國 立 清 華 大 學

碩 士 論 文

題目：虛擬輸入、動態輸出佇列於網路處理器系統上之應用與研製

The Design and Implementation of Virtual Input Queue / Dynamic Output Queue On Network Processor System

所別：通訊工程研究所

學號：885610　姓名：陳闕民

指導教授：黃能富 教授

中華民國九十年六月

摘 要

網路的使用愈來愈普及，因此有許多新的服務及應用來因應使用者的須求。為了因應網路的速度，網路設備廠商就會將這些網路服務製成硬體的功能。但開發 ASIC 所需的時間冗長費時，往往趕不上網路的速度；另一方面，製成 ASIC 後，若有問題需要修改，所需的人力物資成本非常高，為了解決原先網路設備開發上的缺點，於是就有了網路處理器(Network Processor)的概念；其主要構想是設計一顆專門用來處理網路資訊處理器，可以利用它來設計相關處理網路服務的程式。好處是它內部即有處理網路資訊的指令，可減少開發產品的時間；另外，它可依使用者所須，開發不同的應用，並且可以隨時修正更改，減少設計錯誤造成的風險，增加網路設備設計上的彈性，因此其為將來網路處理最佳解決方案。

但網路處理器的概念目前尚未成熟。就我們所選擇的網路處理器而言，由於其硬體架構的關係，會有與縱橫式(Crossbar)交換器類似的前端資料阻塞(Head of line blocking)問題，因此我們設計虛擬輸入佇列(VIQ)，讓輸入封包標頭先轉送到記憶體，使新的封包能夠及時被處理，解決輸入端的瓶頸。另外若輸出佇列只有一個，將不能及時處理大量連續的輸出封包，因此設計了動態輸出佇列(DOQ)，使連續輸出資料不會受影響，如此，一方面可補強原先輸入端設計架構的不足，另一方面讓原先輸出端的架構，能發揮最好的使用性。

接著我們依此架構，設計數個測試條件，證明同時有虛擬輸入佇列及動態輸出佇列的情況下，所能提昇的效能及結果分析，讓網路處理器發揮最佳的效能。

目錄

圖形列表

第一章

導論

1-1 網路的發展

身處二十一世紀初的現在，電腦幾乎已成為每個家庭必備的資訊家電，而大部分電腦族使用電腦的目的多是用來上網，主要原因就是方便性。透過網際網路，我們可以很容易找尋到我們有興趣的資訊，網路從早期提供學術研究之用，到現在已普遍深入大眾生活；網際網路從最原始用來搜尋資料，到現在更能透過它讓我們生活更便利，食、衣、住、行、育、樂幾乎都被網路 e 化了。

以台灣為例，使用網路的人口，從 1996 年 4 月開始最初 40 萬人開始使用網路，到 2001 年初 626 萬人[1]，在短短的幾年間，上網的人口幾乎占了台灣人口 2200 萬人的將近 3 成[2]，大概每年增加 5 成人數的倍數在持續增加中，成長的比例相當的驚人。使用網際網路的人數一增多，相對的網路資料的流量就增大，最直接衝擊的就是網路頻寬的問題；因此網路硬體也在這幾年中漸漸變化，主要就是透過硬體設備線路技術的加強，增加足夠的網路頻寬容納日益增多的網路流量。因此，在硬體頻寬規格，也從早期 10Mbps 的速度到 100Mbps，到最近相當熱門的超高速乙太網路 (Gigabit Ethernet) ，已成為目前的要求標準，隨著如此的發展，將來一定還會有更快的標準，因此也唯有增加網路硬體上的效能，才能提供增加的網路頻寬，足以應付愈來愈多的多媒體網路應用。

1-2 網路處理器在網際網路中所扮演的角色

再來我們從硬體角度切入，交換器是置於網路主要幹道上，交換傳遞網路封包的網路設備，其內部架構，只是單純用簡單的電路交換晶片處理，對於目前的網路頻寬尚還能應付，但隨著網際網路愈來愈普及，網路的應用也朝著多媒體發展，硬體的頻寬也相對的增加，因此目前的硬體架構，可能就無法應付及時龐大的網路資料。除此之外新的網路服務，例如：排程(Scheduler)、封包運作管理(Packet Manipulation)、封包分類(Classifier)等技術，也不是舊硬體所能應付的；另一方面，若每家網路設備公司，為了開發前面所提的網路服務，必須另外多花人力、時間開發相關的硬體晶片，也是相當耗時的。

因此若有一顆處理器，專門設計用來處理網路封包，提供各家網路設備公司，依造自己的須求，設計相關的網路設備，如此減少開發時間及成本。基於這樣的考慮，目前較有名的電腦晶片設計廠商，如：Intel、IBM、MMC 及 Vitesse [3][4][5][6]，都有開發這類功能的網路處理器(Network Processor)；但是因為才剛起步，各家並無一套共識的硬體標準，但將來也不一定會有統一的標準，因此架構上特性有別，但目的都是設計一顆能完全解決超高速乙太網路流量及能即時提供新的網路服務須求的最佳晶片，能有效即時處理網路封包的網路處理器，由其為主再搭配相關硬體，購成一部高階網路處理設備。

1-3 為何須要 VIQ / DOQ 及硬體上的限制

前面提到，網路處理器（NPU）目前還不是一顆公認電腦內部必備晶片(IC)，因此各家所開出來的硬體規格不一。也因為剛起步，一顆網路處理器真正必備的功能、特性為何，各硬體廠商也尚在摸索，

也就是在還未正式成為工業界使用的商品前，還未能完全確定怎樣的硬體規格才能達到所謂輔助中央處理器，又能有效處理網路封包的功能，因此硬體上多少都會有些限制。

以目前這樣一個專門處理網路封包的晶片而言，最直接的應用，就是作為一個專門處理封包的交換器。且在網路頻寬愈來愈大，處理速度的要求愈來愈高的情況下，網路處理器，應用在交換器算是很適合。但以硬體為架構的縱橫式交換器(crossbar switch)會有一些硬體上的問題，例如前端資料阻塞(Head of line blocking)現象，造成封包處理效能的減低，目前解決方式是加輸入佇列(Input Queue)或輸出佇列(Output Queue)於交換器及路徑器前後，改善效能。基於這個理由，我們考慮到網路處理器亦有可能會有這個問題，因此便考慮將目前原有交換器上的技術，虛擬輸入佇列(Virtual Input Queue, VIQ)及動態輸出佇列(Dynamic Output Queue, DOQ)應用在網路處理器的硬體架構上，接下來後面的章節會再詳細討論，因會更能發揮網路處理器的效能。

本論文其他章節安排如下，第二章將介紹網路處理器，並將比較數顆不同廠商所生產的網路處理器，且分析我們所選擇採用 Vitesse IQ2000 的原因並介紹其特性；第三章則介紹交換器的種類，及提昇交換器效能所使用的方式；第四章將會提出 Vitesse IQ2000 效能處理的瓶頸，並如何將解決交換器效能所用佇列的觀念應用於此網路處理器系統上，並說明我們所設計的虛擬輸入佇列及動態輸出佇列架構整合方式，第五章將依第四章所提的架構，設計數個測試，將其效能及結果作分析，最後第六章為總結及未來的展望。

第二章

網路處理器介紹

就整個超高速乙太網路網路交換設備解決方案而言，網路處理器主要的目的是即時處理轉送網路上的封包，功能是為用來處理在超高速網路環境中，維護網路相關資料、控制封包流程正確性而特別設計。因此我們便有了以網路處理器為核心的高頻寬網路處理系統，接著我們將介紹數個網路處理器的架構及比較硬體特性，然後說明為何我們選擇 Vitesse IQ2000 網路處理器及其架構的詳細介紹。

2-1 網路處理器特性介紹及比較

目前工業界已發展網路通訊相關晶片的公司，大部分皆有研發網路處理器功能的晶片。因此我們將目前市面上已開發網路處理器晶片等幾家公司（例如：MMC、Intel、IBM 及 Vitesse 等）[3][4][5][6]的網路處理器所公布的資料，接著大略介紹如下：

2-1-1 Intel IXP1200 網路處理器硬體特性

Intel IXP1200 網路處理器架構如圖 1 所示，其硬體特性如下：

- 可程式微引擎（ Programmable Microengines ）：
  - 包含六個可程式微引擎，執行頻率為 200 MHz，包含專為網路處理而設計的指令集。
  - 每個微引擎(Microengine)包含 4 個程序(Thread) 。
  - 大暫存器集合：128 個普通指令暫存器(General-Purpose Registers)及 128 個傳輸指令暫存器(Transfer Registers) 。
  - 1K x 32-bit 指令控制儲存空間。

- Integrated StrongARM Core **中央處理器：**
  - **高效能、低耗電的** 32-bit **內崁式**(embedded)**精簡指令集處理器**(Embedded RISC processor) 。
  - 16 Kbytes **指令快取**(instruction cache) 。
  - 8 Kbytes **資料快取**(data cache) 。
  - **只使用一次即更換資料的迷你快取**(mini-cache) 512 bytes。
- 64-bit SDRAM **介面：**
  - **位址最高可達** 256 Mbytes。
  - **頻寬最高可達** 800 Mbytes / sec。
- 32-bit SRAM **介面：**
  - **位址最高可達** 8 Mbytes SRAM。
  - **頻寬最高可達** 400 Mbytes / sec。
  - **位址最高** 8 Mbytes FlashROM，**供啟動** StrongARM Core。
- **高頻寬** I/O Bus (IX Bus)：
  - 64-bit，**最高達** 85 MHz **操作頻率。**
  - 4.2 Gbps **頻寬。**
  - 64-bit **或** dual 32-bit Bus **選擇。**
- 32-bit，66 MHz PCI **介面：**
  - PCI 2.2 **版規格。**
  - 264 Mbytes/sec **連續傳送模式**(burst mode)**操作。**
  - Dual DMA channel。
- **其他特性：**
  - Serial UART port。
  - **即時時脈**(Real Time Clock) 。
  - **四個** general-purpose I/O pins。

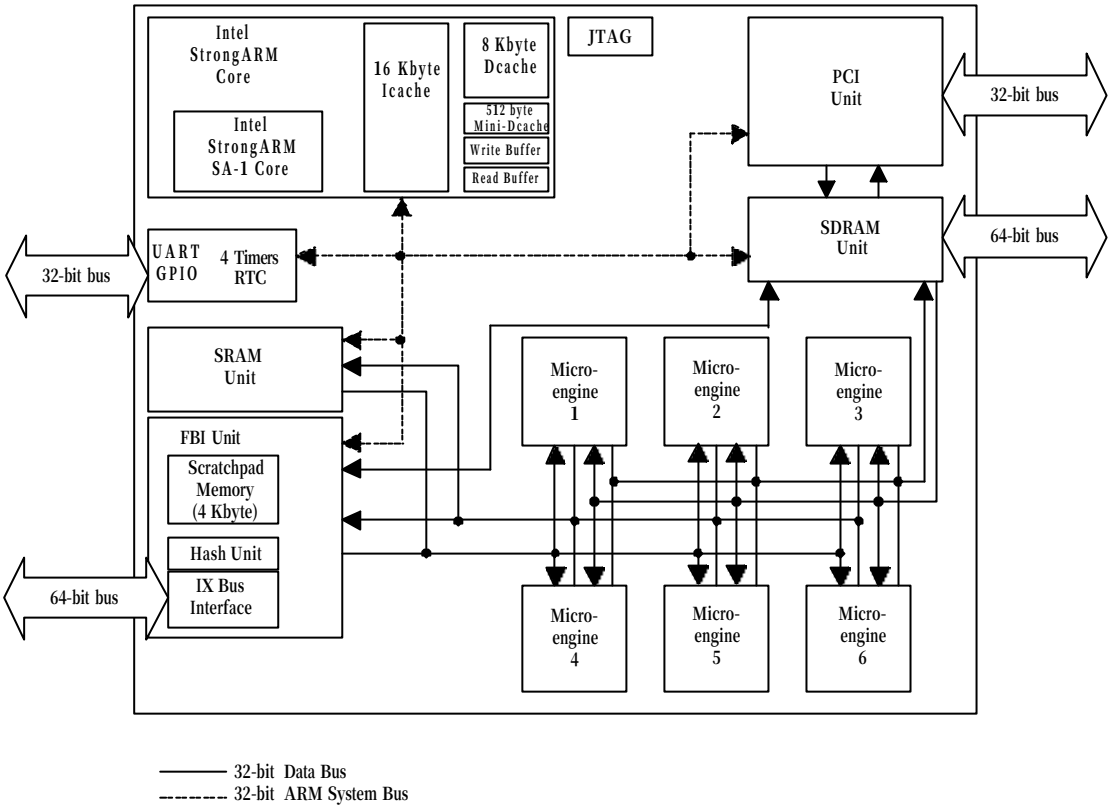

圖 1　Intel IXP1200 **網路處理器架構方塊圖**

2-1-2 IBM IBM32PR161EPXCAC133 網路處理器硬體特性

IBM IBM32PR161EPXCAC133 **網路處理器架構如圖** 2 **所示，其硬體特性如下：**

- **可程式協定處理器**(Programmable Protocol Processors) **：**
  - **包含** 16 **個可程式微引擎** (Programmable Microengines)**，執行頻率為** 133MHz**，包含專為網路處理**

而設計的指令集。

- 3-stage pipeline (fetch, decode, and execute) 。
- **普通目的暫存器**(general purpose register)、**特殊目的暫存器**(special purpose register) 。
- 8 **個指令快取。**
- 2 **個** Protocol Processor **作特殊目的用：**1 **個為** Guided Frame Handler，**另一個為** Generic Tree Handler。
- **每個** Protocol Processor **都包含** 7 **個** coprocessors，**提供** Data Store、Checksum、Enqueue、Interface、String Copy、Counter **及** Policy **的功能。**

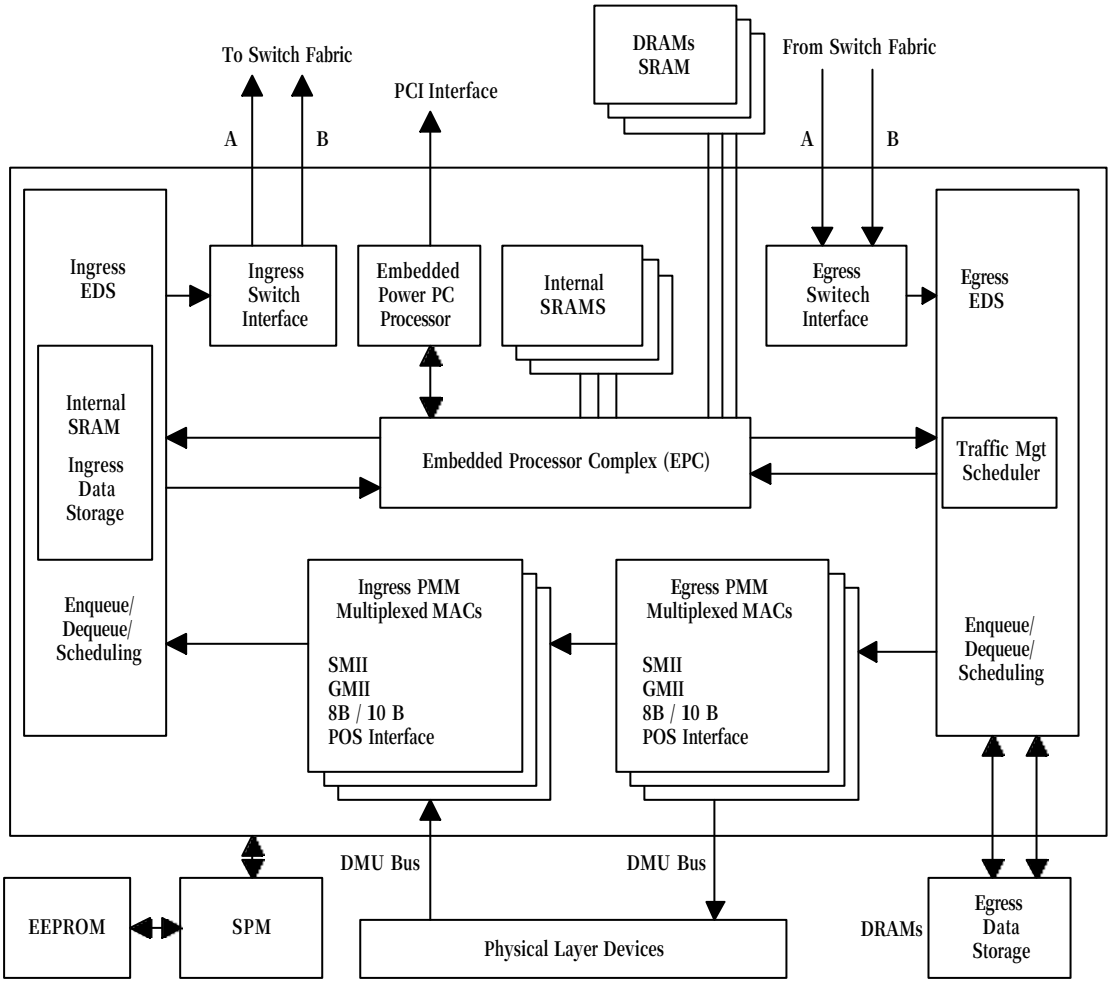

圖 2　IBM IBM32PR161EPXCAC133 **網路處理器架構方塊圖**

- **內崁式** PowerPC **處理器，提供網路連接裝置附加的設計彈性：**
  - 40 **個** Fast Ethernet / 4-Gb MACs **介面。**
  - **二個** 3.5 Gbps **高速** DASL **串列埠**(serial port)**，最多可提供** 1024 Fast Ethernet **埠。**
- **使用高容量** DDR DRAM**：**
  - **能夠儲存較大的** Forwarding table**。**
  - **能夠儲存較多的** Traffic flow queues**。**
  - **成本較傳統** SRAM **低。**

2-1-3 MMC GPIF-207 網路處理器硬體特性

MMC GPIF-207 **網路處理器架構如圖** 3 **所示，其硬體特性如下：**

- XPIF Core **模組：**

每個 XPIF Core **包含** Packet Transform Engine (PTE)**及其相關聯的** Data FIFO **和** 4 **個** Channel Control Modules**，還有內部記憶體對應控制暫存器**(internal memory-mapped control registers)**及** 1 **個** Micro-controller**。**

- Packet Transform Engine (PTE)**：**PTE **可透過。** microcontroller **所下的參數執行封包的轉換，並且每個** XPIF **可同時處埋** 16 **個** Ethernet frames**。**
- Microcontrollers**：**Microcontroller **是** XPIF **的核心，提供使用者設計封包的處理流程，共有** 4 **個** Channel**，**2 **個** ingress channel**、**2 **個** egress channel**，供可同時處理** 16 **個** tasks**。**
- Statistics Engine**：每個** XPIF **都包含一個硬體架構的** Statistics Engine**，負責收集傳送** ingress **及** egress **的封包**

資訊。

- Interface 模組：

Slicer 提供 Switch MUX 及 MMC switch fabric 之間的介面；External GGI block 提供 MAC MUX 及任何遵守 GGI 標準所設計裝置的介面。

  - Slicer：負責將 Ethernet 和 ATM 封包格式互換。
  - Generic Gigabit Interface (GGI) blocks：提供 Gigabit Packet Transform (PTE) 及有 GGI 規格的 generic packet-based 裝置之間的介面。

- MUX 模組：此模組就如同縱橫式交換器的運作方式，置於代理者(Agents)，例如：置於 Slicer 或 GGI，和 PTE 之間。

- Global Resource 模組：在 GPIF 中的二個 microcontrollers 並沒有直接連接，而是透過 Global Resource 模組，負責傳送及維護雙方共同的資料。

- Internal and External Memory Access 模組：

  - Memory Access Unit：透過此介面存取外部搜尋機制(External Search Machine)及記憶體的資料。
  - Policy Engine：此封包分類模組可以藉由 microcontrollers 作封包分類 (packet-classification) 或查尋函式 (lookup function)的轉換，這些函式範圍包括，Layer-2 address matching、Layer-3 longest match 到 Layer-4 wildcard 及 range-access control。
  - External Search Controller：這個模組控制 GPIF 和外部搜尋引擎裝置(External Search Engine device) 的資料傳輸工

作。

- External Search Coprocessor / Memory Interface：此記憶體介面可透過程式設定成數種形式，例如：pipelined SCD、pipelined DCD、ZBT 及 flow through，分別針對 32-bit 或 64-bit 的資料寬度。

- Management Control 模組：
  - MII Management Control Interface：microcontrollers 經由此 serial MII Management Control Interface 控制 GGI 裝置，在主樸模式區分，在此介面上以 GPIF 為主，GGI 裝置為樸的關係構成。

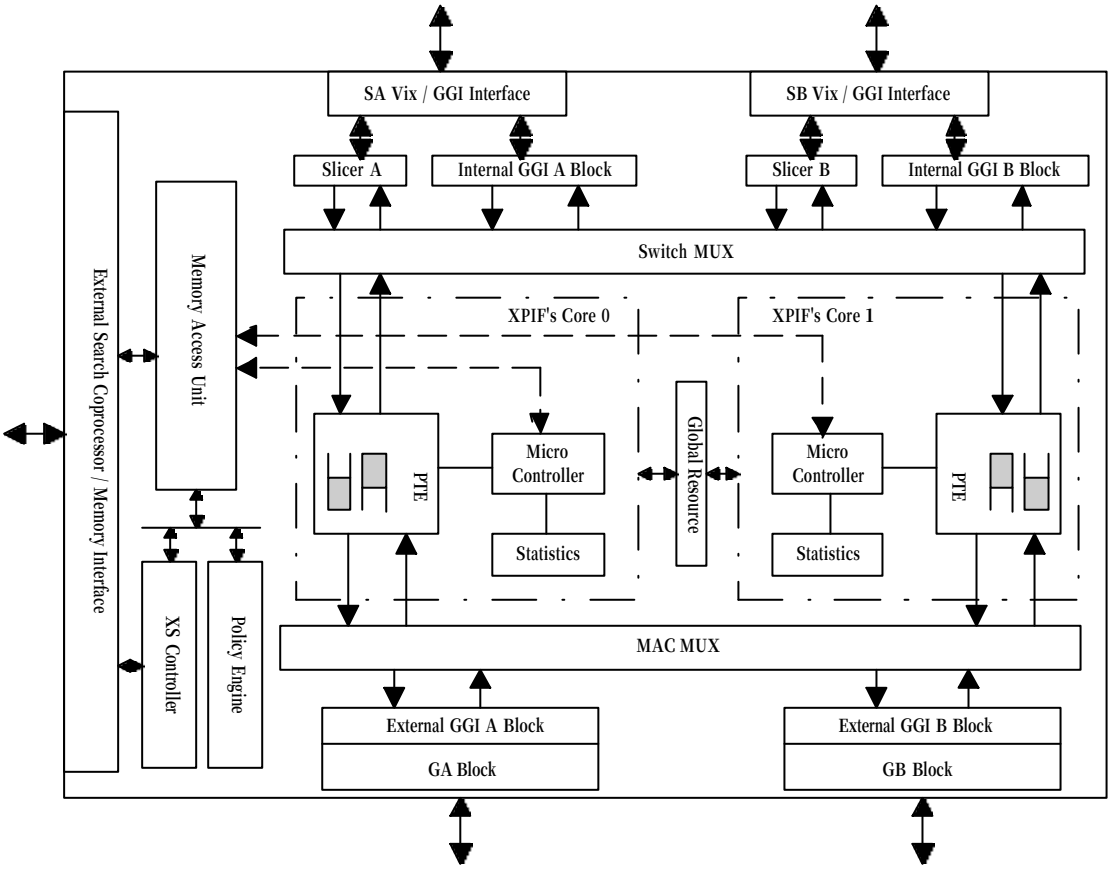

圖 3　MMC GPIF-207 網路處理器架構方塊圖

- Slave-MII Management Control Interface：此 Slave-MII block 可以藉由 CPU 去存取 Slicers 裡面的暫存器, 若 GPIF 在 GGI 或 Mixed 模式下正在運作，此介面會啟動 Slicer 經由此介面去讀取或寫入狀態 (status) 及配置 (configuration) 暫存器經而不是透過 Vix 模式。

2-1-4 Vitesse IQ2000 網路處理器硬體特性

Vitesee IQ2000 網路處理器架構如圖 4 所示，其硬體特性如下：

- 封包處理引擎 (PPE)
  - 包含四個封包處理引擎，執行頻率為 200MHz 的精簡指令集中央處理器 (FACET RISC CPU)，包含專為網路處理而設計的指令集。

每個封包處理引擎包含：

◆硬體加速直接記憶體存取輔助處理器 (DMA coprocessor) 。

◆5 stage pipeline。

◆查詢輔助處理器 (Lookup coprocessor) 。

◆2 Kbytes 內部資料儲存區。

◆5 個硬體執行程序 (1 個核心程序、4 個使用者程序)，每個程序皆有 32 個一般暫存器。

◆2 Kbytes 內部資料標頭緩衝區 (header buffer) 儲存區，可存 16 個 128 bytes 大小的資料標頭 (headers) 。

- 4 個高速週邊連接 Focus Bus 介面：
  - 在 Focus 16 模式下總頻寬為 3.2Gbps。

- 100 MHz 的介面速率。

- MIPS CPU Interface：32Bit / 64Bit MIPS CPU Interface。
- **順序管理器**（Order Manager）**：分派封包資料標頭緩衝區及保存記錄封包順序。**
- **溢出佇列模組**（Overflow Queue Module）**：當佇列**(Queue)**超過主封包資料標頭緩衝區所能存放時，其可控制將超過多出的佇列先暫時存放在溢出佇列模組中，同時可由使用者設定要啟動或關閉此功能。**
- **記憶體控制器：**
  - **最快可達** 800MHz、2 bytes、12.8 Gbps **頻寬。**
  - **最多可接** 32 **個** RDRAM，**上限最多** 2 GB **記憶體。**
- **封包輸入模組**（PIM）**：**
  - 4 **個** PIM，**但目前我們只用到支援** GMAC **的** PIM-A **及** PIM-B。
  - **標準** Focus16 **介面。**
  - **時脈速率達** 100 MHz。
- **封包輸出模組**（POM）**：**
  - 4 **個** POM，**但在此我們只用到支援** GMAC **的** POM-A **及** POM-B。
  - **標準** Focus16 **介面。**
  - **時脈速率達** 100 MHz。
  - 65 **個輸出佇列。**
  - **服務品質保證硬體支援，易作頻寬管理。**

- Smart Buffers (SB) ：
  - **內部包含** 8 Kbytes **封包資料標頭儲存區。**
  - **智慧輸出佇列管理。**
- **緩衝區配置** / **歸還**(Buffer Allocator / Deallocator , BAD)：
  - **硬體配置** / **歸還資料緩衝區。**
  - 8 **個資料槽，每個** 16 Mbytes**，共可放** 128 Mbytes **資料緩衝區。**

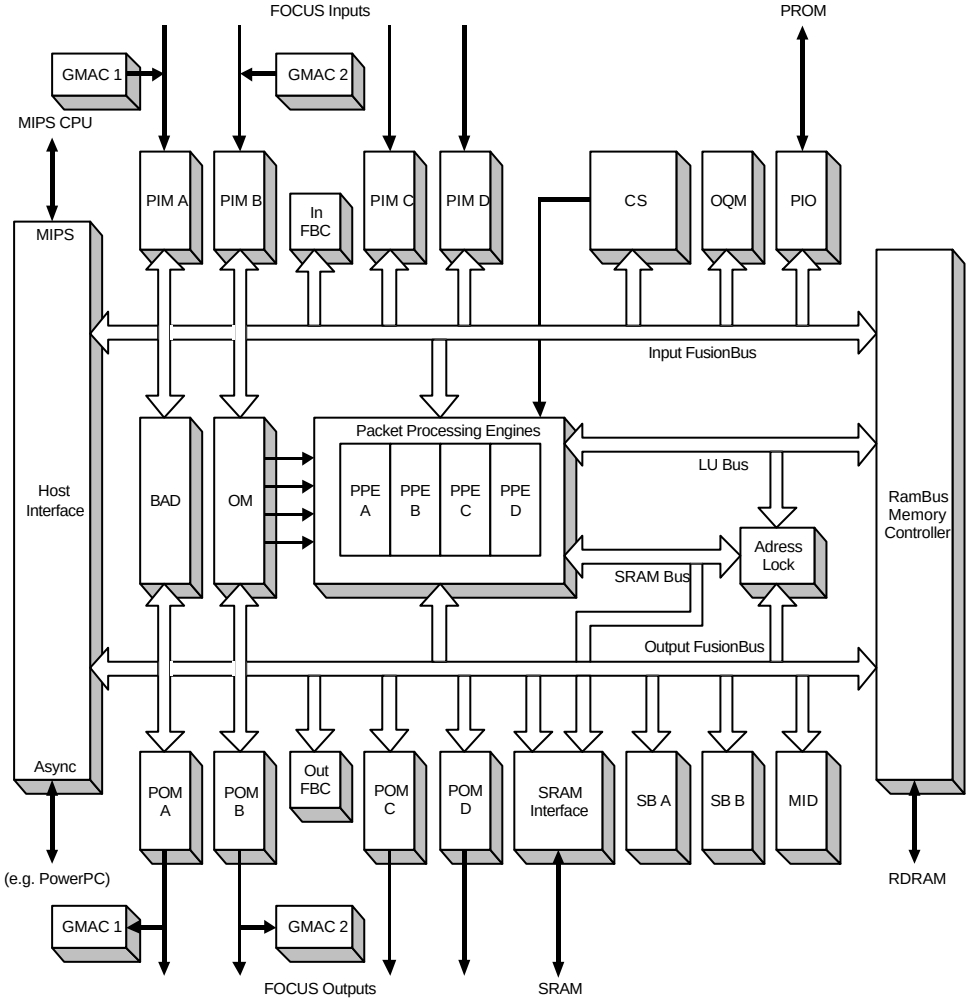

圖 4　Prism IQ2000 **網路處理器架構方塊圖**

- PROM/IO Controller (PIO)：

- 8 或 16 bit 資料匯流排。
- 8 個 PROM 或 IO 區。
- PROM/Flash-EPROM 介面。
- 整合 Integrated 16550 UART 介面。

- Gigabit MACs：
  - 雙向 Gigabit Ethernet MACs。
  - 每向都有 256 bytes 先入先出 FIFO 佇列。
  - 支援 GMII 或 TBI PHY 連接器。
  - 使用 802.3 Pause Frames 對稱或非對稱流程控制器。
- SRAM 介面：
  - 提供封包處理引擎以 32 bit 100MHz 速度存取 ZBT SRAM 記憶體。
  - 最大可達 4 Mbytes 大小。

2-1-5 各廠牌網路處理器相同特性比較

由前幾節幾家廠牌網路處理器特性中，我們將其相同性質的部分各別列出比較，製成下表，由表中可以明顯看出，各廠牌的優劣處和其架構的特色，其中空白部分為其所提供資料沒有特別說明，無從比較。

	Vitesse	Intel	IBM	MMC
Product	IQ2000	IXP1200	IBM32PR161 EPXCAC133	GPIF-207
No. of CPUs	4	6 + 1	16 + 1	2

Clock Speed	200 MHz	200 MHz	133 MHz	200 MHz
Multithread	4	4	2	16
Stage	5	5	3	7
Pipeline	5	5	3	4
No. of Reg.	32	256	-	-
Instruction Size	32 Kbytes	4 Kbytes	-	16 Kbytes
Memory Interface	RDRAM	SDRAM	DDR DRAM	SSRAM
Memory Max Size	2 G bytes	256 Mbytes	64 Mbytes	16 Mbytes
SRAM Max Size	4 Mbytes	8 Mbytes	4 Mbytes	-
External Bus Bandwidth	12.8 Gbps	5.44 Gbps	3.5 Gbps	6.4 Gbps
CPU Interface	External MIPS CPU	Embedded StrongARM	Embedded PowerPC	External PowerPC
Power	7 W	6 W	20 W	4 W

2-1-6 選擇 Vitesse IQ2000 晶片的原因

我們從目前市面上已開發的網路處理器晶片等幾家公司（例如：MMC、Intel、IBM 及 Vitesse 等）[3][4][5][6]的網路處理器所公布的特性分析，最後決定採用 Vitesse 的公司所開發的網路處理器晶片 Prism IQ2000 [7]，之所以為用這顆網路處理器的原因如下：

- 硬體架構適合：
  - 已經整合２組 GMACs，正好適用我們的超高速乙太網路架構。
  - 共享式記憶體無組塞架構正適合網路服務品質保證管理所須。
  - 非黏著（glueless）32 / 64 bit 中央處理器介面，更易整合網路處理器和中央處理器模組。
- 速度快：
  - 記憶體使用高速 RDRAM 其容量可達 512 MB，12.8Gbps 的頻寬。
  - 4 個封包處理引擎，其為 200 MHz 精簡指令集處理器。
  - 每個封包處理引擎包含 1 個核心處理程序 Kernel Context 及 4 個使用者處理程序，所以意謂者同時最多可以有 16 個封包被處理（4x4）。
  - 25.6 Gbps 的內部排線速度。
  - 支援 ZBT SRAM 介面。
- 減少開發時程：
  - 提供公板（evaluation board）硬體及設計電路圖，能讓想開發超高速乙太網路硬體產品的公司，能較快進入狀況，

減少人力開發的時間，提供最快速解決方案。

2-2 系統硬體架構

前一節將系統最主要的特性及內部方塊圖說明後，本節我們以網路處理器為核心，建構出 Vitesse IQ2000 網路處理器系統架構如圖 5 所示，詳細說明如下：

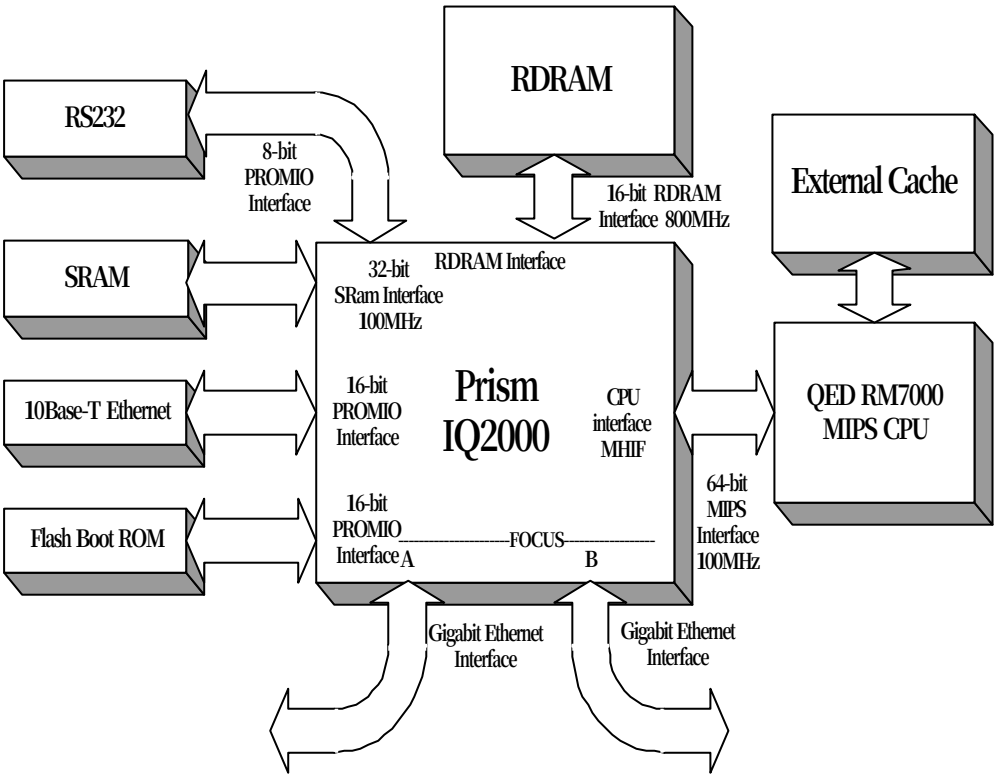

圖 5 Vitesse IQ2000 網路處理器系統方塊圖

- **中央處理器模組**

IQ2000 CPU 介面提供兩種和外部中央處理器連接的形式。第一種即我們所採用的主機介面（MIPS Host Interface），又稱 MHIF；此種介面所連接的是 MIPS 系列的中央處理器，其特性高頻寬，在此

介面上，IQ2000 相當一個主要系統的中央處理器，其速度夠快而提供 MIPS 所提出的要求及資料的存取。

另一種形式的介面叫非同步式主機介面（Asynchronous Host Interface），又稱(AHIF)；此種介面頻寬較小且慢，但優點是較不複雜。此介面下的 IQ2000 僅相當一個週邊的晶片而不是系統晶片，因此透過此介面和中央處理器傳送資料是相當慢的，因此我們不採用此架構。

- External Cache 模組

在 MIPS 中央處理器外接外部快取，將中央處理器常用資料存放於此，減少中央處理器和網路處理器搶 RDRAM 的使用權，增加系統效率。

- RDRAM 模組

  256MB RDRAM 存放中央處理器資料、網路處理器封包資料等。

- SRAM 模組

  存放自定資料。

- RS232 介面

  可透過 RS232 介面進入系統設定相關參數。

- Flash ROM 模組

  透過 PROM/IO 介面存放開機碼及其他系統相關設定資訊。

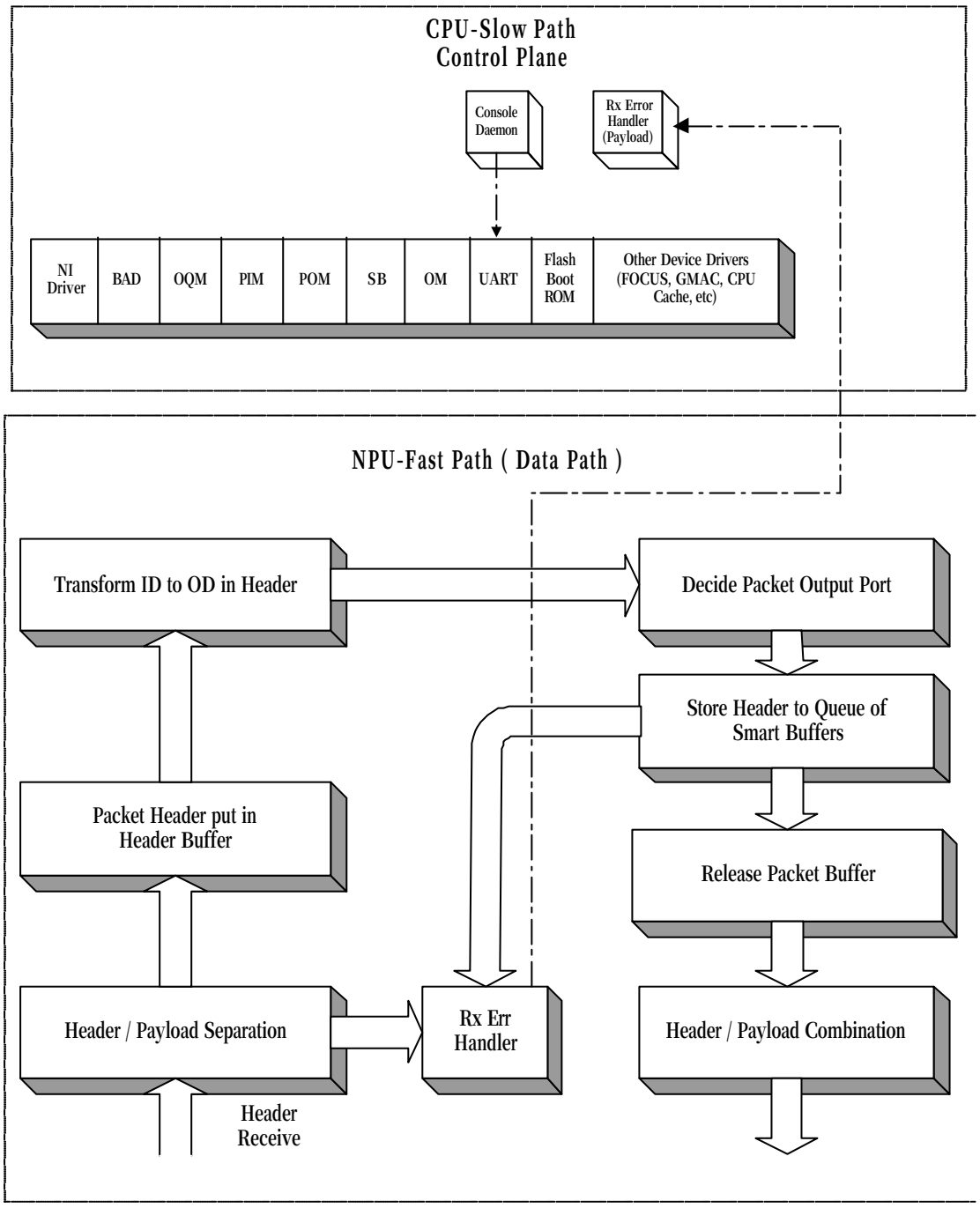

圖 6　Vitesse IQ2000 **網路處理器軟體架構**

2-3 軟體架構

Vitesse IQ2000 **網路處理器軟體架構如圖** 6 **所示，此架構主要分成兩個執行路徑，一是在網路處理器執行的路徑稱之為快速路徑，主**

要是處理網路封包資料轉送的流程，例如：檢查封包的正確性、決定封包轉送方向、封包格式的確認及轉換及服務品質保證的控制等，所有和網路封包及時流量處理動作，都在這一路徑運作；另一在中央處理器執行的路徑稱之為慢速路徑，主要是用來控制及設定系統相關的參數資料，在此路徑中運算的資料，是較不迫切的，不一定要立刻回應，例如：SNMP、Console 設定、Telnet 等，但此兩路徑是缺一不可的，一旦沒有其中一邊，整個系統將不能正確運作。

首先注意中央處理器所處理的非即時路徑 (Slow Path) ，在系統開始執行時，它必須負責將相關的網路處理器設定參數初始化，並將開機的程式段載入至快閃記憶體 (Flash Boot ROM) ，然後驅動 BAD、OQM、PIM、POM、SB、OM、UART、FOCUS、GMAC 及中央處理器快取的驅動程式，並執行一長駐的主機操作程序 (console daemon) ，在網路處理器執行時，負責處理錯誤封包內容的動作。

再來注意網路處理器所負責的部分，在此我們先看 Vitesse IQ2000 網路處理器軟體架構如圖 6 所示，快速路徑流程，一邊配合 Vitesse IQ2000 網路處理器硬體架構如圖 7 所示，即可容易明白。封包自 Focus Bus 收進 PIMA 或 PIMB 後，每個封包標頭(Header)由 4 PPE 中其中 1 個 PPE 處理，並存放於此 PPE 16 個封包標頭緩衝區(Header Buffer)其中之一，若封包大過 Header，則將其他封包的資料內容(Payload)，送到 RDRAM 存放，並同時確認封包格式的正確性。若不正確且是短封包，則直接送到錯誤處理器 Error Handler 釋放所占的空間；若是正確的封包資料，則將其輸入描述(Input Descriptor)內容轉換成輸出描述(Output Descriptor)格式，並確認其輸出的埠為何，然後將整個封包標頭送往相對的 Smart Buffer 存放，並釋放所占用封包標

頭的空間。最後 POM 將此封包標頭結合其相對的封包內容組合，之後，若是原先錯誤的長封包則在此時送至錯誤處理程序去丟棄，原先正確的封包會組合成正確封包格式送出完成封包轉送動作。

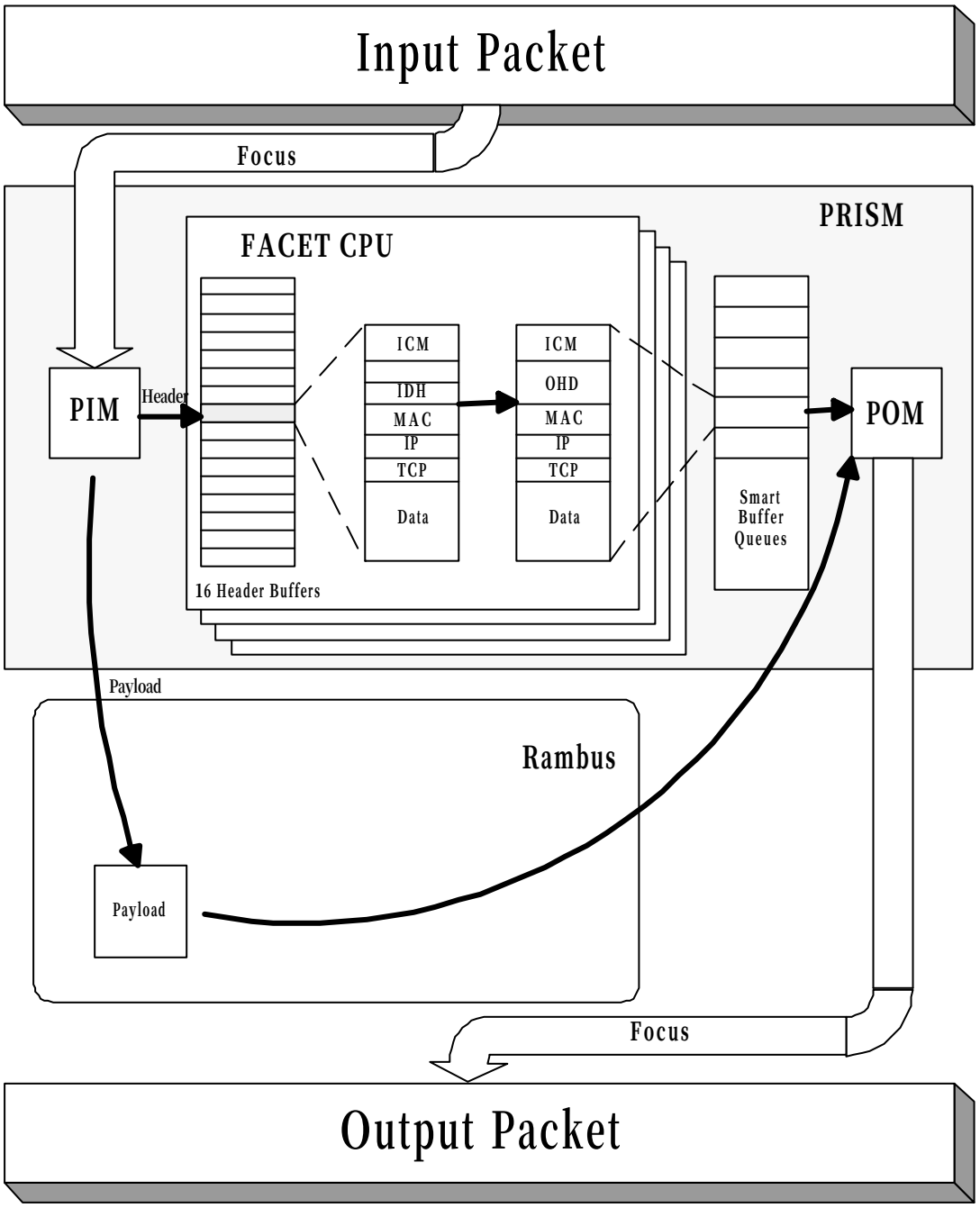

圖 7　Vitesse IQ2000 網路處理器封包處理流程

再來注意網路處理器所負責的部分，在此我們先看 Vitesse IQ2000 網路處理器軟體架構如圖 6 所示，快速路徑流程，一邊配合 Vitesse IQ2000 網路處理器硬體架構如圖 7 所示，即可容易明白。封包自 Focus Bus 收進 PIMA 或 PIMB 後, 每個封包標頭(Header)由 4 PPE 中其中 1 個 PPE 處理，並存放於此 PPE 16 個封包標頭緩衝區(Header Buffer)其中之一，若封包大過 Header，則將其他封包的資料內容(Payload)，送到 RDRAM 存放，並同時確認封包格式的正確性。若不正確且是短封包，則直接送到錯誤處理器 Error Handler 釋放所占的空間；若是正確的封包資料，則將其輸入描述(Input Descriptor)內容轉換成輸出描述(Output Descriptor)格式，並確認其輸出的埠為何，然後將整個封包標頭送往相對的 Smart Buffer 存放，並釋放所占用封包標頭的空間。最後 POM 將此封包標頭結合其相對的封包內容組合，之後，若是原先錯誤的長封包則在此時送至錯誤處理程序去丟棄，原先正確的封包會組合成正確封包格式送出完成封包轉送動作。

2-4 封包格式

本節介紹 Vitesse Prism IQ2000 的封包格式：

2-4-1 封包輸入格式

封包輸入模組 (Packet Input Module, PIM)將自 Focus Bus 收進來的封包，拆解成封包資料標頭及封包內容兩部分，並決定封包類型，下面會依序介紹：

2-4-1-1 輸入封包格式

PIM 會將接收到的封包依其封包資料標頭大小分成：Header、Complete 及 Long 三種類型。資料標頭的長度，由初始參數中的 HEADER_SIZE 欄所決定，PIM 決定封包類型以此為依據，接著我們將以上三種類型，詳述如下：

- Header Packet TYPE：

若此封包只包含第一個 FCell，且這個 FCell 的長度僅小於或等於封包資料標頭大小，則為 Header Packet 類型。

- Complete Packet TYPE：

若此封包只包含第一個 FCell，但這個 FCell 的長度僅大於封包資料標頭大小，則為 Complete Packet 類型。

- Long Packet TYPE：

若此封包包含不只一個 FCell，則為 Long Packet 類型。

在每個 PPE 都一個 128 bytes * 16 大小的記憶空間，同時存放著 16 個每個大小為 128 bytes 封包資料標頭，除了網路封包資料標頭格式內容，它還定義輸入完成訊息 (Input Completion Message) 、輸出描述器及輸入描述器的位置，存放著封包相關資料，例如：封包正確性、封包型態是否為 TCP 封包、長度及輸入埠為何等。另外還可以讓使用者自訂空間存放參數，因此不是 128 bytes 皆存放網路封包的資料。

當一個封包自 PIM 接收後，它會先將資料往封包標頭緩衝區儲存，當存滿且此封包還未收完整，其餘後面的資料，存到 RDRAM 的資料緩衝區(Data Buffers)的位置，此即封包內容的部分。因為網路封

包被拆成兩部分，由此我們可以發現，將那些屬於資料標頭的封包若再傳完後立即處理，不須再浪費時間等待處理封包內容的部分，則可以增快系統執行效率。

2-4-1-2 輸入資料鏈

若進來的封包是 Complete Packet 或 Long Packet 的類型，則封包就會被分成封包標頭及封包內容兩個部分，各自存在封包標頭緩衝區及封包內容緩衝區中。

PIM 可將數個資料緩衝區接在一起，當一個封包被寫入一個資料緩衝區，ＰＩＭ就會將其存放長度和 BUFSIZE (此值內定為 256，亦可自定為 512、1024 或 1536) 作比較。若封包長度大於 BUFSIZE - 8 則會從 Free Buffer Cache 新增另一個 DBP，並將此 DBP 寫入剛才資料緩衝區最後一個 Word64 的最前面 2 bytes，FCell 的資料將會被填入剛才資料緩衝區的最後 8 個 bytes，指向下一個新的 Data Buffer。以此種方式，將可串起一個非常大的封包。

2-4-1-3 輸入封包格式總結

Vitesse IQ2000 網路處理器資料標頭格式如圖 8 所示，可以清楚的瞭解每個 PPE 有 2 Kbytes 的空間，即可存放 16 個 128 bytes 大小的封包標頭緩衝區空間，另外每個封包標頭緩衝區的內部配置順序依序為 ICM、OHD、IHD、真正的封包標頭及使用者自訂參數。

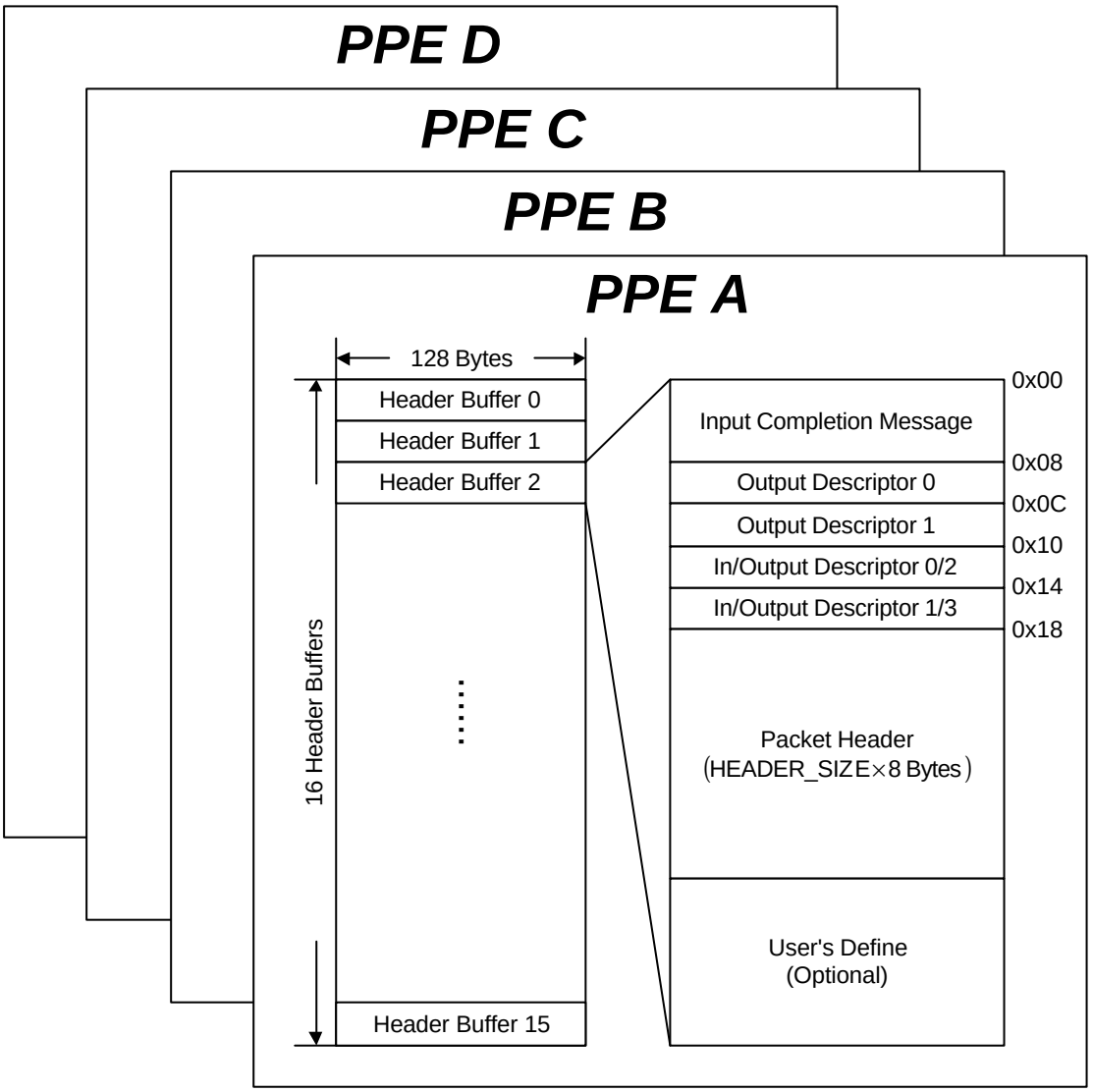

圖 8　Vitesse IQ2000 **網路處理器資料標頭格式**

Vitesse IQ2000 **網路處理器輸入封包格式如圖** 9 **所示，在輸入封包描述** (IHD) **中有二欄對應到資料緩衝區為：**Buffer Space (BSPC) **及** Data Buffer Pointer (DBP) **。但這兩個欄位的值要是封包是** Complete **及** Long **類型才有意義。**

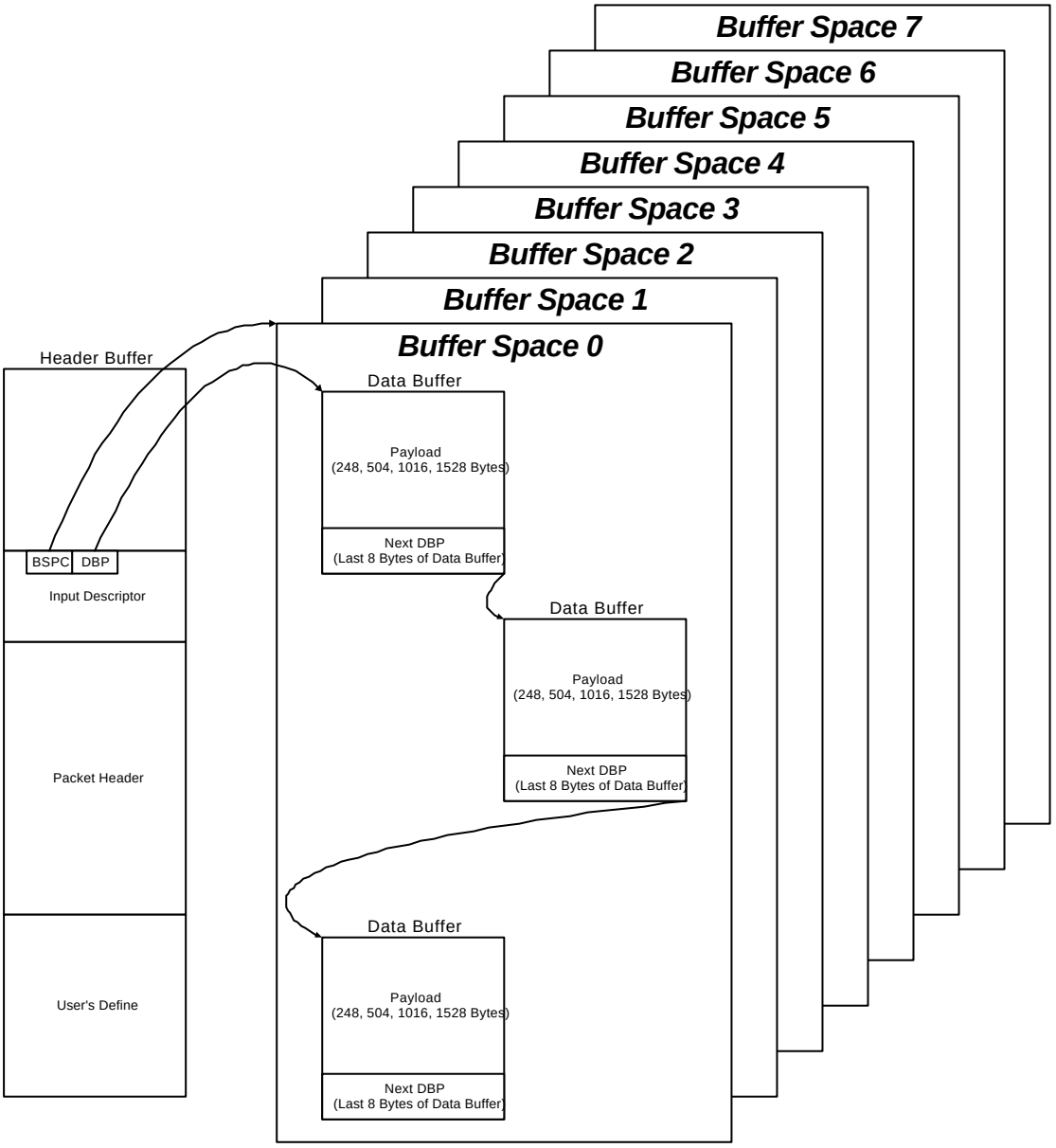

圖 9　Vitesse IQ2000 **網路處理器輸入封包格式**

2-4-2 封包輸出格式

封包輸出模組（Packet Output Module, POM）**從** Prism IQ2000 **送出至連接至** Focus Bus **的外部裝置，從** Prism IQ2000 **的** SB（Smart Buffer）**讀取放至** Focus Cell（FCell）**中的封包格式稱為輸出資料標**

頭（Output Header, OHD）；其結構分為兩部分，一為描述封包的輸出描述，另一為真正封包的前段封包資料。我們將和輸入封包格式一樣，分成兩部份來討論，一為輸出封包類型（Output Packet Type）及輸出封包緩衝區鏈（Output Data Buffer Chain），並在最後作一個總結。

2-4-2-1 輸出封包格式

總共有五種輸出格式可以設定，各為：資料標頭（Header），完整（Complete），傳送（Transfer），歸還（Deallocate），及無效（Invalid），以下將依序詳細說明：

- Header TYPE：

封包資料標頭即為完整封包。

- Complete TYPE：

傳送封包資料標頭及封包內容後，並歸還 DBP。

- Transfer TYPE：

傳送封包資料標頭及封包內容後，但並不歸還 DBP。

- Deallocate TYPE：

歸還 DBP，但不轉送任何資料。

- Invalid TYPE：

丟棄這個 OHD，不作任何事情。

實際上，POM 是從 SB 的輸出佇列讀取 OHD 然後將資料送出。從 SB 輸出的佇列稱為 Smart Buffer Queue，Smart Buffer Queue 有點

像輸入端的 Header Buffer Space，因此每個 Smart Buffer Queue 一樣每 128 bytes 為一個單位，存放 OHD。欲 POM 如何處理封包，則設定 Output Descriptor 的封包類型欄位即可。

和輸入端相同的，OHD Size 直接影響到從 SB 中抓取佇列中資料的大小。這個參數在初始化中的 POM 暫存器即可設定，其設定值*8 即為真正實際要抓的資料長度，因為 OHD 一般直接由封包標頭緩衝區所轉換而成，而封包標頭緩衝區中 ICM 欄位並不會傳送，因此不會將 OHDR_SIZE 設為 0。

2-4-2-2 輸出資料緩衝區鏈

輸出資料緩衝區鏈和輸入端一般。如果欲輸出的封包類型為 Complete 或 Long 類型，則封包將分為封包資料標頭及封包內容兩部分，唯一的不同是輸入部份的 Input Descriptor 及 Header Part 存放於 Header Buffer Space，而輸出部分的 Header Part 及 Output Descriptor 存放於 Smart Buffer Queue。Header Buffer Space 是位於 PPE 內部的 SSRAM，但是 Smart Buffer Queue 位於 SB 的內部記憶體及 RDRAM 中。但實際上，設定時已經取消 SB 的內部記憶體存放權利，因此，Smart Buffer Queue 只能存放於 RDRAM 中。又輸入及輸出的緩衝區鏈都相同，因此輸出的資料緩衝區大小就由輸入端來決定。

2-4-2-3 輸出封包格式總結

Vitesse IQ2000 網路處理器 Smart Buffer 佇列格式如圖 10 所示，由圖中可以清楚明白，每個封包標頭為 128 bytes，每個 Smart Buffer Queue 可存放 4000 個封包標頭。每個 Smart Buffer 由輸出描述及輸出資料標頭所組成。

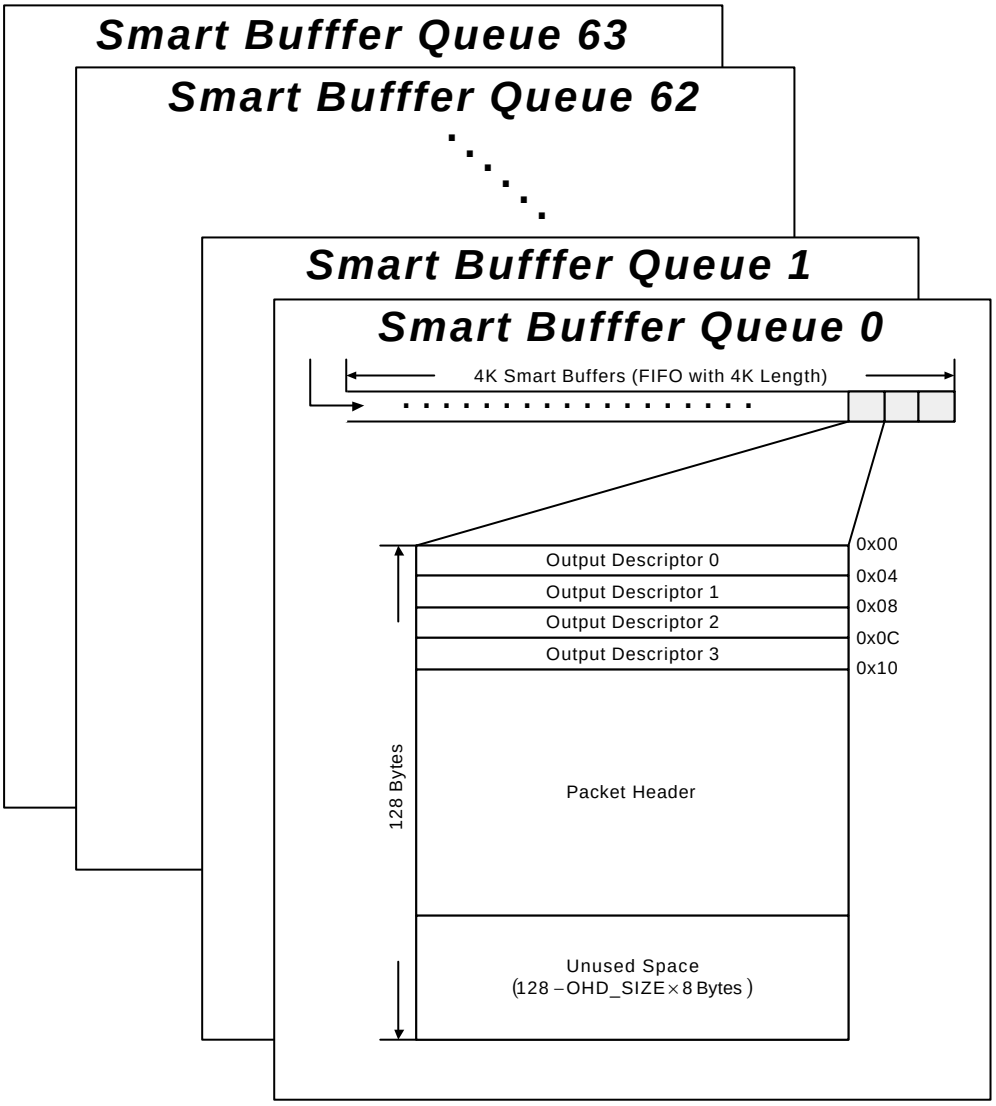

圖 10　Vitesse IQ2000 **網路處理器** Smart Buffer **佇列格式**

Vitesse IQ2000 **網路處理器輸出封包格式如圖** 11 **所示，從中瞭解輸出描述一般有輸入描述兩個和資料緩衝區有關的欄位，分別是** Buffer Space (BSPC) **及** Data Buffer Pointer (DBP) **，但相同的這兩個欄位的值要是封包是** Complete **及** Long **類型才有意義。**

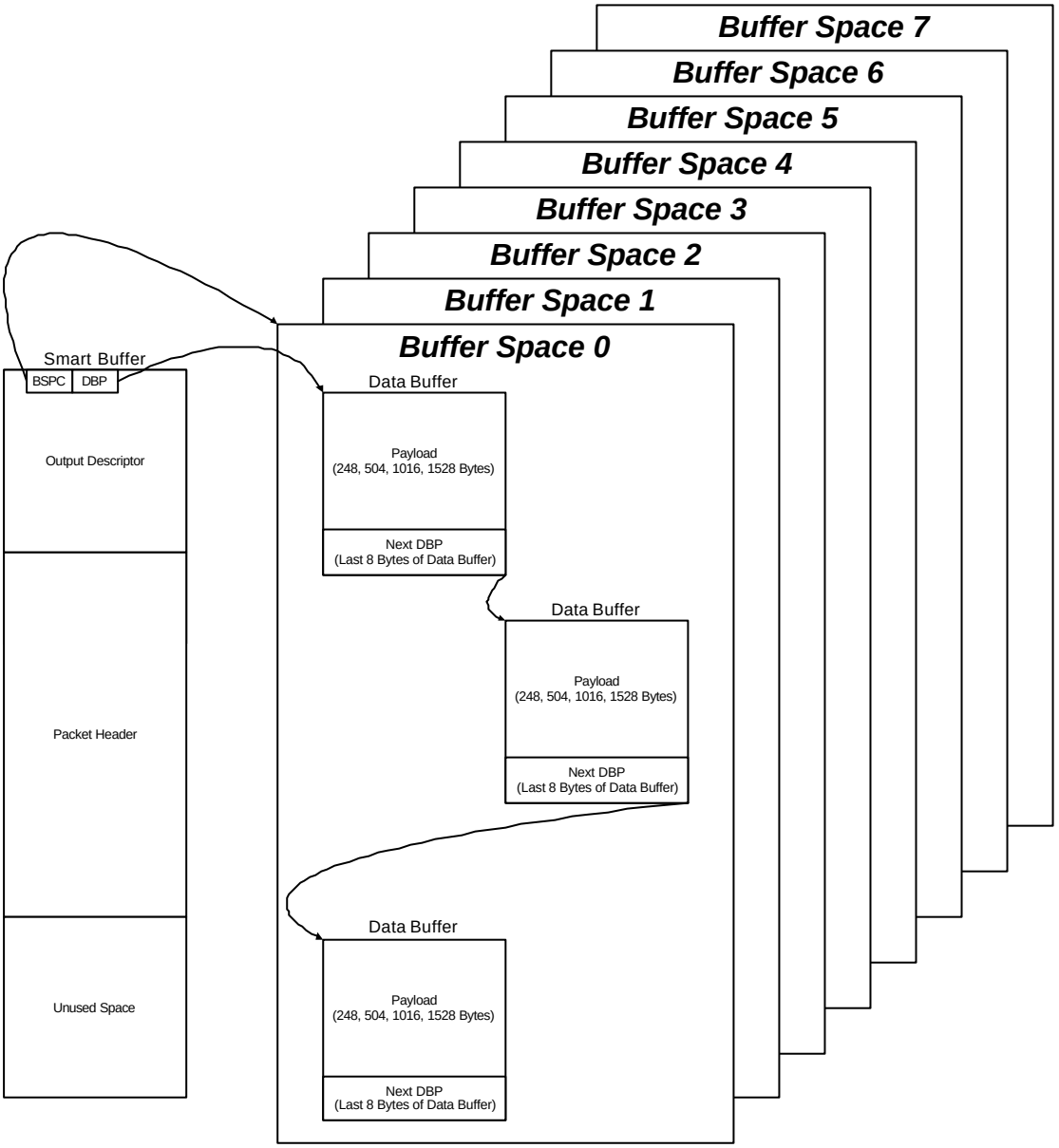

圖 11　Vitesse IQ2000 **網路處理器輸出封包格式**

2-5 封包標頭緩衝區的直達式處理

由前幾節瞭解網路處理器會將接收到的封包分成兩個主要部分封包標頭及封包內容。又只要封包標頭類型只占封包標頭的部分，沒

有封包內容的部分；但若我們執行的程序都須等到有封包內容才能接著處理下個步驟, 此種情況在多數接收封包大於 HEADER_SIZE 時沒有影響, 但若是多數封包都小於或 HEADER_SIZE 的話一定會降低系統的效能。網路處理器考慮到這個問題，因此提供一個非常有用的模式，只要封包標頭收完即先告知，如此只要是小封包的話即可馬上往下個步驟處理，不用在花時間等待收封包內容，此種處理方式稱為直達式(Cut Through)處理方式；Vitesse IQ2000 網路處理器封包標頭處理程序如圖 12 所示，即表達這個處理方式的每個狀態。主要分成兩個迴圈，一為專門處理 Header TYPE 的小迴圈，此為加快系統效能的處理流程，也是 Cut Through 和一般處理流程不同的精神所在；另一為一般的封包處理流程，即 Complete TYPE 及 Long TYPE。以下針對這個狀態圖的每個狀態及狀態專換的條件加以說明：

- 狀態解釋：
  - EMPTY：

沒有資料在封包標頭緩衝區中。

  - UNASSIGNED：

有輸入封包標頭在封包標頭緩衝區中，但還未開始處理。

  - IN PROCESS：

開始處理收到的封包標頭，開始處理並決定往那個路徑送。

  - LOCAL CLEAR TO SEND：

封包已經完整接收，並決定要送的埠，之後並且將封包標頭緩衝區所占的空間釋放。

  - TRANSFERRED：

已經將封包標頭緩衝區的格式轉換成輸出封包標頭格式，但還不能釋放送出，有可能是封包內容還未完全接收或是此封包不是從輸入端放最久的封包。封包已從接收程序，轉換到釋放程序，所以改由釋放程序負責處理。

- CLEAR TO SEND：

輸出封包標頭格式已轉換完成，且封包內容也接收完全，可以由釋放程序準備釋放。

- RELEASING：

輸出封包標頭以被釋放程序送到 Smart Buffer 並釋放。

- 狀態轉換條件：

- EMPTY to UNASSIGNED：

當有封包經由 Fusion Bus 的 CPUH_BASE 位址寫入封包標頭緩衝區。

- UNASSIGNED to IN PROCESS：

當程序讀取 ASNHDR 暫存器，並回傳封包標頭緩衝區的號碼。

- IN PROCESS to LOCAL CLEAR TO SEND：

當 Order Manger 對這個被驅動的封包且啟動 CTS 訊號，表示封包標頭緩衝區已經完整接收，並不會再有其他封包內容的資料。

- LOCAL CLEAR TO SEND to EMPTY：

當 Order Manger 已將此封包的號碼寫入 BUF_OUT，準備釋放。

- IN PROCESS to TRANSFERRED：

當程序寫到 RELHDR 暫存器，且封包標頭緩衝區還未 CTS，並將處理程序，轉換到釋放程序。

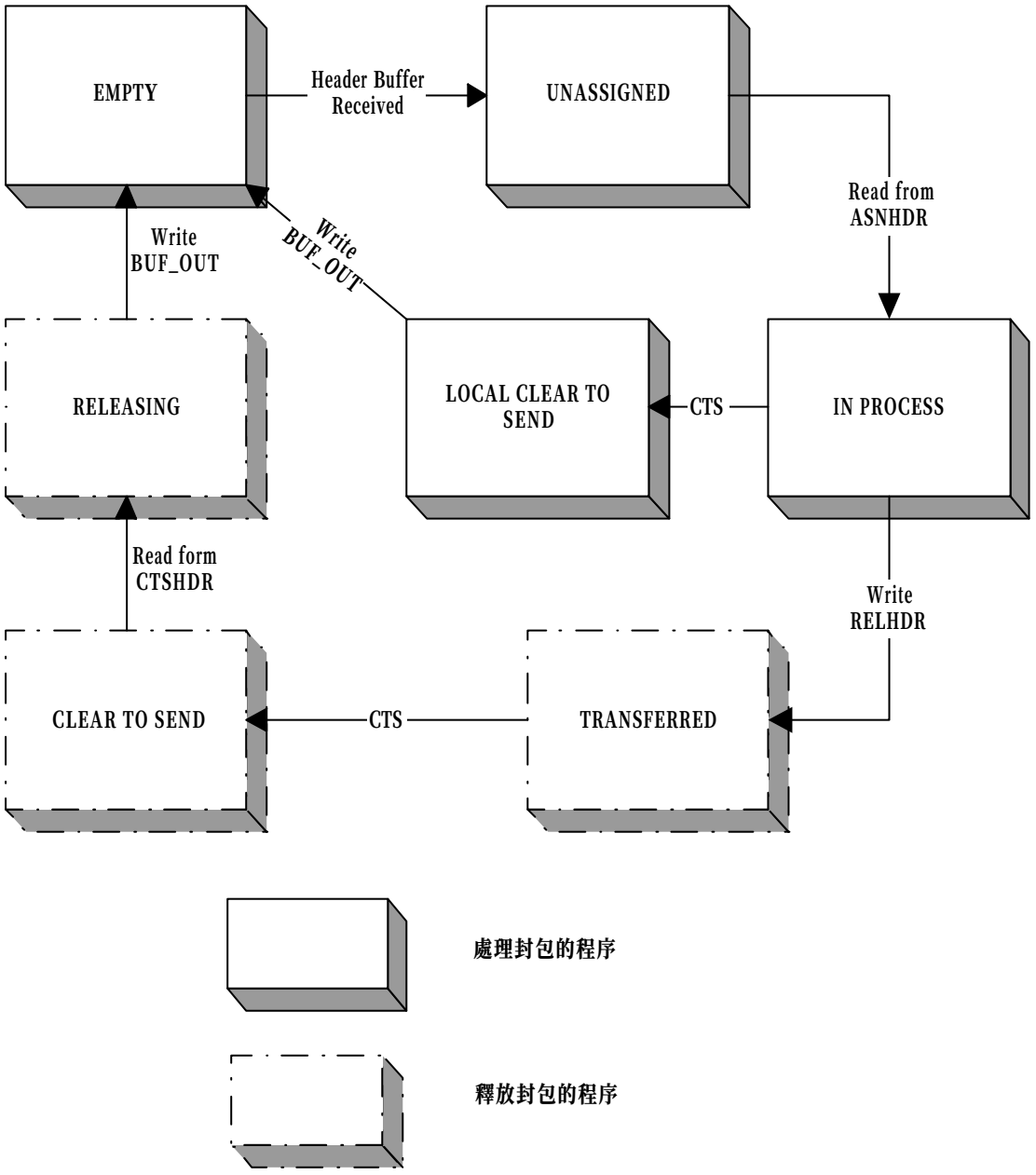

圖 12　Vitesse IQ2000 **網路處理器封包標頭處理程序圖**

- TRNASFERRED to CLEAR TO SEND：

當此封包標頭的被 Order Manger **驅動** CTS **的訊號。**

- CLEAR TO SEND to RELEASING：

當釋放程序讀取 CTSHDR **暫存器，其內容已是現在的封包號碼，表示此封包將被釋放程序所釋放。**

- RELEASING to EMPTY：

當 Order Manger **已將此封包的號碼寫入** BUF_OUT**，準備釋放。**

第三章

交換平台種類介紹

一般將交換平台(Switch Fabric)分成二大類，一為以記憶體為架構的共享記憶體式(Share Memory)交換平台[8]，一為硬體為架構的縱橫式交換平台 [9]，接著將分別就其特性優劣詳細說明及幾種改進效能的方式。

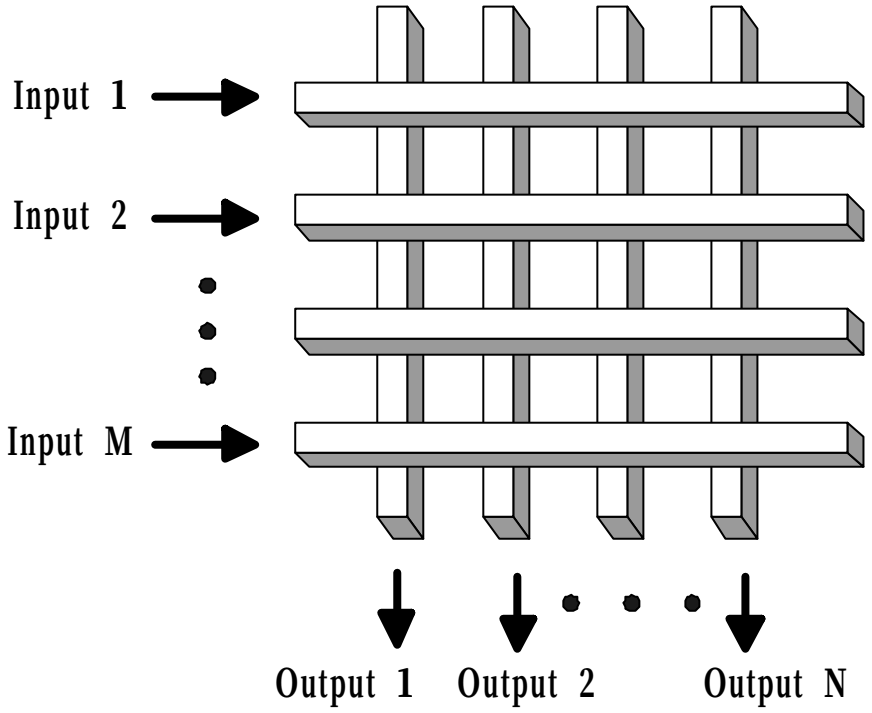

圖 13　縱橫式交換平台

3-1 縱橫式交換平台

縱橫式交換平台架構如圖 13 所示，此種交換平台設計的觀念是易製作成硬體。由此圖可以明瞭，橫軸為輸入端，假設有 M 個輸入

端；縱軸為輸出，假設為 N 個，任何從輸入端進來的封包，輸入到交換平台後，交換平台會自動比對輸出封包目的輸出為址為何，自動將封包由橫軸轉到對應的輸出縱軸排線上，將封包送出交換平台。因此，這是即時的將封包轉送，速度很快；但此種直接硬體對應轉送的機制速度雖快，但有個非常嚴重的問題，即是當同一時間，不同輸入端接收到的封包，有兩個以上其目的位址都指向同一輸出端，即同時轉向同一個縱軸，則就會發生硬體輸出上的錯誤，因為硬體不知要將哪個輸入轉送到此一輸出，此種問題即為縱橫式交換平台上有名的前端資料阻塞(Head of line blocking) 的問題 [9]，簡稱 HOL。它會使輸出效能減低，實驗證明在每個輸入端送往輸出端的資料量是很平均的情況下，此現像將造成系統效能少於 60%以下 [9][10]，因此有了以下結論：

- 優點：速度快，因為硬體線路直接對應
- 缺點：有前端資料阻塞的問題，造成系統效能降低

3-2 無阻塞共享記憶體交換平台

共享記憶體交換平台架構如圖 14 所示，它和縱橫式交換平台一樣，假設有 M 個輸入端，N 個輸出端，此種交換平台設計的觀念，是以記憶體為核心，由每個輸入端進來的封包全部都儲存於記憶槽(Memory Pool)中，儲存的方式是檢查封包目的位址，將其對應到輸出端的記憶體佇列存放，之後由接輸出端檢查屬於自己的記憶體佇列是否有封包存在，若有即將封包依序送出。也因為如此的特性，即使發生如同縱橫式交換平台一樣，同時有不同輸入端的封包要送往同一

個輸出，因為會儲存於記憶體中，所以不會遺失封包，也沒有所謂前端資料阻塞的問題，因此又稱此種交換平台為無阻塞共享記憶體交換平台；另外，因為封包是藉由記憶體轉送，所以轉送速率就會受限於記憶體最大的傳輸速率。

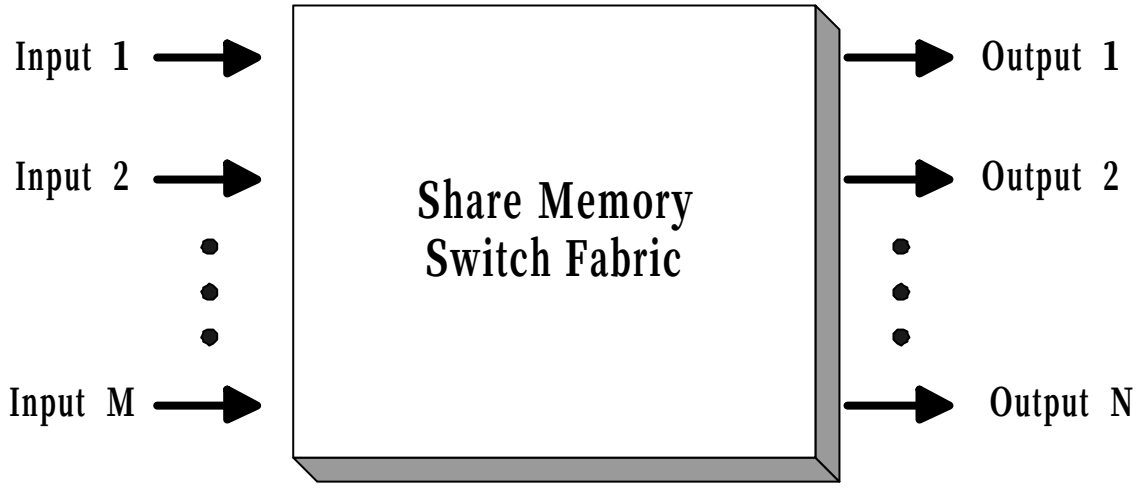

圖 14　共享記憶體式交換平台

- 優點：以記憶體為媒介轉送封包，不會發生前端資料阻塞的問題
- 缺點：轉送速度受限於記憶體最大的傳輸率

3-3 提昇交換平台效能的方式

為了因應網路封包的傳輸速度，目前交換平台都採用硬體的架構，即縱橫式交換平台的方式，因為硬體有高速轉送資料的特性，才能夠應付高頻寬資料的傳輸；但之前提到，此種架構有前端資料阻塞的問題，因此目前通常將縱橫式交換平台作了以下幾種方式改良，解決前端資料阻塞問題，達到 100% [9][10]的使用效能：

3-3-1 輸出佇列(Output Queue)

輸出佇列 [10]主要就是用來解決在縱橫式交換平台上所發生前端資料阻塞的問題，輸出佇列縱橫式交換平台其架構如圖 15 所示，其在每個輸出端，都加上一個佇列，因此欲輸出的封包就會先存放在此佇列後才被依序送出，因此，即使同一時間有兩個以上的輸入封包，其輸出端相同，也依然可正常運作，即不會有前端資料阻塞的情況發生，系統效能也能達 100%。但先決條件是，內部系統傳輸資料的速度，必須是輸出端連節的速度的 N 倍，N 為輸出端的個數，如此的條件，將造成系統成本的增加，為其最大的缺點，但是這樣架構還有另一項優點，就是佇列能控制封包輸出的時間，即可以達到服務品質保證的功能。

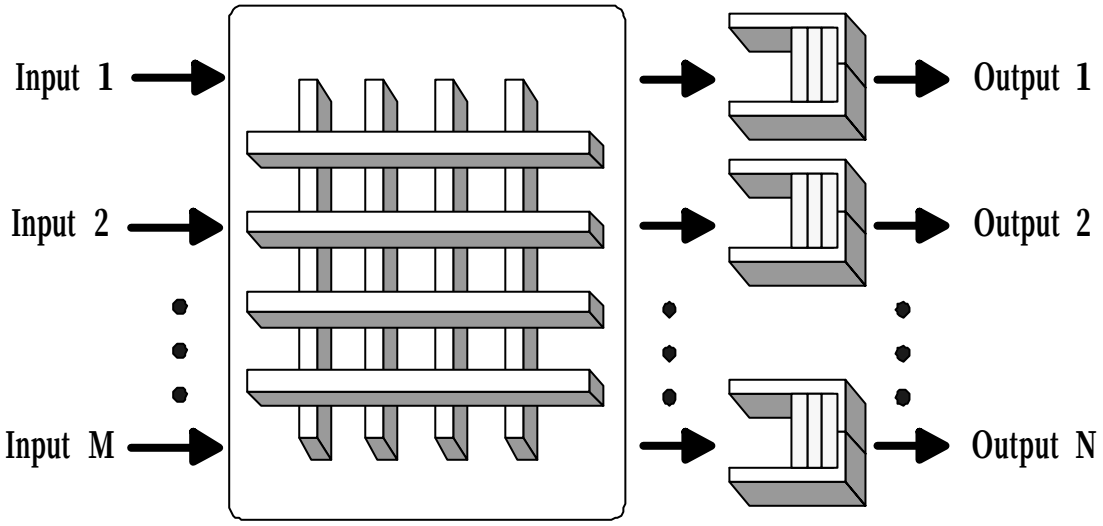

圖 15　輸出佇列縱橫式交換平台

3-3-2 輸入佇列(Input Queues)

輸入佇列縱橫式交換平台架構如圖 16 示 [11][12][13]

[14][15][16]，有鑑於輸出佇列架構的高成本，不符合經濟效益，於是有了此一架構，其相當於有著輸出佇列效能的輸入佇列，主要設計重點在於每個輸入端，即配置 N 個佇列，其中 N 為輸出端的個數，在封包進來時，即決定要放到哪個佇列存放；此種架構不但解決原先輸入佇列中前端資料阻塞的問題，達到 100%的輸出效能，同時不用像輸出佇列架構一樣，要求系統內部硬體的速度，降低成本，因此最適合須處理大量封包的高速交換平台上。

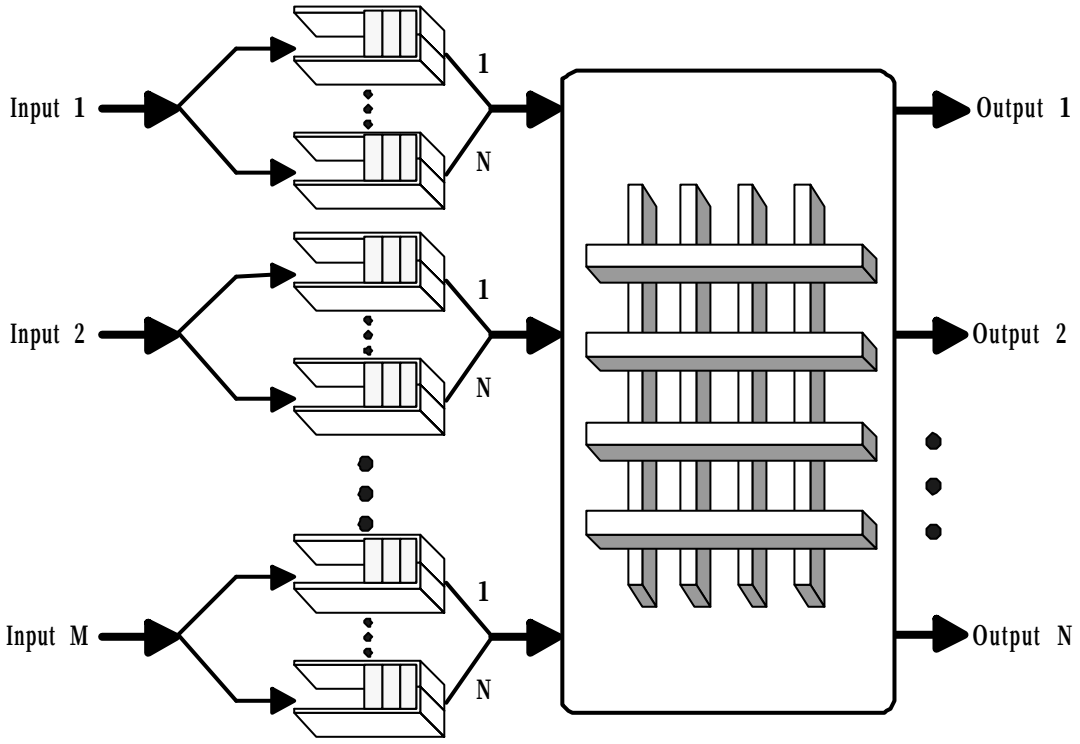

圖 16　輸入佇列縱橫式交換平台

3-3-3 整合輸入輸出佇列(CIOQ) 縱橫式交換平台

既然輸入佇列、輸出佇列各有其優點，於是便研究報告將此兩者優點合而為一，成為新的交換平台架構，稱之整合輸入輸出佇列 (Combined Input Output Queue, CIOQ) **縱橫式交換平台架構** [10][17][18]**如圖** 17 **所示。此架構交換平台兼有輸入佇列不用提高硬體速度即可解決前端資料阻塞問題，又有輸出佇列提供服務品質保證**[11]**，此種架構，非常適合須要高速處理能力的交換平台，因為不僅須要高的處理效能，同時會要求服務品質保證能力，此為最完善的交換平台架構。**

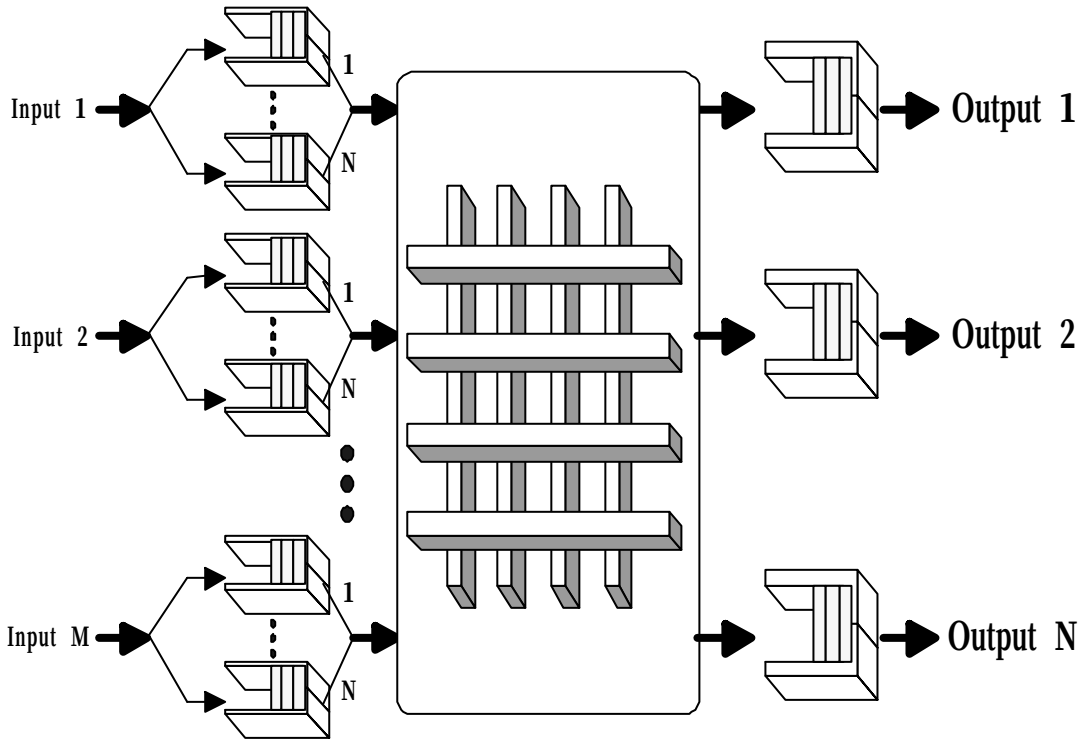

圖 17 **結合輸入輸出佇列式縱橫式交換平台**

高速交換平台架構，為了簡化硬體設計複雜度及提高封包轉送的速度，通常在封包進度交換平台時會切割成固定大小的 cell，並在輸出之前，再把相關的 cell 集合成原本正確的封包格式傳送出去，達成高速處理的目的。

第四章

將 VIQ 及 DOQ 應用在網路處理器系統上

本章將說明如何利用輸入佇列及輸出佇列提昇網路處理器的效能，並介紹我們所設計虛擬輸入佇列及動態輸出佇列的架構及整合方式。

4-1 將虛擬輸入佇列及動態輸出佇列應用於網路處理器的理由

第三章提過，雖然本系統是共享記憶體的架構，原則上應該不會發生類似縱橫式交換平台才會發生的前端資料阻塞問題，但由於硬體設計架構，內崁式記憶體不可能太大，造成系統運作時會有類似前端資料阻塞的情況。雖然硬體本身有設計 OQM (Overflow Queue Manger) 的機制，將來不及處理的封包放到 RDRAM，但其所能處理的個數有限，因此要靠虛擬輸入佇列及動態輸出佇列來補強硬體上的不足，如此發揮更佳的系統效能。

原先系統執行的流程是每個 PPE 內的 Context 所執行的流程都相同，即每個 Context 負責處理 1 個封包，從接收進來到傳送出去，都是由同一個 Context 負責，由之前的介紹知道，系統只有 4 個 PPE，其每個 PPE 內有 4 個 Context，即表示系統同時會處理 16 個封包，但一個封包處理的流程，亦為生命週期，即從接收到封包標頭緩衝區，到送到 Smart Buffer 後釋放，其間會經過很多其他程處理，例如要檢查封包的類型、封包正確性、決定封包送出時間等等程序，讓封包的

生命週期加長，但此情況會下面要新進來的封包，有負面的影響，原因即是，封包生命週期越長，其封包標頭緩衝區就沒有其他的空占存放新的封包，則執行時間一長，新的封包會因為進不來，時間過長失去時效，最後被丟棄，進而大大影響系統處理封包的效能。

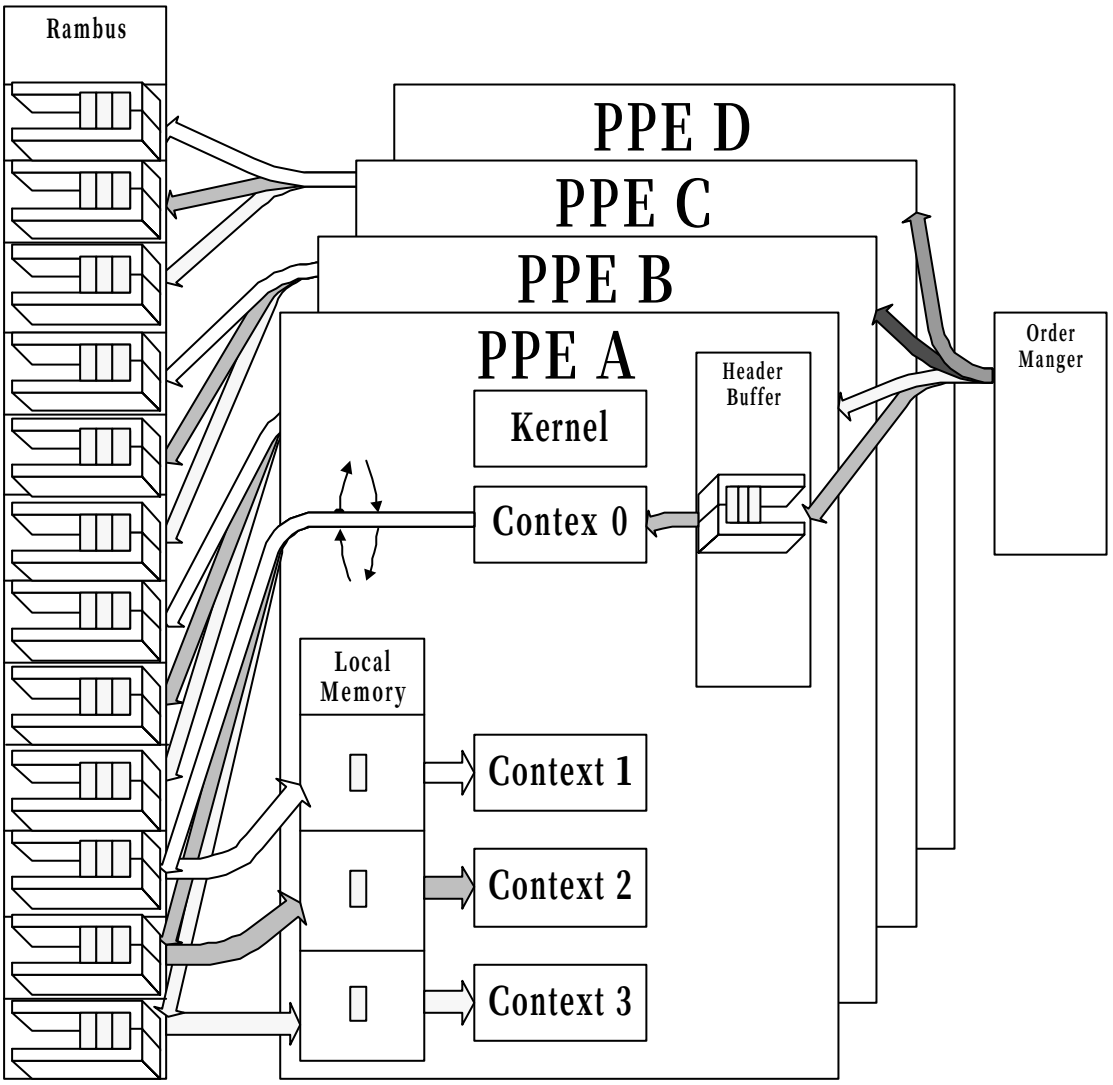

圖 18　虛擬輸入佇列方塊圖

解決這個問題的重要關鍵，即時將封包占住封包標頭緩衝區的時間。即將每個封包的生命週期，減至最短，即達成目的。最佳的方式是，封包一接收，就在最短時間內將其釋放掉，讓下個封包以最短的時間進入網路處理器處理。於是我們利用之前討論虛擬輸入佇列的觀

念，讓一個 Context 專門負責將接收到的封包送往 RDRAM 儲存，之後音即釋放掉所處理封包，如此便可讓每個封包的生命週期減至最短，達到增加效能的目的。

要送出的封包資料標頭，會先置於 Smart Buffer 中的某個佇列，若沒有機制去偵測此佇列是否被填滿時，則超過的封包資料標頭將無法再放入而被丟棄，同時和此被丟棄標頭相關的封包內容所占的空間，將無法被釋放。若此情形不斷發生，將造成記憶體被占住無法使用，最後造成系統當機，因此須要有一偵測的機制，當正在運作的佇列快滿時，則選擇延遲送出或更換佇列，防止當機情況發生。

另外，一個網路處理器系統，其應除了轉送(forwarding)封包之外，還可以處理很多網路上的應用，例如之前所提：網路管理、排程、服務品質保證等。因為若輸出只有一個佇列，當發生有大量資料要同時送出的情況，就會發生不能如預期的時間送出，而讓管理的能力大打折扣，此網路處理器的硬體，每個輸出的 Smart Buffer 都提供 8 個群組(Group)，每個群組包含 8 個佇列，共 64 個佇列可以讓使用者自由運用調度，可以是循序佇列，也可以讓其為有優先權的佇列，因此若我們可以運用的佇列愈多，則同時可送出大量資料的能力就愈高。因此我們運用其所提供動態調整優先權佇列的特性，每個 Smart Buffer 選連續 5 個群組，即同時每邊管理 40 個佇列；因此群組設優先權 1-5(號碼愈大優先權愈高)，每個群組亦設優先權 0-7，以“永遠存優先權最低群組，佇列由優先權高往低依序存放，滿時往下個低優先權佇列放，若放到最低佇列則表示此群組全滿，則動態將此群組優先權往上加 1，並指向下一個最低優先權的空群組存放封包”，以此達成動態輸出佇列的功能。

4-2 虛擬輸入佇列設計架構

虛擬輸入佇列方塊圖如圖 18 所示是虛擬輸入佇列架構的執行流程，又因為網路處理器特性，有同時 4 個 PPE 在執行，因此我們只看其中一個 PPE，其他 3 個執行方式亦同，接著詳述如下：

1. Context 0 負責接收送到網路處理器的封包, 其會隨時檢查封包標頭緩衝區有無新進的封包，若有新進封包，則用循環（round robin）的方式依序以直接記憶體存取（DMA），將封包資料標頭存到 3 塊 RDRAM Address 的其中一個，並更新新增的 RDRAM 所對應的計數器（Counter），目的是為通知 Context1~3 有新進封包，並立即將封包釋放，加速下個封包進來的速度。

2. Context 1~3 亦會隨時檢查其所屬計數器以知道 RDRAM 有無新進的封包，如有新進封包，則會將 RDRAM 中的封包，以直接記憶體存取方式將其存到所對應內部記憶體當中，讓後面的程序處理，以達到虛擬輸入佇列的目的。

4-3 DOQ 設計架構

動態輸出佇列方塊如圖 19 所示是動態輸出佇列架構的執行流程，網路處理器特性，有同時 4 個 PPE 在執行，因此我們只看其中一個 PPE，其他 3 個執行方式亦同，接著詳述如下：

1. 當所以應該執行的步驟在網路處理器內執行完成後，會決定將封包傳往何處，Context 1~3 會隨時檢查目前要放的 Smart Buffer 的 Queue 是否已滿，若已滿再檢查現在是不

是這個群組的最後一個佇列（ 即此群組的佇列 0)。

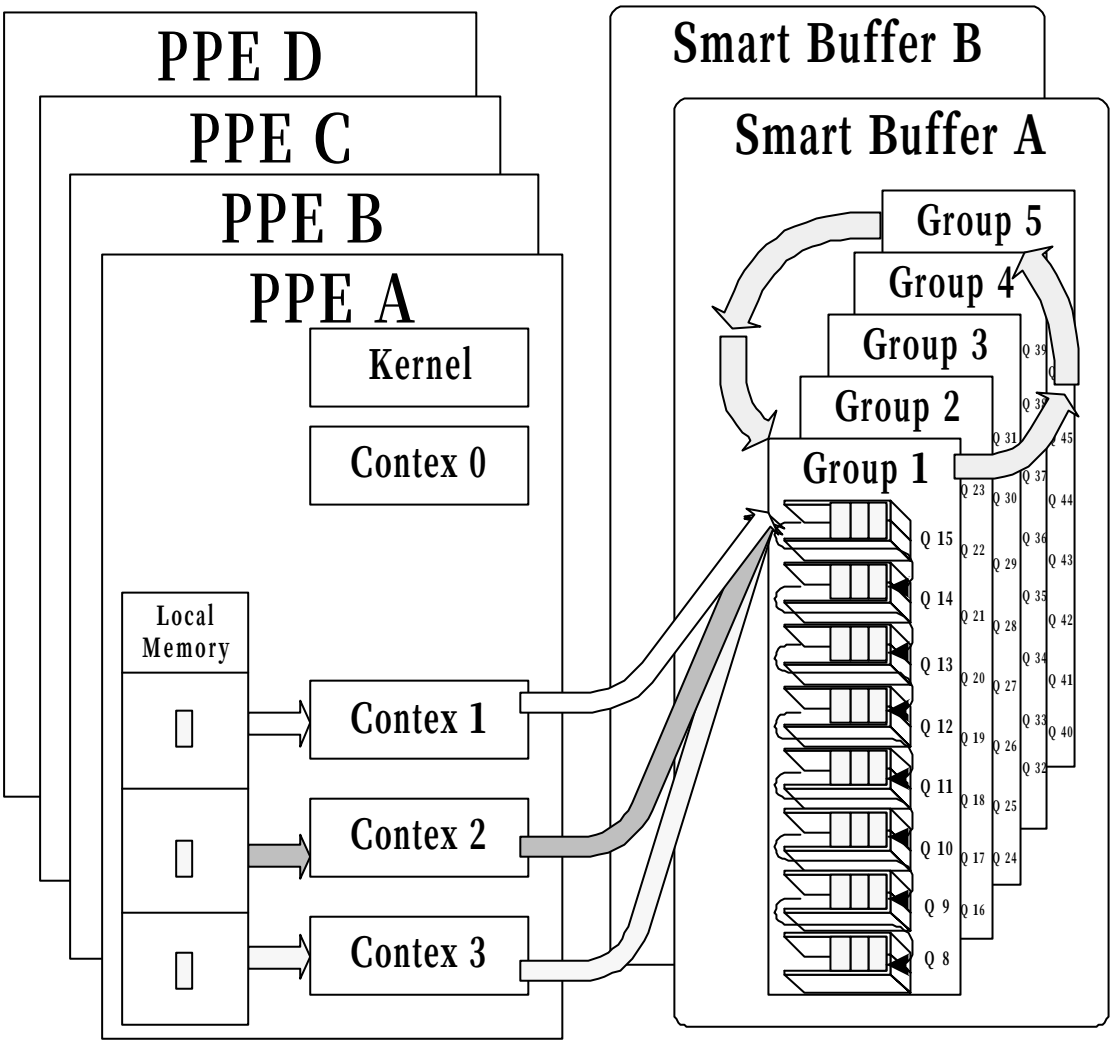

圖 19　動態輸出佇列方塊圖

2. 若是此群組最後一個佇列，則表示此群組有封包，則再檢查下個群組有無封包存在，若有封包，則表示所有的群組都有封包，則無新的空佇列可以存放，則要將接下來的封包都丟棄。

3. 若下個群組沒有封包存在，則動態調整群組的優先權，即將下個群組的優先權設成最小，將原先的群組的優先權依

次往上增加 1，再從新的群組的最高優先權佇列 7 開始放欲送出的封包，如此完成動態輸出佇列的處理流程。

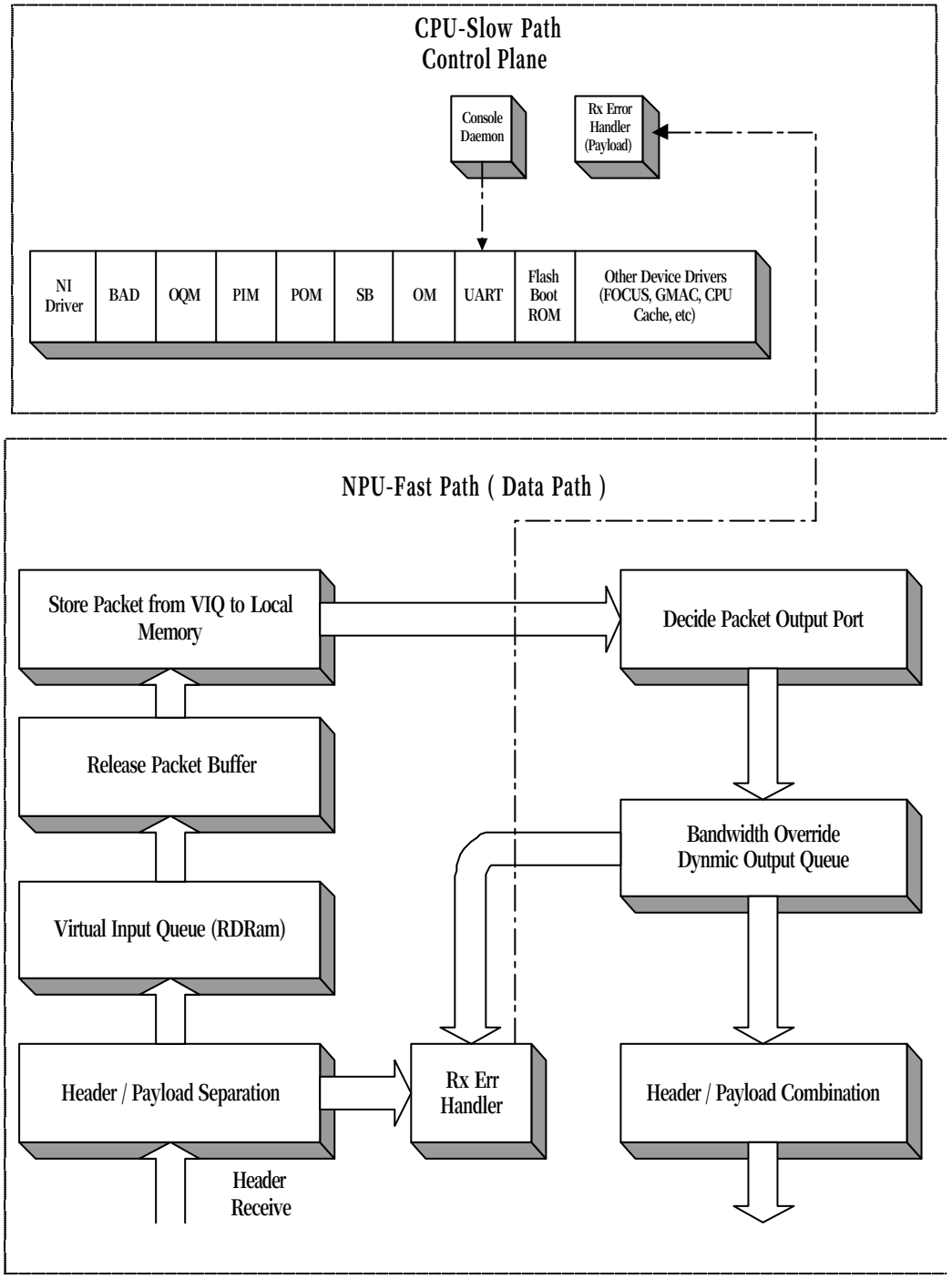

圖 20　加入 VIQ / DOQ 架構封包處理流程

4-4 整合方式

原先系統執行的流程為，每個 PPE 內的 Context 所執行的流程都相同，即每個 Context 負責處理 1 個封包，從接收進來到傳送出去，都是由同一個 Context。

如上圖 20 加入虛擬輸入佇列及動態輸出佇列架構封包處理流程，此架構主要分成兩個執行路徑，一是在網路處理器執行的路徑稱之為快速路徑，主要是處理網路封包資料轉送的流程，例如：檢查封包的正確性、決定封包轉送方向、封包格式的確認及轉換及服務品質保證的控制等，所有和網路封包及流量處理動作，都在此一路徑運作；另一在中央處理器執行的路徑稱之為慢速路徑，主要是用來控制及設定系統相關的參數資料，在此路徑中運算的資料，是較不迫切的，不一定需要立刻回應，例如：SNMP、Console 設定、Telnet 等，但此兩路徑是缺一不可，一旦沒有其中一邊，整個系統將不能正確運作。

首先，先看網路處理器所負責的快速即時路徑，封包自 GMAC 收進 PIMA 或 PIMB 後, 封包資料標頭會放在 4 個 PPE 其中之一的封包標頭緩衝區中，若封包大過封包標頭，則將其他封包的資料內容，送到 RDRAM 存放，並在此時確認封包格式的正確性，若不正確且是短封包，則直接送到錯誤處理器 Error Handler 釋放所占的空間；若是正確的封包資料，則用直接記憶體存取方式，將封包資料標頭送到虛擬輸入佇列即 RDRAM 存放，之後便可以將此封包釋放，讓新進的封包能馬上接收處理，此時另一個程序，會將放在虛擬輸入佇列的封包資料標頭一樣以直接記憶體存取方式存放至內部記憶體，然後讀取封

包相關參數，決定此封包要往哪個埠送出，並決定要送至動態輸出佇列即 Smart Buffer 位置；之後，若是原先錯誤的長封包則在此時就會送至錯誤處理程序去處理，若是一般正確的封包，則硬體便會自動將封包資料標頭及封包內容結合送出，完成一整個封包即時處理的流程。

另一個由中央處理器所處理的非即時路徑，在系統開始時，它必須負責將相關的網路處理器設定參數初始化，並將開機的程式段載入至快閃起啟記憶體（ Flash Boot ROM ），然後驅動 BAD、OQM、PIM、POM、SB、OM、UART、FOCUS、GMAC 及中央處理器快取的驅動程式，並執行一長駐的主機操作程序（ console daemon ），作為處理錯誤封包內容的動作。

第五章

VIQ / DOQ 測試平台與結果

本章將虛擬輸入佇列及動態輸出佇列實作後，介紹整個效能的測試環境並設計數個測試，依其結果分析效能改變的原因。

5-1 測試環境

在開始測試之前，先介紹實驗所須的相關設備及其連接關係，測試環境的連接方式如圖 21 所示，說明如下：

- 硬體須求：

  電腦一部、網路處理器系統一部及產生封包的 SmartBits 一部其中須含二張 Gigabit 的介面卡。

- 連接方式：

  1. 將兩對光纖線分別連接於網路處理器系統及 SmartBits 的兩端，讓 SmartBits 產生封包輸入網路處理器系統，作測試用。
  2. 將一條 RS232 的線連接於電腦 com port 和網路處理器系統 console 二端，讓電腦方便由終端程式顯示網路處理器系統目前的使用狀況。
  3. 將一條 10/100 網路線連於電腦和網路處理器系統兩端，讓 Gigabit 在初始化時，透過此介面抓取電腦中的初始設定碼，及在測試時下載網路處理器程式碼。

- 運作方式：

1. 將網路處理器系統重置後，其會自動連接至電腦內抓取相關硬體初始設定參數，下載至本身系統內部，完成開機及系統初始動作。

2. 於電腦端執行網路處理器系統所附之 Source level debug 程式，連接至網路處理器系統後，於組譯完成後，下載網路處理器程式碼至網路處理器系統的 Code Storage 內並執行。

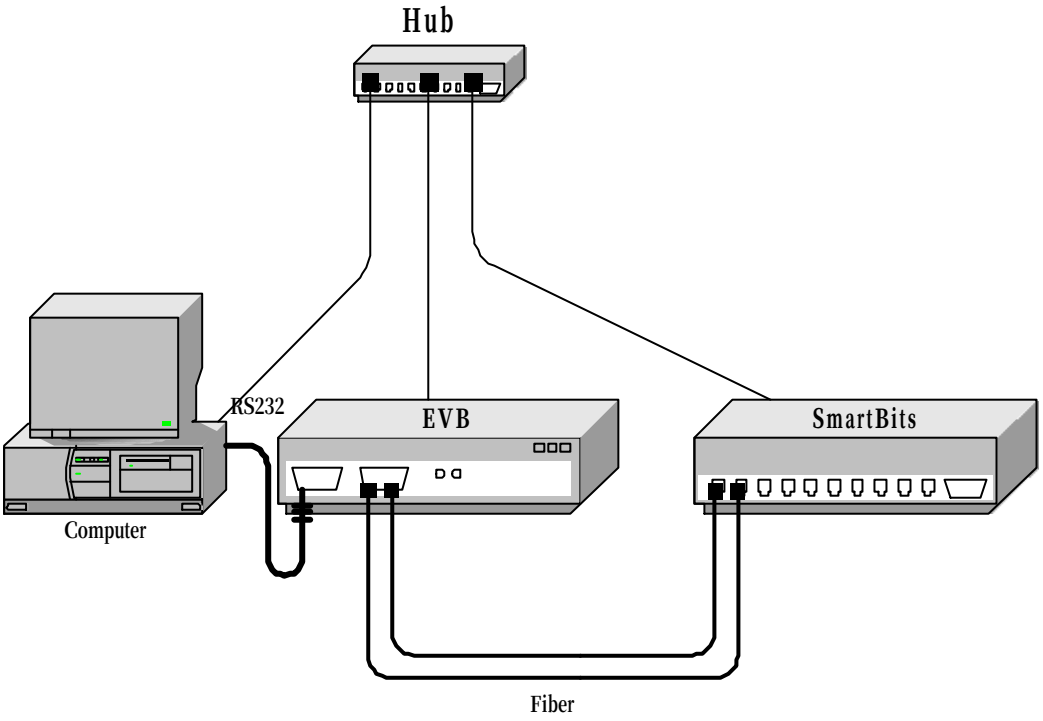

圖 21　測試環境

3. 於電腦端執行 SmartBits 應用程式，並設定產生封包的方式、速度及順序等相關參數，由 64 bytes、128 bytes 封包開始測試一直到 1518 bytes 封包，以每次連續一秒送 1 Gbps / (封包大小+最小 Gap)個封包數目，然後直接執行，讓 SmartBits 產生一連串封包灌入網路處理器系統，並讓其轉送回

SmartBits，藉此偵測網路處理器系統的執行效能。

4. 依之前所設計之虛擬輸入佇列及動態輸出佇列觀念，設計數個虛擬輸入佇列及動態輸出佇列組合方式，讓 SmartBit 依序產生不同大小的封包，將所得的測試數據，記錄於下。

5-2 測試參數設定及結果

■ 測試一：

條件：以系統最初的方式，即無任何虛擬輸入佇列及動態輸出佇列的情況下執行。

目的：產生一原始的效能數據，以為參考比較。

測試數據：

表一：

Frame	64	128	256	512	1024	1280	1518
Percentage	91.1%	82.2%	100%	100%	100%	100%	100%

試測結論：在沒有加入任何虛擬輸入佇列及動態輸出佇列的情況下，可明顯看出其 64 bytes 封包在小迴圈的效能，較 128 佳，但還未達 wire speed，到 256 bytes 才能達 wire speed。

■ 測試二：

條件：加入虛擬輸入佇列，方式為一個 Context 專門負責接收封

包，並以循環方式，輪流存放於 3 塊不同的 RDRAM 中，並由其他 3 個 Context 負責從 RDRAM 傳至內部記憶體開始處理。

目的：驗證加入虛擬輸入佇列是否會明顯增加效能。

測試數據：

表二：

Frame	64	128	256	512	1024	1280	1518
Percentage	71.1%	70%	100%	100%	100%	100%	100%

測試結論：從以上的數據，發現 64 bytes 封包，非但未增加效能，反而下降，這應該是小封包是走小迴圈，其封包標頭進來後，等於一個完整封包，不須多花時間等待封包內容，此時若馬上處理後送出，和先送至 RDRAM 再傳回處理，前者應該會較快，這也是為何 64 bytes 效能反而較差的原因，至於走大迴圈 128 bytes 以上的封包，居然效能也沒能增加，其原因是輸出的封包都塞在 SB 的佇列中來不及處理，造成有些封包因此而被丟棄而使效能降低；另一方面，此程式不像測試一都是單純將封包轉送出去，其所須要執行的程式更多，也因此降低了部分的效能，因此推論須要將動態輸出佇列的功能加入，才能提昇效能。

■ **測試三：**

條件：取消虛擬輸入佇列，改以 4 **個群組，**32 **個佇列的動態輸出佇列取代。**

目的：在未加虛擬輸入佇列的情況，測試只單純接動態輸出佇列可否增加效能。

測試數據：

表三：

Frame	64	128	256	512	1024	1280	1518
Percentage	58.1%	36.6%	67%	100%	100%	100%	100%

測試結論：於理論中，動態輸出佇列只是用來處理在輸出同時大量資料時，能應負即時大量流出資料用的。由實驗數據得，效能並沒有增加，反而效能下降許多，表示效能不能增加的主要瓶頸在於輸入端，讓每個封包的生命週期太長，來不及釋放掉占用封包標頭緩衝區的空間，讓新的封包增加進來替換的速率，再加上執行動態輸出佇列所須要的指令較多，因此造成效能下降較虛擬輸入佇列多的原因。

■ **測試四：**

條件：加入虛擬輸入佇列，並於輸出端加入 4 **個群組，**32 **個佇**

列的動態輸出佇列，兩者並行運作。

目的：由測試三得知主要瓶頸在於虛擬輸入佇列，若解決此一瓶徑即加入虛擬輸入佇列，測試虛擬輸入佇列及動態輸出佇列同時運作，基於有虛擬輸入佇列的條件下，動態輸出佇列可否輔助讓效能更佳。

測試數據：

表四：

Frame	64	128	256	512	1024	1280	1518
Percentage	41.1%	87%	100%	100%	100%	100%	100%

測試結論：因為虛擬輸入佇列端讓封包進來的速率增加，解決輸入端的瓶頸問題，可是顯由 128、256 bytes 封包看出效能增加的情況，但 64 bytes 依然無法提昇，推論是因為 64 bytes 封包因為沒有封包內容，當其收到封包標頭時，應該就可以將其送出後釋放空間，但我們一樣讓其執行虛擬輸入佇列及動態輸出佇列的程序，因此使其占住更多程式執行的時間，因此 64 bytes 封包才會下降。

- 測試五：

條件：將 64 bytes 的封包在收到之後，直接送到 Smart Buffer 處理，而不放到虛擬輸入佇列中，其他格式的封包，依然透過虛擬輸入佇列及動態輸出佇列處理。

目的：由測試四虛擬輸入佇列及動態輸出佇列同時運作結果發現，除了 64 bytes **的封包外，其他格式的封包效能均有改善，而** 64 bytes **的封包因為沒有封包內容，因此** 64 bytes **的封包只要接收完封包標頭即代表整個封包已收完整，此時應該可以立即送到** Smart Buffer **處理釋放，但若像測試四一樣，還將其送到虛擬輸入佇列中，反而是浪費時間處理不必要的動作，基於以上推論，測試** 64 bytes **的效能可否改善。**

測試數據：

表五：

Frame	64	128	256	512	1024	1280	1518
Percentage	91%	87%	100%	100%	100%	100%	100%

測試結論： 由測試數據可知，64 bytes **因為本測試讓動態輸出佇列有** 40 **個佇列，因此和測試五比較起來，實際上明顯使效能增加，因此證明我們的理論正確！**

5-3 實驗的結果及增加效能的比例

內崁式記憶體的容量有限，網路處理器在設計之初也考慮到此問題，因此另外設計 Overflow Queue Manger **用來將多餘來不及處理的封包，所存放到** RDRAM**，但是** OQM **所能管理的封包個數有限，因此虛擬輸入佇列即發揮提昇系統效能的主要作用。**

前一節以循序漸進的方式，以我們所規劃虛擬輸入佇列及動態輸出佇列設計數個的測試，經由所得的數值我們可以明瞭，若能以最短的時間釋放每個封包占用的封包標頭緩衝區，讓新的封包有機會以最短時間，不延遲為封包進入網路處理器，如此便能解決輸入端的瓶頸，但若單純只有虛擬輸入佇列，輸出依然只用一個佇列來處理輸出的封包，則大量輸出封包將會造成佇列來不及應付，使系統效能依然不能提昇，因此輸出端須再配合動態輸出佇列，才能處理大量連續的輸出封包，如此虛擬輸入佇列及動態輸出佇列相互配合，才能提昇系統效能。

但在 64 bytes 的封包格式須特別以原先直達式方法處理，因為 64 bytes 的封包無封包內容部分，若其還是進入虛擬輸入佇列及動態輸出佇列的處理程序，不但沒有必要，反而增加無意義的程式執行時間，因為在 64 bytes 封包中，其封包資料頭收完，應可立即送往 Smart Buffer 的佇列處理輸出了，因此特別將 64 bytes 的封包以原先 cut through 方式處理。

因此經由測試，我們證明了唯有虛擬輸入佇列及動態輸出佇列同時運作，才能讓高頻寬網路處理系統發揮最佳的系統效能。

第六章

結論

目前網路處理器的發展，還尚未成熟，因此並沒有所謂的固定的硬體架構，各家晶片廠商都在發展最適合網路設備廠商所須求的網路處理器，使其有最佳的處理效能。因為其可加快設計的時間及減少開發成本，相信網路處理器必會成為未來網路設備中必要的原件。

輸入佇列及輸出佇列及整合輸入輸出佇列觀念，原本是用來解決縱橫式交換器上所發生前端資料阻塞問題而提出的解決方案，目的即是讓交換器能 100%發揮因有的系統效能，並額外達到提供服務品質的保證，本篇論文利用這種觀念，將其變化用應用在高頻寬網路處理系統上，因此提出虛擬輸入佇列及動態輸出佇列的架構，並實際應用在本系統中，以提昇其封包處理效能。

由前一章所得的實驗結果，證明我們的理論的正確性，但目前因為硬體設計的限制，其處理封包的速度趕不上封包進來的速度，因此須要虛擬輸入佇列及動態輸出佇列來提昇網路處理器的效能。

以這個系統的架構，若只是用在當交換器上，其實有點大材小用，因為光是網路處理器內部，就設計 4 個處理器作網路封包的處理，理所當然，其功能不只限於此，以目前超高速乙太網路為標準，頻寬愈來愈大，網路多媒體化的發展趨勢，網路資料量愈來愈大也是必然的，以這個觀點來看，頻寬的管理、以量計費的應用、網路監控及網路負載平衡等，都是高頻寬網路處理系統的應用範圍，如此的系統架構應該很有應用發展潛力。

參考資料

[1]. **我國網路用戶數成長情形**, "http://www.find.org.tw/img_focus/focus20010306_1.gif"

[2]. **目前台灣人口統計**, http://www.teputc.org.tw/env_news/199906/88061901.htm

[3]. Intel Corporation web site, "http://www.intel.com/"

[4]. IBM Corporation web site, "http://www.ibm.com/"

[5]. MMC Corporation web site, "http://www.mmc.com/"

[6]. VITESSE Corporation web site, http://www.vitesse.com/

[7]. IQ2000 Network Processor, http://www.vitesse.com/products/categories.cfm?family_id=5&category_id=16

[8]. Massoud R. Hasemi, Alberto Leon-Garcia, "A Multicast Single-Queue Switch with a Novel Copy Mechanism", *IEEE INFOCOM'98*, San Francisco California, 1998

[9]. Cheng-Shang Chang, Wen-Jyh Chen, Hsiang-Yi Huang, " Birkhoff-von Neumann Input Buffered Crossbar Switches ", *IEEE INFOCOM'00*, Tel Aviv, Israel, 2000, pp. 1614 -- 1623

[10]. Shang-Tse Chuang, Ashish Goel, Nick McKeown, Balaji Prabhakar, "Matching Output Queuing with a Combined Input Output Queued Switch", *IEEE INFOCOM'99*, New York, March 1999, pp. 1169-- 1178

[11]. Mark W. Goudreau, Stavros G. Kolliopoulos, Satish B. Rao, "Scheduling Algorithms for Input-Queued Switches: Randomized Techniques and Experimental Evaluation", *IEEE INFO-*

COM'01, Alaska USA, April 2001

[12]. D. N. Sepanos, P. I. Antoniadis, "FIRM: A Class of Distributed Scheduling Algorithms for High-speed ATM Switches with Multiple Input Queues", *IEEE INFOCOM'01*, Alaska USA, April 2001

[13]. Adisak Mekkittikul, Nick McKeown, "A Practical Scheduling Algorithm to Achieve 100% Throughput in Input-Queue Switches", *IEEE INFOCOM'98*, San Francisco California, 1998

[14]. Matthew Andrews, Sanjeev Khanna, Krishan Kumaran, "Integrated Scheduling of Unicast and Multicast Traffic in an Input-Queued Switch", *IEEE INFOCOM'98*, San Francisco California, 1998

[15]. Shizhao Li, Nirwan Ansar, "Input-Queued Switching with QoS Guarantees", *IEEE INFOCOM'98*, San Francisco California, 1998

[16]. Matthew Andrew, Lisa Zhang, "Achieving Stability in Network of Input-Queued Switches", *IEEE INFOCOM'01*, Alaska USA, April 2001

[17]. M. Ajmone Marsan, E. Leonardi, M. Mellia, F. Neri, "On the Stability of Input-Buffer Cell Switches with Speed-up", *IEEE INFOCOM'* 01, Alaska USA, April 2001

[18]. E. Leonardi, M. Mellia, F. Neri, M. Ajmone Marson, "Bounds on Average Delays and Queue Size Averages and Variances in Input-Queued Cell-Based Switches", *IEEE INFOCOM'01*, Ala-

ska USA, April 2001