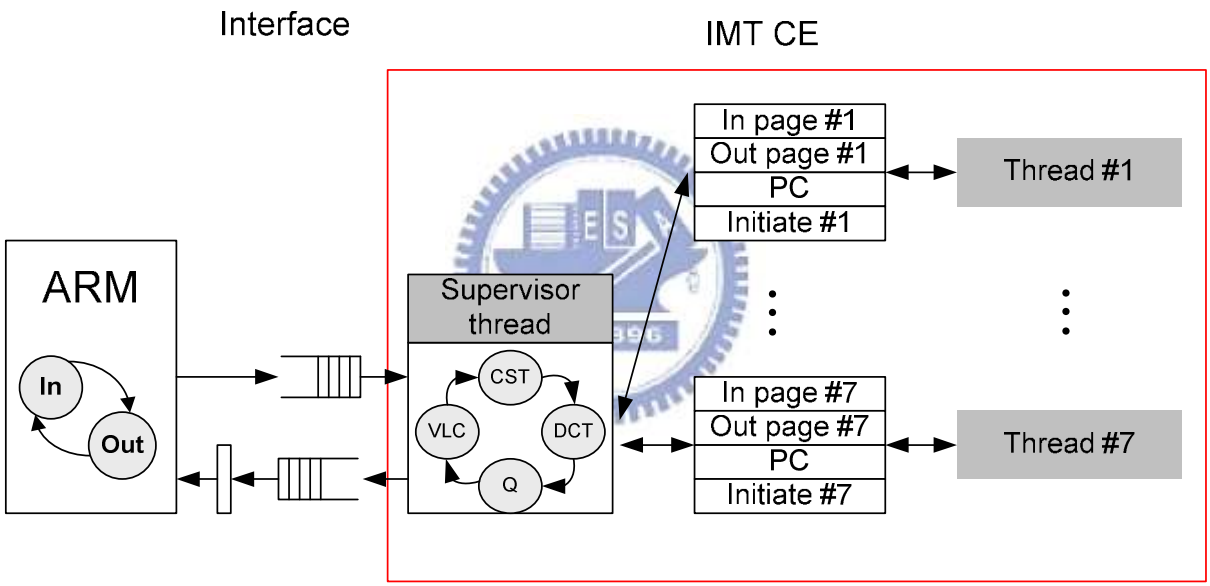


圖 4-11Process Management 由 DSP 來處理

4.3.4 Case III-Process Management 的工作由 HPI 來執行

Case III 為我們所提出的模型，詳細的操作情形請參考第三章的說明。圖 4-12 表示 HPI 負責 Process Management 的工作，分派 Process 給 DSP 來運算，與主處理器之間的溝通是經由 FIFO Queue 配合 I/O Process 來處理。

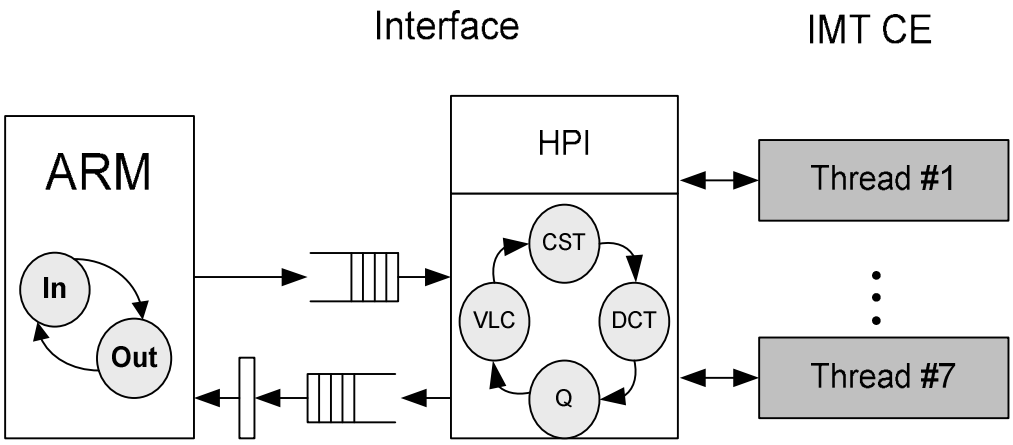


圖 4-12 Process Management 的工作交由 HPI 來處理

4.3.5 模擬結果

■ DSP 利用率計算 I

圖 4-13 表示這三個 cases 執行 JPEG Encoding 所花費的執行時間（X 軸為 JPEG 圖片的張數，Y 軸為其對應的執行時間）。

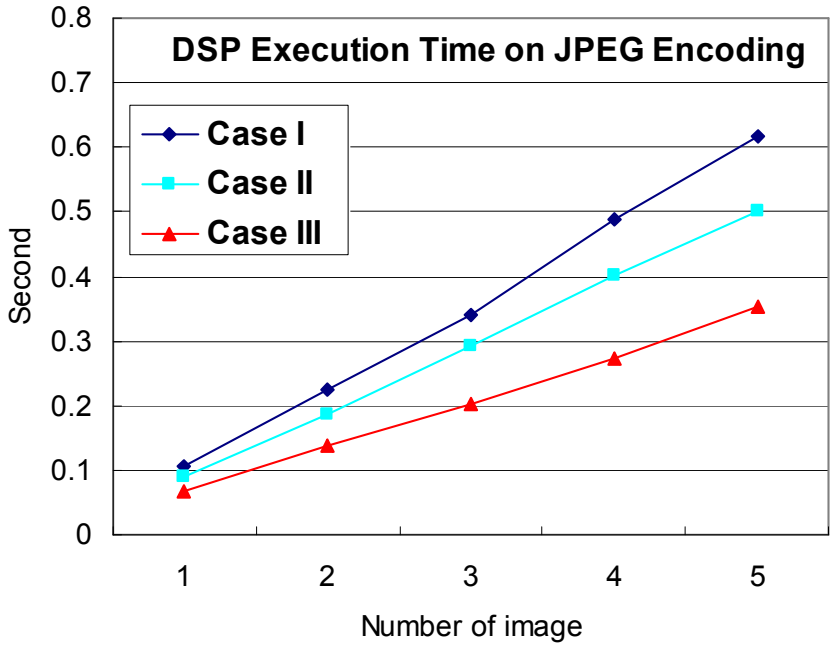


圖 4-13 DSP 執行 JPEG 的執行時間

以下公式為表 DSP 利用率的計算方式：

$$U=\frac{(\text{Total Program Count})/(\text{DSP Operation Freq.})}{(\text{Total Execution Time})} \quad (4\text{-}1)$$

經過公式計算，DSP 的利用率分別為：

- ✓ Case I：55.5%
- ✓ Case II：66.7%
- ✓ Case III：93.4%

■ DSP 空閒的時間

圖 4-14 表示 DSP 空閒的時間（DSP 空閒時間 ＝DSP 總執行時間– DSP 實際花在運算的時間）。其結果顯示 Process Management 的工作交由 HPI 來處理，相對於 Case I/Case II，將可節省 11/7 倍的時間花在實際的運算。

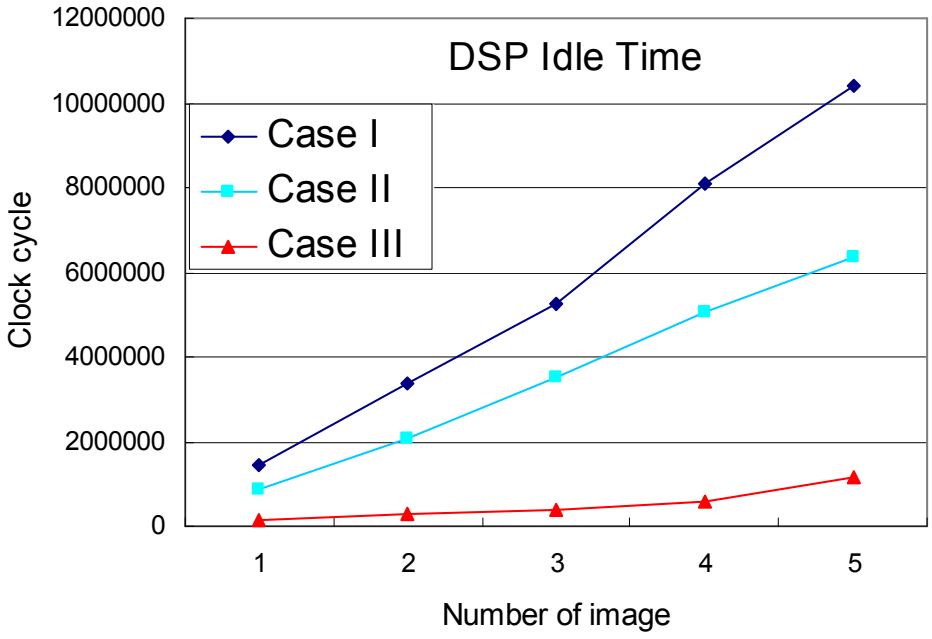


圖 4-14 DSP 空閒的時間

■ DSP 利用率計算 II（將 DSP 可運算的執行緒標準化到 7 個）

圖 4-15 表示 DSP 執行 JPEG 所花費的時間，其 DSP 內可運算的執行緒都以 7 個來計算其 DSP 的利用率：

- ✓ Case I：54.9 %
- ✓ Case II：78.6%
- ✓ Case III：92.3%

DSP 的利用率在 case II 為 78.6%，跟之前的比起來增加了 11.9%（相當於 Supervisor Thread 所佔據的利用率），其餘部分 21.4%（100-78.6）的時間是花在兩個 Processes 之間的溝通（當一個 Process 結束後，Supervisor Thread 還需要花些時間來更改 Table，經過一段時間後，另一個新的 Process 才會被分派），而執行緒每個時間週期都在做切換，會讓此段時間的溝通變的較沒效率。

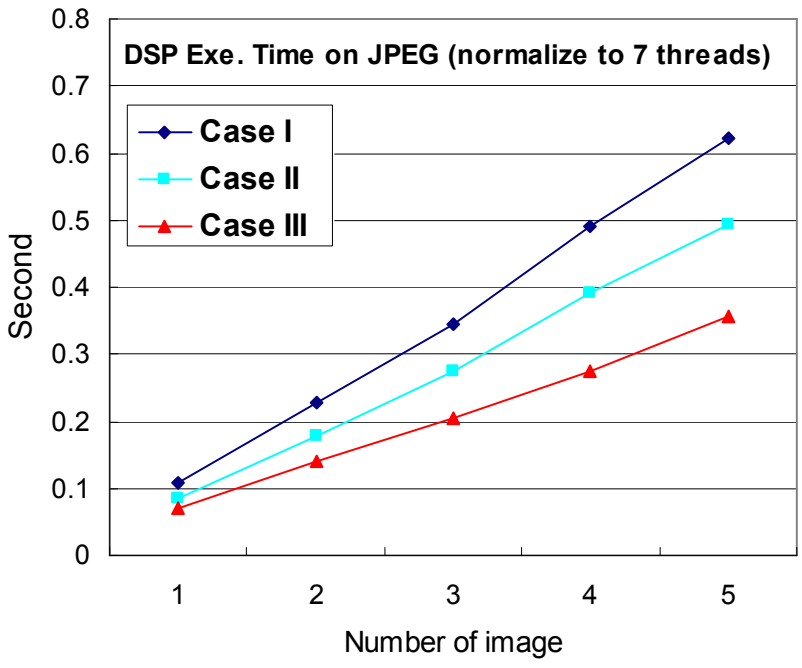
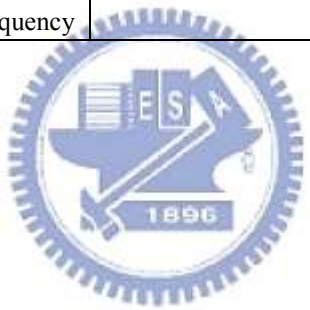


圖 4-15 DSP 執行 JPEG 的執行時間（DSP 可運算的執行緒標準化到 7 個）

如表 4-8 為使用 TSMC 0.13μm CMOS 製程技術的合成結果，提出的 HPI 所使用的電晶體數量相對於 DSP 處理器本身增加了 6.25%，是幾乎可以乎略的。

表 4-8 合成結果

Technology	TSMC 0.13μm CMOS technology
Core gate count	129536
HPI gate count	8103
Operating Frequency	100MHz



5 總結及未來工作

我們首先提及在嵌入式系統，傳統單一的通用處理器已經無法滿足現代行動多媒體所需的運算量，而以異質性整合的雙核心處理器來處理這些多樣性的任務，已經是一個普遍且正確的做法。然而我們發現 DSP 的利用率通常都不高，其原因來自於(1)指令的延遲時間造成管線裡充滿著管線延遲；(2)有限的 ILP，使得像 VLIW 架構的處理器其指令的 Issue Slot 充滿著無效的運算；(3)繁複的處理器之間的溝通與 Process Management 造成兩處理器之間的相互等待，而沒有在做運算；這些都使 DSP 的利用率都在 60%以下。我們利用了 IMT 的架構來實作 DSP 處理器，IMT 執行緒切換的方式為 Cycle-by-Cycle，可以避免 Short Stalls 和 Long Stalls；另外我們提出 HPI 來負責 Process Management，使得處理器之間的溝通次數降低，進而提升 DSP 的利用率；另外，Process 的分派是 Priority-based 的，高優先權的 Process 將會優先分派，而 DSP 的 Thread Slot 不會被低優先權的 Process 佔住而慢下來。

第二章說明指令延遲時間造成管線延遲的原因，資料相依性阻礙了 ILP。現階段高效能

處理是將以 TLP 來獲得更高的平行度，來解決 ILP 所面臨的問題。多執行緒處理器依執行緒切換的方式可分為 IMT 與 BMT。IMT 可避開因指令延遲時間與資料相依性所造成的管線延遲；而 BMT 執行緒的切換是以 Even Driven 方式，當遇到一個很長的延遲時間（如 L2 Cache Miss）才做切換。而另一種同時 Exploiting TLP 與 ILP 的 SMT，其架構建立在 Superscalar 處理器上，SMT 架構非常複雜，同時在多個執行緒中探詢平行指令，造成設計上的困難。SMT 架構，常被運用在伺服器的處理器市場中。最後提及兩處理器的溝通機制；並以 TI OMAP 為例，說明其溝通的步驟與相關需注意的議題。

第三章說明了 Data Flow Process Network 的基本概念，以及所提出的 HPI 如何以此概念來完成 Process Management 的工作。HPI 內部維護了三個 Tables：分別為 Process List Table、Queue Table 和 Dispatch Table。一個 Process 要能分派到 DSP 上必須先滿足三個條件：(1)DSP 有無閒置的 Thread Slot；(2)有無 Page 可供 Process 儲存運算完的結果；(3)選一個具有高的優先權 Process 來分派。DSP 的執行緒藉由 Dispatch Table 的 Thread Enable 來 Initialization。執行緒的 Termination 藉由特定的指令(SlotFree)加在運算 Process 的最後的一行來為之。資料記憶體分成固定大小的 Page，記憶體管理單元則在動態的去 Allocate 或 De-Allocate 這些 Page 給一個新的 Process 或一個終止的 Process 。處理器之間的溝通分三部份：(1)Synchronization 為 Mailbox；(2)Communication 為經由事先定義好的 FIFO Queue；(3)I/O Process 負責協助處理器之間的溝通。

第四章說明將所提出的 DSP 核在 ARM Versatile 上實現；另外在實驗設計方面，以 JPEG Encoding 為其多媒體應用程式，設計了三個 Cases：(1) Case I：將 Process Management 的工作由 ARM 來執行；(2) Case II：將 Process Management 的工作由 DSP 來執行；(3) Case III：將 Process Management 的工作由 HPI 來執行。從模擬的結果得知，所提出的 HPI 來負責 Process Management，其 DSP 的利用率為 93%，而其所使用的電晶體數量大約為 DSP 處理器的 6.25%，是幾乎可以忽略的。

我們現在正朝向讓 HPI 的設計能夠支援多個輸入與多個輸出（MIMO）的模型。對於 DSP 核部分，我們將增加其指令的 Issue Bandwidth 到一次可以 Issue 兩道指令，其

架構會類似於 IMT + VLIW。另外，我們將增加其管線的階數，從 5 級增加到 8 級，8 個執行緒的 IMT 允許 8 級的管線內不會產生 Short Stalls，並且 8 級的管線將允許有更複雜的資料路徑（CISC-like），使 DSP 擁有更高的效能。

參考文獻

[1] A. Gatherer, T. Stetzler, M. McMahan, and E. Auslander, "DSP-based architecture for mobile communications: past, present and future," in *IEEE Commun. Mag*., vol.38, pp.84-90, Jan. 2000

[2] J. Chaoui, K. Cyr, S. Gregorio, J. Giacalone, J. Webb, and Y. Masse, "Open multimedia application platform: enabling multimedia applications in third generation wireless terminals through a combined RISC/DSP architecture," *ICASSP*'01, vol. 2, pp. 1009-1012, May,2001

[3] M. S. Lam and R. P. Wilson, "Limits of control flow on parallelism," in *Proc. 18th ISCA,* May 27-30, 1992, pp. 46-57

[4] D. W. Wall, "Limits of instruction-level parallelism. in *Proc. Int. Conf. ASPLOS-IV,* April 8-11,1991, pp. 176-188

[5] M. Michael, T. Y. Yeh, Y. Patt, M. Alsup, H. Scales, and M. Shebanow, "Single instruction stream parallelism is greater than two," in *Proc. 18th ISCA,* May 27-30, 1992, pp. 46-57

[6] Evaluating real-time operating systems for DSP. [Online]. Available: http://www.rtcmagazine.com/home/article.php?id=100498&pg=2

[7] RTOS to DSP: what makes it tick? [Online]. Available: http://www.openlicensesociety.org/docs/RTOS_for_DSP.pdf

[8] J. L. Henessy and D. A. Patternson, *Computer Architecture – A Quantitative Approach,* 4th Edition, S.F.:Morgan Kaufmann Publishers, 2007

[9] Theo Ungerer, Borut Robič, and Jurij Šilc, "A survey of Processors with explicit multithreading," *ACM Computing Surveys,* 35(1), 2003

[10] Inter-processor communication. [Online]. Available: http://www.3dsp.com/pdf/InterProcessorCommunication.pdf

[11] Linux DSP Gateway specification. [Online]. Available: http://dspgateway.sourceforge.net/pub/3.1/DSP_Gateway31_spec.pdf

[12] PrimeCell vectored interrupt controller (PL190). [Online]. Available: http://www.arm.com/pdfs/DDI0181E_vic_pl190_r1p2_trm.pdf

[13] E. A. Lee and T. M. Pards, “Dataflow process networks,” in *Proc. IEEE,* vol83, 1995

[14] Independent JPEG Group, http//www.ijg.org

[15] Versatile Platform Baseboard for ARM926EJ-S. [Online]. Available: http://www.arm.com/

[16] AMBA Specification Rev 2.0, ARM Limited, 1999

作者簡歷

卓志宏，1976 年 4 月 11 日出生於彰化縣。2002 年取得明新科技大學電機工程學系學士學位。2007 年在劉志尉教授指導下，取得碩士學位。本篇論文「適於雙核心多媒體系統晶片之多重執行緒協同處理器介面」為其碩士論文。