東 海 大 學 電 機 工 程 學 系

碩 士 論 文

利用緩衝器合併演算法之低功率晶片網路路由器設計

Low Power NoC Router Design using Buffer Merging Algorithm

研 究 生： 陳豪澤

指導教授： 蔡坤霖

中華民國 一〇一 年 六 月

摘要

隨著半導體工業以及製程技術的快速進步，在單一晶片內能放置的電晶體數目愈來愈多。一般而言，一個晶片內能夠整合許多的子系統，稱之為單晶片系統。單晶片系統可以結合許多應用，如電腦系統、數位訊號處理與多媒體等。傳統單晶片系統使用匯流排來負責內部通訊，但當系統變得愈來愈複雜時，晶片內部的通訊將會成為影響晶片處理效率的主要原因之一。因此，有學者將網路封包傳輸概念應用於晶片上以進行資料的交換，稱為晶片網路。晶片網路是一個能幫助設計者應付複雜單晶片系統設計挑戰的新型架構。雖然晶片網路是晶片上通訊問題的有效解決方案，但其存在嚴重的資源限制。緩衝器占晶片網路總面積與功率消耗的主要部分，同時緩衝器容量對不同應用的效能有重要影響。在本論文中，提出一種針對特定應用之單晶片系統的晶片網路路由器之緩衝器設計方法。此方法針對不同的特定應用之單晶片系統在映射後，該演算法可根據其資料流量分布，實現網路中各個路由器輸入緩衝器單元數量的分配。實驗結果顯示，使用該緩衝器合併演算法，緩衝資源獲得了更有效的利用，且最為重要的是於 0.18μm 製程技術下，相較於均勻分配緩衝器的典型路由器而言，緩衝器合併設計的實現可使路由器在傳輸效能變化不大的情況下，依負載情況的不同而能夠分別節省 20%至 70%的晶片面積與功率消耗。

關鍵字：單晶片系統、晶片網路、映射、緩衝器合併

ABSTRACT

As the advance of semiconductor industry and process technology, more and more transistors can be placed in single chip. Generally speaking, a chip that integrates a number of subsystems, is called system-on-chip. System-on-chip design provides integrated solutions to many applications such as computer system, digital signal processing, and multimedia. One of the major challenges of designing an system-on-chip is the communication among all the components on the chip. Traditional communication scheme of system-on-chip is based on bus system. Some researchers used the concept of communication network, and proposed the network-on-chip. Network-on-chip is a new communication architecture which helps to meet the challenge of designing a complex system-on-chip. The network-on-chip approach was proposed as an effective solution for on-chip communication problem, however it is by far resource limited. The input buffers of a typical on-chip router take a significant portion of the silicon area and power consumption of network-on-chip, and the performance of an network-on-chip is greatly affected by the amount of buffer size. In the thesis, an application-specific system-on-chip buffer merging method that can be used to customize the router design is proposed. When given a mapped application-specific system-on-chip and the data flow distribution, the proposed method can assign the number of the input buffers for each input channel in different routers of whole chip. The experimental results show that the buffer can be utilized more effectively after using the buffer merging method. Based on synthesized designs in 0.18μm technology, the proposed buffer merging method provides similar performance with the uniform buffer allocation method, and the total silicon area and power saving goals can be achieved.

Keyword: System-on-Chip, Network-on-Chip, mapping, buffer merging

目次

圖目次

表目次

第一章 導論

1.1 研究動機

近年來科技快速發展，不論是消費性電子產品、電腦、汽車電子、通訊或網際網路基礎設備等，各個領域之間逐漸朝向整合科技的趨勢發展。大型且功能複雜的電子系統，已成為現今積體電路設計上的一大難題。

在過去數年間，隨著晶片製程的幾何尺寸不斷縮小，單一晶片內已可成功地放置數千萬顆電晶體。由於特殊應用積體電路(Application-Specific Integrated Circuit, ASIC)設計技能的增強，意味著更多的元件功能可輕易地被整合並實現於單一晶片內。

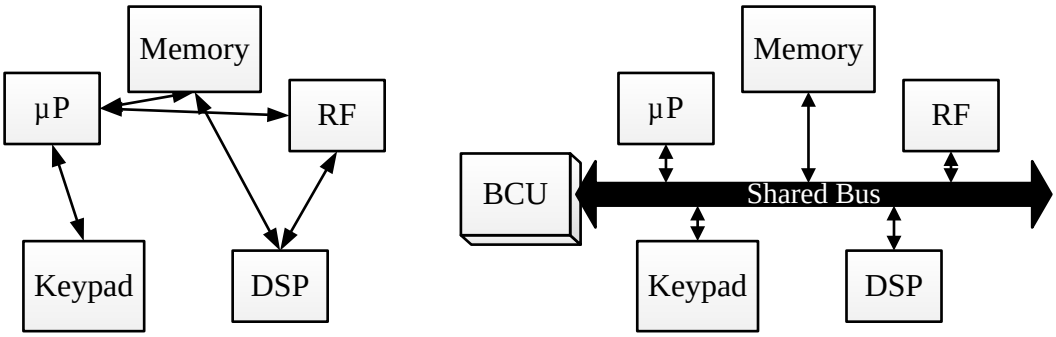
系統單晶片(System-on-a-Chip, SoC)係指將原本分處在不同晶片，負責不同功能的矽智財(Silicon Intellectual Properties, SIP)模組，加以連結整合至單一晶片內，使其能有效率地工作。先進的 SoC 採用奈米 CMOS 製程技術，從系統角度出發，將嵌入式軟體、數位電路、類比電路、混合訊號及射頻電路在內等各層次電路設計緊密結合在單一晶片內，實現整個系統的功能。如此高度整合的晶片能為許多複雜應用提供有效的解決方案，且滿足系統產品對於輕薄短小、低耗電量、低價格及高效能之設計訴求，SoC 設計因此廣泛應用於消費電子、數據處理和通訊三大領域。

未來，在功能及使用者需求的考量下，SoC 的設計將日益複雜。在 SoC 中，可能包含數以百計的 SIP，然而在晶片面積以及晶片功率消耗的限制下，改善系統效能及晶片內全域資料傳輸能力，儼然已成為一大設計挑戰[1]。

傳統上，晶片內部資料傳輸幾乎採用點對點連線(Point-to-Point, P2P)或是共享匯流排(shared bus)的方式進行連接，如圖 1-1(a)所示。若晶片規模小，內部只有數個 SIP，則使用 P2P 的連線方式效率較高。但 P2P 連線的數目會隨著系統複雜度的提高而增加，並可能出現一些難以控制的電子特性，因此 P2P 的連線方式並不具備良好的可擴充性。

共享匯流排的好處是擁有較為簡單的拓樸結構且佔用的晶片面積較小，其基本結構的改良，如 ARM AMBA [2]、IBM CoreConnect [3]、Wishbone [4]等協定，

更成為現今 SoC 廣泛使用的系統匯流排結構。圖 1-1(b)為典型的共享匯流排連接多個特定功能的 SIP 所組合而成的 SoC,其中大多數需要通訊的模組元件均憑藉著這條傳輸媒介來進行。雖然共享匯流排的連線方式相較於 P2P 有較好的可擴充性，但在一條相當長的匯流排中，其寄生阻抗和電容皆相當高，反而會影響資料傳輸延遲時間。因總長度增加及 SIP 數目增加，每個位元經由匯流排的傳輸延遲變得相當大，延遲時間最終將超過單一時脈(clock)週期，致使資料無法在單一時脈內完成傳輸的動作。且共享匯流排在一時間點上僅允許單一對 SIP 存取，而其它同時競爭的作業程序則必須擱置等待，即無法同時支持一對以上的 SIP 進行通訊，換言之，匯流排上所連接的各個 SIP 必須去共享相同的通訊頻寬，也因此限制了晶片上的傳輸效能。此外，匯流排控制單元(Bus Controller Unit, BCU)的延遲及設計複雜度也會隨著系統規模的擴充而增加，因而更限制了系統的可擴展性。解決這些問題的辦法之一是將匯流排分成多個區段並使用分層的架構[5]。然而，對於以匯流排為資料傳輸基礎的 SoC 來說，多區段匯流排仍有其侷限性，眾多的 SIP 必須共享總頻寬，導致以匯流排為基礎的資料傳輸結構將面臨嚴重的通訊瓶頸[6]，無法承載高流通量的資料，且勢必無法滿足未來 SoC 高速、高可靠度及高複雜度的設計挑戰。



(a) P2P　　(b) Shard Bus

圖 1-1　傳統單晶片系統內部通訊架構

為解決晶片內部通訊的問題，一些研究團隊主張以一般網路通訊的概念運用於 SoC 設計上。此概念允許晶片內部的各個 SIP 去耦化(decoupling)，意即不再需要晶片全域同步(global synchronization)，SIP 之間只要在傳輸資料時同步即可。這種新的設計方式被稱為晶片網路(Network-on-Chip, NoC)，其基本概念的實現如圖 1-2 所示。SIP 之間的資料傳輸可以藉由封包(packet)的形式傳遞，並利用晶片內部路由器(router 或稱 switch)溝通[7][8]。NoC 的出現大幅減少晶片內部全域繞線之需

求[9]，其特點具有(1)較佳的傳輸效能；(2)頻寬的延展性；(3)並行性(concurrency)；(4)可模組化(modularity)及可重複使用化(reusability)等。由於 NoC 的高度規則性與良好的可控制結構，因而讓其擁有高度效率和可靠性，且規則性的結構亦使得 NoC 可模組化和網路設計容易被重複使用。此外，同時進行也易可達成，因 NoC 內的連線可以同時地被不同封包使用，所以增加了傳輸線的使用率。因此 NoC 可以提供未來 SoC 互連問題許多的解決方案。

NoC 的基本功能元件組成，如圖 1-3 所顯示。每一個 SIP 可能為負責資料處理計算的處理元件(Processing Element, PE)或儲存元件(Storage Element, SE)，且皆被視為一個計算節點，而各計算節點之間透過網路介面單元(Network Interface Unit, NIU)與路由器所建立的路由節點相連接，再藉由各個路由節點及其之間的拓樸網路來完成資料封包的相互傳輸。

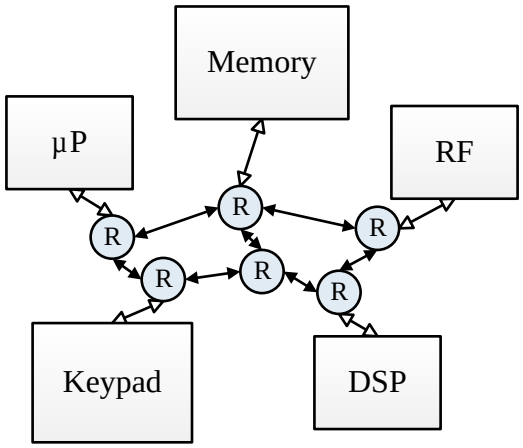


圖 1-2　晶片網路基本概念

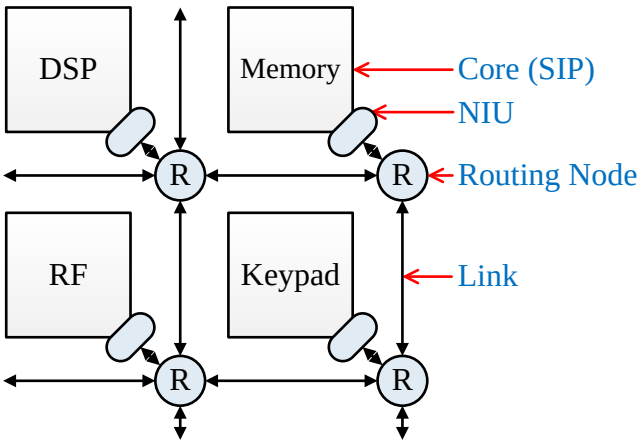


圖 1-3　2D mesh 拓樸，NoC 基本功能元件組成

NIU 可實現計算節點與通訊網路兩者間的分離，提高 SIP 於設計上的再重複使用率，能縮減 SoC 的設計時間。NIU 主要功能為轉換上述兩者之傳輸資料格式，使它們能夠彼此接收和識別對方的資料。

路由器為 NoC 中最重要的組成元件，所有 NoC 的特性以及其所提供的功能都必須藉由路由器來達成。路由器主要連接本地端的計算節點與周圍相鄰的路由器，其傳輸的方式是根據網路上的路由演算法對資料封包傳輸路徑進行選擇，將附帶路由資訊的封包自起始節點(source node)傳送至目的節點(destination node)，其主導著訊息或資料的流向，進而實現網路各節點之間的相互通訊，於 NoC 中扮演相當重要的角色。一個優良的路由器需要有效的利用路由節點之間的鏈結頻寬(link bandwidth)來因應具特殊傳輸需求的系統。

儘管 NoC 為晶片內部資料傳輸問題帶來有效解決方案，但其存在嚴重的資源限制。因此於 NoC 設計階段，準確地評估其網路效能、面積成本及功率消耗以避免昂貴的重新設計成本為之必要。在路由器設計上，緩衝器(buffer)一般以 FIFO 佇列(First In First Out queue)實現，緩衝器不但是路由器內最重要的構成元件，同時對於上述評估條件的影響也最為嚴重。緩衝器個數或緩衝區大小的調整，都將嚴重影響硬體資源的消耗與鏈結頻寬如何有效率地分配予資料封包，因此對全局網路傳輸率而言，為一個相當重要的影響因素。實際上，緩衝器是為提供緩衝空間(buffer space)來存放因資源衝突(如網路的壅塞、輸出埠的競爭和路由器內部計算延遲等)而無法立即被轉送至輸出鏈結的資料封包。故總體而言，考量如何有效地連接所有 SIP 並管理所需的晶片內部資料傳輸功能，以獲得較高的成本效益和效能為 NoC 設計之關鍵。

欲減少 NoC 硬體設計成本，必須盡可能地降低緩衝器使用個數或緩衝區空間。以圖 1-4 為例，其顯示了不同 FIFO 尺寸的面積相對於單一典型路由器總面積之比例。由圖中可得知，FIFO 尺寸為 4 個 filt (flit 為流量控制的最小單位；一個資料封包由數個 flit 所組成)時，FIFO 於該路由器所占面積比例約為 61%，而當 FIFO 尺寸增加至 32 個 filt 時，FIFO 所占面積比例大幅提高至 96%，如此說明了緩衝器元件佔用了路由器總面積的顯著部分。

隨著製程技術不斷演進，造成電路元件的靜態功率消耗(static power consumption)呈現指數型的上升，因此未來不得不重視此問題。於先前文獻研究結果表明，緩衝器為路由器結構中最大的靜態功率消耗者，其消耗量約占總體路由器靜態功率消耗的 64% [10]。同樣的，緩衝器消耗路由器大部分的動態功率(dynamic power)，

且此消耗量隨著資料流量增加而快速增加[11][12]。圖 1-5 說明了不同 FIFO 尺寸相對於單一典型路由器平均功率消耗之比例。由圖中可得知，除 FIFO 尺寸為 2 個 flit 時，FIFO 於該路由器平均功率消耗所占比例低於 50%外，其餘所測試的不同 FIFO 尺寸所造成的平均功率消耗之比例皆高於 50%，甚至在 FIFO 尺寸為 32 個 flit 時，該平均功率消耗之比例更高達 90%。由此可知，不僅於面積方面，緩衝器在路由器功率消耗上同樣扮演主要的消耗者。

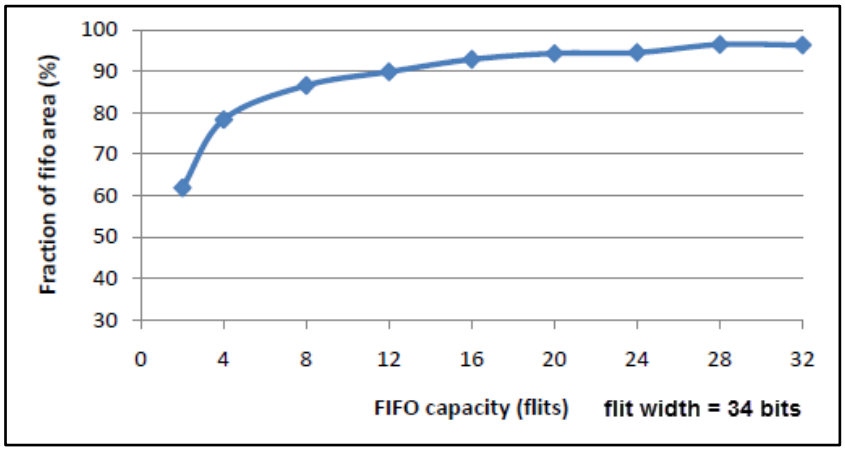


圖 1-4　不同 FIFO 尺寸相對於單一典型路由器總面積之比例

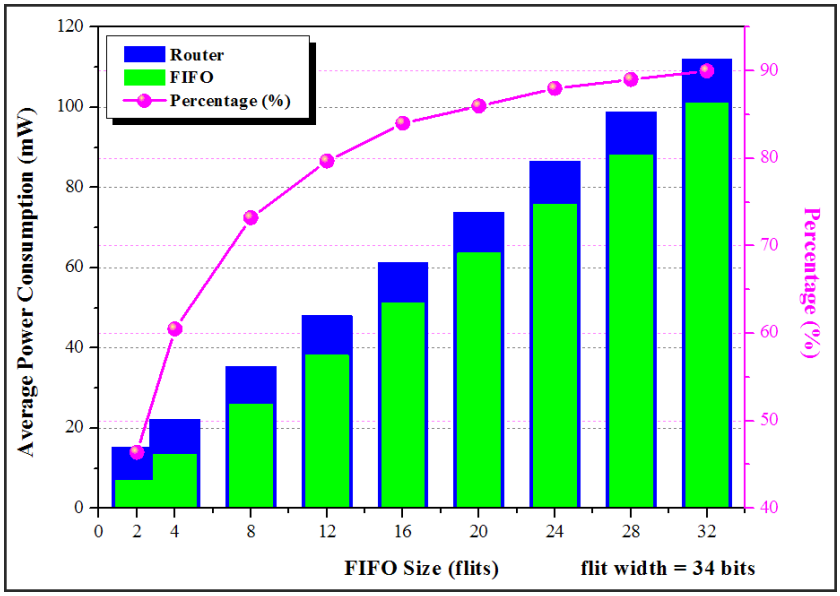


圖 1-5　不同 FIFO 尺寸相對於單一典型路由器平均功率消耗之比例

另一方面，FIFO 尺寸的不同亦對於 NoC 效能指標影響甚巨[13]，效能指標包括網路的傳輸延遲(latency)及資料傳輸率(throughput)。傳輸延遲係指資料封包自起始節點傳送至目的節點所需時間。優良的 NoC 設計除必須提供較低的資料傳輸延

遲外，也必須具有高的資料傳輸率；傳輸率為對於每一 flow (source-destination pair) 在網路單位時間內其目的節點所能接收的資料量，它衡量了一個網路傳輸資料的能力和效率。於圖 1-6 和圖 1-7 可得知，配置有較大的 32 個 flit 緩衝空間有助於獲得較佳的網路傳輸延遲及資料傳輸率。因此，欲達成硬體資源成本的節省和最小的功率消耗而一味縮減緩衝器尺寸並非一合適方法，其原因在於網路效能與緩衝資源之間存在著複雜的依賴關係。

因此在有限資源的 NoC 設計中，針對特定應用之單晶片系統(application-specific SoC)，如何藉由提升緩衝空間的利用率，進而達到緩衝資源使用的最佳化，且能同時保有原本的網路效能表現，成為本論文研究動機。

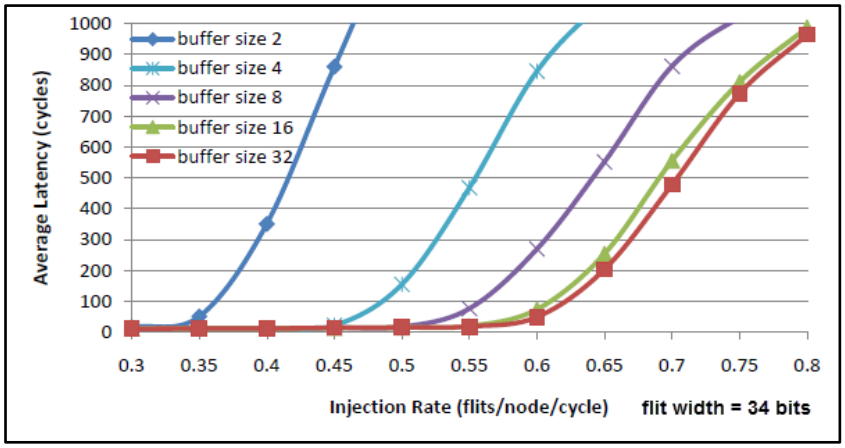


圖 1-6　不同緩衝器尺寸之平均傳輸延遲曲線
(3×3 mesh 拓樸採用隨機交通樣式)

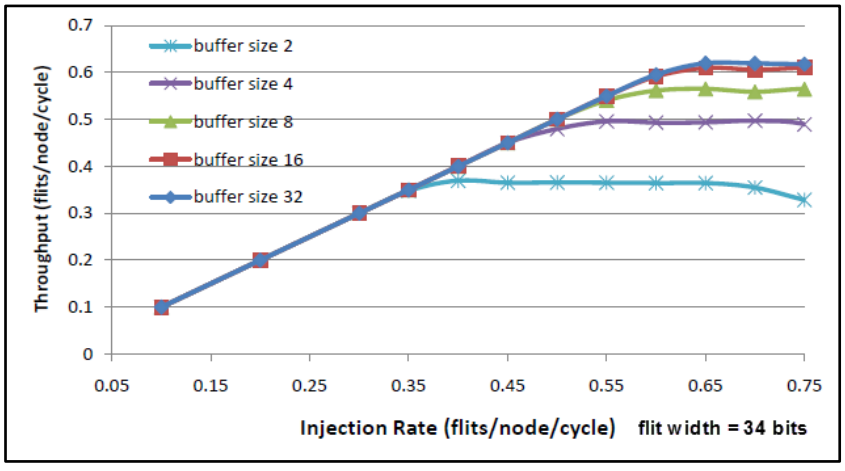


圖 1-7　不同緩衝器尺寸之傳輸率曲線
(3×3 mesh 拓樸採用隨機交通樣式)

1.2 相關研究

除緩衝器尺寸外，緩衝器的結構對於 NoC 效能表現同樣有直接的影響。此章節針對緩衝器的結構作說明。

在以交叉交換(crossbar switching)的路由器中，緩衝器存在原因，主要是為因應三種情況的發生[14]：(1)當多個封包在同一時間點抵達相同的輸出埠，而該輸出埠在該時間點上只允許接收其中一個封包;(2)封包在等待路由路徑的仲裁選擇時，路由器必須對該封包進行緩衝儲存；(3)當網路出現阻塞情況，導致封包無法依據其路由路徑傳送至鄰近路由節點。此外，緩衝器在路由器中依其座落位置可分為三個型態：輸出緩衝器(output buffer)、共享式中心緩衝器(shared central buffer)和輸入緩衝器(input buffer)。

輸出緩衝器一般用於當路由器進行封包交換的速率大於輸出通道且緩衝器有足夠大的緩衝空間可容納所有輸入的封包時，主要原因是因為多個封包可能在同一時間點上被配置到相同輸出埠。然而，如此高速度的交換技術並非容易實現，且輸出緩衝器在結構上必須具有多個寫入埠(write port)以完全對應到路由器所有輸入埠，因而造成邏輯延遲的增加[14]。

共享式中心緩衝器採用單一中心記憶體(single central memory)用以處理各個方向資料的輸入緩衝儲存，其主要優點為對任一個輸入埠所接收的封包具備高度彈性動態地配置其有效緩衝空間。然而，該結構卻存在兩個關鍵缺陷[14][15]：(1)對高效能路由器的實現而言，記憶體的頻寬將遭遇瓶頸，即為使記憶體資料的寫入與讀取可同時進行，記憶體所提供的頻寬至少須為輸入埠數量的倍數。而高頻寬緩衝器的實現，則需要多個 flit 寬度的記憶體以及更複雜的機制用以記錄不同方向的封包儲存於哪一個記憶體片段。此類複雜的設計間接衍生出第二個問題；(2)傳輸延遲的增加。為滿足高頻寬記憶體的寬度，將需要花費額外的時脈週期以完成資料的寫入或讀出動作。

於先前的 NoC 研究文獻中，輸入緩衝器常被採用。因為輸入緩衝器可簡單地藉由單一寫入埠的佇列實現，如此可有效的降低晶片面積與功率消耗。以下針對輸入緩衝器結構設計進行分析說明。輸入緩衝器的配置大致可分為四種型態[14][16]：

1. FIFO 緩衝器：圖 1-8 描繪出使用單一 FIFO 佇列作為各輸入緩衝器的路由器，其中，緩輸入緩衝器藉由一個 4×4 交叉開關(crossbar switch)連接至輸出埠。由

圖可知，各個輸入緩衝器為單一固定長度的佇列所構成，新進的封包由該佇列的後端寫入，於該佇列的前端讀出並通過交叉開關到達輸出鏈結。由於佇列的長度固定，路由器可容易地追蹤鄰近路由器之有效緩衝空間並確保封包只在鄰近路由器具有可用緩衝空間時進行傳輸。單一輸入埠對應單一佇列的結構設計可有效節省晶片面積，但面對資料流量負載較重的情況，卻容易遭遇 Head-of-Line(HoL)問題，造成網路不必要的阻塞。HoL 意指當一個封包位於佇列的前端，同時其目標輸出埠可能因被其它封包所使用而處於忙碌狀態，導致該封包在傳輸上必須等待。此情況連帶造成緩衝儲存於相同佇列中後端所有封包的阻塞效應，即便它們的目標輸出埠為空閒狀態，也將無法進行傳輸，因而浪費重要的資源，如緩衝空間、內部連線和輸出埠等，嚴重影響網路效能。

為更有效率的使用輸出埠，並增加路由器的資料傳輸率，資料封包必須根據其路由路徑所指定輸出埠的不同而被隔離。該目標的實現可利用靜態地配置且完全連接(Statically Allocated Fully Connected, SAFC)的緩衝器結構達成。

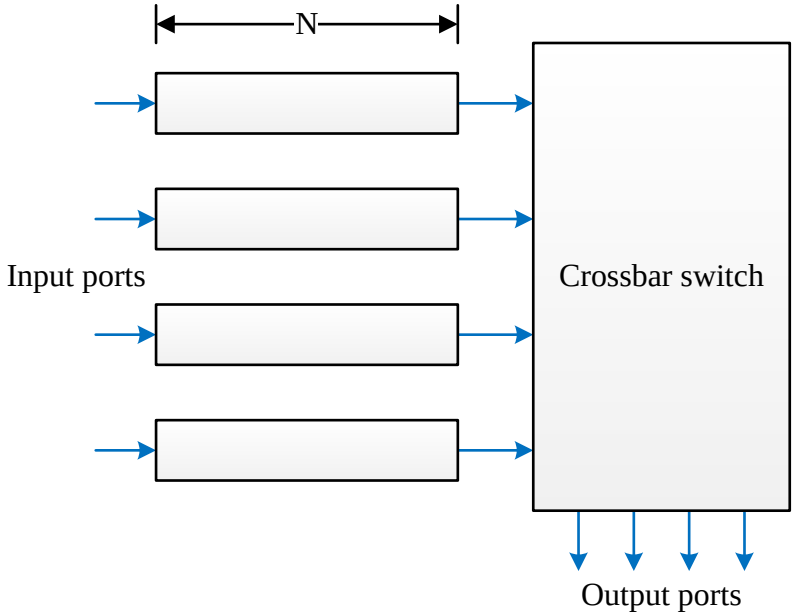


圖 1-8　FIFO 緩衝器

2. SAFC 緩衝器[17]：為了解決 HoL 問題，SAFC 路由器對每一個輸入緩衝器作靜態地佇列分割，而被分割出的佇列彼此之間完全獨立且分別對應至各個輸出埠。以圖 1-9 為例，分割的佇列總數為 16 個。此外，由於單一輸入緩衝器

具有 4 個分割佇列，因此需要 16×4 的交叉開關亦或如圖 1-9 所描繪的 4 個 4×1 的交叉開關，用以支持所有可能的封包傳輸路徑。

當封包緩衝儲存於相應的佇列中，對於欲使用相同輸出埠的封包而言，彼此之間會相互競爭，但位於佇列前端的封包將不再對其它封包造成阻塞。SAFC 緩衝器相對於 FIFO 緩衝器能提供較高的資料傳輸率，然而，SAFC 緩衝器存在著幾個缺點，以圖 1-9 為例：(1)因為有效緩衝空間被靜態地分割後，對於任何輸入封包而言，每一個輸入埠的有效緩衝空間只有四分之一，故緩衝空間利用率相較 FIFO 緩衝器為低；(2)封包在傳送之前，必須於當下路由器中執行預先路由選擇(pre-routing)，以得知該封包在下一個路由節點中欲傳輸使用的輸出埠，如此才可順利將該封包傳送至相應的緩衝佇列。然而，預先路由選擇的實現，將增加路由電路(routing circuit)的設計複雜度及硬體資源；(3)SAFC 路由器的設計相對較複雜，其必須建立額外的控制器，同時將該控制器硬體複製多份，以對配置的佇列及交叉開關有適當的管理。但在有限資源的 NoC 實現下，硬體的複製動作將導致晶片面積的浪費。

為了減少硬體資源的消耗，SAFC 路由器可透過靜態地配置多個佇列(Statically Allocated Multi-Queue, SAMQ)緩衝器結構替代。

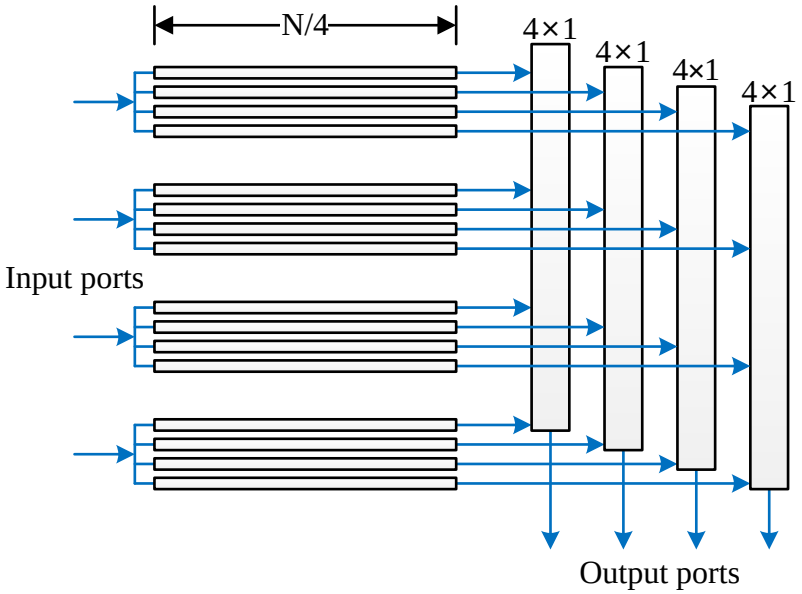


圖 1-9　SAFC 緩衝器

3. SAMQ 緩衝器：SAMQ 路由器同 SAFC 路由器於每一個輸入緩衝器中靜態地配置多個佇列，但該多個佇列彼此間不再相互獨立，而是被群組化，使得該

多個佇列讀取埠由 4 減為 1，藉此改善了 SAFC 路由器設計上的複雜度。如此緩衝器配置方案亦被稱為虛擬通道(Virtual Channel, VC)[18]。當多個封包經相同輸入埠分別緩衝儲存於各佇列中，而後，在一時間點上只允許單一個佇列的封包被讀取並傳遞至交叉開關，如圖 1-10，也因此，SAMQ 路由器不須對多個交叉開關進行管理。然而，靜態地分割緩衝器，使得緩衝空間的利用率依舊無法獲得提升，一個較好的方法是藉由動態地配置多個佇列(Dynamically Allocated Multi-Queue, DAMQ)緩衝器以增加使用效率。

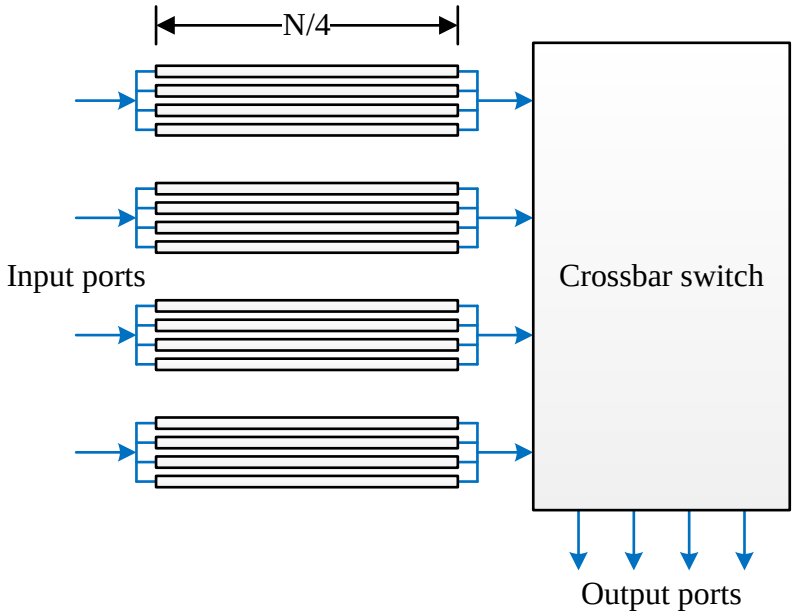


圖 1-10　SAMQ 緩衝器

4. DAMQ 緩衝器[19][20]：每一個輸入緩衝器中分別為各個輸出方向設置一個緩衝佇列，佇列長度則依據所接收的封包做動態地配置，並藉由共享方式來提高緩衝空間利用率，如圖 1-11。DAMQ 緩衝器的資料路徑必須依靠鏈結串列(linked list)系統來組織。但其中因 head pointer、tail pointer 和 free pointer 頻繁的更新，將造成複雜的電路設計，且對於資料的寫入與讀取將花費 3 個時脈週期的延遲時間，不易被晶片網路採納使用。除此之外，由於封包緩衝儲存於相同的佇列當中，因此必須遵守 FIFO 佇列的讀取順序，加上 DAMQ 結構通道數量有限，故 HoL 阻塞依然明顯。

由於 SAMQ 路由器被現今晶片網路研究大量採用，故本論文以 SAMQ 路由器為基礎並進行改良，企圖保留 SAMQ 緩衝器優點，並改善其緩衝空間利用率不佳的缺點。

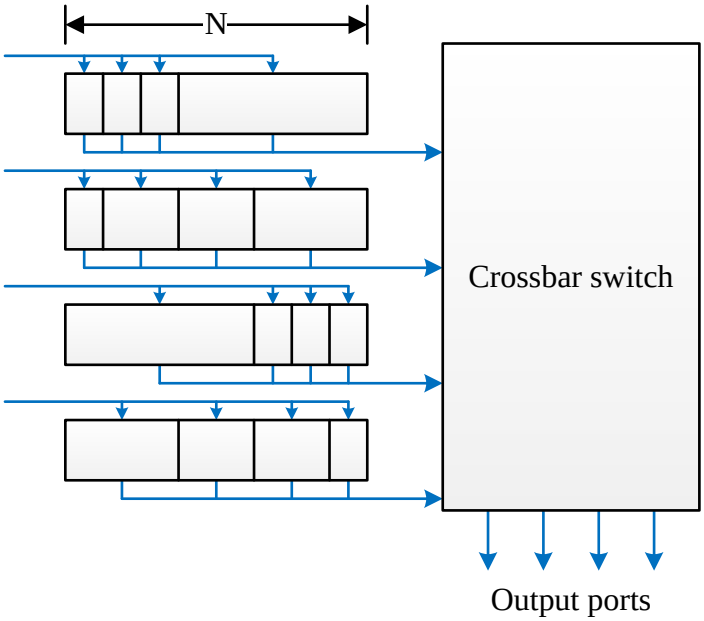


圖 1-11　DAMQ 緩衝器

1.3 論文章節組織

本論文共分五章，第一章概述本論文研究動機、相關研究和論文章結組織編排。第二章說明與本論文相關之研究背景，包含網路拓樸、路由演算法、交換技術等。第三章介紹本論文主要之緩衝器合併演算法，並針對不同的應用提出新的路由器架構。第四章驗證本論文所提出的架構，並針對模擬數據分析。第五章則為結論。

第二章 背景

本章針對現有 NoC 設計相關技術作分類整理與探討。涵蓋的主題與內容主要有：拓樸網路、路由演算法及交換技術的介紹與分析比較。

2.1 拓樸網路

拓樸網路(topology network)意指在 NoC 中靜態地配置路由節點和鏈結通道，決定其實體佈局及連接。拓樸的選擇為 NoC 設計的首要步驟，其後之路由策略、交換技術和路由器結構設計皆以拓樸為基礎而建立。

拓樸結構對於整體網路的效能表現和成本效益有極大的影響，效能和成本亦成為拓樸選擇所依據的重要特徵條件。效能方面，拓樸結構決定了資料傳輸可能經過的路由節點數量多寡及其之間鏈結的長度，從而影響網路延遲變化。此外，拓樸結構亦決定了各路由節點之間存在的傳輸路徑的多樣性，而影響該網路是否能將交通流量傳遞開來，符合頻寬要求。成本方面，取決於實現拓樸網路中每一個路由器之節點度以及晶片上拓樸佈局複雜度(接線長度、密度等)。同時，拓樸結構所提供的傳輸路徑距離長短也會直接影響網路功率的消耗。

以下章節，將具體指出影響上述效能和成本兩項特徵條件變化之衡量指標及普遍用於 NoC 的拓樸結構介紹。

2.1.1 拓樸比較之衡量指標

造成拓樸產生效能和成本差異的主要衡量指標包括：

1. 節點度(node degree)：表示在拓樸網路中，每一個路由節點與其它路由節點相互連接的鏈結數量，即路由器所需要的輸入/輸出埠數量。節點度通常作為網路成本的衡量指標，因高節點度勢必造成路由器結構輸出入埠的增加，如此雖可有更多的資料傳輸替代路徑選擇機會，但同時也增加了實現上的複雜度與額外面積成本。

2. 平均跳躍數(hop count)：表示資料從起始節點傳送至目的節點的過程中所經過的鏈結數量。由於每一個路由節點和鏈結通道的穿越，皆會招致資料傳輸上

的延遲，即使過程並無壅塞也將如此，故平均跳躍數常作為衡量網路延遲的指標。而平均最小跳躍數可藉由網路上所有可能的 flow 之平均最小跳躍數計算獲得。

3. 最大通道負載(maximum channel load)：用來評估網路所能支持的最大頻寬。換句話說，於網路飽和前，每一個路由節點每秒可注入網路的最大 bits 數(bit per second, bps)。通道負載可藉由多種方法計算得知，典型方式為使用機率分析。即使在路由選擇和交換技術尚未決定前，通道負載依然可藉由理想路由選擇與理想流量控制的假設方式來計算之。

4. 路徑多樣性(path diversity)：可提供路由演算法更多的彈性選擇以平衡交通流量負載並且避開網路中壅塞或失效的通道。

2.1.2 Mesh 和 Torus 形式拓樸結構

1. CLICHÉ：S. Kumar 等人提出一個基於網狀(mesh)排列型式的互連拓樸結構，並稱為 CLICHÉ(Chip-Level Integration of Communicating Heterogeneous Elements)[7]，在本論文中以 mesh 拓樸稱之，如圖 2-1 為一個 4×4 mesh 拓樸結構範例。其中，所有非邊緣路由節點皆有五個埠，四個連接相鄰路由節點，另一個連接 local 的 SIP，形成一網狀結構。Mesh 拓樸結構中，路由節點有著和計算節點相同的數量，而各節點彼此之間的溝通藉由兩條單向的鏈結所構成的通訊通道來達成。Mesh 拓樸結構簡單易於實現且可擴充性良好，為目前較為廣泛採用的拓樸結構之一。但該拓樸結構的網路直徑和平均距離較大，尤其對兩對角線節點之間的資料傳輸而言。因此，當網路規模擴充後，較長的資料傳輸距離將導致功率消耗的提升。

2. Torus：為改善 mesh 拓樸結構的缺點，W. J. Dally 和 B. Towles 提出一個相似於 mesh 拓樸的 2D torus 拓樸結構[21]，如圖 2-2 為一個 4×4 torus 拓樸結構範例。2D Tours 拓樸與 mesh 拓樸主要差異在於 tours 拓樸結構藉由增加額外的鏈結，將每行和每列的兩邊緣路由節點相連接而形成一個環狀鏈結，即每一個路由節點皆有四個埠與鄰近路由節點相連接。如此藉由額外的鏈結，明顯縮短平均距離，減少資料傳輸上的跳躍數，從而降低功率消耗。同時，因每一個路由器節點度的一致性，其擴充性也獲得了加強。但不幸是，如此設置將使得網路接線更加複雜化，且過長的環狀鏈結亦會造成額外的傳輸延遲。

3. Folded Torus：folded torus 的提出是為避免 torus 拓樸結構中長距離鏈結所造成的額外傳輸延遲[22]。該拓樸結構採用錯位鏈結的方式來代替 torus 拓樸中的環狀鏈結，如圖 2-3。如此設置方式縮短了鏈結的長度，因而解決了 torus 拓樸架構的缺點，不僅提高了整體網路效能且更易於 VLSI 的實現。

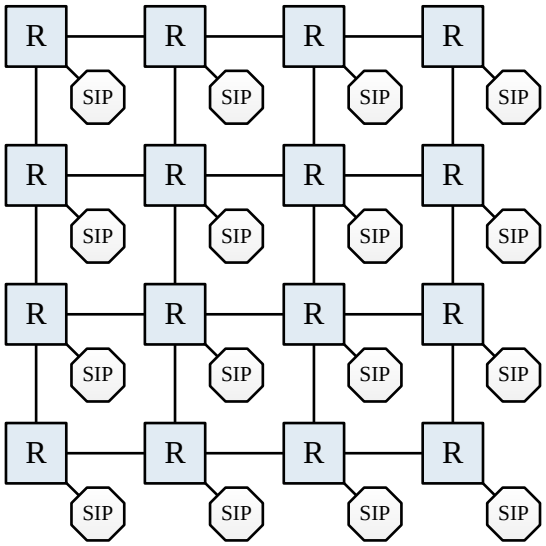


圖 2-1　4×4 mesh 拓樸結構

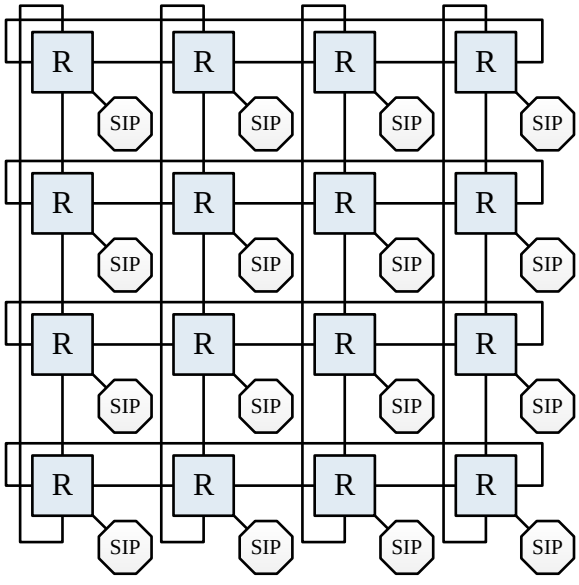


圖 2-2　4×4 torus 拓樸結構

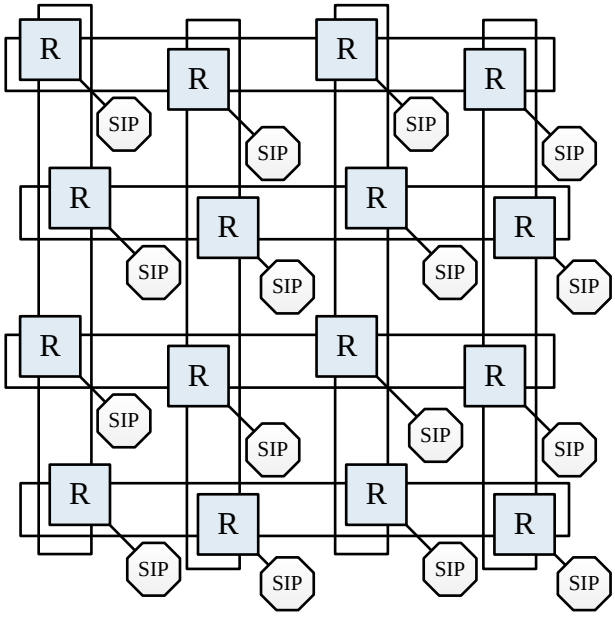


圖 2-3　4×4 folded torus 拓樸結構

2.1.3 Tree 形式拓樸架構

1. 平衡二元樹(balanced binary tree)：圖 2-4 為一個平衡二元樹拓樸結構。其中，葉節點(leaf node)為 SIP 所建立之計算節點，而所有非葉節點均為路由器所建立。該拓樸結構特點是節點之間不存在迴圈，故不會有死結情況的產生。但其存在著單一父節點(parent node)的特性，尤其是根結點(root node)，容易成為通訊瓶頸所在。

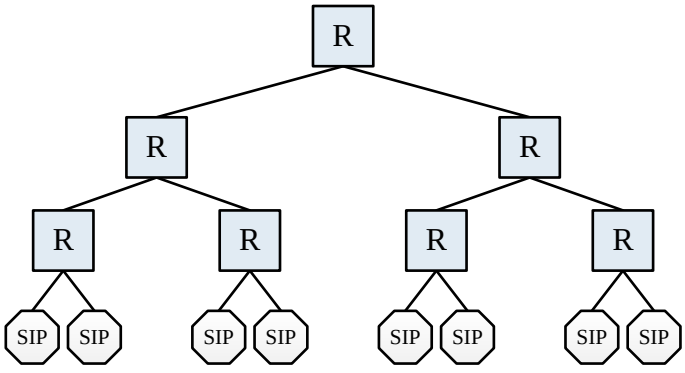


圖 2-4　平衡二元樹拓樸架構

2. Scalable, Programmable, Integrated Network (SPIN)：SPIN 為一種可擴充的 fat tree 拓樸結構[23]。在 fat tree 結構中，每一個葉節點處為計算節點，非葉節點

處皆為路由節點且可以有多個父節點，可藉此改善平衡二元樹主要缺點，如圖 2-5 為基本 SPIN 示意圖，具有 16 個葉節點，兩層共 8 個路由節點。其中，由於每一個父節點在樹的每層都會被複製四次，使得階層 1 的各路由節點具有相同數目的父節點與葉節點，實現了良好的可擴充性，當網路規模變大時，仍然具有較佳的網路效能。當各計算節點之間需要通訊時，首先會在底層路由節點中查找本身子節點是否存在欲相互通訊的目的節點。若無，則進一步向上層路由節點查詢，如此通訊方式減低了系統路由的壓力。但該拓樸結構的交換複雜，且接線複雜度高，增加了晶片面積的成本。

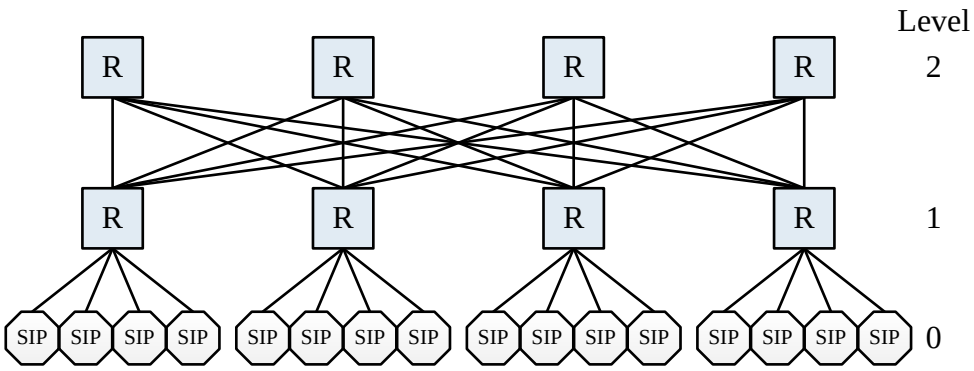


圖 2-5　SPIN 拓樸架構

3. Butterfly Fat Tree(BFT)：BFT [24]與 SPIN 結構相似，如圖 2-6 為具有 16 個計算節點和 6 個路由節點之 BFT 拓樸架構。該拓樸結構中路由器及其鏈結的使用都大為減少，改善了 SPIN 增加面積成本的缺點。除此，BFT 拓樸架構的最大優點為任兩個計算節點之間通訊的距離為相等。

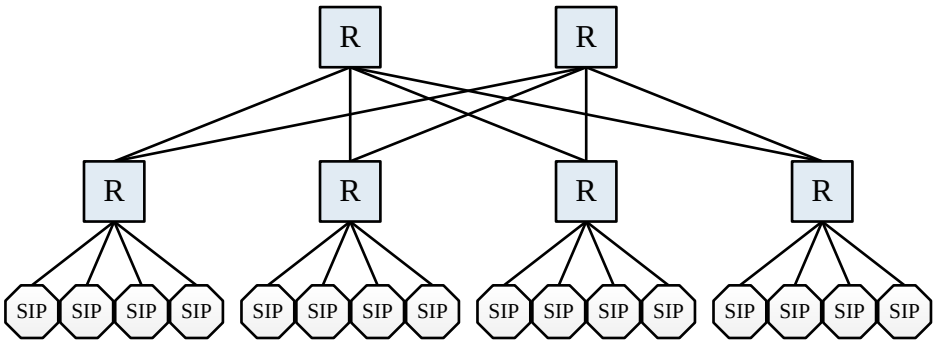


圖 2-6　BFT 拓樸架構

2.1.4 Ring 形式拓樸架構

為解決 SoC 中匯流排結構存在可擴充性差的缺陷，F. Karim 等人提出 OCTAGON 拓樸結構[25]。OCTAGON 拓樸的基本模型為路由節點和計算節點數量分別各為 8 個，且彼此間透過 12 條雙向鏈結相互連接，如圖 2-7 所示。該拓樸結構最大特點是平均傳輸距離短，即在一個基本 OCTAGON 拓樸網路中，任兩個計算節點之間的通訊至多只需經過兩個跳躍數(經由一個中間節點)即可完成。另一方面，該結構中每一個路由節點皆可輕易連接另一個基本 OCTAGON 拓樸單元，達成多維度的擴充延伸，如圖 2-8 所示。但該結構擴充後很明顯導致接線的高密度且使路由器設計更加複雜。除此，隨著 SIP 數量的增加，連接兩個 OCTAGON 拓樸單元的橋接節點(bridging node)很容易成為通訊的瓶頸。當該橋接節點失效時，整個擴充後的拓樸結構將被分割為獨立的三個部分。

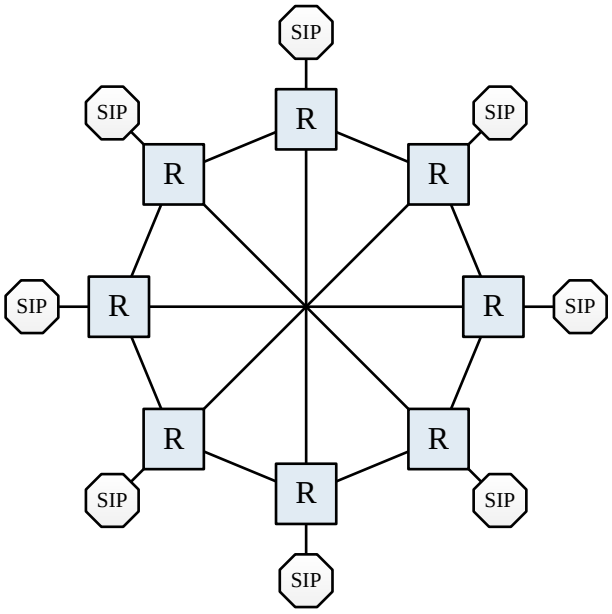


圖 2-7　OCTAGON 拓樸單元

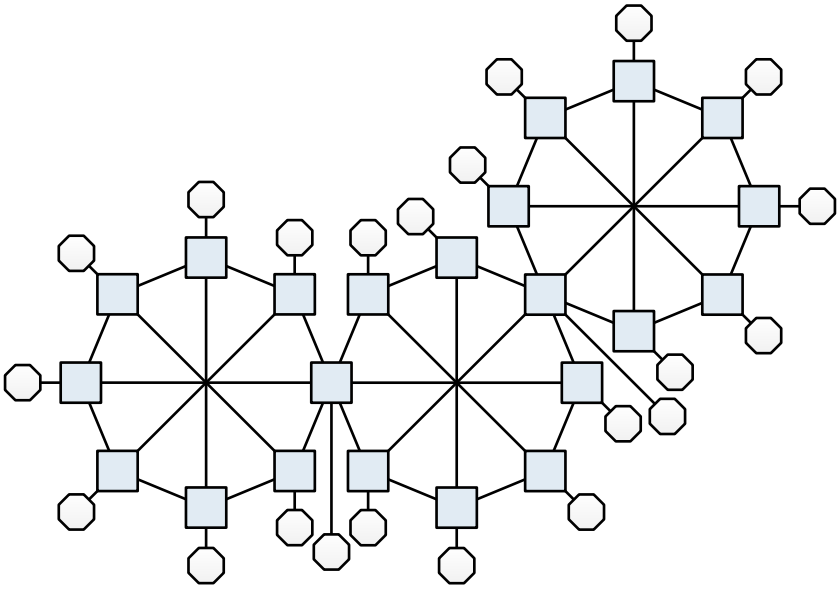


圖 2-8　擴充 OCTAGON 拓樸架構

2.2 路由演算法

路由演算法(routing algorithm)用於決定資料封包在網路中如何選擇傳輸路徑到達其目的節點，即於特定拓樸網路中，從起始節點至目的節點之間選擇一條路徑予以封包傳輸。路由演算法對 NoC 的資料傳輸率及效能表現有直接影響。

路由演算法的目的在於對拓樸網路所提供的傳輸路徑中，均衡地分配交通流量負載，以避免熱點(hotspot)產生並且盡量減少資源衝突的情況，從而改善網路的傳輸延遲和資料傳輸率。愈是均衡的通道負載，其網路效能表現愈理想，但此特性卻明顯地牴觸了其它重要約束條件：盡可能地保持較短的傳輸距離，意即資料封包的傳輸只需經過較少的節點數即可送達至目的節點，但選擇此一路徑將可能導致網路通道負載不平衡。因此，設計或選擇路由演算法，評估通道負載的均衡及傳輸距離的長短必須慎重考量。

另一方面，NoC 通訊上的傳輸延遲及功率消耗也有相當大程度關係到路由演算法，故實現路由機制所帶來的影響也須仔細考慮。路由電路會拉長路徑延遲並增加路由器面積，且於功率方面，儘管路由電路通常功率消耗量低，但具體路徑選擇影響了所經節點數的多寡，因此也直接地影響了功率消耗。

2.2.1 路由演算法的分類

路由演算法一般劃分為確定性路由演算法(deterministic routing algorithm)、模糊性路由演算法(oblivious routing algorithm)和適應性路由演算法(adaptive routing algorithm)三類，係根據如何選擇於起始節點至目的節點間存在的有效傳輸路徑而予以分類[26]。

1. 確定性路由演算法：對資料傳輸而言，確定性路由演算法於起始節點至目的節點之間只會選擇唯一且相同的路徑，即使起始節點至目的節點之間存在有多條可能路徑。雖然目前許多路由演算法已經被提出，但最為普遍使用於 NoC 中的確定性路由演算法莫過於最簡單的維序路由演算法(dimension-ordered routing algorithm)，例如 X-Y 或 Y-X 路由演算法。維序路由演算法採用 dimension by dimension 的方式傳輸資料。如圖 2-9 為一個 2D mesh 拓樸網路，X-Y 路由演算法首先於起始節點(0,0)沿 X 軸傳送封包，當到達與目的節點有相同的 X 座標節點時，再轉向沿 Y 軸繼續傳送至目的節點(3,2)。

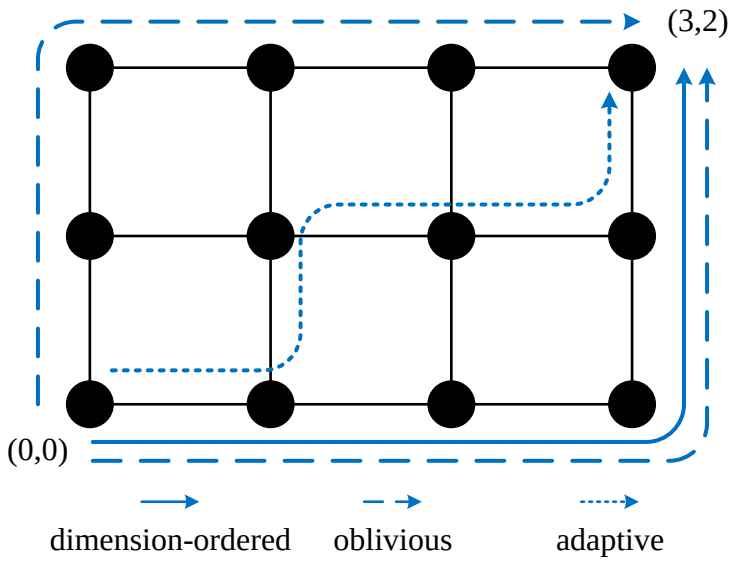


圖 2-9　路由演算法的分類

採用確定性路由演算法的好處是其邏輯電路設計較為簡單且於網路無壅塞情況下可提供低延遲傳輸。但隨著資料注入率(injection rate)的提高，卻很可能導致資料傳輸率的下降，其原因在於確定性路由演算法無法對網路壅塞情況作出動態調整，意即確定性路由演算法忽略了潛在於特定拓樸網路之下的路徑多樣性，因而無法避開網路壅塞區域。也正因為如此，確定性路由演算

法對於平衡網路通道負載方面而言較不適宜。儘管如此，實際上確定性路由演算法易於實現且無死結(deadlock-free)的優點，依然使其相當普遍使用於 NoC 設計中。

2. 模糊性路由演算法：包括了確定性路由演算法作為一子集。模糊性路由演算法容許資料的傳輸透過不同路徑來進行，其中路徑的選擇並不考量任何網路當前狀態，如網路壅塞程度。換句話說，路由器可在可供選擇路徑之中隨機性地選擇其一用以傳輸資料。以圖 2-9 為例，資料封包將於(0,0)傳輸至(3,2)，傳輸路徑的選擇可隨機挑選 X-Y 路由路徑亦或 Y-X 路由路徑。但值得注意的是，此隨機選擇 X-Y 或 Y-X 路由路徑將導致死結情況的發生；反之，倘若一致性的只考慮 X-Y 路由路徑，則是為無死結模糊性路由演算法。

   設計或選擇模糊性路由演算法須詳細考慮資料區域性(data locality)與負載平衡兩者之間的取捨[27][28]。另一方面，因無考量網路狀態使得模糊性路由演算法易於實現與分析，但由於考慮網路狀態可能改善路由效能，因此一般認為模糊性路由演算法在實際效果上不如適應性路由演算法，但在[29]，該作者認為適應性路由演算法即使付出了設計複雜度的代價，但並未帶來應有的效能提升，更可能由於不當的設計導致效能下降，因為設計者無法得知整個全局網路的狀態。同時該作者根據應用程序的網路路由模型進行分析，靜態的配置模糊性路由器並証明了其優越性。

3. 適應性路由演算法：為最複雜的路由演算法。適應性路由演算法可適應於網路交通狀態，並依據此狀態資訊作為路由決策之條件，其中狀態資訊可能包含一個路由節點或者鏈結的狀態、緩衝佇列長度、通道負載資訊等。以圖 2-9 為例，資料封包於(0,0)發送至(3,2)，最初封包依 X-Y 路由路徑到達節點(1,0)，且接收到壅塞資訊於此節點東方輸出鏈結，因而將封包轉向傳送於北方輸出埠。適應性路由演算法的實現邏輯相較於確性路由演算法複雜，且當網路只有輕度壅塞時，資料傳輸的延遲較大。

   局部或全局網路資訊皆可作為適應性路由演算法設計之決策判斷條件。理論上，一個良好的適應性路由演算法效能應優越於模糊性路由演算，但實際上則不然。主要因為許多適應性路由演算法僅考慮局部路由節點狀態資訊(如佇列佔用率和佇列排隊延遲)，用以判斷網路壅塞程度及選擇輸出鏈結，如此路由方式能平衡局部負載，但往往造成網路全局負載的不平衡[30]。使用通道緩衝器管理機制(channel buffer management mechanism)可迅速的獲得遠端

壅塞資訊，藉以改善之。

另一方面，適應性路由演算法亦可被分類為最短路徑(shortest path)和非最短路徑(non-shortest path)[31]。在低密度交通流量下(無壅塞)，非最短路徑適應性路由演算法因額外的傳輸節點數增加了網路傳輸延遲及功率消耗，反觀最短路徑適應性路由演算法則實現了較好的效能結果；而在高密度交通流量下(壅塞)，非最短路徑適應性路由演算法的選擇可能較合適，因它可避開壅塞的鏈結，減少資料傳輸延遲時間。

2.2.2 死結和活結

設計或選擇路由演算法，不僅須考慮延遲、功率、傳輸率及可靠度等影響層面，大多數應用程序更要求網路無死結的保證。死結(deadlock)是一種資料永遠阻塞的情況。當多個資料封包處於等待其它封包釋放佔用的資源時，將無法繼續前進。倘若這些處於等待狀態中的封包排列形成一環狀時，死結將發生於網路之中。

圖 2-10 顯示四個路由節點，且每個路由器具有單一封包大小的輸入緩衝空間。在此情況下，packet 0 從節點 A 之西方輸入埠進入，而後抵達緩衝儲存於節點 B 之西方輸入緩衝器，且繼續停留等待在緩衝器中直至獲得使用節點 C 之北方輸入緩衝器，而節點 C 之北方輸入緩衝器正同時被 packet 1 佔據使用，其等待且欲使用節點 D 之東方輸入緩衝器。以此類推，將導致死結的產生並且阻擋限制其它封包的傳輸，進而癱瘓全局網路的運行。藉由使用死結避免方法(deadlock avoidance methods)，設計一個路由演算法或特殊路由器架構以保證網路的無死結亦或具檢測死結的發生並調節排除之死結恢復方法(deadlock recovery methods)，能有效防止網路死鎖的發生。

活結(livelock)的發生同樣能造成網路陷入困境。不同於死結的是活結發生後，資料封包仍可持續於網路中移動傳輸，但此類封包將無法抵達其目的節點。活結的發生主要涉及非最短路徑適應性演算法，因其具有封包錯誤指向(misrouting)的特性。對每一個資料封包限制其錯誤指向的最大次數可避免活結的存在[15]。

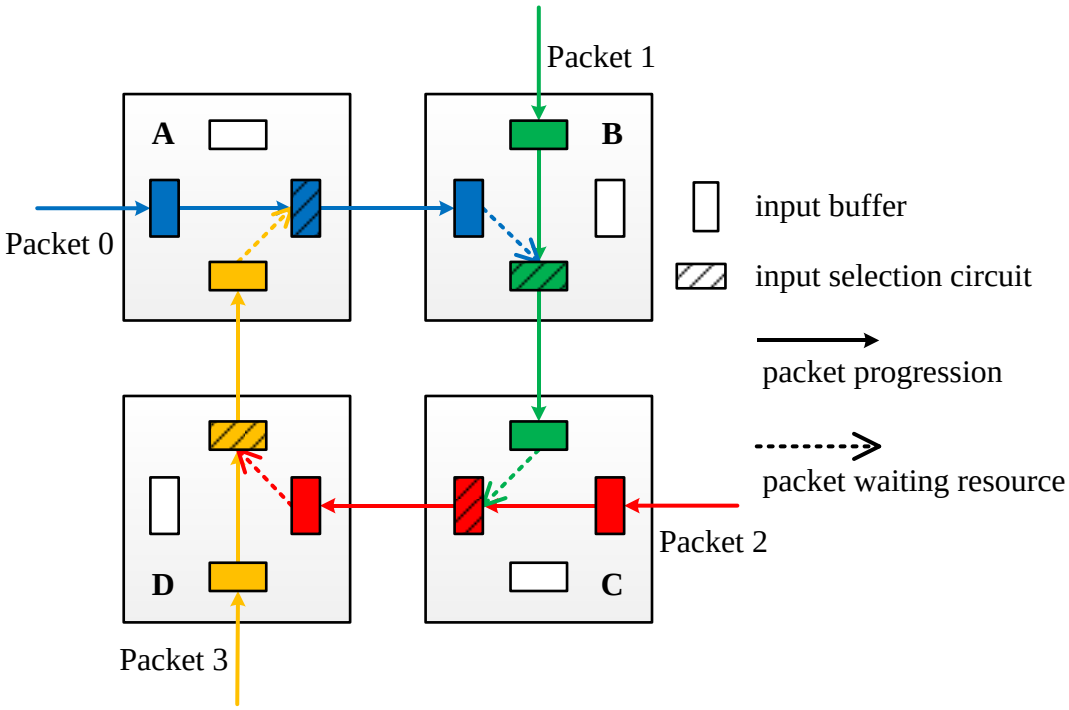


圖 2-10　環狀資源等待路徑造成網路死結的發生

2.2.3 轉向模型路由演算法

由上述得知，死結問題對於網路效能及資料傳輸正確性有相當大程度的影響。無死結路由演算法可藉由轉向行為來描述之，如圖 2-11 顯示了於 2D mesh 拓樸網路下所有可能的路由轉向行為。然而，允許所有可能轉向的運行可能導致資源等待路徑形成一個環狀相依關係而產生網路死結。故以下針對 NoC 中較普遍被採用的 2D mesh 拓樸網路之轉向模型路由演算法作一介紹。

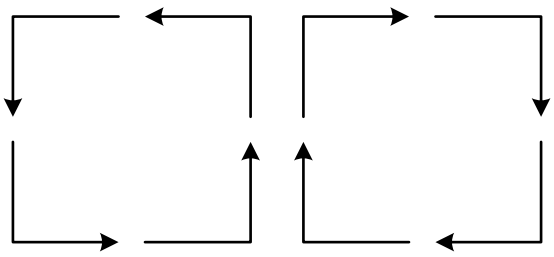
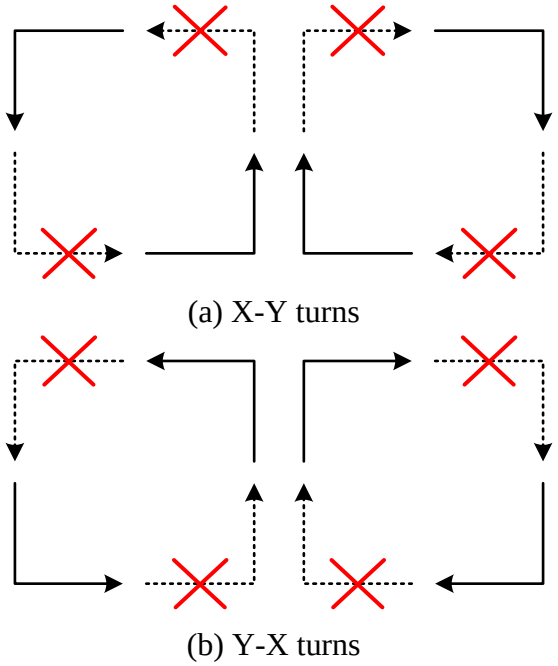


圖 2-11　2D mesh 拓樸網路中可能的路由轉向

1. X-Y 路由演算法[7]：為最簡單且最為普遍被使用的確定性之維序路由演算法。其資料封包傳輸自起始節點沿最短路徑，先以 X 軸方向傳送，待封包抵達與

目的節點相同 X 座標節點時再轉向以 Y 軸方向傳送至目的節點。於圖 2-12(a)可知，資料傳遞移動於東方向和西方向則被允許轉向至南方向和北方向，而資料傳遞移動於南方向和北方向是不允許任何轉向。同理可得，圖 2-12(b)為 Y-X 路由演算法轉向限制示意圖。如此設置轉向限制條件致使圖 2-11 中四個轉向行為中的兩個轉向行為不被允許運行，故環狀資源等待路徑不存在且無死結發生可能。但如此的轉向限制卻容易造成網路的阻塞，因為封包在到達任一目的結點前都先以 X 軸方向開始傳輸，如果多筆資料封包同時處於同一 X 軸方向上傳輸，對於 X 軸方向上的路由器的東或西輸入通道容易造成阻塞。



(a) X-Y turns

(b) Y-X turns

圖 2-12　X-Y 路由演算法及 Y-X 路由演算法

2. West-First 路由演算法：為非最短路徑適應性路由演算法但同樣能保證網路無死結[32]。其基本概念是先將資料向西方向傳送，再適時地向東、南、北方向轉向傳送。圖 2-13 顯示 West-First 路由演算法只禁止北轉西(NW turn)轉向及南轉西(SW turn)轉向，也就是每一個環狀資源等待路徑只禁止一轉向行為。如此最少轉向限制條件的設置相較於 X-Y 路由演算法而言，增加了路徑選擇的彈性。

此外，藉由 West-First 路由演算法概念，可加以衍生出其他無死結路由演算法，例如 North-Last routing、Negative-Last routing [32]。

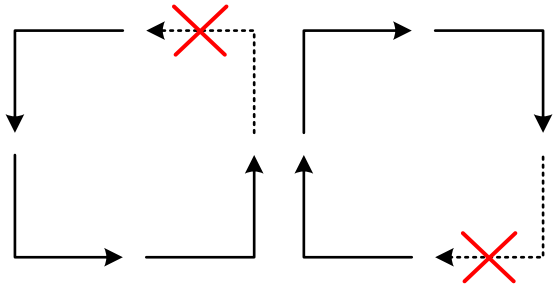


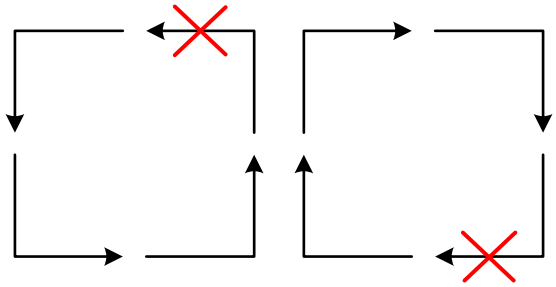
圖 2-13　West-First 路由演算法

3. 奇偶轉向模型(odd-even turn model)：為適應性路由演算法，就 2D mesh 拓樸網路而言，依據行座標數值，把所有的行分為偶數行(even column)與奇數行(odd column)。奇偶轉向路由模式的基本觀念就是藉由限制某些路由轉向能發生的行為，以達到無死結的特性。更明確地說，它靠著下面兩個規則(圖 2-14)來管制轉向行為[33]：

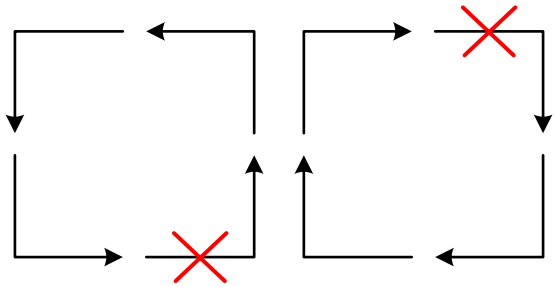
   (1) 任何封包不可在偶數行做東轉北(EN turn)轉向，並且任何封包不可在奇數行做北轉西之轉向(NW turn)。

   (2) 任何封包不可在偶數行做東轉南(ES turn)轉向，並且任何封包不可在奇數行做南轉西之轉向(SW turn)。

   上述的管制規則其實是避免死結現象必有的環狀資源等待路徑最東邊的一行的存在，因此，死結即無產生的可能。另一方面，有別於其它轉向模型演算法而言，奇偶轉向模型演算法藉由資料封包的相關位置來限制封包轉向，如此設置為其本身提供了多樣性路由路徑的優點，尤其在網路具有故障鏈結(faulty link)時，將需要較多的路由路徑來避免之。



(a) odd column



(b) even column

圖 2-14　奇偶轉向模型演算法之禁止轉向規則

2.3 交換技術

交換技術(switching technique)是藉由 link level 的流量控制型態而分類，因此亦稱為網路流量控制(network flow control)。交換技術決定如何配置網路資源(例如緩衝器、鏈結通道)予以資料封包傳輸於網路中。不同的交換技術會導致不同的資料傳輸延遲時間，故交換技術為影響網路效能的重要原因之一[34]。

此章節介紹目前主要使用於 NoC 中的封包交換技術及常見的四種流量控制技術：儲存轉發、虛擬直通、蟲洞及虛擬通道。

2.3.1 封包交換

封包交換(packet switching)允許資料數據在沒有明確地 path-setup 時傳輸。一般而言，計算節點所產生的資料數據稱作訊息(message)，為一連續位元組。當一個訊息進入 NoC 中，此訊息首先會被劃分為數個長度固定的資料封包。封包為路由選擇基本單位，故含有路由資訊，其大小並無硬性規定，可根據需求設置，通常範圍從 128 bits (16 bytes)到 512 Kbits (64 Kbytes)，典型為 1 Kbit (128 bytes)。而後，每一個資料封包可再更進一步劃分為若干個流量控制單位(flow control unit)，

即資料微片(flit)。Flit 為頻寬與緩衝儲存分配的基本單位，通常大小範圍從 16 bits (2 bytes)到 512 bits (64bytes)，典型為 64 bits (8 bytes)。因此單一固定長度的封包可分解為以下 flit 類型(如圖 2-15)：

1. Head Flit：封包的第一個 flit，其包含封包的路由資訊(通常為封包的 header 欄位)。
2. Data Flit (body flit)：接續 head flit 之後的 flit，其包含封包資料或網路控制訊息等。
3. Tail Flit：接續 data flit 之後的 flit，即封包的末端。Tail flit 包含封包資料或網路控制訊息等。

Phit 為最小的實體傳輸單位(physical transfer unit)且其大小相應於實體鏈結通道寬度，大小範圍在 1 bit 至 64 bits 之間，典型為 8 bits。一般而言，phit 傳輸穿越一個實體鏈結需花費一個時脈週期，因此一個 flit 的傳輸需花費數個時脈週期，但為簡單說明，在此假設 phit 大小相等於 flit 大小。故當進行一個完整的訊息傳輸時，對計算節點之間而言，是簡單地傳送/接收數個封包，但從網路觀點來看，路由節點間必須執行多次的 flit 交換來完成一個封包的傳送與接收。如此交換技術可提升鏈結通道的使用率，且針對未來 SoC 應用在各種需求之間實現了重要的成本效益的平衡。

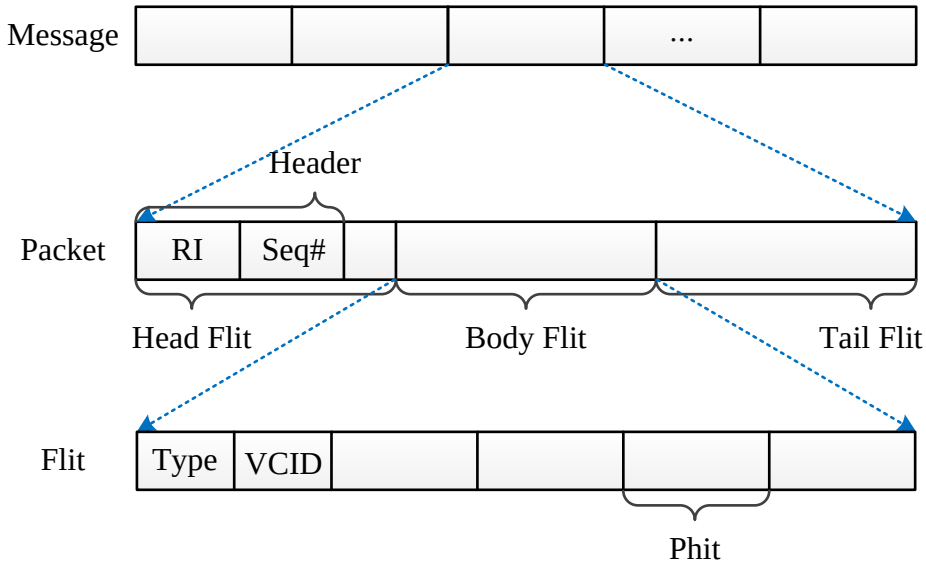


圖 2-15　訊息的組成

2.3.2 儲存轉發流量控制

儲存轉發(store and forward)流量控制[15][26]係指當封包到達中介路由節點後，會先將完整封包接收儲存於緩衝器內，之後依據路由演算法執行傳輸路徑的判斷選擇，當輸出通道空出且下一個欲傳輸的路由節點有足夠的緩衝空間可容納完整封包時，才將封包往下一個路由節點進行傳送。

圖 2-16 描述一個封包從起始節點傳送至目的節點且使用儲存轉發流量控制。由圖中可觀察得知,若一個封包由四個 flit 組成,則在傳輸路徑上的每一個節點中，封包的傳遞都將導致四個時脈週期的延遲時間，而此延遲時間將隨著封包長度增加而增加。除此，儲存轉發流量控制要求每一個路由節點須有足夠的緩衝空間用以緩衝儲存完整封包，故如此高的緩衝空間需求，加上高傳輸延遲的缺點皆使其不適用於 NoC 設計。

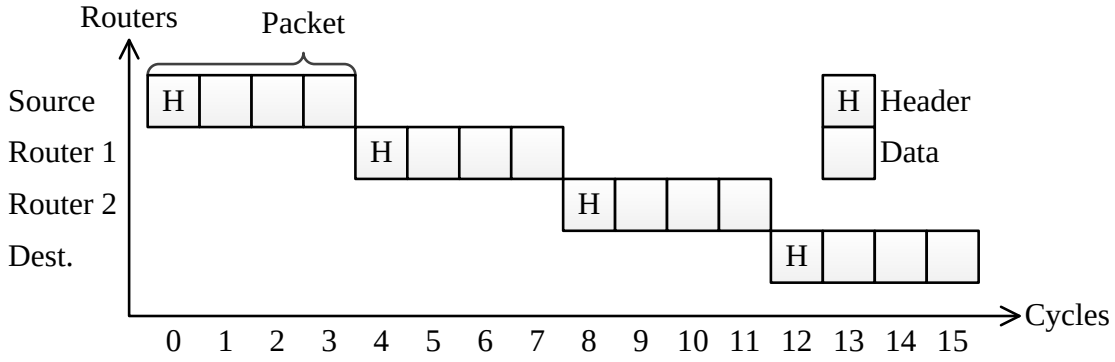


圖 2-16　儲存轉發流量控制範例

2.3.3 虛擬直通流量控制

虛擬直通(virtual cut-through)流量控制[35]改善了儲存轉發流量控制所造成的資料傳輸延遲，原因是虛擬直通流量控制在封包的 header 到達中介路由節點後，不需等待完整封包的接收儲存後即可立即選擇判斷下一個欲傳輸的路由節點，若當輸出通道空出且欲傳輸的路由節點有足夠緩衝空間可保證該封包的完整接收儲存時，則將封包進行傳送。反之，則將封包先行阻擋(blocking)，直至欲傳輸的路由節點有足夠緩衝空間可完整接收儲存該封包時才繼續傳送。

圖 2-17 描述一個封包從起始節點傳送至目的節點且使用虛擬直通流量控制。由圖 2-16 可知,使用儲存轉發流量控制需花費 16 個時脈週期傳送一個完整封包，

而使用虛擬直通流量控制只需花費 7 個時脈週期即可完成一個完整封包的傳送，如圖 2-17 所示。雖然虛擬直通流量控制克服了儲存轉發在資料傳輸延遲上的缺點，但虛擬直通流量控制與儲存轉發同屬於 packet-based 流量控制，所以依舊需要大量的緩衝空間於每一個路由節點當中，特別是在封包長度較大時，緩衝空間亦須隨之增加，因而直接影響資料傳輸效率、功率消耗及面積成本，使其在有限面積的 NoC 設計上難以獲得適用。

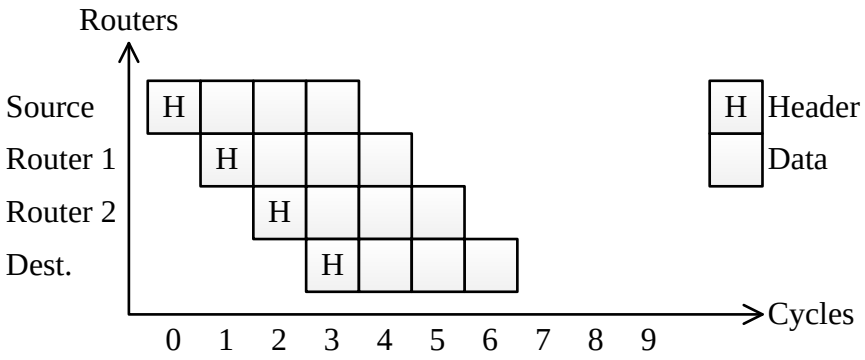


圖 2-17　虛擬直通流量控制範例

2.3.4 蟲洞流量控制

蟲洞(wormhole)流量控制[22]的概念源自蚯蚓爬行模式，蚯蚓爬行時頭部向前蠕動，身體尾隨其後，前進時不斷佔據前方路徑而放棄身後路徑。蟲洞流量控制將封包分解成若干個 flit，而後，以 flit 為單位執行管線(pipeline)方式傳輸，如圖 2-18 所示。蟲洞流量控制是藉由 head flit 直接從起始節點至目的節點間開闢一條傳輸路徑。當封包的 head flit 到達中介路由節點後，路由器根據 head flit 的路由資訊立即做出路徑選擇：

(1) 若所選擇路徑通道空閒(idle)且欲傳送的路由節點有足夠容納一個 flit 的緩衝空間可用，由於路由資訊只包含於 head flit 中，故 head flit 不必等待同一封包中的其它 flit 到達即可直接進行傳送，而同一封包中的其它 flit 則遵循 head flit 的傳輸路徑向前傳遞，同時間 tail flit 釋放所佔用的節點資源。

(2) 若所選擇路徑通道非空閒且欲傳送的路由節點緩衝空間已滿，則 head flit 必須在當下路由節點等待資源，而散佈在 head flit 傳輸路徑上的同一封包的其它 flit 則在其當下路由節點位置不動，而不從網路中移除。當 head flit 所在路由節點滿足(1)中條件時，同一封包中其它 flit 才可再次向前移動傳輸。

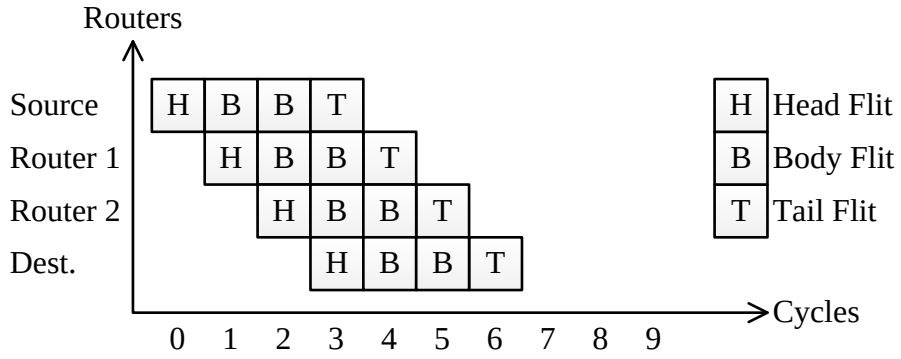


圖 2-18　蟲洞流量控制範例

在蟲洞流量控制傳輸過程中，flit 以不可錯亂的順序前進，head flit 控制路由資訊，其它 flit 緊隨其後，於此過程中不可被其它 flit 中斷，否則傳輸將出錯。蟲洞流量控制屬 flit-based，這使得每一個路由節點需要的緩衝空間大為降低，節省晶片面積，並且大大提高了資料的傳輸效率，但須值得注意的是通道阻塞的問題。虛擬直通流量控制允許封包的 header 被阻塞時，封包後面的資料部分仍可繼續前進傳輸並完整儲存於發生阻塞的路由器中，而不阻塞其它路由節點；但對於蟲洞流量控制而言，一旦 head flit 被阻塞，所有跟隨的 flit 將佔用傳輸路徑上各路由節點資源，導致阻擋了其它封包的前進而造成網路壅塞。如圖 2-19，藍色 flit 欲從節點 E 沿虛線路徑傳輸至節點 F，但因黃色 flit 傳輸上遭遇了阻塞情況，使得節點 A 與節點 B 之間的鏈結通道呈現空閒狀態，但該鏈結通道的使用權依然保留予以黃色 flit，從而導致該藍色 flit 傳輸上的延遲等待。

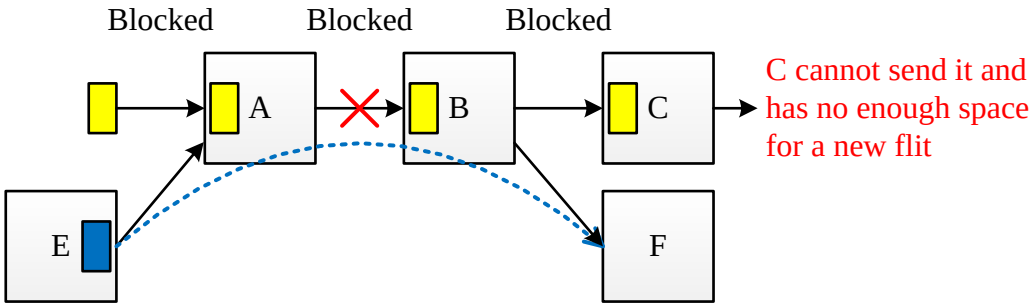


圖 2-19　蟲洞流量控制中通道阻塞問題範例

2.3.5 虛擬通道流程控制

HoL 與鏈結通道阻塞問題的發生，主要原因皆歸咎於路由器中的每一個輸入埠皆只有單一個緩衝佇列造成。蟲洞流量控制可搭配虛擬通道以減緩 HoL 與鏈結

通道阻塞問題產生機率，為目前使用最多的流量控制方法。虛擬通道實際上是在路由器輸入緩衝器中保留多個緩衝佇列，而一個緩衝佇列被視為一個虛擬通道緩衝器，且多個虛擬通道緩衝器彼此間共享相同的實體鏈結/鏈結頻寬於兩個比鄰路由節點之間，故每一個單向虛擬通道可藉由兩個路由器之間獨立管理的緩衝佇列對來實現，如圖 2-20 所示。而如圖 2-19 所描述的鏈結通道阻塞情況發生時，藍色 flit 仍然可透其它虛擬通道來穿越實體鏈結傳輸資料，因此資料傳輸阻塞的機率將大為降低。

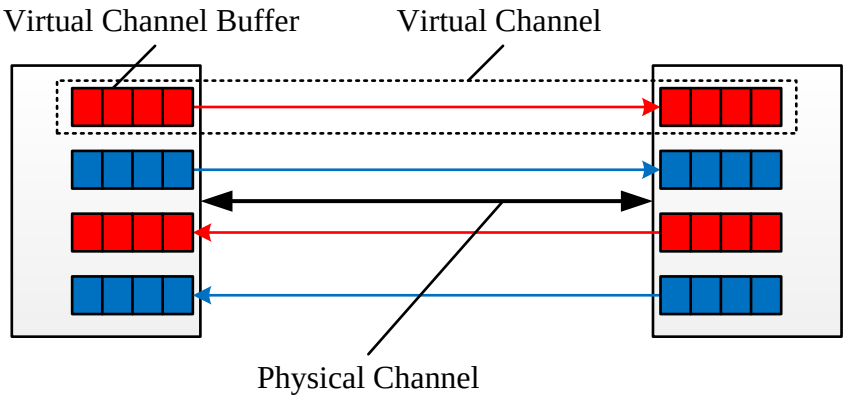


圖 2-20　一個實體通道被分為兩個(紅色和藍色)虛擬通道

由上述可得知，增加虛擬通道除提高了路由器的節點度，可用於避開網路中的壅塞節點和失效節點外，另一方面，虛擬通道的設置增加了實體鏈結的使用率，從而提升全局網路的資料傳輸率。但一味的增加虛擬通道數量將會消耗更多的晶片上資源，同時增加虛擬通道控制電路的複雜度，降低每一個虛擬通道實際分配到的頻寬。針對 2D mesh 拓樸網路而言，[36]指出虛擬通道數量超過 4 個以後，對其效能的提高並不明顯。

2.3.6 通道緩衝器管理

對於所有流量控制方法，其發送端(上游路由節點)皆必須藉由一通道緩衝器管理機制與接收端(下游路由節點)溝通以得知是否有有效的緩衝區空間用以儲存資料。倘若接收端無可用緩衝區空間，則發送端必須停止傳送資料以防止緩衝區溢出(buffer overflow)，導致資料的遺失。通道緩衝器管理的計數單位取決於所使用的流量控制方法；儲存轉發和虛擬切通流量控制以封包為緩衝器計數管理單位，而蟲洞和虛擬通道流量控制以 flit 為緩衝器計數管理單位。

兩個普遍被使用的通道緩衝器管理機制為 on/off management 和 credit-based management [37]：

1. On/off management：為最簡易的緩衝器管理方法。如圖 2-21 所描述，當發送端不斷傳送 packer/flit 至接收端，導致接收端的有效緩衝器數量減少達到一個臨界值 Xoff 時，則觸發一個 *off* 訊號至接收端，告知發送端應停止傳送 packet/flit。臨界值 Xoff 的設定必須保證 *off* 訊號在經傳輸延遲後且被發送端接收的這段時間中，發送端持續進行傳送的 packet/flit 在抵達接收端後將有緩衝區可使用。

   而當接收端緩衝區可再接納新的資料時且有效緩衝器數量增加至一個臨界值 Xon 時，則觸發一個 *on* 訊號至發送端，通知繼續進行 packet/flit 的傳送。臨界值 Xon 的設定必須確保 *on* 訊號經傳輸延遲後被發送端接受，且傳送新的 packet/flit 抵達接收端的這段時間中，發送端緩衝區中仍然有 packet/flit 可持續進行傳送。

   為保證 Xoff/Xon 的設定滿足上述條件，on/off management 有最小緩衝器容量的限制，其計算涉及鏈結長度、發送端觸發 *off* 訊號的 overhead 和接收端接收該 *off* 訊號並停止傳送 packet/flit 的 overhead，故增加了實現上的複雜度。

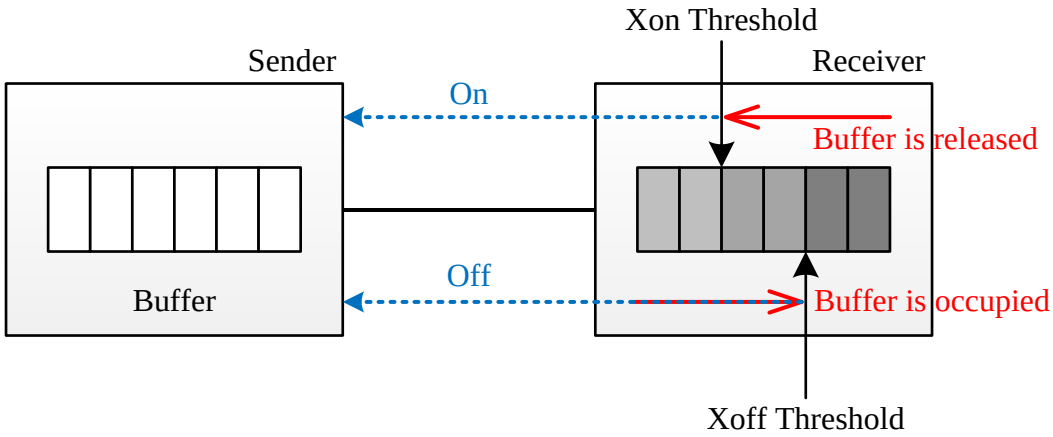


圖 2-21　On/off management

2. Credit-based management：發送端利用一個 credit count 持續追蹤接收端有效的緩衝器數量。如圖 2-22 描述，接收端在釋放本身一個緩衝器的同時，發出一個相應的 credit 至發送端，發送端接收到此 credit 後便自動增加其 credit count。

發送端可依據 credit count 來傳輸相同數量的 packet/flit，且在傳送一個 packet/flit 後，減少本身一個對應的 credit。若 credit count 為零，表示接收端所有緩衝器皆已滿且發送端無法再傳送任何資料，直至從接收端接收一個新的 credit 為止。

Credit-based management 可使通道緩衝器獲得善用，且於實現方面不需考量實體鏈結長度或發送端/接收端的 overhead。Credit-based management 在兩相鄰路由器間，對於每一個緩衝佇列須存在著 $log_2(B)$條反向訊號線用作 credit count 編碼，其中 B 表示佇列中緩衝器數量，故相對於 on/off management 的每一個緩衝佇列只需要單一條反向訊號線而言，credit-based management 較多的額外接線開銷為其主要缺點。

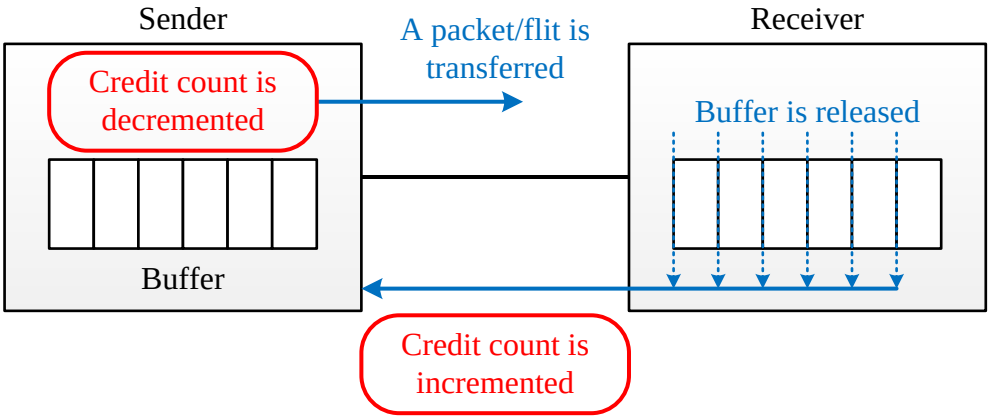


圖 2-22　Credit-based management

第三章 緩衝器合併演算法之路由器設計

在給定網路節點連接拓樸結構和映射(mapping)條件下，本論文所提出的緩衝器合併演算法可根據不同路由路徑上的資料流量負載情況，對每一個路由器的輸入緩衝器單元(input buffer unit)進行合併計算，從而給出最終的緩衝器配置方案。該合併演算法所需的環境配置及參數描述如下。

3.1 問題定義

1. 已知條件：

(1) 系統任務圖(Task Graph, TG)：一個特定 SoC 應用電路，其各個 SIP 之間的通訊可以藉由一有向(directed)任務圖的形式表示，圖 3-1 為一個視訊物件平面解碼器(Video Object Plane Decoder, VOPD)任務圖[38]。假設該任務圖以 $G(T, E)$作為表示，其中 T 為系統元件標號集合，每一個頂點(vertex) $t_i \in T$ ，而 t_i 表一個選擇的 SIP i 標號。E 為資料通訊集合，在 $t_i, t_j \in T$ 之間的有向邊(directed edge)以 $e_{i,j} \in E$ 表示之，$e_{i,j}$ 代表 SIP t_i 和 SIP t_j 之間的通訊，且每一條邊 $e_{i,j}$ 皆擁有權重(weight)，記為 $comm_{i,j}$ ，其顯示從 t_i 到 t_j 所需的通訊頻寬，圖 3-1 中資料傳輸頻寬以 Mbytes/sec 為單位。

(2) 拓樸網路結構：本論文所選擇拓樸結構為 2D mesh，且其同樣可定義為一個有向圖，如圖 2-1 為具 16 個節點之 2D mesh 拓樸結構。假設該圖以 $T(C, R, L)$ 表示之，其中每一個頂點 $r_i \in R$ ，表示拓樸當中的一個路由節點，且每一條有向邊 $l_{i,j} \in L$ ，表示從 r_i 到 r_j 的實體鏈結，使得 $r_i, r_j \in R$ 。同樣地，C 表示一個 core(SIP)且 $c_i \in C$ ，其透過一個 NIU 連接至 r_i 且 $|C| = |R|$ ，即 C 與 R 之間存在著 one by one 的對應關係。鏈結 $l_{i,j}$ 的權重記為 $bw_{i,j}$ ，顯示鏈結 $l_{i,j}$ 所提供的有效頻寬，其可由式(1)計算得知。

$$bw_{i,j} = phit\_size \times freq / 8 \quad \text{(MBps)} \tag{1}$$

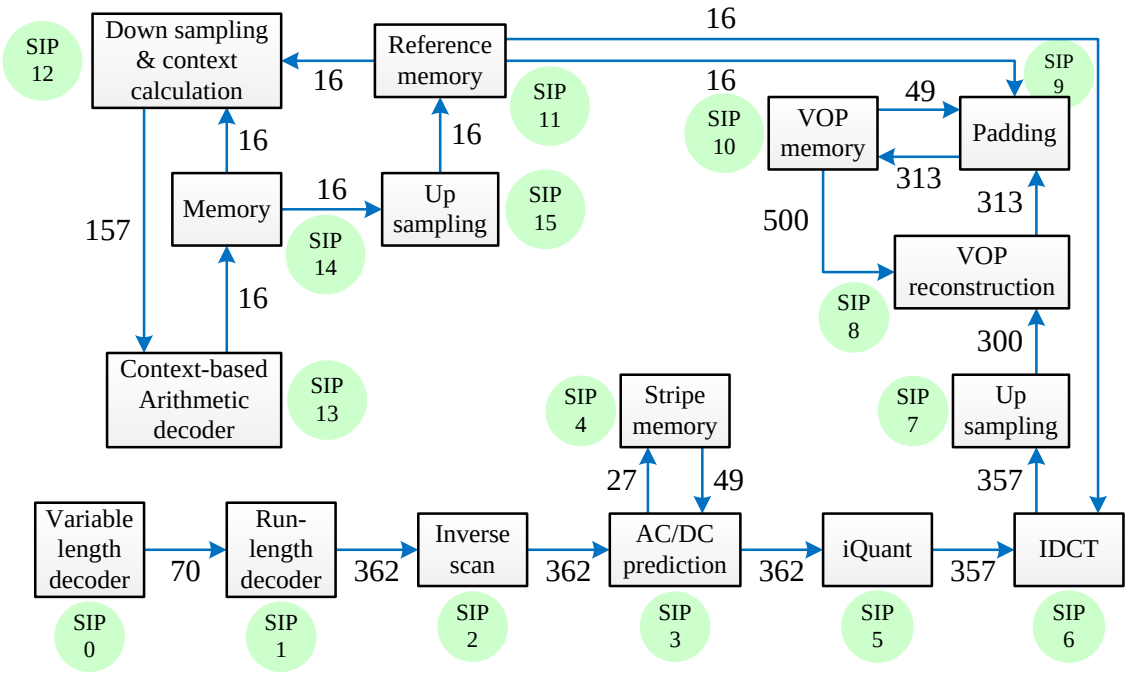


圖 3-1　Video Object Plane Decoder 任務圖

2. 定義：

(1) 操作頻率(operating frequency)：係指 NoC 系統的工作頻率，其設定為經合成(synthesis)過程後電路可允許之最大時脈頻率。

(2) 映射：映射意謂按照一定的最佳化規則將 $G(T, E)$指定的任務分配到目標拓樸結構 $T(C, R, L)$上，從而實現該特定應用。更確切的地說，映射決定哪一個 core 節點應該被所選定的 SIP 所映射，而最佳化的映射目標可能為功率消耗、資料傳輸延遲或鏈結使用率等多方面系統優化。當$|T| \le |C|$且被定義為 one by one 的映射形式時，任務圖 $G(T, E)$映射至 NoC 拓樸結構 $T(C, R, L)$可藉由一函式 *map* 描述：

$$map: T \to C,\ \text{s.t. } map(t_i) = c_j,\ \forall t_i \in T,\ \exists c_j \in C \tag{2}$$

在映射的過程，本論文分別採用隨機(random)映射及 PERMAP[38]映射演算法以進一步探究映射方式的好壞差別對於緩衝器合併演算法的影響。其中，PERMAP 演算法利用佇列模型有效地對 SoC 進行效能預測與分析，進而使 SIP 映射至 NoC 拓樸結構上，以達到 SIP 之間的平均傳輸延遲能夠最小化。PERMAP 演算法在執行上快速且穩定，且對於不同的拓樸結構和路由演算法，PERMAP 演算法同樣能為其提供高品質解決方案。

3. 參數計算：

在完成映射的執行後，每一個 SIP 在 NoC 拓樸結構上的位置則已確定。因此，對於本論文提出的緩衝器合併演算法所需的Φ矩陣參數即可進行計算獲得。

對於任何已知的路由器，Φ矩陣表示穿越該路由器的資料流通率(traffic flowing rate)，而Φ矩陣中每一個元素 λ_{ij} 則代表從該路由器的輸入埠 i 到輸出埠 j 的資料流量。以圖 3-1 為例，其通訊頻寬負載為已知，故 λ_{ij} 可藉由式(3)計算得出：

$$\lambda_{ij} = \sum_{f_c \in E} \lambda(f_c) R(f_c, i, j) \tag{3}$$

在式(3)，對於圖 3-1 中每一個 source-destination pair(即每一條邊 $e_{i,j} \in E$)皆被視為一個 flow，且表示為 f_c，$c = 1, 2, \cdots |E|$，其值代表該 flow 之通訊頻寬並以 $\lambda(f_c)$ 表示之。所有的 flow 可透過函式 D 完整的表達：

$$D = \left\{ \begin{array}{l} f_c : \lambda(f_c) = comm_{i,j},\ c = 1, 2, \cdots |E|,\ \forall e_{i,j} \in E \\ \text{with } source(f_c) = map(t_i),\ dest(f_c) = map(t_j) \end{array} \right\} \tag{4}$$

而式(3)中，R 代表所採用的路由演算法，其定義如下：

$$\begin{array}{l} R(f_c, i, j),\ f_c \in E,\ i, j \in \{E, W, S, N, L\} \\ \left\{ \begin{array}{l} 1,\ \text{if flows from port } i \text{ to port } j \\ 0,\ \text{otherwise} \end{array} \right\} \end{array} \tag{5}$$

因此，式(3)表示將經過該路由器的輸入埠 i 到輸出埠 j 的所有資料流量相加總，故對於 NoC 中每一個路由器，其矩陣Φ可完整描述如式(6)。其中，因本論文在實驗模擬過程使用 X-Y 路由演算法屬最短路徑路由演算法，所以在Φ矩陣中的元素 λ_{ii} 恆為 0，意即任何方向的輸入埠不會傳送資料至相同方向的輸出埠。另外，於Φ矩陣左手邊的 L、N、E、S 和 W 標籤代表輸入埠的方向，而上方的標籤則代表輸出埠的方向。

$$\Phi = \begin{array}{c} \\ \mathrm{L} \\ \mathrm{N} \\ \mathrm{S} \\ \mathrm{E} \\ \mathrm{W} \end{array} \begin{array}{c} \begin{array}{ccccc} \mathrm{L} & \mathrm{N} & \mathrm{S} & \mathrm{E} & \mathrm{W} \end{array} \\ \begin{bmatrix} \lambda_{LL} & \lambda_{LN} & \lambda_{LS} & \lambda_{LE} & \lambda_{LW} \\ \lambda_{NL} & \lambda_{NN} & \lambda_{NS} & \lambda_{NE} & \lambda_{NW} \\ \lambda_{SL} & \lambda_{SN} & \lambda_{SS} & \lambda_{SE} & \lambda_{SW} \\ \lambda_{EL} & \lambda_{EN} & \lambda_{ES} & \lambda_{EE} & \lambda_{EW} \\ \lambda_{WL} & \lambda_{WN} & \lambda_{WS} & \lambda_{WE} & \lambda_{WW} \end{bmatrix} \end{array} \tag{6}$$

由於本論文欲合併的緩衝器單元處於路由器輸入端，因此只須考量計算路由器各方向輸入通道的資料流量負載，其值的計算對於單一條最短路由路徑而言，可藉由式(7)獲得一參數 $x_{i,j}^{c}$，並由式(8)計算出該輸入通道的資料流量負載。以圖 3-2 為例，北方輸入通道的資料流量負載 N_{in} 可由 λ_{NE}、λ_{NW}、λ_{NS} 和 λ_{NL} 相加總得出。以此類推，可獲得其它方向輸入通道 L_{in}、S_{in}、E_{in} 和 W_{in} 各值。對於 NoC 中每一個路由器而言，其各方向輸入通道的資料流量負載將可藉由一陣列 $Array_k$ 表示之，如式(9)。

$$x_{i,j}^{c} = \begin{Bmatrix} \lambda(f_c), \text{ if } l_{i,j} \in path\big(source(f_c),\ dest(f_c)\big) \\ 0 \quad , \text{otherwise} \end{Bmatrix} \tag{7}$$

$$Dir_{in} = \sum_{c=1}^{|E|} x_{i,j}^{c},\ Dir \in \{L, N, S, E, W\},\ \forall i, j \in 1, 2, \cdots |R| \tag{8}$$

$$Array_k \left[L_{in}, N_{in}, S_{in}, E_{in}, W_{in} \right],\ k = 1, 2, \cdots |R| \tag{9}$$

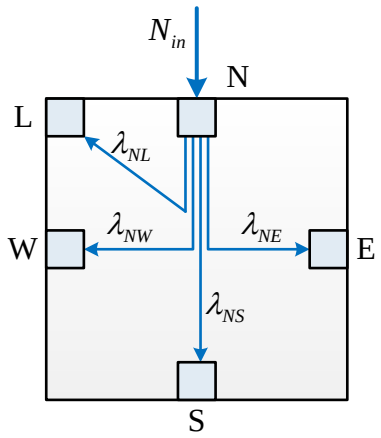


圖 3-2　輸入埠之資料流量負載計算範例

3.2 緩衝器合併演算法

對於上述 Dir_{in}，實際上存在著一個頻寬約束(bandwidth constraint)條件，其可由不等式(10)表示之，而該限制條件正是用以執行緩衝器合併的重要判斷依據。

$$Dir_{in} \le bw_{i,j},\ Dir \in \{L, N, S, E, W\},\ \forall i, j \in 1, 2, \cdots |R| \tag{10}$$

圖 3-3 為本論文所提出的緩衝器合併演算法執行流程，採用 C 語言實現。該演算法執行過程描述如下：

1. 針對上一節已知條件中的系統任務圖 $G(T, E)$和拓樸網路 $T(C, R, L)$分別以隨機方式及 PERMAP 演算法進行映射，並決定出路由演算法 R。

2. 在完成映射的 NoC 系統中，對於每一個路由器的輸入埠 i，根據路由演算法 R 對所有的 flow 經過該輸入埠 i 的資料流量負載進行相加總，可得陣列 $Array_k$。

3. 根據式(1)以計算 NoC 中每一條鏈結 $l_{i,j}$ 所能提供的有效頻寬 $bw_{i,j}$。

4. 進行頻寬約束條件的判斷並接續執行緩衝器單元的合併，詳細演算過程如圖 3-4 所描述。

   (1) 合併開始以陣列 $Array_k$ 及有頻寬約束條件 *bw* 作為輸入資料，並宣告所需變數及陣列。其中，*flag_ar* 陣列和 *dir_ar* 陣列分別用以判斷 $Array_k$ 中各個元素合併狀態與否及合併後各方向的判定；*size* 變數代表 $Array_k$ 中存在元素個數，其值相等於路由器輸入緩衝器單元數量。

   (2) 該演算法以函式 merge 為主體，執行上首先呼叫函式 find_min_two，該函式於 *flag_ar*[*i*]為 1(表示尚未合併)的條件下，尋找 $Array_k$ 中具最小兩個值的元素 *min1* 和 *min2*，並將相對應之迴圈數值 *i* 分別指定於 *flag1* 和 *flag2*。

   (3) 執行 31 至 33 行的條件判斷，因為該三個判斷式中包含有一值大於等於 *bw* 的判斷，因此若判斷條件成立後即跳出迴圈完成 $Array_k$ 的合併動作；若條件不成立則將 *min1* 和 *min2* 相加總並指定於變數 *sum*，而後判斷該 *sum* 值是否小於等於 *bw*。

   若 *sum* 小於等於 *bw*，則進行 36 至 39 的合併描述，其中會呼叫函式 find_min_one 以找尋在 *flag_ar*[*i*]為 1 條件下 $Array_k$ 的最小值作為下一回合

的合併選擇；若 *sum* 大於 *bw*，則跳出迴圈完成演算法的執行。

(4) 函式 find_min_one 執行完畢後，函式 merge 將進入下一迴圈並從第 42 行開始執行條件判斷，該判斷式中同樣包含有一值大於等於 *bw* 的判斷，若條件成立則跳出迴圈完成合併動作；若條件不成立則將 *sum* 和 *min1* 相加總並判斷其值是否小於等於 *bw*。

若 *sum* 和 *min1* 相加值小於等於 *bw* 則執行 45 至 47 行的合併描述且再次呼叫函式 find_min_one 尋找下回合的合併值；若 *sum* 和 *min1* 相加值大於 *bw*，則呼叫 find_min_two 尋找 $Array_k$ 未合併的兩個最小值作為下一回合的合併判斷。

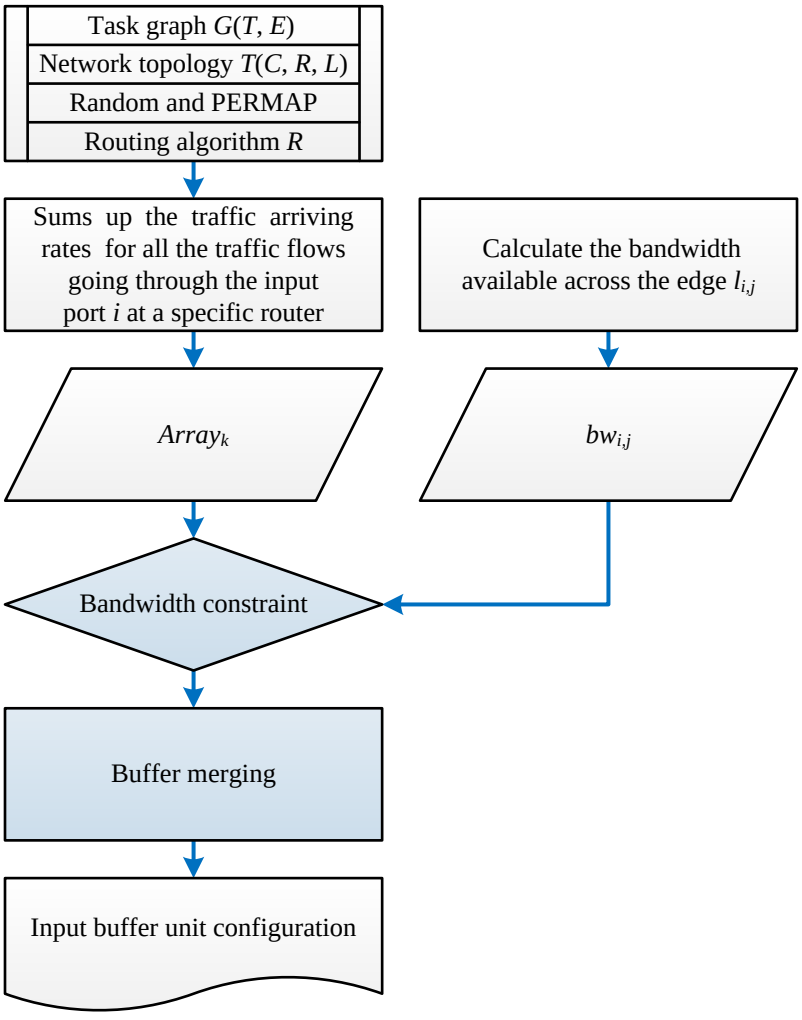


圖 3-3　緩衝器合併演算法流程圖

```
1.  int bw = available bandwidth across each link      // obtained by Eq.(1)
2.  int Array_k[]  = {L_in, N_in, S_in, E_in, W_in}    // obtained by Eq.(9)
3.  int flag_ar[] = {1, 1, 1, 1, 1}
4.  char dir_ar[]= {L, N, S, E, W}
5.  size = size of Array_k
6.  int min1 = 0, min2 = 0
7.  int flag1 = 0, flag2 = 0
8.  void find_min_two (void) {
9.     for i = 0 to size-1 do {
10.      if (flag_ar[i] == 1) {
11.        find out two minimum values of the Array_k
12.        assign the minimum values to min1 and min2 respectively
13.        assign the i values that corresponding to min1 and min2 to flag1 and flag2 respectively
14.      }
15.    }
16. }
17. void find_min_one (void) {
18.    for i = 0 to size-1 do {
19.      if (flag_ar[i] == 1) {
20.        find out minimum value of the Array_k and then assign to min1 in addition to once merge
21.        assign the i value that corresponding to min1 to flag1
22.      }
23.    }
24. }
25. void merge (void) {
26.    int sum = 0
27.    int buf_units = size
28.    find_min_two()
29.    for i = 0 to 10 do{
30.      if (find_two_min_value) {
31.        if (min1 >= bw) buffer units can't merging; break;
32.        if (min2 >= bw && min1 == 0) dir_ar[flag1] and dir_ar[flag2] can merging; break;
33.        if (min2 >= bw && min1 != 0) buffer units can't merging; break;
34.        sum = min1 + min2
35.        if (sum <= bw) {
36.          buf_units = buf_units – 1    // dir_ar[flag1] and dir_ar[flag2] can merging
37.          flag_ar[flag1] = 0
38.          flag_ar[flag2] = 0
39.          find_min_one()
40.        } else break;
41.      } else {
42.        if (sum == 0 && min1 >= bw) buf_units = buf_units – 1; break;
             // dir_ar[flag1] can merge with the buffer unit that previous to merge
43.        sum = sum + min1
44.        if (sum <= bw) {
45.          buf_units = buf_units – 1
46.          flag_ar[flag1] = 0
47.          find_min_one()
48.        } else {    // dir_ar[falg1] can't merging
49.          find_min_two()
50.        }
51.      }
52.    }
53. }
```

圖 3-4　緩衝器合併演算法

3.3 路由器結構設計

在有限面積和功率限制條件下，設計一個滿足傳輸延遲和資料傳輸率需求的路由器，是現今多核心系統(many-core system)規模尺寸縮小趨勢下主要面臨的挑戰。路由器結構複雜程度隨著頻寬需求而增加。當資料高傳輸率需求不被要求時，路由器的建構可更加簡單化，故可實現較低的面積成本和功率消耗；反之，降低 NoC 的傳輸延遲和資料傳輸率需求表現時，路由器設計上的挑戰儼然而生。

路由器結構由記憶體、控制邏輯、功能單元和交叉開關等關鍵元件所構成，這些元件共同實現路由選擇、流量控制和路由器管線化功能，主導著緩衝器和鏈結頻寬的使用，因而影響全局網路資料傳輸率。路由器結構亦決定了其關鍵路徑(critical path)的延遲，更深入影響 per-hop 及整體網路延遲時間。除此，路由器結構關係著其本身內部電路元件的行為模式，對於網路功率消耗有直接影響，且晶片面積成本更明顯地取決於路由器架構的選擇與設計。

在此章節，介紹應用於 NoC 中最典型的虛擬通道路由器結構和本論文所提出的具緩衝器合併演算法之虛擬通道路由器結構並說明其工作原理及設計。

3.3.1 典型虛擬通道路由器結構

圖 3-5 顯示一個基於 2D mesh 拓樸，採用蟲洞流量控制搭配 credit-based 通道緩衝器管理機制的典型虛擬通道路由器區塊連接結構，因此該路由器(以 mesh 拓樸結構中非邊緣路由節點為例)具有五個輸入埠和五個輸出埠，且分別對應連接至四個鄰近路由節點方向及 local 的計算節點[39]。此路由器主要組成元件包括：輸入緩衝器單元、路由計算(Routing Computation, RC)單元、虛擬通道配置(Virtual-channel Allocation, VA)邏輯、開關配置(Switch Allocation, SA)邏輯和交叉開關。在此，假設一個輸入緩衝器單元具有四個虛擬通道緩衝器且對應至一個輸入埠，且每一個虛擬通道為 4 個 flit 寬度的 FIFO 佇列。另外，圖 3-5 中 P_i 表示輸入埠編號；P_o 表示輸出埠編號；V_i 表示輸入緩衝器單元編號，而 v_i 表示緩衝器單元 V_i 內虛擬通道緩衝器編號。

在所有 flit 抵達路由器輸入埠(P_i)後，虛擬通道識別器(virtual channel identifier)將對其虛擬通道識別(VCID)欄位進行解碼，而後，該 flit 將被緩衝儲存於相應的輸入虛擬通道佇列(v_i)中。當資料封包的 head flit 完整儲存輸入虛擬通道(v_i)後，欲開始傳送該 head flit，路由器必須獲取下列兩項資訊[40]：

(1) 輸出埠(P_o)：head flit 欲傳輸的輸出埠編號 P_o。

(2) 輸出虛擬通道(v_o)：head flit 欲傳輸的輸出埠 *Po* 之虛擬通道編號 v_o。輸出虛擬通道(v_o)實質上為鄰接路由器的輸入虛擬通道(v_i)。

虛擬通道路由器的基本工作原理及步驟如下所述。

1. RC 單元：主要負責引導該 head flit 至適當的輸出埠(P_o)，藉由對封包的目的地欄位進行檢查，並且回覆適當的 P_o 值和有效的 v_o 值以建立傳輸路徑。其 P_o 值和 v_o 值的產生取決於所採用的路由演算法。RC 單元的執行屬於 per-packer 的操作模式，對於每一個資料封包僅僅執行一次計算，即只對封包的 head flit 進行計算動作。

2. VA 邏輯：經過路由計算階段後，該 head flit 所在的輸入虛擬通道(v_i)將對 VA 邏輯提出輸出虛擬通道(v_o)的使用要求。VA 邏輯主要為配置一個有效的輸出虛擬通道(v_o)給予 head flit 傳輸使用，其配置動作涉及仲裁機制，對於至少兩個以上的 head flit 皆請求使用相同輸出虛擬通道(v_o)時做出仲裁行為，並決定出唯一授權使用者。VA 邏輯的執行同樣屬於 per-packet，僅對封包的 head flit 進行輸出虛擬通道(v_o)的配置。

3. SA 邏輯：當該 head flit 成功配置了輸出虛擬通道(v_o)且藉由 credit count 得知該輸出虛擬通道(v_o)的緩衝空間為有效時，其所在的輸入虛擬通道(v_i)便向開 SA 邏輯提出使用輸出埠(P_o)的請求。SA 邏輯主要功能為配置目標輸出埠(P_o)予以 flit 傳輸使用，其配置過程中除提供交叉開關行為動作的控制訊號以建立輸入虛擬通道(v_i)至輸出埠(P_o)之間的路由器內部傳輸路徑外，還須解決可能的衝突，包括[40]：

   (1) 輸入端的衝突：在每一個時脈週期，一個輸入緩衝器單元(V_i)中只有一個虛擬通道佇列(v_i)可以進行資料傳輸。

   (2) 輸出端的衝突：在每一個時脈週期，一個輸出埠(P_o)只可以從一個輸入虛擬通道佇列(v_i)中接收資料。

   SA 邏輯的仲裁機制除對所有競爭的輸入虛擬通道(v_i)必須給予平等的機會進行資料傳輸外，同時也必須致力於實體輸出通道的最大使用率以獲得較高的網路傳輸率。另外，SA 邏輯的執行屬於 per-flit 操作模式，即配置的對象為欲透過該路由器進行傳輸的所有 flit，不僅僅是封包的 head flit。

4. 交叉開關：在 SA 階段配置成功後，head flit 便獲准通過交叉開關至正確的輸出埠(P_o)並且向欲傳輸的輸出虛擬通道(v_o)的路由節點進行傳送。同時，flit 的 VCID 欄位將被更新為輸出虛擬通道(v_o)。在 head flit 離開所在的輸入虛擬通道(v_i)後，同屬於相同封包的 body flit 和 tail flit 可從 SA 階段開始進行傳輸。一旦當 tail flit 獲准通過該路由器的交叉開關後，則 VA 邏輯同時釋放相應之輸出虛擬通道(v_o)的使用權。

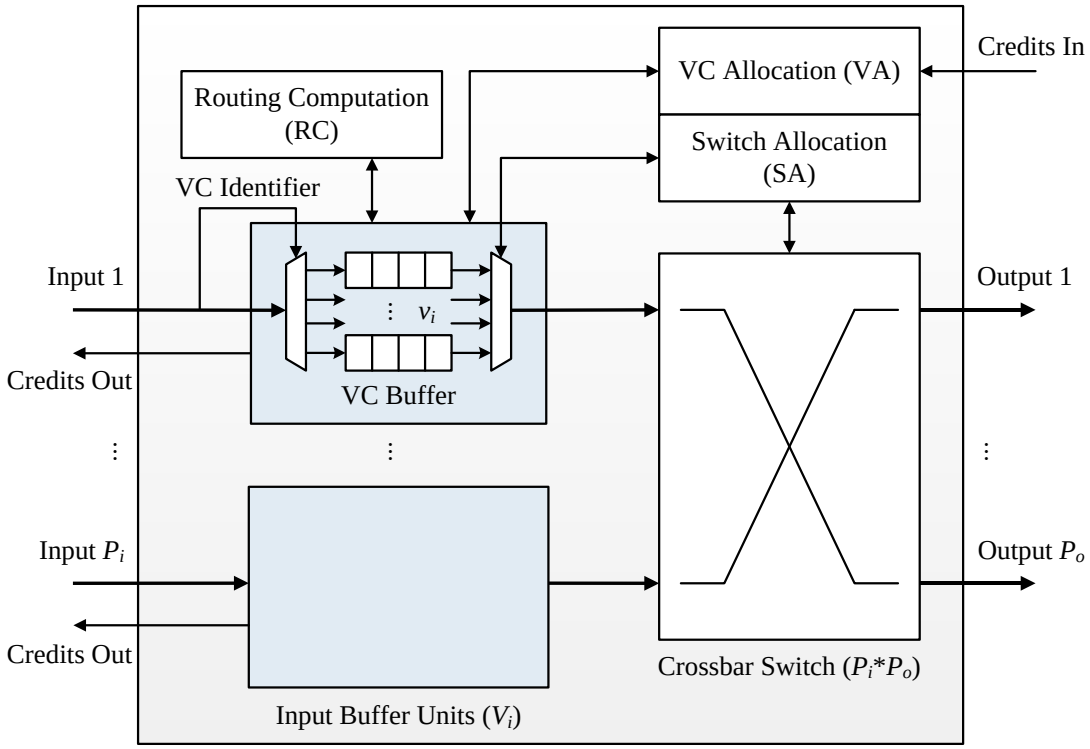


圖 3-5　虛擬通道路由器區塊結構

3.3.2 路由器管線

管線技術的支持，可提高路由器各模組電路的使用率，進而提升路由器執行效率。圖 3-6 為一個基本蟲洞流量控制具有虛擬通道的路由器管線化範例。該路由器管線化於 flit 階層。當每一個封包欲從路由器輸入埠傳送至輸出埠，其 head flit 必須經過緩衝器寫入(Buffer Write, BW)、RC、VA、SA、交換器(交叉開關)穿越(Switch Traversal, ST)和鏈結穿越(Link Traversal, LT)階段，而這些階段期間將依據實際時脈頻率(圖 3-6 假設每一個階段花費一個時脈週期)適當地嵌入一個管線化通道，即每個階段皆有自己的管線通道。

在封包的 head flit 抵達輸入埠後，首先被解碼且根據其 VCID 緩衝儲存於 BW

管線階段(cycle 1)。緊接著，執行 RC 步驟來決定適當的輸出埠(cycle 2)。接著，head flit 接受其輸出虛擬通道的配置與仲裁於 VA 管線階段(cycle 3)。而後，如果成功配置了輸出虛擬通道，head flit 前進至 SA 管線階段(cycle 4)，此階段執行交叉開關的輸出/輸入埠匹配。一旦獲得了輸出埠使用權後，head flit 從輸入虛擬通道中被讀取出並進行 ST 管線階段(cycle 5)來穿越交叉開關。最後，head flit 執行 LT 階段(cycle 6)，藉由輸出鏈結通道傳送至下一個路由節點。

緊接 head flit 之後，除了 RC 和 VA 階段，body flit 和 tail flit 皆遵循與 head flit 相同的管線階段，因為它們繼承了 head flit 所建立的路由路徑及輸出虛擬通道的配置。最後，當 tail flit 離開路由器，由 head flit 所建立的輸出虛擬通道配置將被解除。

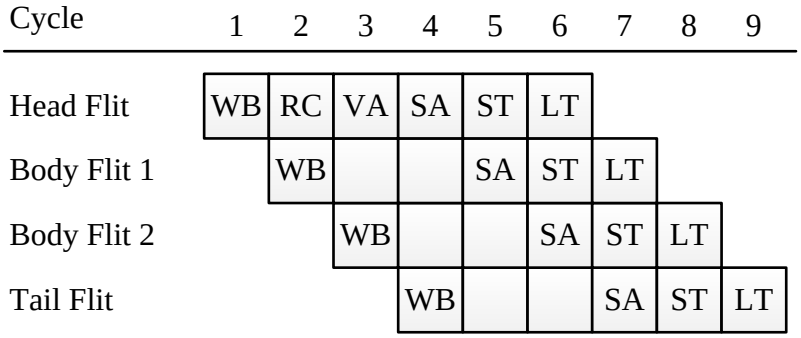


圖 3-6　基本蟲洞流量控制具有虛擬通道的路由器管線化範例

3.3.3 具緩衝器合併演算法之路由器設計

為了符合本論文所提出的緩衝器合併演算法運算結果，必須對上述典型虛擬通道路由器結構做適當修改以確保其能夠正常運作。圖 3-7 為基於 2D mesh 拓樸且具緩衝器合併之虛擬通道路由器塊結構範例。其中，P_i 表示輸入埠編號；P_o 表示輸出埠編號；V_i 表示輸入緩衝器單元編號，而 v_i 表示緩衝器單元 V_i 內虛擬通道緩衝器編號。

由圖 3-7 可得知，該路由器結構在經過修改後，輸出/輸入埠的數量並無增減，且主要組成元件與典型虛擬通道路由器並無太大差異，主要差別在於輸入緩衝器單元的使用個數。以圖 3-7 為例，經緩衝器合併演算法計算後，原本編號 1、2、3 的輸入埠各自對應的輸入緩衝器單元(V_1、V_2、V_3)被合併成同一個單元(假設編號為 V_1)，且資料的輸入藉由一個多工器作選擇性輸入。換言之，源自編號 1、2、3 輸入埠的資料將共享輸入緩衝器 V_1 內虛擬通道緩衝器 v_i 的使用，如此即可大幅度減

少緩衝器使用數量，節省晶片面積及功率消耗。

除此之外，為配合輸入緩衝器單元合併前後的個數不一情況，其 VA 邏輯、SA 邏輯及交叉開關的設計規格也需一併進行調整，此部分在以下章節詳細說明。

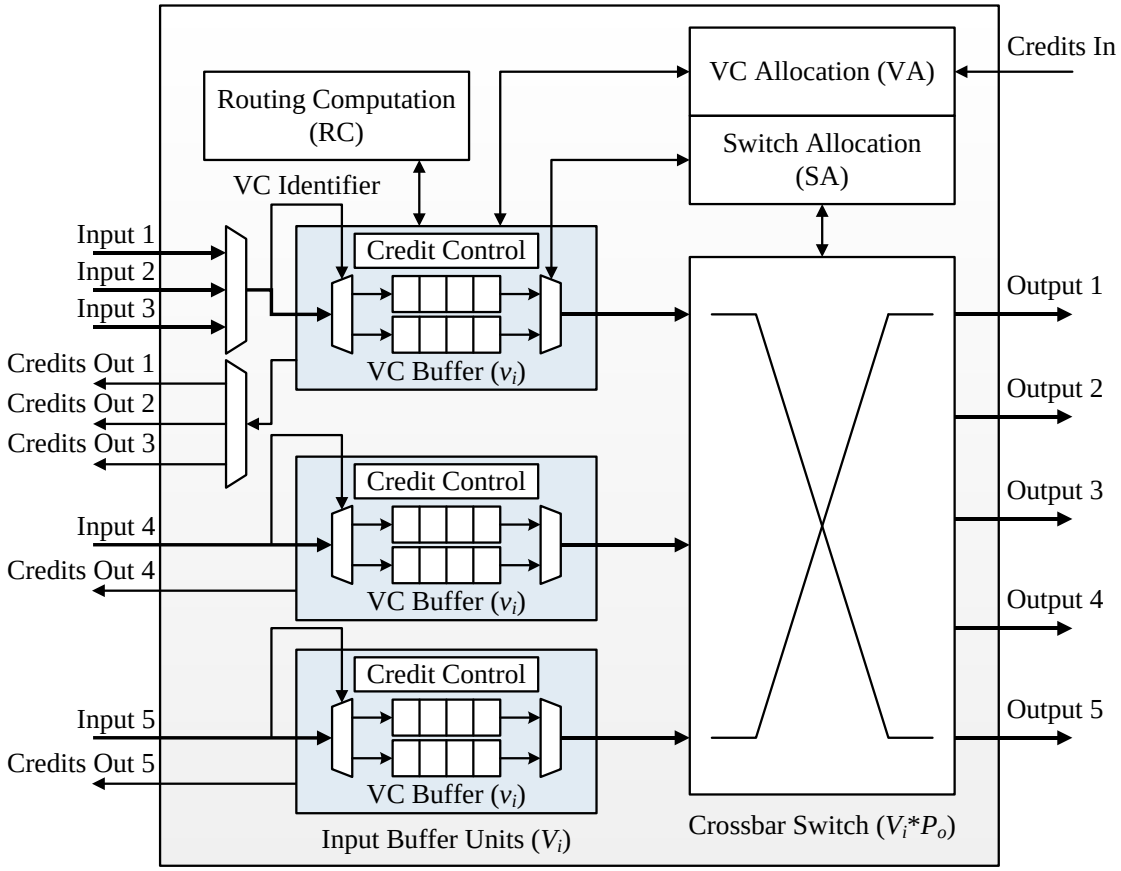


圖 3-7　具緩衝器合併之虛擬通道路由器區塊結構範例

3.3.4 具緩衝器合併之虛擬通道配置邏輯設計

VA 邏輯的設計複雜度依據 RC 單元所回覆的輸出虛擬通道(v_o)數量範圍有直接關係。由於本論文採用 X-Y 路由演算法，RC 單元將只回覆單一特定輸出埠(P_o)，但對於該特定輸出埠(P_o)之輸出虛擬通道，則採用有效者即予以回覆。為簡化實現複雜度，兩個階段的仲裁設計為所需要，如圖 3-8 所描述。第一階段仲裁(VA1)將每一個輸入虛擬通道(v_i)對於有效輸出虛擬通道(v_o)的請求減至為一個，如此可以保證該特定輸出埠(P_o)的單一個輸出虛擬通道皆可被每一個輸入虛擬通道(v_i)所請求。因此在 VA1 將需要 V_iv 個 v:1 仲裁器(arbiter)，v 表示緩衝器單元 V_i 內虛擬通道緩衝器 v_i 的數量。換句話說，每一個輸入虛擬通道(v_i)即需要一個 v:1 仲裁器。其中，

V_i 的具體個數在運行緩衝器合併演算法後將呈現不盡相同的數量於網路中所有路由器，如五個輸入埠對應四個輸入緩衝器單元亦或五個輸入埠對應三個輸入緩衝器單元，甚至是五個輸入埠對應一個輸入緩衝器單元等情況皆可能發生。在獲得 VA1 仲裁勝利的輸入虛擬通道(v_i)之請求得以繼續進行至第二階段仲裁(VA2)。VA2 將需要 $V_o v$ 個 V_iv:1 仲裁器，即每一個輸出虛擬通道(v_o)需要一個仲裁器。其中，V_o 為輸出緩衝器單元編號，意同鄰接路由器的輸入緩衝器單元編號(V_i)，且因為路由器本身的五個實體輸出通道所連接對應到的輸出緩衝器單元分處鄰近不同的路由器，故 V_o 的具體各數固定為五個(以 mesh 拓樸結構中非邊緣路由節點為例)。此外，VA2 階段的仲裁器規格為 V_iv:1，主要是考量所有輸入虛擬通道(v_i)可能皆請求相同的輸出虛擬通道(v_o)情況而所設計。

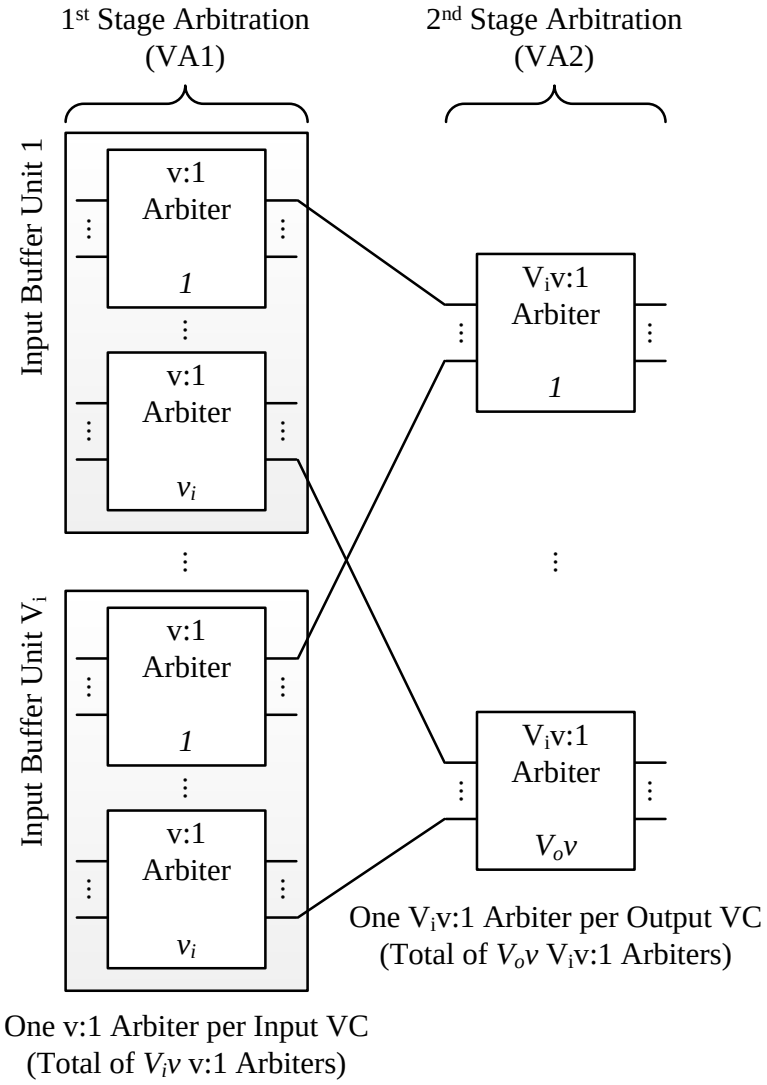


圖 3-8　具緩衝器合併之虛擬通道配置邏輯設計

3.3.5 具緩衝器合併之開關配置邏輯設計

SA 邏輯對於所有輸入虛擬通道佇列(v_i)內的資料欲經由交叉開關傳送至目標輸出埠(P_o)的請求進行仲裁並授予權限。此 SA 邏輯的設計同樣實現於兩個仲裁階段，如圖 3-9。因為單一輸入緩衝器單元(V_i)中的所有虛擬通道佇列(v_i)皆可能請求相同目標輸出埠(P_o)，故彼此之間勢必會產生競爭行為。因此，第一階段仲裁(SA1)對相同輸入緩衝器單元(V_i)內欲存取相同輸出埠(P_o)之虛擬通道佇列(v_i)彼此之間的相互競爭作出裁決並決定出唯一授權者。緊接著，所有輸入緩衝器單元(V_i)中的各個被授權者將進行第二階段的仲裁(SA2)，並於此階段決定出該輸出埠實際唯一的使用者。此 SA 邏輯的設計非常相似於 VA 邏輯，唯一不同的是仲裁器的使用個數及其規格。SA1 需要 V_i 個仲裁器，即每一個輸入緩衝器單元(V_i)需要一個 v:1 仲裁器；SA2 則需要 P_o 個 V_i:1 規格的仲裁器，即每一個輸出埠(P_o)必須搭配一個仲裁器，其中，V_i:1 的仲裁器規格為考量各個輸入緩衝器單元(V_i)中經 SA1 決定出的被授權者，其欲存取的輸出埠(P_o)皆為相同的最糟情況而設計。

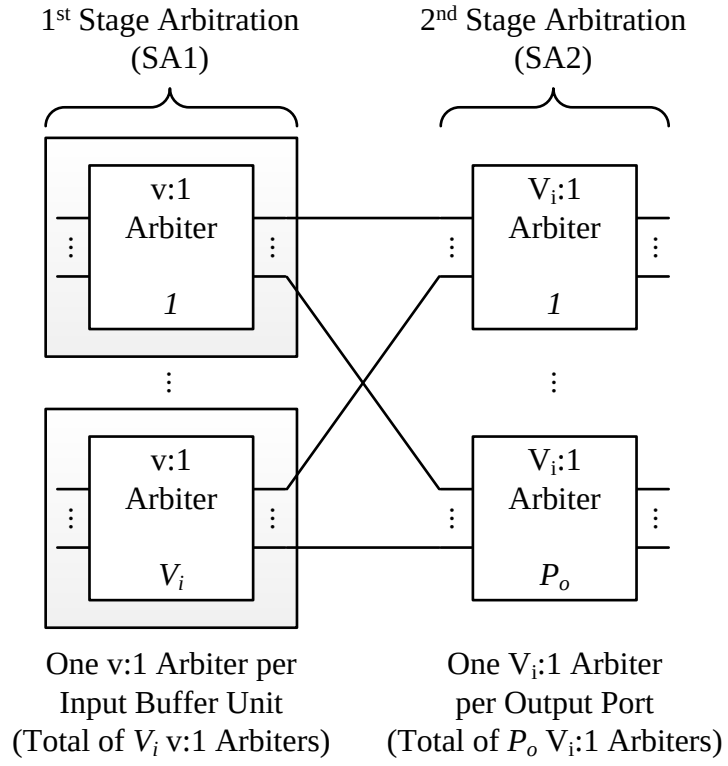


圖 3-9　具緩衝器合併之開關配置邏輯設計

3.3.6 具緩衝器合併之交叉開關設計

交叉開關主要連接路由器輸入緩衝器單元與輸出埠之間的傳輸路徑，執行路由器最基本的封包交換動作。交叉開關基本結構由多個多工器(multiplexer)所建立，如圖 3-10。其中，多工器的控制訊號來自於 SA 邏輯，主導連線的建立，亦即該訊號決定哪一個輸入緩衝器單元(V_i)應該被連接至哪一個輸出埠(P_o)。在緩衝器合併演算法的執行下，多工器的數量最多為五個(以 mesh 拓樸結構中非邊緣路由節點為例)，分別對應至不同輸出埠方向，而每一個多工器的輸入埠(C_i)編號數量為可變動地，分別對應至合併後的輸入緩衝器單元編號 V_i。

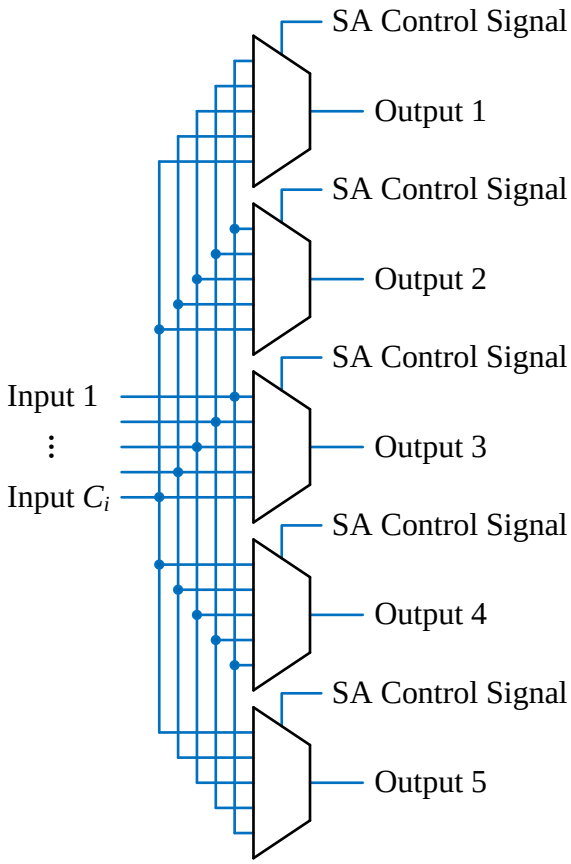


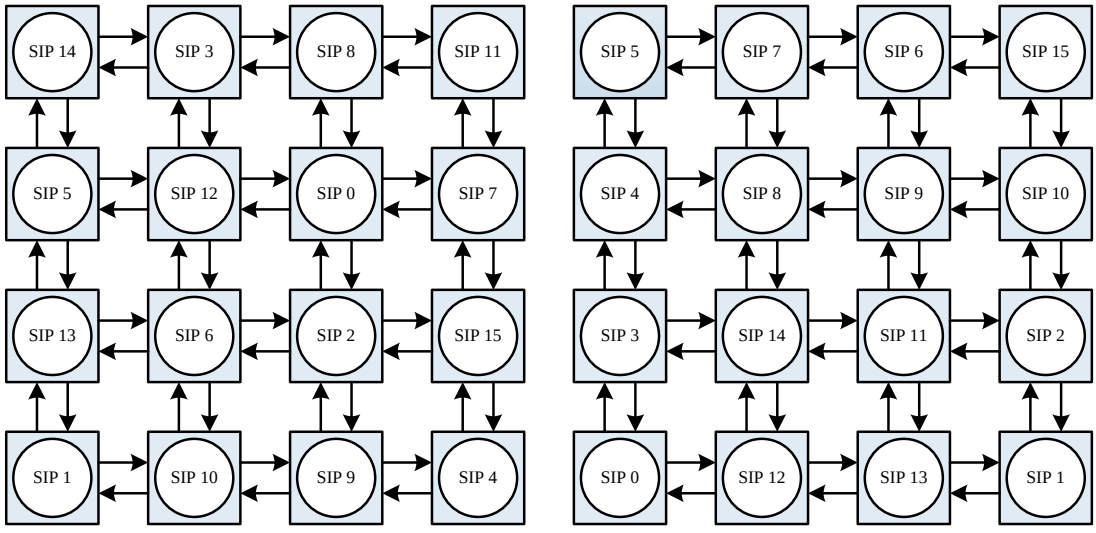
圖 3-10　具緩衝器合併之交叉開關設計

第四章 模擬與實驗結果

此章針對本論文所提出的緩衝器合併演算法，進行模擬與分析，並作詳細的報告與討論。

4.1 模擬環境

為了實際測試緩衝器合併演算法對於 NoC 系統的可行性與正確性。本論文對圖 3-1 所表示的 VOPD 任務圖進行實作與模擬。實作過程中，本論文首先針對映射問題將指定的任務分配到目標 NoC 拓樸結構，更將其分別劃分為隨機映射與 PERMAP 映射以便觀察不同的映射策略對於本論文所提出的緩衝器設計方法具體的影響程度，如圖 4-1 所示。接續使用 verilog 硬體描述語言進行路由器電路設計，並以 Mentor Graphics Modelsim 做暫存器轉移層次(Register Transfer Level)的編譯和功能模擬驗證，後續接以 Synopsys Design Compiler 合成軟體搭配 ARM 產商所提供 TSMC 0.18μm 製程的 CMOS 標準元件庫進行電路合成並觀察電路面積、功率及 timing 變化情形。而後，進行邏輯閘層次(gate level)的功能模擬與比對並轉儲 VCD(Value Change Dump)檔案。最後再載入 VCD 檔案於 Synopsys PrimeTime(PX) 以做更進一步的電路 timing 分析與功率消耗評估。



(a) random　　(b) PERMAP

圖 4-1　映射策略

在路由器設計過程，採先實現一個典型虛擬通道路由器，再進而建立出完整的 NoC 系統，以供後續模擬數據比較的基準。之後，執行緩衝器合併演算法並修改路由器設計及整體網路系統以符合演算結果。詳細的 NoC 模擬環境設定及路由器設計規格如表 4-1 所列。其中，由於路由器的運作上存在著許多仲裁機制，為使仲裁結果獲得公平，仲裁器的實現採用矩陣仲裁器(matrix arbiter)[15][39]，其除具備循環式(round-robin)的仲裁機制外，更提供了 least recently served 的優先策略以確保公平性。除此，矩陣仲裁器對於較少的輸入請求還具有執行快速且電路設計簡單的優點。

表 4-1　模擬環境設定及路由器設計規格

Configuration	Experimental sets
Network topology	4×4 2D mesh
Routing algorithm	X-Y routing algorithm
Flow control	Wormhole flow control
Channel buffer management	Credit-based management
Virtual channels	2 VCs per physical channel
Pipeline stage delay	Router: 4 ; Channel: 1 (cycles)
Arbiter implementation	Matrix arbiter
Mapping policy	Random; PERMAP
Operating frequency	100 MHz
Buffer size	4 flits
Packet length	8 flits
Flit width	34 bits

於測試方面，由於本論文實作網路尺寸為 4×4 2D mesh，因此需提供 16 個處理器以進行網路資料封包的交換。但由於實際上處理器的面積很大，故在測試 NoC 是否正常工作時，並未嵌入真實的處理器，而是運用 verilog-testbench 來模擬每一個處理器傳送/接收封包的行為，藉此測試該網路是否正常的運作。本論文將測試程式與 NoC 作整合測試，檢測每一個處理器是否能夠正確地運用封包的方式來進行資料交換，並且測定出該網路的資料傳輸效能。圖 4-2 為測試程式之模組區塊結構示意圖[15]。其中，為了模擬處理器行為，在交通樣式產生(traffic pattern generation)模組中包含封包產生器(packet generator)單元，其主要功能為根據特定的交通樣式、封包規格和注入率(injection rate)以產生測試封包；由於本論文將封包劃分為多個 flit 進行傳輸，故注入率定義為每一時脈週期，平均輸入至網路系統中的 flit 數量。

來源佇列(source queue)具體的規格大小設計必須大於等於測試封包的產生數量，主要原因為在網路系統滿載(full)時，來源佇列必須提供足夠的緩衝空間以儲存測試封包，避免封包的遺失(loss)。除此，測試封包自產生到來源佇列之前將經過一道標記程序，該程序主要對 flit 進行數量的計數標記及測試的起始時間(start time)標記，故值得注意的是，所有測試封包的傳輸延遲包含了兩部分：封包尚未進入網路系統而處於來源佇列中所花費的緩衝儲存時間及封包進入網路系統後傳輸所花費的時間。相應的另一標記程序由輸出模組(ejection module)所執行。該模組會對接收自網路中輸出的 flit 進行數量計數與測試結束時間(finish time)的標記，因此單一 flit 的傳輸延遲可藉由被標記的測試結束時間與起始時間的相減獲得，進而計算出全局平均網路傳輸延遲。網路傳輸率則定義為測試期間輸出模組所接收到的 flit 數量除以總測試時間。

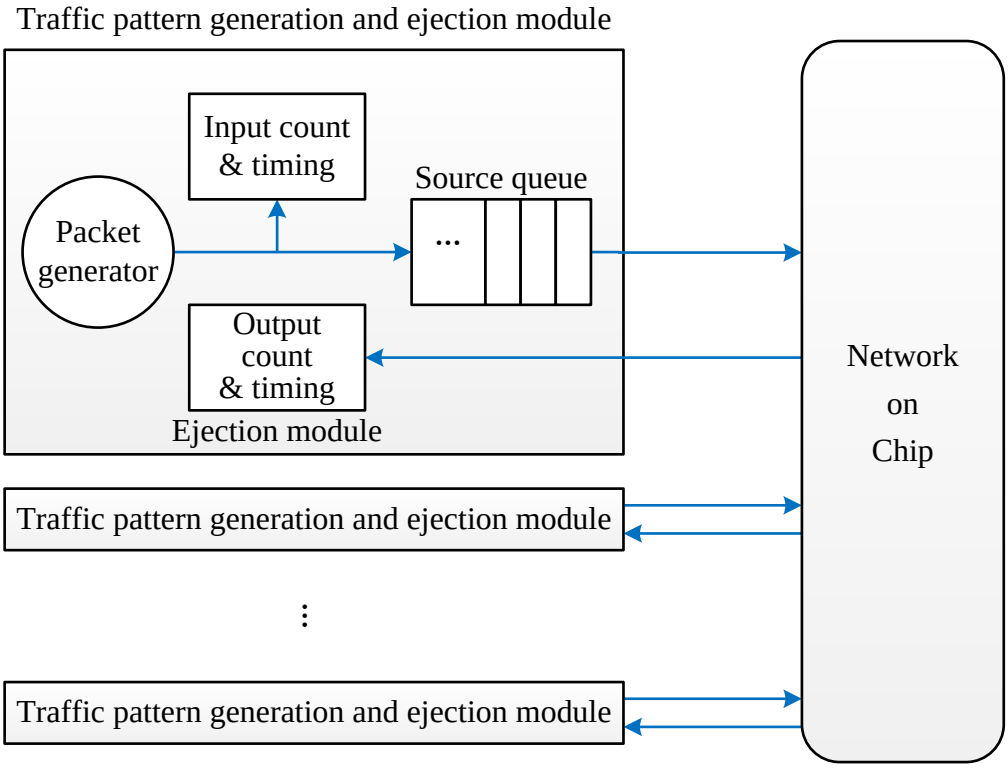


圖 4-2　測試程式之模組區塊結構

4.2 實驗結果與討論

表 4-2 顯示對於單一路由器而言，採用本論文所提出的緩衝器設計方法與均勻分配緩衝器的典型路由器實驗結果比較。因為各個路由器通道負載的分佈不同，所以路由器經改良後分別存在著具 4 個緩衝器單元並依序遞減至 1 個緩衝器單元結構的路由器。在 propagation delay 方面，由於路由器管線並無改變，因此在典型路由器與各個改良後的路由器於網路不壅塞的情況下同樣維持在 5 個時脈週期。面積成本方面，因為緩衝器單元的遞減，在緩衝器單元數量為 1 時，相對於典型路由器的面積改善百分比上可達 74%的節省，這對於受限的 NoC 資源而言是很可觀的。功率消耗方面，因為緩衝器為路由器主要功率消耗部分，因此在緩衝器單元遞減的情況下，使得在緩衝器單元數量為 1 時，同樣可節省達 74%的功率消耗表現(此功率消耗指動態功率消耗)。

表 4-2　典型路由器與改良後路由器比較數據

	General Router (5 Buf. Unit)	Modified Router (4 Buf. Unit)	Modified Router (3 Buf. Unit)	Modified Router (2 Buf. Unit)	Modified Router (1 Buf. Unit)
Propagation Delay (cycles)	5	5	5	5	5
Area (um^2)	222226.81	175055.13	136894.67	95291.38	55836.95
Percentage (%)	100	21	38	57	74
Power (mW)	15.7131	12.5345	9.8725	6.7721	3.9385
Percentage (%)	100	20	37	56	74

另一方面，以 mesh 角度為觀點，表 4-3 為完整 VOPD mesh 的模擬比較數據。其中，更分別劃分為不具緩衝器合併機制的隨機映射及 PERMAP 映射和具有緩衝器合併機制的隨機映射及 PERMAP 映射，共四種不同模擬方案。由於不具緩衝器合併機制的隨機映射及 PERMAP 映射僅僅差異於 SIP 排列位置的不同，路由器內部緩衝器單元數量並無改變，因此由表 4-3 可得知其面積大小為相等，功率消耗部分亦是如此。在隨機映射具有緩衝器合併機制的模擬情況下，緩衝器單元因為被合併而獲得數量上的減少，故相較於不具緩衝器合併機制的映射方案而言，其面積和功率表現皆可獲得 40%的改善。而相對於隨機映射，較佳的 PERMAP 映射演算方式且具緩衝器合併機制的模擬方案，對於不具緩衝器合併機制的映射方案於

面積和功率的改善幅度更可達到 51%的優異表現。

表 4-3　VOPD mesh 改良前後面積和功率比較數據

	Random(PERMAP) mapping without buffer merge	Random mapping with buffer merge	PERMAP mapping with buffer merge
Area (um2)	4078040.04	2436295.31	1987460.82
Percentage (%)	100	40	51
Power (mW)	273.6485	162.8558	134.7578
Percentage (%)	100	40	51

於傳輸延遲部分，圖 4-3 為隨機映射具緩衝器合併機制與不具緩衝器合併機制平均延遲曲線圖，圖 4-4 為 PERMAP 映射具緩衝器合併機制與不具緩衝器合併機制平均延遲曲線圖。由圖中可觀察出，具或不具緩衝器合併機制對於隨機映射及 PERMAP 映射而言，其平均延遲表現皆是為相似。換言之，在採用緩衝器合併機制下，雖然緩衝器單元數量獲得了減少，但大致仍然可保有緩衝器未合併前的平均延遲表現，除了在圖 4-3 中注入率為 1 時，可發現緩衝器合併前後平均延遲相差了 15 個時脈週期，而在圖 4-4 中注入率為 1 時，緩衝器合併前後平均延遲相差了 3 個時脈週期。其主要原因為雖然緩衝器的合併是經過計算的，理論上合併前後延遲表現應為相當，但當網路系統趨向飽和導致網路阻塞時，以緩衝器作為通道的數量多寡，對於紓緩網路阻塞的情況還是有所影響。該問題於未來可藉由網路壅塞的評估與預測，且讓可合併的目標緩衝器單元內的虛擬通道緩衝器數量於網路壅塞或較不壅塞處可分別適當地進行增加或減少以改善之。除此，隨機映射的 15 個時脈週期差異相較於 PERMAP 的 3 個時脈週期差異為大，乃歸咎於映射演算方式的較佳或較差所導致。

圖 4-5 為圖 4-3 與圖 4-4 的合併，以隨機映射具緩衝器合併機制與不具緩衝器合併機制平均延遲曲線為一組且 PERMAP 映射具緩衝器合併機制與不具緩衝器合併機制平均延遲曲線為一組的觀點分析，可清楚得知映射的好壞對於 NoC 平均傳輸延遲的影響，較佳的 PERMAP 映射演算可提供相較於較差的隨機映射方式有較低的平均延遲表現，但各組中緩衝器合併前後依舊保有相似的平均延遲。由此可知，影響 NoC 的傳輸延遲表現明顯來至於映射演算方式的差異，而本論文所提出的緩衝器合併演算方法並不會對 NoC 系統造成不良影響，反而於面積及功率方面可提供大幅度的節省。此外，映射演算方式的好壞對緩衝器合併演算方法的影響

則僅僅於面積和功率改善幅度的多或少。以表 4-3 為說明，只考慮具有緩衝器合併機制的隨機映射及 PERMAP 映射數據，較佳的 PERMAP 映射方式相較於隨機映射方式在面積和功率改善方面，分別可節省 18%的面積開銷和 17%的功率消耗。

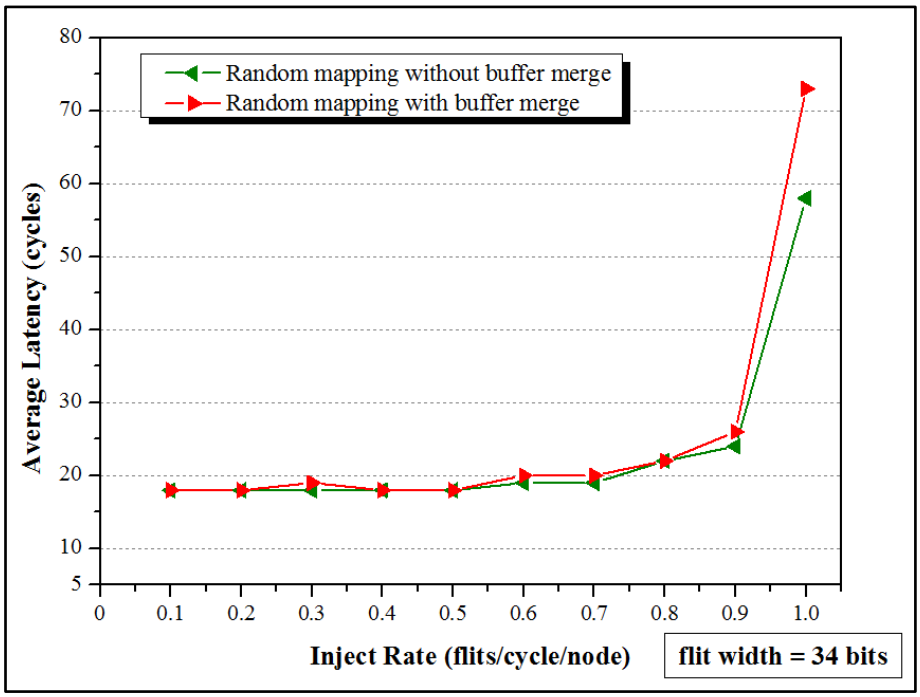


圖 4-3　隨機映射之緩衝器合併前後平均延遲曲線圖

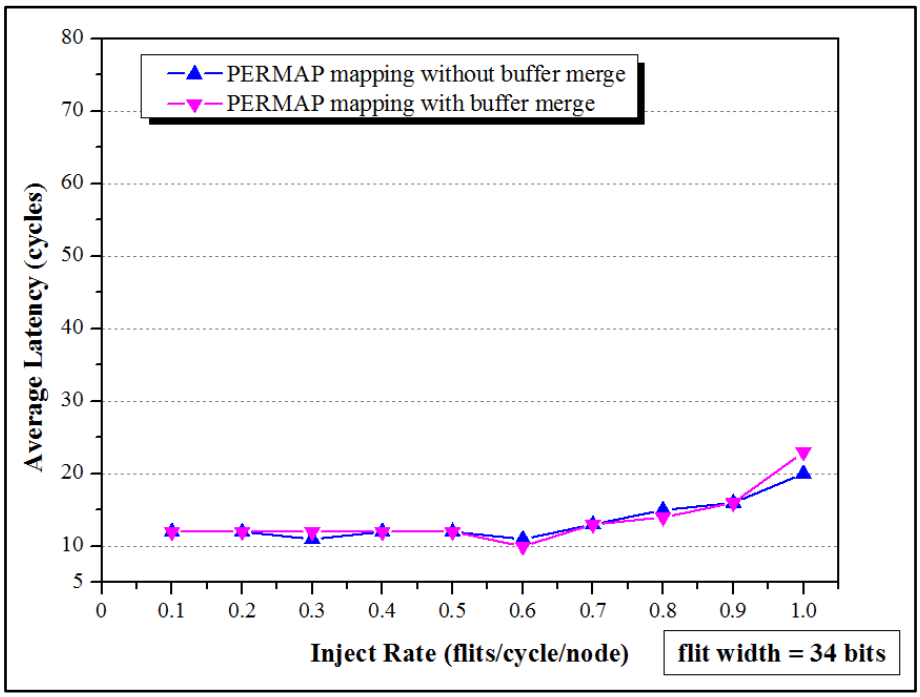


圖 4-4　PERMAP 映射之緩衝器合併前後平均延遲曲線圖

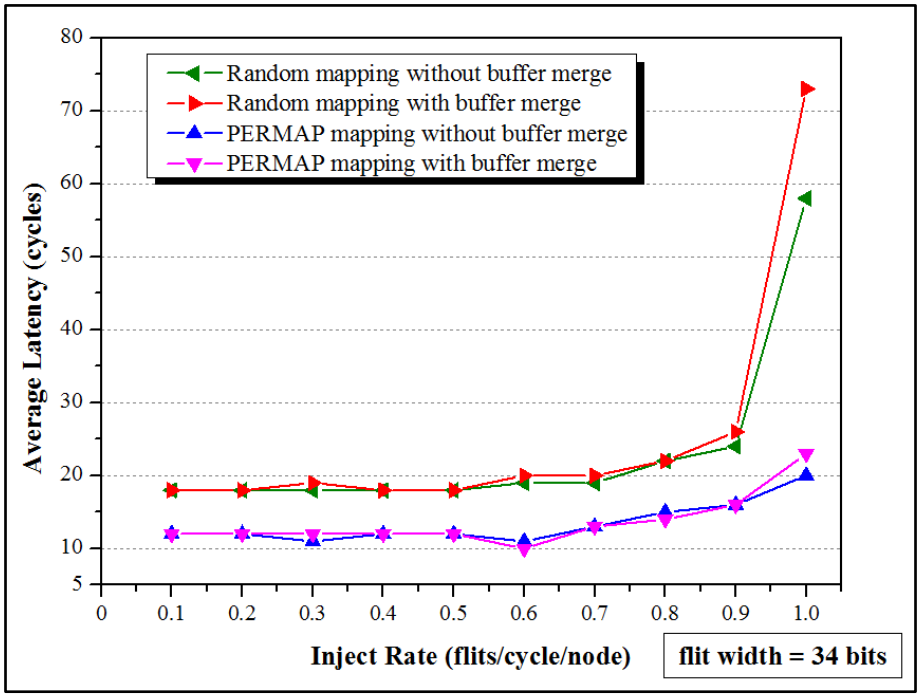


圖 4-5　映射演算差異之平均延遲曲線圖

第五章 結論與未來研究

5.1 結論

對於資源受限的 NoC 系統，為了降低系統成本並最佳化系統性能，每一個路由器的輸入緩衝器單元數量需要針對定應用流量特徵進行有效分配。本論文提出一種針對特定應用之單晶片系統的 NoC 路由器之緩衝器設計方法。該演算法首先估算出不同路由器每一個輸入埠的負載大小，隨後根據輸入通道的負載分布情況採用合併方式來實現緩衝資源的適當分配，使得在滿足系統性能約束的前提下，能最小化系統實現成本。實驗結果顯示，該設計方法可以更加合理地分配緩衝資源；針對單一路由器而言，在可接受網路性能情況下，依負載情況的不同，分別可節省約 20%至 70%的晶片面積與功率消耗。

此外，從何合併演算法計算過程可以看出，該演算法可應用於任意拓樸結構的網路模型中，其本質是與網絡拓樸無關的。同時，該演算法對於其它確定性路策略下的緩衝資源分配同樣適用。

5.2 未來研究

本論文所提出的緩衝器合併方法實際上是基於貪婪演算法來執行，而後續研究工作將針對性地加入更多的網路特徵條件，如網路壅塞的評估預測等，建立其模型並加以分析統計，讓可合併的目標緩衝器不再只限於以一個單元為考量，而是一個緩衝器單元內的虛擬通道緩衝器數量也可進行增加或減少，同時以其它演算法，如遺傳演算法等，進行更符合實際網路狀況的緩衝器合併選擇，以利於網路效能的提升。

最後，對於映射議題方面，多數研究文獻因其改良訴求的不同，大多主要針對功率或延遲方向進行改良，其映射方法未必可以真正符合緩衝器合併的最佳映射結果。因此，針對每一個路由器輸入埠而言，建立使其能具有最低流量需求的映射演算，同時，提高系統工作頻率，使得整個 NoC 緩衝器合併設計能夠更加完善是為未來研究的努力方向。

參考文獻

[1] L. Benini and G. De Micheli, "Networks on chip: a new paradigm for systems on chip design," in Proceeding of *Design, Automation and Test in Europe* (DATE), pp. 418－419, 2002.

[2] AMBA Open Specification, http://www.arm.com/products/system-ip/amba/amba-open-specifications.php, 2011.

[3] CoreConnect Bus Architecture, https://www-01.ibm.com/chips/techlib/techlib.nsf/products/CoreConnect_Bus_ Architecture, 2011.

[4] SoC Interconnection: Wishbone, http://opencores.org/opencores,wishbone, 2011.

[5] H. Cheng-Ta and M. Pedram, "Architectural energy optimization by bus splitting," *IEEE Transaction on Computer-Aided Design of Integrated Circuits and Systems*, vol. 21, no. 4, pp. 408－414, Apr. 2002.

[6] C. Grecu, P. P. Pande, A. Ivanov, and R. Saleh, "Structured interconnect architecture: a solution for the non-scalability of bus-based SoCs," in Proceeding of *ACM Great Lakes Symposium on VLSI* (GLSVLSI), pp. 192－195, Apr. 2004.

[7] S. Kumar, A. Jantsch, J. P. Soininen, M. Forsell, M. Millberg, J. Oberg, K. Tiensyrja, and A. Hemani, "A network on chip architecture and design methodology," in IEEE Proceeding of *Computer Society Annual Symposium on VLSI*, pp. 105－112, 2002.

[8] E. Rijpkema, K. Goossens, A. Radulescu, J. Dielissen, J. van Meerbergen, P. Wielage, and E. Waterlander, "Trade-offs in the design of a router with both guaranteed and best-effort services for networks on chip," in IEE Proceeding of *Computers and Digital Techniques*, vol. 150, no. 5, pp. 294－302, Sep. 2003.

[9] M. Horowitz and W. Dally, "How scaling will change processor architecture," in IEEE Proceeding of *International Solid-State Circuits Conference* (ISSCC), vol.1, pp. 132－133, Feb. 2004.

[10] C. Xuning and P. Li-Shiuan, "Leakage power modeling and optimization in interconnection networks," in Proceeding of *International Symposium on Low Power Electronics and Design* (ISLPED), pp. 90－95, Aug. 2003.

[11] T. T. Ye, L. Benini, and G. De Micheli, "Analysis of power consumption on switch fabrics in network routers," in Proceeding of *Design Automation Conference* (DAC), pp. 524－529, 2002.

[12] W. Hangsheng, P. Li-Shiuan, and S. Malik, "Power-driven design of router micro-architectures in on-chip networks," in IEEE/ACM Proceeding of *Annual International Symposium on Microarchitecture* (MICRO), pp. 105－116, Dec. 2003.

[13] H. Jingcao and R. Marculescu, "Application-specific buffer space allocation for networks-on-chip router design," in IEEE/ACM Proceeding of *International Conference on Computer Aided Design* (ICCAD), pp. 354－361, Nov. 2004.

[14] Y. Tamir and G. L. Frazier, "Dynamically-allocated multi-queue buffers for VLSI communication switches," *IEEE Transaction on Computers*, vol. 41, no. 6, pp. 725－737, June 1992.

[15] W. Dally and B. Towles, *Principles and practices of interconnection networks*, Morgan Kaufmann Pub., 2004.

[16] D. Jianxun Jason and L. N. Bhuyan, "Evaluation of multi-queue buffered multistage interconnection networks under uniform and nonuniform traffic patterns," in Proceeding of *International Conference on Computer Communications and Networks* (ICCCN), pp. 576－583, Sep. 1995.

[17] M. Kumar and J. R. Jump, "Performance enhancement in buffered delta networks using crossbar switches and multiple links," *Journal of Parallel and Distributed Computing*, vol. 1, pp. 81－103, 1984.

[18] W. J. Dally, "Virtual-channel flow control," *IEEE Transaction on Parallel and Distributed Systems*, vol. 3, no. 2, pp. 194－205, Mar. 1992.

[19] Y. Tamir and G. L. Frazier, "High-performance multiqueue buffers for VLSI communication switches," in Proceeding of *Annual International Symposium on Computer Architecture* (ISCA), pp. 343－354, 1988.

[20] Y. Tamir and G. L. Frazier, "The design and implementation of a multiqueue buffer for VLSI communication switches," in IEEE Proceeding of *International Conference on Computer Design* (ICCD), pp. 466－471, Oct. 1989.

[21] W. J. Dally and B. Towles, "Route packets, not wires: on-chip interconnection networks," in Proceeding of *Design Automation Conference* (DAC), pp. 684－689, 2001.

[22] W. J. Dally and C. L. Seitz, "The torus routing chip," *Distributed Computing*, vol. 1, no. 4, pp. 187－196, Springer Berlin/Heidelberg, 1986.

[23] P. Guerrier and A. Greiner, "A generic architecture for on-chip packet-switched interconnections," in Proceeding of *Design, Automation and Test in Europe* (DATE), pp. 250－256, 2000.

[24] P. P. Pande, C. Grecu, A. Ivanov, and R. Saleh, "Design of a switch for network on chip applications," in Proceeding of *International Symposium on Circuits and Systems* (ISCAS), vol. 5, pp. V-217－V-220, May 2003.

[25] F. Karim, A. Nguyen, and S. Dey, "An interconnect architecture for networking systems on chips," *IEEE Micro*, vol. 22, no. 5, pp. 36－45, Sep./Oct. 2002.

[26] N. Jerger and L. Peh, *On-Chip Networks*, Morgan & Claypool Pub., 2009.

[27] L. G. Valiant and G. J. Brebner, "Universal schemes for parallel communication," in Proceeding of *Annual ACM Symposium on Theory of Computing* (STOC), pp. 263－277, May 1981.

[28] T. Nesson and S. L. Johnsson, "ROMM routing on mesh and torus networks," in Proceeding of *Annual ACM Symposium on Parallel Algorithms and Architectures* (SPAA), pp. 275－287, July 1995.

[29] M. A. Kinsy, M. H. Cho, T. Wen, E. Suh, M. v. Dijk, and S. Devadas, "Application-aware deadlock-free oblivious routing," in Proceeding of *Annual International Symposium on Computer Architecture* (ISCA), pp. 208－219, June 2009.

[30] W. J. Dally and H. Aoki, "Deadlock-free adaptive routing in multicomputer networks using virtual channels," *IEEE Transaction on Parallel and Distributed Systems*, vol. 4, no. 4, pp. 466－475, Apr. 1993.

[31] M. Morvarid, M. Fathy, R. Berangi, and A. Khademzadeh, "IIIModes: New Efficient Dynamic Routing Algorithm for Network on Chips," in Proceeding of *International Multi-Conference on Computing in the Global Information* (ICCGI), pp. 57－62, Aug. 2009.

[32] C. J. Glass and L. M. Ni, "The Turn Model for Adaptive Routing," in Proceeding of *Annual International Symposium on Computer Architecture* (ISCA), pp. 278－287, 1992.

[33] C. Ge-Ming, "The odd-even turn model for adaptive routing," *IEEE Transaction on Parallel and Distributed Systems*, vol. 11, no. 7, pp. 729－738, July 2000.

[34] R. V. Boppana and S. Chalasani, "A Comparison Of Adaptive Wormhole Routing Algorithms," in Proceeding of *Annual International Symposium on Computer Architecture* (ISCA), pp. 351－360, May 1993.

[35] P. Kermani and L. Kleinrock, "Virtual Cut-Through: A New Computer Communication Switching Technique," *Computer Networks and ISDN Systems*, vol. 3, no. 4, pp. 267－286, Sep. 1979.

[36] H. Jingcao, Y. O. Umit, and M. Radu, "System-Level Buffer Allocation for Application-Specific Networks-on-Chip Router Design," *IEEE Transaction on Computer-Aided Design* (TCAD), vol. 25, no. 12, pp. 2919－2933, Dec. 2006.

[37] F. Gebali, H. Elmiligi, and M. W. El-Kharashi, *Networks-on-chips: theory and practice*, CRC Press, 2009.

[38] A. E. Kiasari, S. Hessabi, and H. Sarbazi-Azad, "PERMAP: A performance-aware mapping for application-specific SoCs," in Proceeding of *International Conference on Application-Specific Systems, Architectures and Processors* (ASAP), pp. 73－78, July 2008.

[39] R. Mullins, A. West, and S. Moore, "Low-latency virtual-channel routers for on-chip networks," in Proceeding of *Annual International Symposium on Computer Architecture* (ISCA), pp. 188－197, June 2004.

[40] N. Kavaldjiev, G. J. M. Smit, and P. G. Jansen, "A virtual channel router for on-chip networks," in IEEE Proceeding of *International SOC Conference* (SOCC), pp. 289－293, Sept. 2004.