東 海 大 學 電 機 工 程 學 系

碩 士 論 文

以多層次網格為基礎之晶片網路設計

Network-on-Chip Design Based on Multi-level Mesh

研 究 生： 蘇柏豪

指導教授： 蔡坤霖 博士

中華民國 一〇一 年 六 月

東海大學電機工程學系碩士學位

考試委員審定書

電機工程學系研究所 蘇柏豪 君所提之論文

以多層次網格為基礎之晶片網路設計

經本考試委員會審查，符合碩士資格標準。

學位考試委員會 召集人：張延任 (簽章)

委 員：葛坤霖

蕭宇宏

蔡奇宏

鐘玉芳

中 華 民 國 101 年 6 月 21 日

摘要

隨著製程及設計技術的快速進步，系統單晶片的使用日益廣泛。系統單晶片可含有數量龐大的矽智財(Silicon IP; SIP)模組，這些系智財模組在傳統的 SoC 中使用匯流排(bus)負責內部通訊傳輸。然而，在匯流排傳輸模式下，傳輸資料延遲、電路同步、雜訊和功率消耗的問題，均隨著製程縮小而需要大幅度改進。另一方面，當單一晶片內可擺放電晶體總數量急遽增加而愈來愈複雜時，晶片內部的通訊傳輸將影響晶片處理效率，傳統的匯流排傳輸已經漸漸地不敷使用。因此，為了確保各個模組之間資料傳輸與訊號溝通的正確性，網路封包傳輸概念被應用於晶片上，以進行不同矽智財模組間的資料交換。這種將單晶片系統內部的傳輸介面轉換成由網路模式傳輸的方式，稱為晶片網路(Network-on-Chip; NoC)設計。晶片網路具有良好的擴充性及較為可靠的晶片通訊方式。二維網格(2D mesh)拓樸架構在過去的晶片網路設計中被普遍地使用，因為它能使用簡單的路由演算法，並且具有好的擴充性。但是由於二維網格拓樸有相對較大的網路半徑，造成有些長距離的封包傳送有較大的傳輸延遲。因此在這篇論文中我們針對傳統網格拓樸提出一個多層次的設計方法，主要的概念是讓長距離資料傳送可以利用另一層網格拓樸進行快速傳輸。實驗結果顯示，我們所提出來的多層次網格架構在電路面積增加 20%的情況下，可以節省 50%功率消耗，並改善 70%效能，我們所提出來的方式確實可行。

關鍵字：系統單晶片、晶片網路、二維網格、路由器設計、矽智財

ABSTRACT

With the rapid progress of design technology, the use of SoC is increasingly widespread. SoC contains a huge number of Silicon IP (SIP) modules, and the SIP modules often use on-chip buses to transmit the data or control signals. However, the problems of communication latency, circuit synchronization, signal noise and power consumption need to be improved due to narrowing process. To ensure the accuracy of data communication between each module, the network packet transmission concept is applied to on-chip communication, which is called Network-on-Chip (NoC) design.

NoC is a flexible and reliable on-chip communication architecture. 2D mesh topology is often used for NoC design, because it can use a simple routing algorithm, and has good scalability. However, the 2D mesh topology has relatively large radius of the network, and causes larger transmission delay for long distance packets. In this paper, we propose an improved design for 2D mesh topology. The main concept is long distance data transmission take another layer of mesh to achieve fast communication goal.

The experimental results show that the multi-level mesh architecture can save 50% power, and improve 70% performance with 20% area increase.

Keyword：System-on-Chip (SoC), Network-on-Chip(NoC), mesh, router design, Silicon IP (SIP)

目錄

圖目錄

表目錄

第一章 序論

1.1 晶片網路的發展

近十年來，在科技快速發展下，不論是消費性電子產品、電腦、汽車電子、通訊和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，面對比較大型且功能複雜的系統，在功率、效能、面積和延遲的需求已成晶片設計上的挑戰。

特殊應用積體電路(Application-Specific Integrated Circuit, ASIC)是為了滿足低成本和縮短開發週期要求而應運而生的。在過去幾年間，隨著晶片製程的技術運用到奈米級尺寸，單一晶片內已可放置數千萬顆電晶體。由於 ASIC 設計技能的增強，意味著更多的元件功能已可輕易地被整合並實現於單一晶片內，稱之為系統單晶片(System-on-a-Chip, SoC)。

系統單晶片將原本分處在不同晶片內，負責不同功能的矽智財(Silicon Intellectual Properties, SIP)模組，加以整合至單一晶片內，從整個系統的角度出發，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的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整合的解決方案，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，因此 SoC 設計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。在一個 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 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。

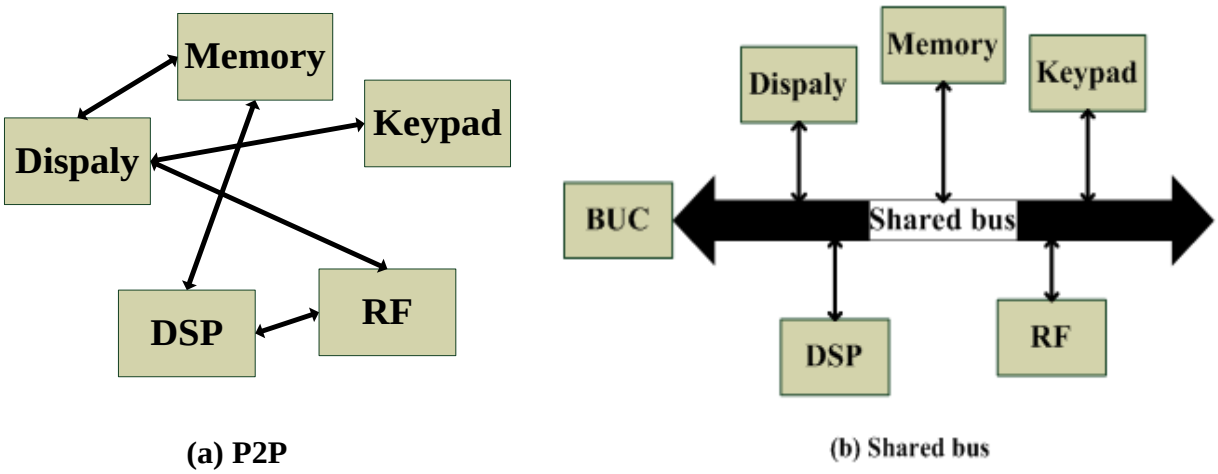


圖 1-1　傳統單晶片系統內部通訊架構

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，其基本結構的變形，如 ARM AMBA [2]、IBM CoreConnect [3]、Wishbone [4]等協定，更成為現今 SoC 整合廣泛被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。因匯流排總長度增加及 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。且共享匯流排在同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。故總體而言，SoC 經常由幾十或幾百個 SIP 所組成，而這些 SIP 必須共享總頻寬，導致以匯流排為主的資料傳輸結構將面臨嚴重的通訊瓶頸[6]，無法承載資料的高流通量，且無法滿足 SoC 高速、高可靠度及高複雜度的設計挑戰。為了解決晶片內部通訊的問題，一些研究學者主張將一般網路通訊的概念運用於 SoC

設計上。這樣的概念允許晶片內部的各個矽智財模組去耦化(decoupling)，意即不再需要晶片全域同步(global synchronization)，矽智財模組之間只要在傳遞訊號時同步即可。這種新的方式稱為晶片網路(Network-on-Chip; NoC)。晶片網路的基本架構如圖 1-2，每個矽智財模組之間的資料，可以藉由封包(packet)的形式傳遞，並利用晶片內部路由器(router)進行溝通[7][8]。晶片網路的出現大幅減少晶片內部全域繞線的需求[9]，其特點是具有(1)較佳的傳輸效能;(2)頻寬的延展性;(3)可模組化(modularity)及可重複使用化(reusability);(4)並行性(concurrency)等。NoC 的規則性與好的可控制結構，使其擁有高源效率和可靠性，規則的結構也使得 NoC 可模組化和設計容易被重複使用。另外，同時進行亦可達成，主要是因為 NoC 內的連線可以同時地被不同封包使用，所以它增加了傳輸連線的使用率。

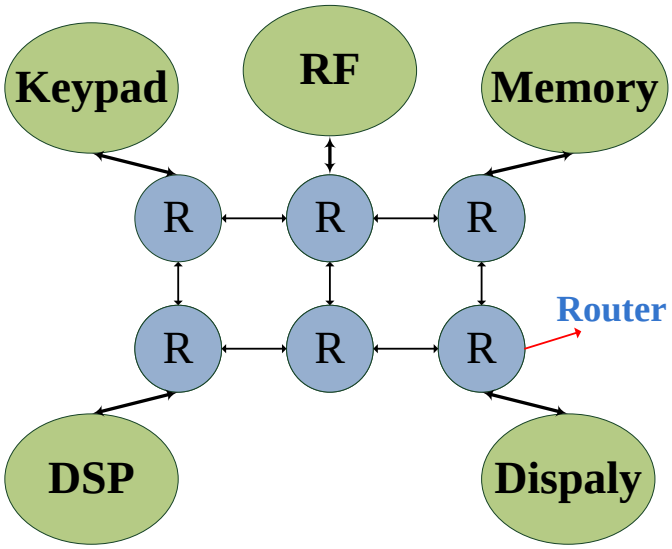


圖 1-2　晶片網路基本概念

1.2 基本晶片網路架構

本節藉由一個簡單的網格(mesh)互聯拓樸結構例子介紹基本 NoC 的基本架構。如圖 1-3 所示。每一個 SIP 可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 SIP 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

樸網路完成資料封包的相互傳輸。

NIU 可實現計算節點與通訊網路兩者的距離，可提高 SIP 於設計上的再重複使用率，縮短 SoC 的設計時間。NIU 的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。

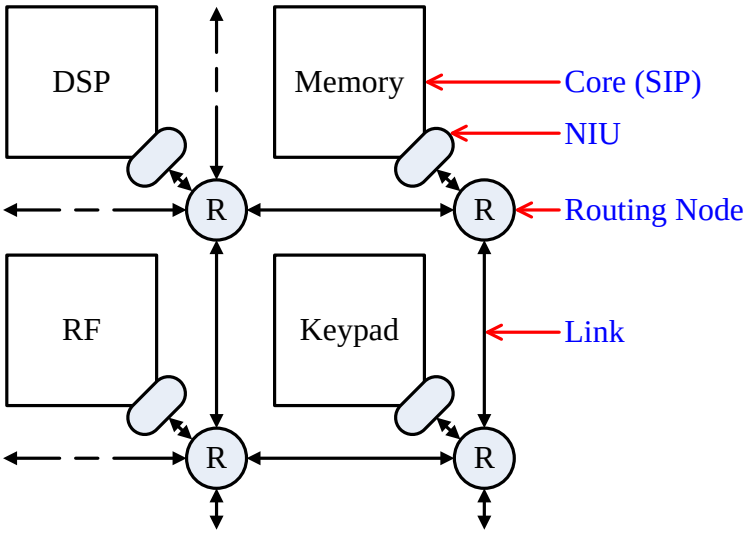


圖 1-3　二維網格拓樸，NoC 基本功能元件組成

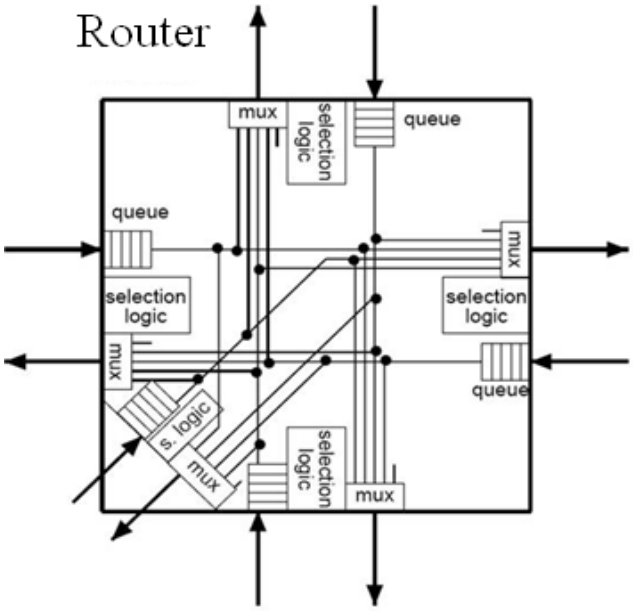


圖 1-4　路由器內部接線圖

在晶片網路中，路由器是最重要的組成元件，所有晶片網路的特性以及其所提供的功能都必須藉由路由器完成。路由器主要連接本地端的 SIP 與周圍相鄰的路由器，其傳輸的方式需要根據網路上的路由演算法(routing algorithm)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)，其仲裁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。如圖 1-4 所示，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)因應具特殊傳輸需求的系統。當然路由器也可以根據不同的尋徑策略傳送訊息，並藉由多工器(multiplexer)將多個輸入訊息傳到輸出埠。另外，當壅塞發生時，路由器也可以將資料儲存於內部的緩衝佇列中。

1.3 研究動機

以二維網格(2D mesh)為基礎的 NoC 被廣泛地運用在 SoC 設計上，主要的原因在於二維網格具有簡單的尋徑演算法及網路可延展性。由於尋徑演算法簡單，且路由器之間的連線單純，因此路由器電路在二維網格架構下可以有效地實行。

然而，二維網格在資料傳輸上有較大的平均距離。一般而言，長距離傳輸的資料封包需要經過許多路由器方能到達目的地，因而導致較大的傳輸延遲及較高的功率消耗。如圖 1-5 所示，若從起始節點 source node (SN)將資料傳輸至目的節點 destination node (DN)，需要經過 11 個路由器，亦即需經過 11 個路由器的傳輸延遲並耗費 11 個路由器之運算功率。當網格逐漸成長，對角節點的傳輸延遲將更為嚴重。

有鑑於二維網格在設計上仍有其缺陷，本論文提出一個多層次的網格拓樸架構，此方法能夠降低長距離封包的傳輸延遲。多層次網格架構的基本想法為藉由增加第二層網格，將長距離的資料封包利用第二層網格進行傳輸，以縮短其傳輸時間，並解決壅塞問題。一

個例子如圖 1-6 所示，紅色節點和藍色線段組成第二層的網格，第二層網格有較少的節點，因第二層節點為第一層節點的子節點，第二層節點的通道跨過多個第一層網格的節點。因此，讓資料封包於第二層網格傳輸，可以大幅減少原來所需經過的路由器個數。

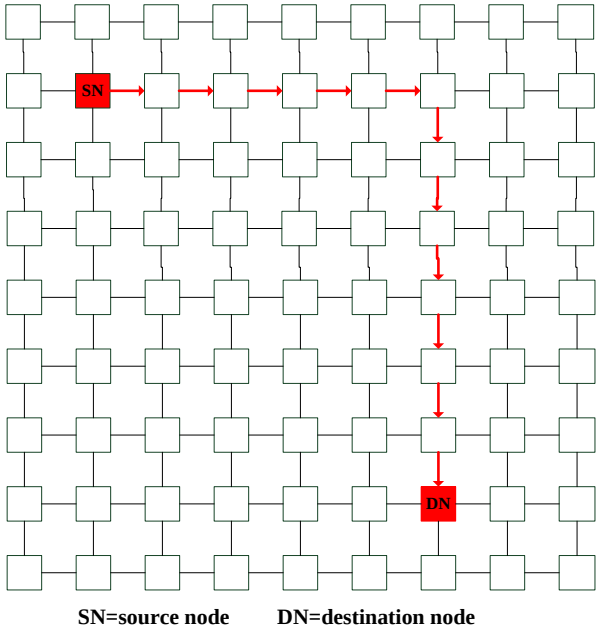


圖 1-5　傳統晶片網路網格架構範例

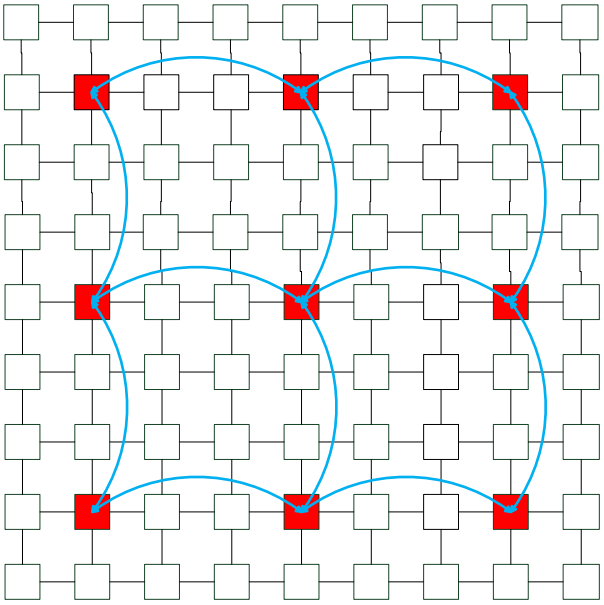


圖 1-6　二層網格晶片網路平面圖

1.4 章節組織

此篇論文為二維網格的多層次架構研究,在第二章我們介紹 NoC 相關的背景知識及近幾年的相關研究工作，內容包含常見的拓樸架構、尋徑方法、路由器架構等。第三章提出以多層次網格為基礎之晶片網路設計，為了使新提出來的網格架構能夠正常運作，我們也針對新架構設計尋徑演算法，並介紹其內部的路由器設計方式。第四章呈現了新架構的實驗模擬結果，我們針對面積、功率消耗及傳輸延遲改善等方面做一分析比較。最後，我們於第五章對於我們所提出來的以多層次網格為基礎之晶片網路設計做一總結。

第二章 背景與相關研究論述

此章節介紹晶片網路相關的研究，包括常見的晶片網路結構及其內部資料傳輸的方式，涵蓋的內容主要有：拓樸網路、路由器架構、尋徑方法及交換技術的介紹與分析比較。

2.1 晶片網路拓樸架構

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。拓樸選擇為 NoC 設計的首要步驟，路由策略、交換技術和路由器結構設計皆以其為基礎而建立。

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，因此效能和成本也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，進而影響網路延遲。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的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 拓樸之比較

由上述所說明，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開銷。

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，皆會產生資料傳輸上的延遲，即使傳輸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。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位元數(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 Tree 形式拓樸架構

- 平衡二元樹(balanced binary tree)：圖 2-1 為一個平衡二元樹拓樸結構。其中，葉節點(leaf node)為 SIP，而非葉節點則為路由器。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，尤其是根節點(root node)，容易成為通訊瓶頸所在。

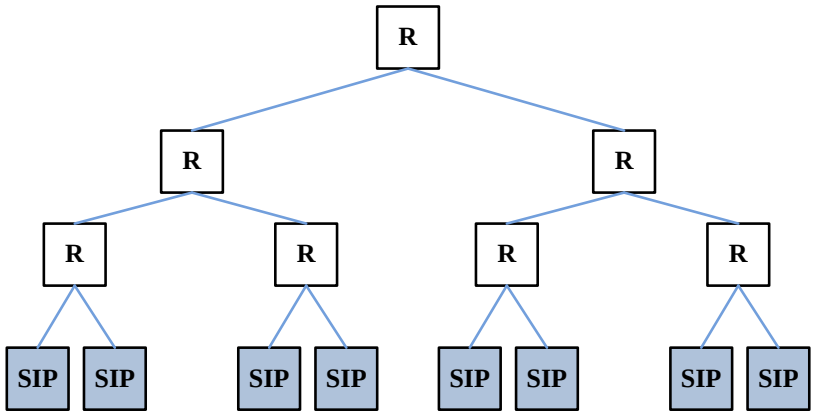


圖 2-1　平衡二元樹拓樸架構

- SPIN(Scalable, Programmable, Integrated Network)：為 Guerrier 與 Greiner 提出了一個稱為的互連模型[10]，他們利用 fat-tree 結構使各個矽智財模組互相連接。在 fat-tree 結構

當中，每個內部節點皆有四個子節點。圖 2-2 顯示了擁有 16 個葉節點的 SPIN 基本架構，葉節點數目表示系統中矽智財模組的數目。在圖 2-2 中，SIP 區塊代表矽智財模組，而 R 區塊則代表路由器模組。在 SPIN 架構下，網路的規模隨(N log N)/8 增加，而路由器的數目將會收斂到 R = 3N/4 ，其中 N 為矽智財模組的個數。

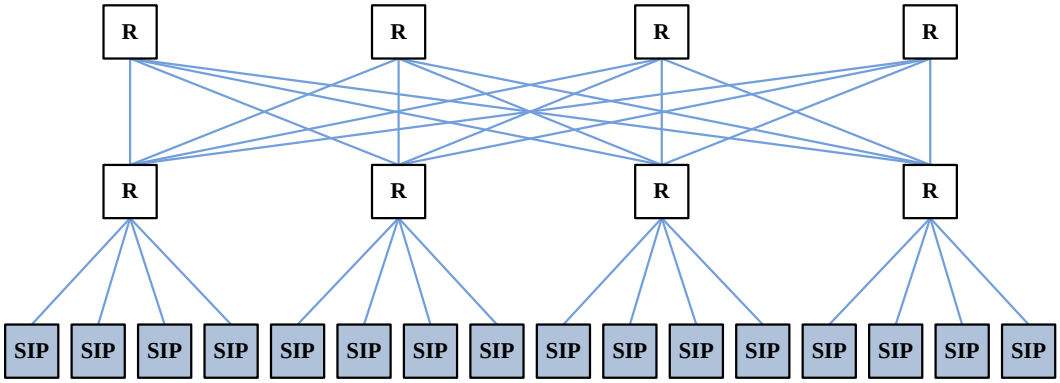


圖 2-2　SPIN 拓樸架構

- Butterfly Fat Tree(BFT)：BFT [11]與 SPIN 結構相似，如圖 2-3 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大為減少，改善了 SPIN 增加面積的缺點。此外，BFT 拓樸架構的最大優點為任兩個節點之間的通訊距離均相等。

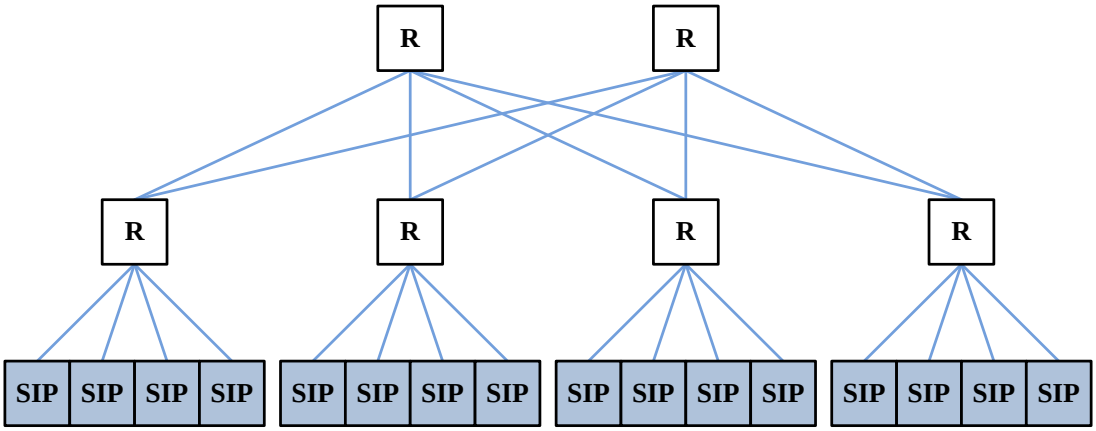


圖 2-3　BFT 拓樸架構

2.1.3 Mesh 與 torus 形式拓樸架構

- 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，稱為 CLICHÉ(Chip-Level Integration of Communicating Heterogeneous Elements)[7]，在本論文中以 mesh 拓樸稱之，圖 2-4 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分別為東、西、南、北與本地(local)，前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。在 Mesh 拓樸結構中，路由節點的數量與計算節點相同，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達成。因此 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，尤其對角線節點之間的資料傳輸影響甚大。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與延遲的增加。

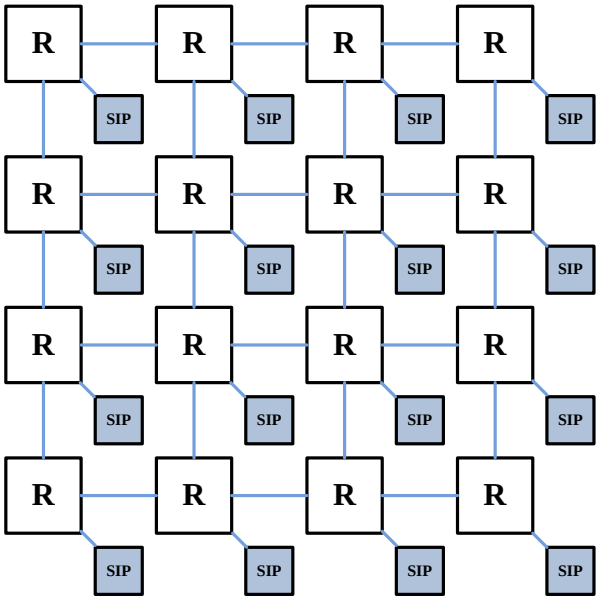


圖 2-4　4×4 mesh 拓樸結構

- Torus：為 Dally 與 Towles 提出了一個類似 mesh 架構的 2D-Torus 架構[12]，如圖 2-5 所示。這個 torus 的基礎架構就像是一個規則的 mesh[13]，唯一的區別在於邊緣的路由器經由連線連接到另一個邊緣的路由器。不同於 CLICHÉ，每個路由器皆有五個出

入埠(port)，其中一個連接到本地的矽智財模組，其餘四個則連接到鄰近的路由器。在此架構下，路由器的數目與矽智財模組的數目亦為相同。如此藉由額外的鏈結，將明顯縮短平均距離，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。但此方法將使得網路接線更複雜化，且過長的環狀鏈結亦會造成額外的傳輸延遲。

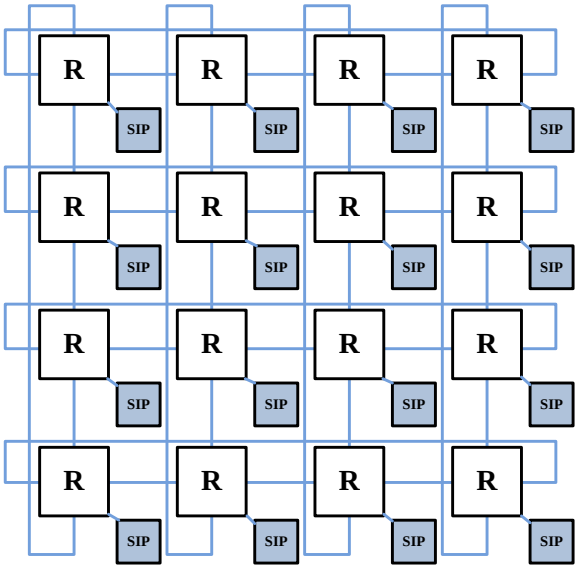


圖 2-5　4×4 torus 拓樸結構

- Folded Torus：folded torus 避免了 torus 拓樸結構中長距離鏈結所造成的額外傳輸延遲[14]。該拓樸結構採用錯位鏈結的方式代替 torus 拓樸中的環狀鏈結，如圖 2-6 所示。所示。如此設置方式縮短了鏈結的長度，解決了 torus 拓樸架構的缺點，不僅提高了整體傳輸效能且更易於 VLSI 的實現。

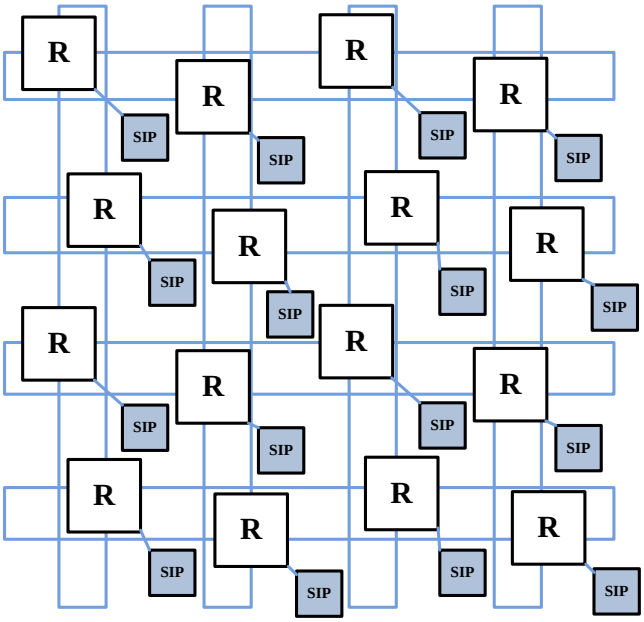


圖 2-6　4×4 folded torus 拓樸結構

2.1.4 Octagon 形式拓樸架構

Karim 等人了提出 Octagon 的 NoC 架構[13]。圖 2-7 顯示了一個基本的 Octagon 架構，此架構由八個節點和十二個雙向連結所組成，每個節點都與一個矽智財模組和一個路由器相連。在這個架構下，任何兩個矽智財模組之間的資料傳輸，都只需要經過兩個路由器，亦即 hop count 為 2。這個架構的擴充方式是：每一個八角形節點被 2-tuple (i,j)索引，對於每一個 i=I, I∈[0:7] ，一 Octagon 是用節點建構{(I,j), j∈[0:7]}，這些 Octagon 經由對應的 i 節點連接起來，每個節點(I,J)屬於兩個 Octagon：一組是{(I,j), j∈[0:7]}所組成，令一組是{(i,J), i∈[0:7]}所組成。然而這些連接點可能成為傳輸瓶頸，Octagon 的架構也可以擴展成多邊形；擴展後的多邊形架構稱為 spidergon[15]。

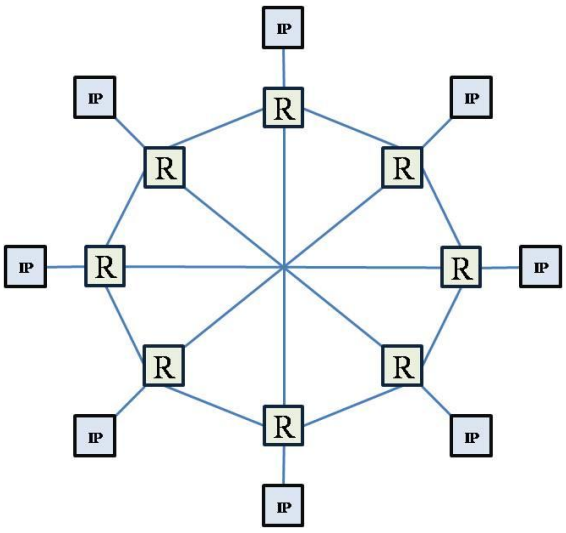


圖 2-7　Octagon 拓樸架構

2.2 尋徑方法

尋徑(routing)的定義是封包從起始(source) 矽智財模組傳送到目的(destination) 矽智財模組的路徑，以下將介紹三種不同的分類方式[16]：

2.2.1 路徑決定於何處

- 起點尋徑(Source 尋徑):在起點尋徑中，封包的路徑由起始模組使用預先計算好的路徑表決定。標頭封包需要帶著路徑上所有路由器的資訊，封包所經過的路由器越多，封包的大小也會跟著增加。起點尋徑不必在封包中保留目的模組位址的欄位，也可節省在每個路由器的尋徑時間[17]。
- 目的尋徑(Distributed 尋徑):在目的尋徑中，路徑決定於每個路由器中的硬體電路計算或是查詢路由器內的尋徑表。由於每個路由器皆須更新封包的下一個路徑，因此路由器會耗費較多的時間計算路徑上。

2.2.2 路徑如何被定義

- 靜態尋徑(Static 尋徑): 靜態尋徑意指不管當前網路狀態為何，封包路徑均為固定路徑。每個封包到達路由器後，由路由器依據其規則決定封包的下一個傳輸點。使用的規則可能為單一的固定路徑，或者是將流量分到隨機或是多個預先定義的路徑上。靜態尋徑的電路較為簡單，亦較為容易製作，但若矽智財模組之間有大量的資料傳輸時，如架構設計不當，容易產生封包壅塞情形，致使整體晶片效能下降。

- 動態尋徑(Dynamic 尋徑): 在動態尋徑當中，封包路徑會根據目前網路狀態而決定。因此，路由器將會替封包選擇較佳的路徑，以避開 NoC 中較為壅塞的區域。動態尋徑方式雖可避開壅塞區域，但在電路設計上較為複雜，必須有額外的記憶體記錄目前網路狀態，對於大型 NoC 而言，此一方式較有機會獲得效能改善。

2.2.3 路徑的長度

- 最短路徑(Minimal): 使用起始矽智財模組與目的矽智財模組之間的最短路徑，封包所經過的每個路由器必須更接近目的節點。XY 尋徑[16]就是一種最短路徑的傳輸方式。

- **非最短路徑(Non-minimal):**封包可以經由各種路徑抵達目標節點，所經過的路徑未必是最短路徑。此一方法是為了避免遭遇流量瓶頸的情況，以避開壅塞的通道。但是非最短路徑可能會導致livelock的情況，及額外的功率消耗[17]。

2.3 尋徑演算法

決定晶片網路的拓樸架構後，須使用尋徑演算法(routing algorithm)決定資料封包在晶片網路中將如何選擇傳輸路徑傳達至目的節點，換言之，在特定拓樸網路裡，尋徑演算法決定如何從起始節點至目的節點之間選擇一條路徑予以封包傳輸。尋徑演算法的選擇對晶

片網路的資料傳輸率及效能表現有直接影響。

使用良好的尋徑演算法可以避免熱點(hotspot)的生成並且能盡量減少資源衝突情況的發生。原因在於拓樸網路提供傳輸路徑時，可以均衡地分配交通流量負載，對此可改善網路的傳輸延遲與資料傳輸率。所以交通流量負載愈是均衡，其網路效能表現愈是理想，不過此特性卻明顯地牴觸了其它約束條件，如果盡可能地保持較短的傳輸距離，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使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。

尋徑演算法在晶片網路的傳輸上對於延遲及功率消耗有相當大的關係，所以要實現尋徑機制所帶來的影響，則須仔細考慮選擇。尋徑電路(routing circuit)會拉長路徑延遲並增加路由器面積，而在於功率方面，儘管尋徑電路之功率消耗量不高，但具體的路徑選擇，卻會影響所經過節點數的多寡，因此直接影響了功率消耗。

2.3.1 尋徑演算法的分類

尋徑演算法一般分為三種類型：確定性尋徑演算法(deterministic routing algorithm)、模糊性尋徑演算法(oblivious routing algorithm)和適應性尋徑演算法(adaptive routing algorithm)，其為根據如何選擇於起始節點至目的節點之間所存在的有效傳輸路徑而予以分類[18]。

1. 確定性尋徑演算法：對資料傳輸而言，即使起始節點至目的節點之間存在著多條可能路徑，確定性尋徑演算法用於起始節點至目的節點之間只會選擇唯一且相同的路徑。雖然目前許多路由演算法已經被提出，但最為普遍使用於晶片網路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 尋徑演算法。維序路由演算法採用 dimension by dimension 的方式傳輸資料。以圖 2-8 為例，圖 2-8 為一個二維網格拓樸網路，X-Y 尋徑演算法於起始節點(0, 0)首先沿 X 軸方向傳送封包，當到達與目的節點有相同

的 X 座標節點時，再轉向沿 Y 軸方向繼續傳送至目的節點(4, 3)。

確定性尋徑演算法的優點是電路設計較為簡單且在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)提高，卻可能導致資料傳輸率的下降，其原因在於確定性尋徑演算法無法對網路壅塞情況作出動態調整。換言之，確定性尋徑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，確定性尋徑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，實際上確定性尋徑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晶片網路設計中。

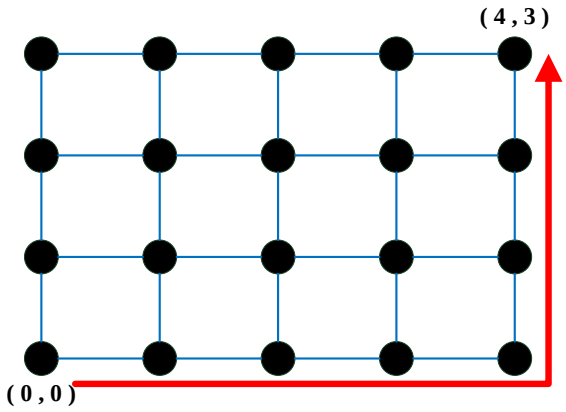


圖 2-8　確定性尋徑演算法例子

2. 模糊性尋徑演算法：模糊性尋徑演算法容許資料的傳輸透過不同路徑進行，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。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(4, 3)，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 路由路徑會導致死結情況的發生；反之，若只考慮 X-Y 路由路徑，則是為無死結模糊性尋徑演算法。所以設計或選擇模糊性尋徑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[19][20]。然而，因未考量網路狀態，使得模糊性尋徑演算法易於實現與分析。但實際上，若能考慮當前網路狀態，則可能改善路由效能，因此一般認為模糊性尋徑演算法在實際效果上不如適應性尋徑演算法。在[21]，該作者認為適應性尋徑演算法即使

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。

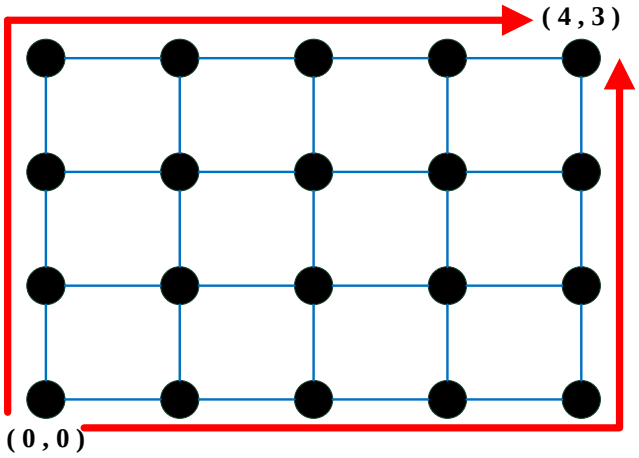


圖 2-9　模糊性尋徑演算法例子

3. 適應性尋徑演算法：為最複雜的尋徑演算法。適應性尋徑演算法可適應於網路交通狀態，並依據網路狀態資訊作為尋徑決策之條件，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 2-10 為例，資料封包於(0, 0)發送至(4, 3)，最初封包依 X-Y 路由路徑到達節點(2, 0)，且於此節點東方輸出鏈結接收到壅塞資訊，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尋徑演算法複雜，且當網路只有輕度壅塞時，資料傳輸的延遲較大。

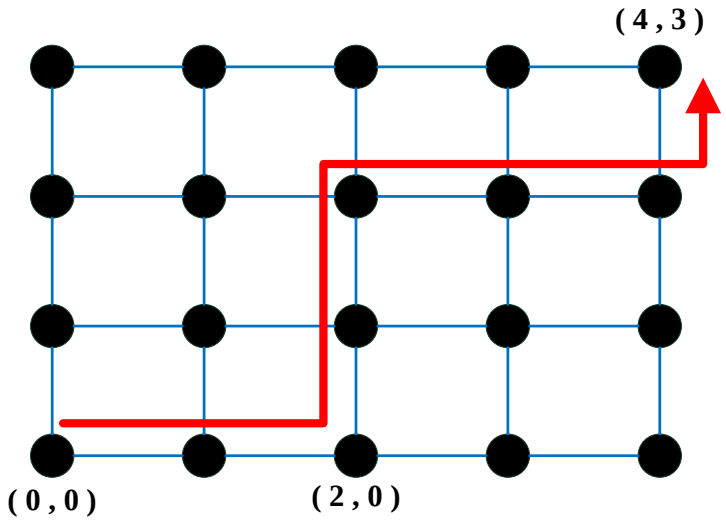


圖 2-10　適應性尋徑演算法例子

局部或全局網路資訊皆可以作為適應性尋徑演算法設計之決策判斷條件。理論上，一個良好的適應性尋徑演算法效能應優越於模糊性尋徑演算法，而實際上則不然，主要原因是許多適應性尋徑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。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[22]。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，適應性尋徑演算法亦可被分類為最短路徑(minimal path)和非最短路徑(non-minimal path)[23]。在低密度(無壅塞)交通流量下，非最短路徑適應性尋徑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，反之最短路徑適應性尋徑演算法則實現了較好的效能結果；而在高密度(壅塞)交通流量下，非最短路徑適應性尋徑演算法的選擇可能是合適的，因它可避開壅塞的鏈結，減少資料傳輸延遲時間。

2.3.2 死結

設計或選擇尋徑演算法，不僅須考慮延遲、功率、傳輸率及可靠度等影響層面，更要求網路無死結(deadlock-free)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。

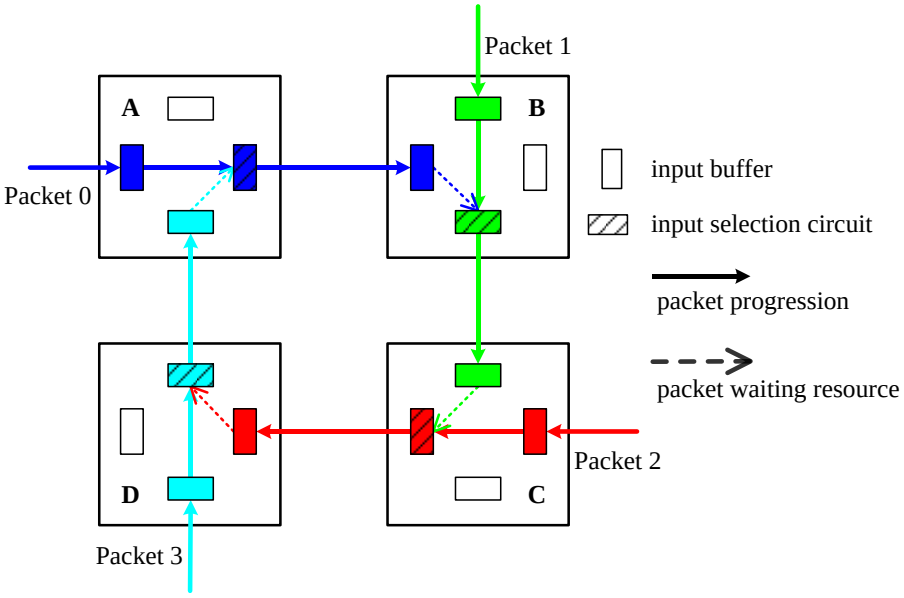


圖 2-11　環狀資源等待路徑造成網路死結的發生

圖 2-11 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)，設計一個尋徑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)，能有效防止網路死結的發生。

2.3.3 轉向模型尋徑演算法

由上節所知，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。無死結尋徑演算法可藉由轉向行為描述之，如圖 2-12 顯示在二維網格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將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。故以下針對晶片網路中較普遍被採用的二維網格拓樸網路之轉向模型尋徑演算法一一作介紹。

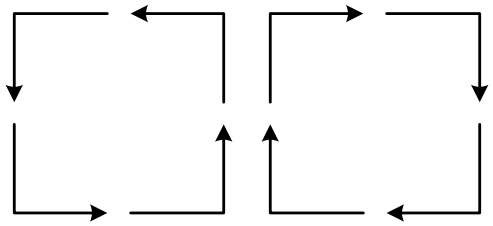
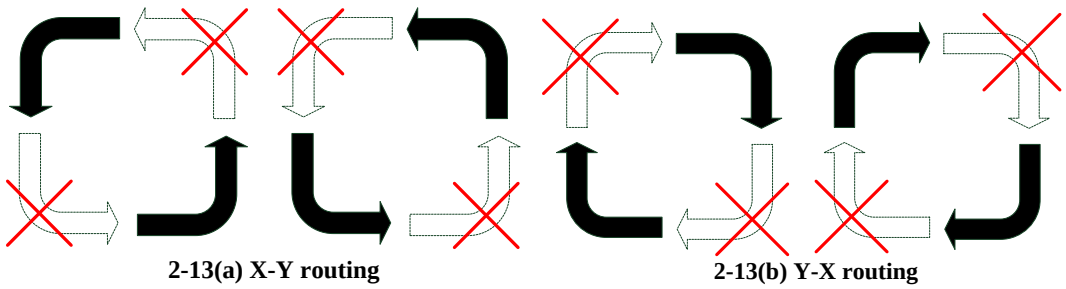


圖 2-12　二為網格拓樸網路中可能的路由轉向

1. X-Y 尋徑演算法[7]：為最簡單且最為普遍被使用的確定性之維序尋徑演算法。資料封包傳輸自起始節點沿最短路徑，其尋徑步驟，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。在圖 2-13(a)可知，資料傳遞移動於東方向與西方向都可允許轉向北方或南方，而資料傳遞移動於南方向和北方向則不允許任何轉向。同理，圖 2-13(b)為 Y-X 尋徑演算法轉向限制示意圖。如此設置轉向限制條件將致使圖 2-12 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。



2-13(a) X-Y routing　　**2-13(b) Y-X routing**

圖 2-13　X-Y 尋徑演算法及 Y-X 尋徑演算法

2. West-First 尋徑演算法：屬於非最短路徑適應性尋徑演算法，且同樣能保證網路無死結[24]。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 2-14 顯示 West-First 尋徑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。

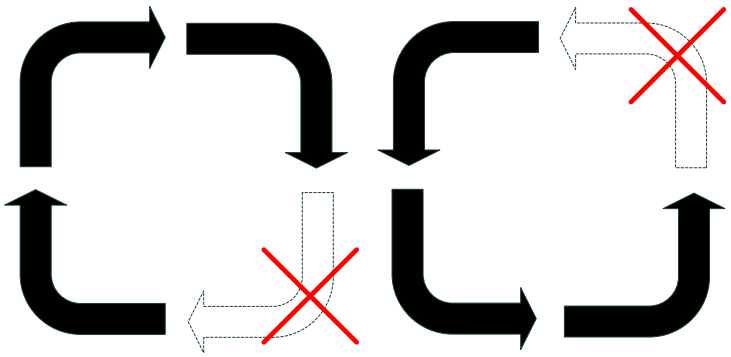
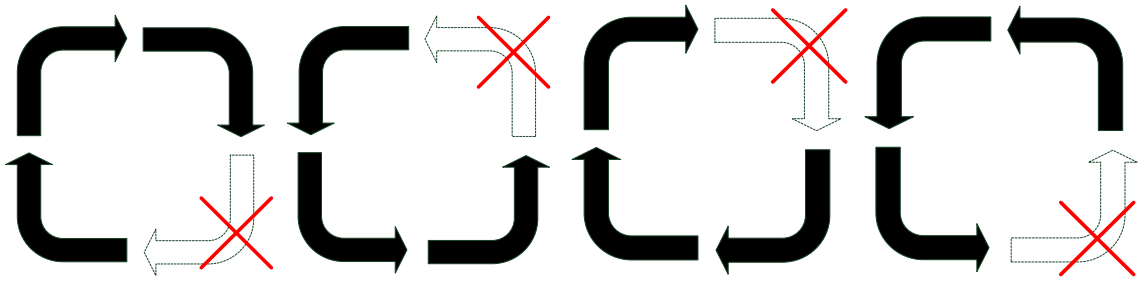


圖 2-14　West-First 尋徑演算法

3. 奇偶尋徑模型(odd-even routing model)：為適應性路由演算法，是另外一種有彈性的二維網格晶片網路尋徑演算法，它可以保證可以避免死結但並不是最短路徑。主要的差異在於奇偶尋徑是由封包位置的關係限制封包的轉向。

以圖 2-15 說明兩個奇偶尋徑模型的規則[25]：

Rule 1：在偶數行的封包不能由東轉向北，在奇數行的封包不能由北轉向西。
Rule 2：在偶數行的封包不能由東轉向南，在奇數行的封包不能由南轉向西。



2-15(a) odd column　　**2-15(b) even column**

圖 2-15　奇偶尋徑模型演算法之禁止轉向規則

上述的管制規則是避免死結現象的環狀資源等待路徑最東邊一行的存在，因此，死結不會有產生的可能。另一方面，有別於其它尋徑模型演算法而言，奇偶尋徑模型演算法藉由資料封包的相關位置限制封包轉向，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避免之。

2.4 訊息交換技術

在這一節中，我們將介紹基本的 NoC 訊息交換技術。第一部份介紹在 NoC 架構下，基本的傳輸單位，第二部分則介紹訊息交換的方法，包括線路式及封包式訊息交換。

2.4.1 封包傳輸單位

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,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，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 2-16)：

- 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)。
- 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。
- 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。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平衡。

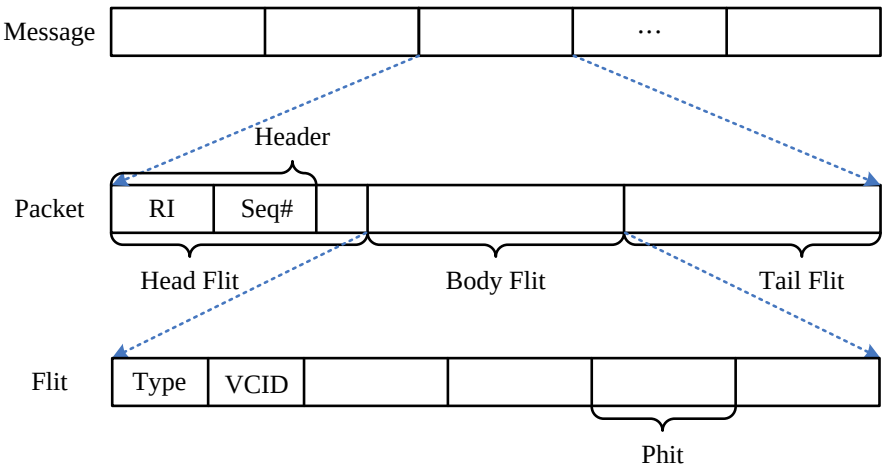


圖 2-16　訊息的組成

2.4.2 訊息交換技術

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[26]。以下分別敘述兩種主要的訊息交換方法。

1. 線路式訊息交換

線路式訊息交換(circuit switching)是一種bufferless的流量控制形式。當一個message被發送時，這個message的head flit記錄從發送端到接收端的所有路徑，若head flit抵達接收端而沒有遇到任何衝突，接收端則發送一個回應(acknowledge)訊號給發送端，緊接著其後的資料將沿著同樣的路徑開始被傳送與接收。相反的，假如傳輸路徑已經被另外一個線路使用，一個負的回應訊號會被傳回發送端，當另一個線路的tail flit傳送完成後，被保留的路徑才會釋放給其他線路使用。

由於完整的路徑頻寬可被電路使用，數據傳輸非常有效率。資料不需要置放於路由器的緩衝區(buffer)當中，如此一來，便可減少路由器的電路面積。此種方式可適

用於資料傳輸頻繁或溝通模式在傳送端和接收端之間為相對靜態時[27]。然而，線路式訊息交換在路徑建立與釋放上較為浪費頻寬，若 NoC 逐漸擴充時，此一方式較不具有好的延展性，因此在先前的 NoC 研究當中，較少使用線路式訊息交換[17]。

2. 封包式訊息交換

不同於線路式訊息交換，在封包式訊息交換(packet switching)的模式下，所傳送的資料都被分割成小封包，每個封包皆記錄目的地的位址，但可能經由不同的路徑抵達接收端，接收端可再依據封包內的序列號將訊息組合起來。在這種情況下，資源分配是每個節點隨時都可以用網路資源，輪流共用，輪流發送封包。然而，在網路壅塞的情況下，必須配合良好的流量控制(flow control)機制，才能達到良好的傳輸效率，否則，其傳輸效率將會低於線路式訊息交換方式。以下，我們介紹幾種常見的封包式訊息交換方法：

(1)儲存轉發流量控制

儲存轉發流量控制(Store and Forward)的特性是一個節點沿著一條路徑等到整個封包被完整的接收，然後再開始試著將封包傳送到下一個節點，因此在封包可以被傳送到下一個router之前整個封包會先儲存在一個輸入緩衝區(input buffer)內，然後在output channel及下一個router的input buffer都可用的情況下，依據packet header的相關資訊，將封包傳送到下一個router。Store and Forward最主要的缺點是網路的延遲時間(network latency)非常高。這個延遲將是連續性的狀態，其時間如圖2-17所示。整體延遲可由公式$(L/BW)\times D$表示，L為封包長度，BW是通道頻寬，D為起點到終點之間的路徑長度。

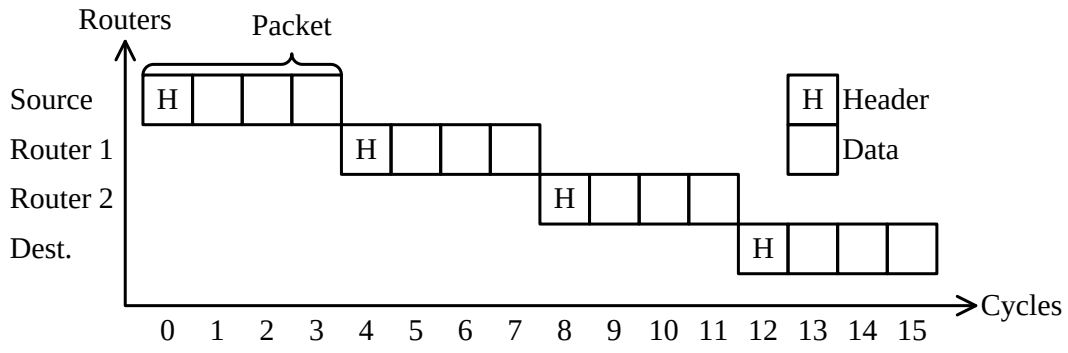


圖 2-17　Store and Forward 的概念

(2)虛擬直通流量控制

虛擬直通(virtual cut-through)流量控制[28]改善了儲存轉發流量控制所造成的資料傳輸延遲，原因是為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，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。

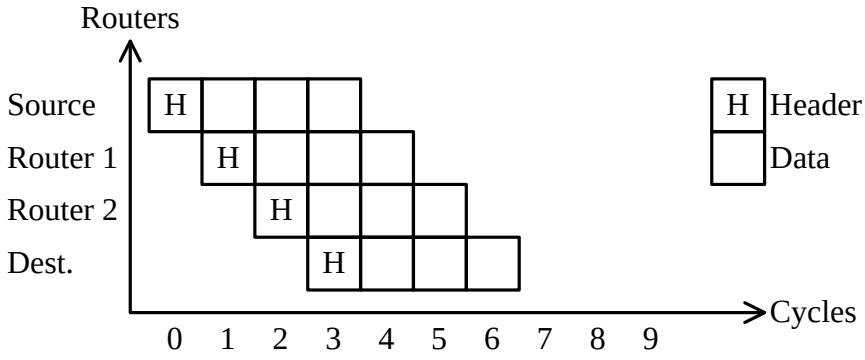


圖 2-18 虛擬直通流量控制概念

圖2-18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2-17及圖2-18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。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、功率消耗及面積開銷，使其在有限面積的晶片網路設計上難以獲得適用。

(3)蟲洞流量控制

蟲洞(wormhole)流量控制[14]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 2-19 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 所在路由節點滿足上個項目符號中條件時，同一封包中其它 flit 才可再次向前移動傳輸。

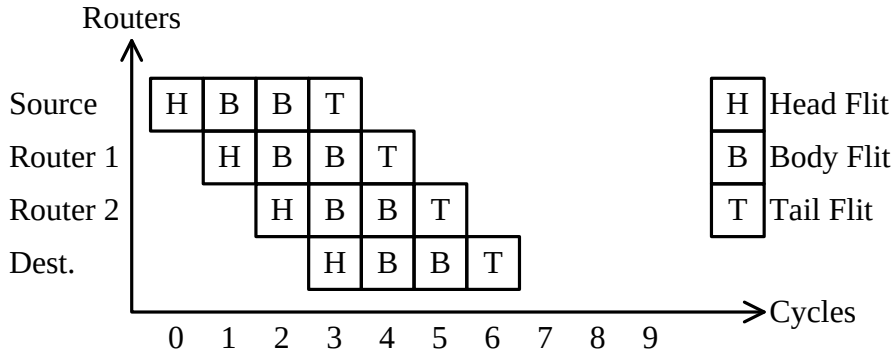


圖 2-19　蟲洞流量控制範例

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，並且大幅提高了資料的傳輸效率，但須注意通道阻塞的問題。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 2-20，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。

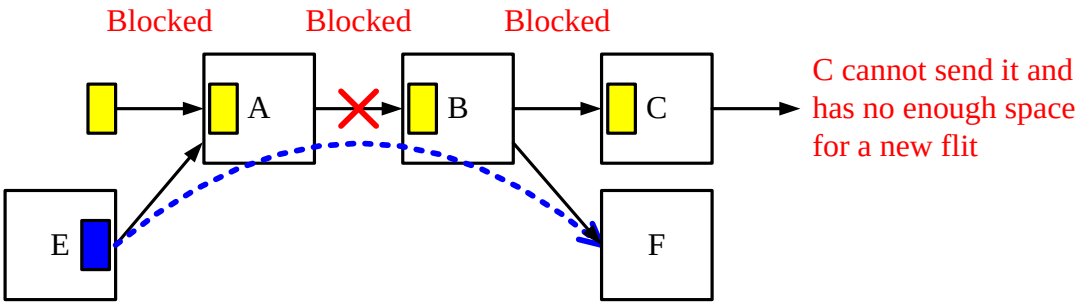


圖 2-20　蟲洞流量控制中通道阻塞問題範例

2.5 虛擬通道

虛擬通道(Virtual Channel, VC)是一種用於晶片網路路由器常見的設計。每個通道藉由提供多個緩衝器(buffers)，virtual channels 分享原通道的實體頻寬(physical channel)，它可以避免死結並降低 head-of-line blocking 的錯誤以增加輸出量[30]。在圖 2-21 的左側，封包 P1 在佇列的開頭，會影響到在相同序列中封包 P2 的前進。若使用虛擬通道，如圖 2-21 的右側，封包 P2 可以選擇其他的虛擬通道前進，而不會受到封包 P1 的影響。實作上，擁有虛擬通道的路由器較為複雜，主因是必須增加虛擬通道仲裁的控制電路。

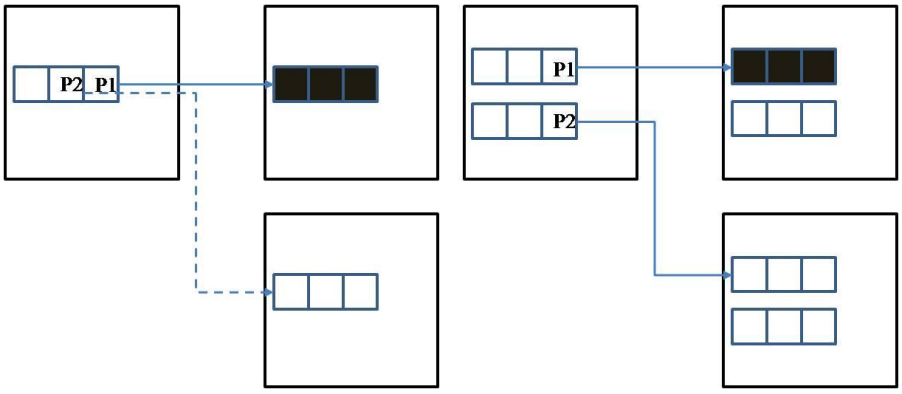


圖 2-21　藉由虛擬通道降低 head-of-line blocking 的影響

圖 2-22 與圖 2-23 是說明虛擬通道如何減緩 wormhole 尋徑通道阻塞問題的另一個例子，圖 2-22 封包因為封包 P2 正在使用介於路由器 1 與路由器 2 之間的通道，所以封包 P1 不能使用，即使 P2 在路由器 2 阻塞而沒有使用路由器 2 的頻寬。圖 2-23 每個路由器擁有

2 個虛擬通道，封包 P1 可以取得路由器 2 的第 2 個虛擬通道，而不受到封包 P2 阻塞的影響。

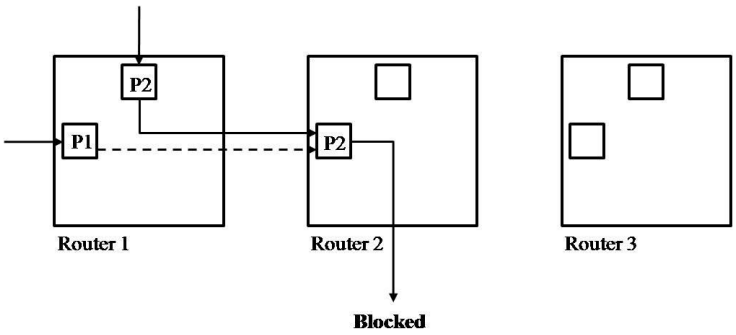


圖 2-22　無虛擬通道之 wormhole

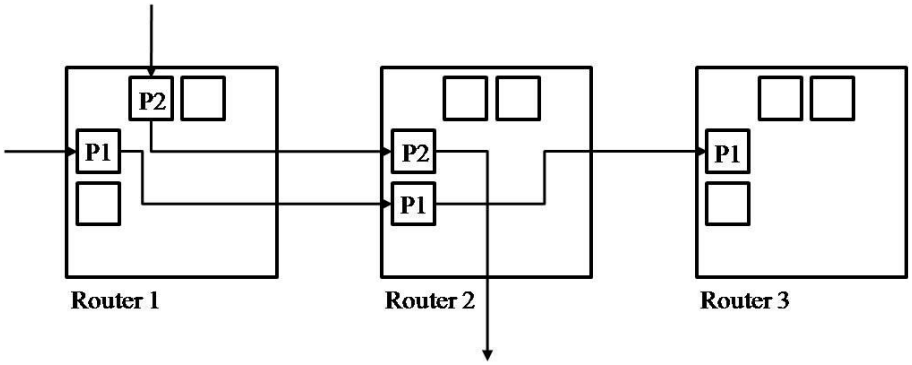


圖 2-23　具備虛擬通道之 wormhole

2.6 路由器架構

圖2-24為一個具有虛擬通道的路由器架構[31]。在圖2-24中，路由器有P個輸入埠與P個輸出埠，每個輸出入埠有V個虛擬通道。在傳輸的過程當中一個packet會被分成一個head flit、一個tail flit及數個body flits。路由器也可設計為具備管線(pipeline)的方式進行傳輸[30]以提高傳輸量。以下，我們以圖2-24路由器架構進行NoC資料傳輸特性做介紹。

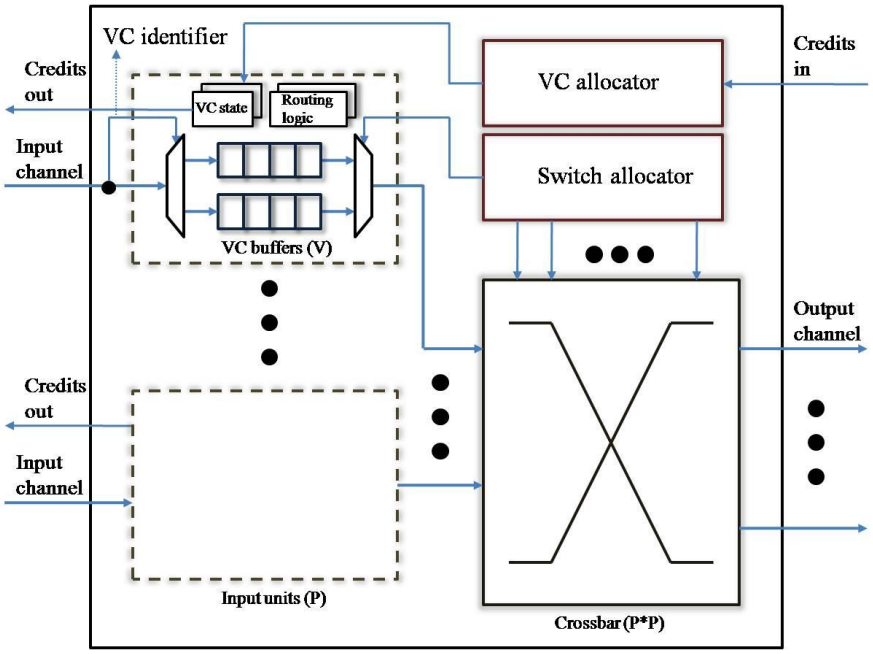


圖 2-24　具備虛擬通道之路由器架構

1. Routing:

當 flit 抵達路由器時，此 flit 的虛擬通道 ID 欄位會被路由器的 VC identifier 解碼，然後這個 flit 會被置放於適當的虛擬通道佇列。若有任何 flit 被置放在 VC_n(n 為虛擬通道編號)中，此 VC_n 的尋徑電路將會檢查 flit 的尋徑欄位並傳回一些有效的輸出虛擬通道資訊，以確認此 flit 可以使用該虛擬通道。

2. Virtual-Channel Allocation:

在尋徑步驟後，VC_n 將會要求 VC Allocator，VC Allocator 從每個 VC_n 收集全部的要求並回覆哪些要求是允許使用輸出虛擬通道。

3. Switch Allocation:

在 VC_n 分配完輸出虛擬通道後，VC_n 向 Switch Allocator 發送要求，假如 VC_n 指定的輸出虛擬通道有空閒的緩衝區，則經由 crossbar 存取需要的輸出埠。

4. Crossbar Traversal:

在 switch allocation 後，flit 可以通過 crossbar 到適當的輸出埠，然後朝向下一個路由器前進並更新 ID 欄位。當 head flit 離開 VC_n 後，相同 packet 的 body flits 便能開始傳送到 Switch Allocation 的步驟，一旦 tail flit 被允許通過 crossbar 通道，它必須通知分

配器，以釋放先前所保留的輸出虛擬通道。

5. Transport Latency

傳輸延遲定義為產生一個 message header 進入起始節點開始和最後一個 tail flit 在目的節點被接收之間的時間（時脈週期）[32]。為了從起始節點到達目的節點，flit 傳輸經過一個由路由器和傳輸線路組成的路徑，根據不同的起始節點、目的節點與尋徑算法，每個封包可能會有不同的傳輸延遲。對於封包 i，傳輸延遲 L_i 為

$$L_i = sender\ overhead + transport\ latency + receiver\ overhead$$

若我們假設 P 是到達目的節點的封包總數，L_i 是每個 message 的延遲，i 介於 1 到 P 之間，平均延遲 L_{avg} 可以表示為

$$L_{avg} = \frac{\sum_{i=1}^{P} Li}{P}$$

6. Energy

當 flit 在內部網路中傳輸時，內部連線以及路由器的電路都會造成能量消耗。flit 從起始節點開始，必須經過多個路由器以及內部連線才能達到目的節點。因此每個 flit 經過一個中間節點(hop)的能量消耗定義為

$$E_{hop} = E_{router} + E_{interconnect}$$

這裡的 E_{router} 和 $E_{interconnect}$ 主要根據電容值與訊號切換次數及每一段的內部接線而定義，定義如下

$$E_{router} = \alpha_{router} C_{router} V_{dd}^2$$

$$E_{interconnect} = \alpha_{interconnect} C_{interconnect} V_{dd}^2$$

α_{router}及C_{router}是路由器的訊號運作頻率與總電容，而$\alpha_{interconnect}$及$C_{interconnect}$則是線段的訊號運作頻率與電容。傳送封包的能量消耗是由 n 個 flits 經過 h 個 hops 組成，可以被計算成如下

$$E_{packet} = n \sum_{j=1}^{h} E_{hop,j}$$

假設 P 為傳送的封包總數，讓 E_{packet} 為第 i 個封包的能量消耗，i 介於 1~P 之間，則每個封包的平均能量消耗為

$$E_{ave} = \frac{\sum_{i=1}^{P} E_{packet_i}}{P} = \frac{\sum_{i=1}^{P} (n \sum_{j=1}^{h} E_{hop,j})}{P}$$

第三章 多層次網格晶片網路設計

在此章節中，將介紹本論文的主要架構以多層次網格為基礎之晶片網路設計，首先介紹三種多層次二維網格架構網路拓樸之分類，接著介紹尋徑演算法應用於此架構中，最後我們將會介紹在此架構中所使用的路由器。

3.1 第二層網格建構方式與分類

在建構第二層網格之前，我們先定義子網格(sub-mesh)與中心節點(central node)。子網格為一個3x3的mesh架構，如圖3-1所示，在子網格當中，我們將座標為(1,1)的節點標示為中心節點。中心節點被用來建構第二層網格，藉由子網格的拓展，並將鄰近的中心節點互相連接，便可形成第二層網格。由於中心節點必須兼具第一層網格與第二層網格的路由器功能，因此中心節點的內部電路架構與其他節點不同，必須由原來5 ports的路由器(router)，增加成9 ports的路由器。關於路由器的內部電路架構，我們將在3.7節詳細介紹。

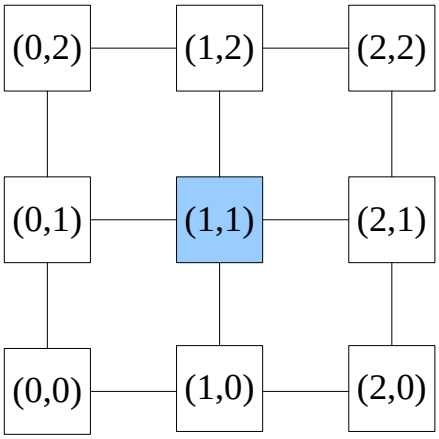


圖 3-1　子網格及中心節點

中心節點對於第一層網格與第二層網格而言都是普通的路由器，藉由增加額外的4 ports到原來5 ports的路由器，我們就可以利用第二層mesh進行資料傳輸。然而，決定資料

封包該走第一層網格或第二層網格的尋徑電路，則被包含於中心節點的判斷電路中。

每一個子網格的排列都會造成資料於傳輸資料時所需之延遲與功率，尤其在長距離封包傳輸的表現更有差異性，所以以下將第二層網格拓樸架構分成三類分別以兩個子網格之中心節點間隔一個普通路由器、間隔二個以及間隔三個，並將此三類架構與未改良之架構作傳輸須經過的路由器數量之比較。

3.2 Case 1 建構方式與分類

Case 1為兩個相鄰的子網格中有其中一邊為兩個子網格所共用之類別，如圖3-2所示。圖3-2為一個簡單的5×3網格，在此圖中分別有兩個子網格，名為Submesh-1與Submesh-2，這兩個子網格共用的邊為圖中標示為綠色之路由器區塊，圖中紅色區塊為中心節點路由器。在Case 1中，我們針對各種不同的網格型態加以討論，共可分為四種子類型。

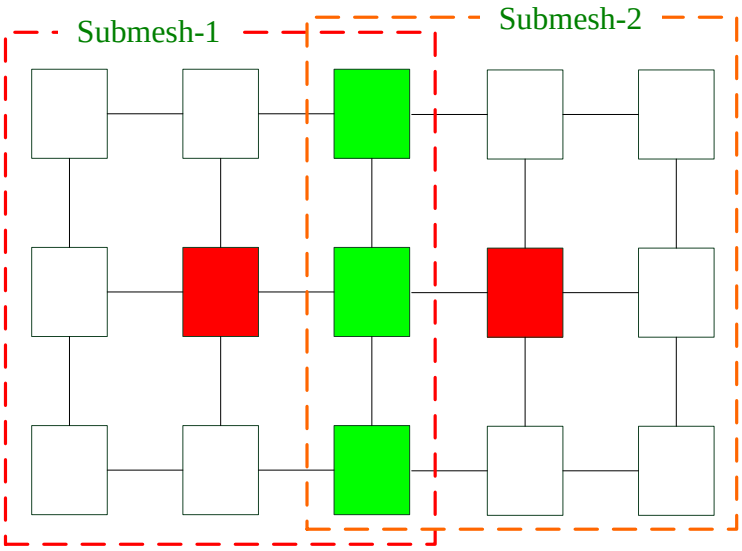


圖 3-2　Case 1 之子網格分佈

1. Type 1A ：在 N=M 且為奇數的情況下，可以找到整個 mesh 最中心的節點，如圖 3-3 所示。將此中心節點作為第二層網格的其中一點，再從最中心節點開始拓展子網格建構出整個網格。

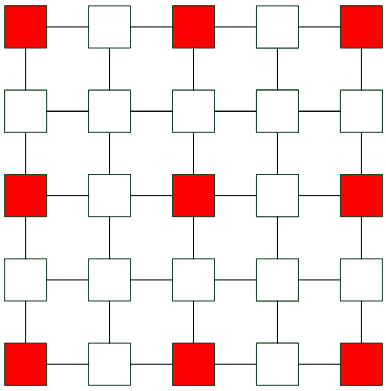


圖 3-3　由子網格拓展成 5×5 Type 1A 的 mesh

2. Type 1B：在 N=M 且皆為偶數時，如圖 3-4 所示，不考慮最外層標示為灰色區塊之路由器，便可以 Type 1A 的方式建構第二層網格。

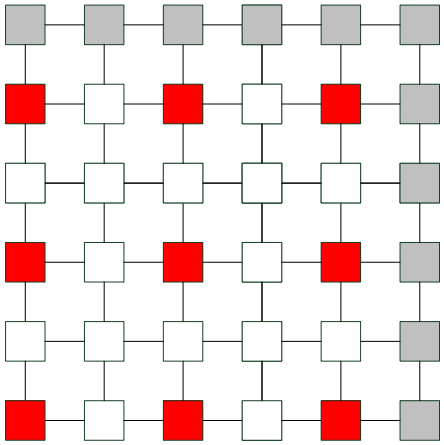


圖 3-4　由子網格拓展成 6×6 Type 1B 的 mesh

3. Type 1C：在 N≠M 且 N 為奇數，M 為偶數的情況下，如圖 3-5 所示。不考慮 M 之最外邊標示為灰色區塊之路由器，便可以 Type 1A 的方式建構第二層網格。

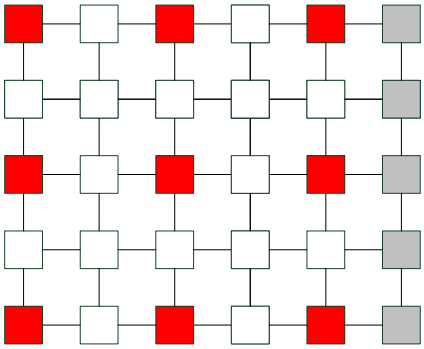


圖 3-5　由子網格拓展成 5×6 Type 1C 的 mesh

4. Type 1D ：在 N≠M 且 N 為偶數，M 為奇數的情況下，如圖 3-6 所示。不考慮 N 之最外邊標示為灰色區塊之路由器，便可以 Type 1A 的方式建構第二層網格

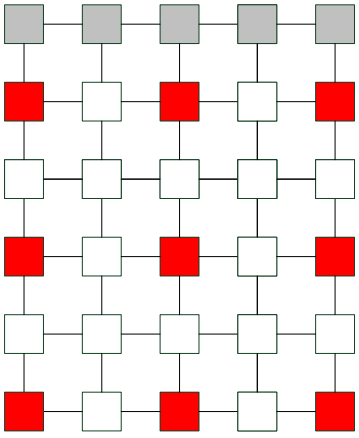


圖 3-6　由子網格拓展成 6×5 Type 1D 的網格

3.3 Case 2 建構方式與分類

Case 2之型態為相鄰的兩個子網格不具交錯邊，如圖3-7所示。

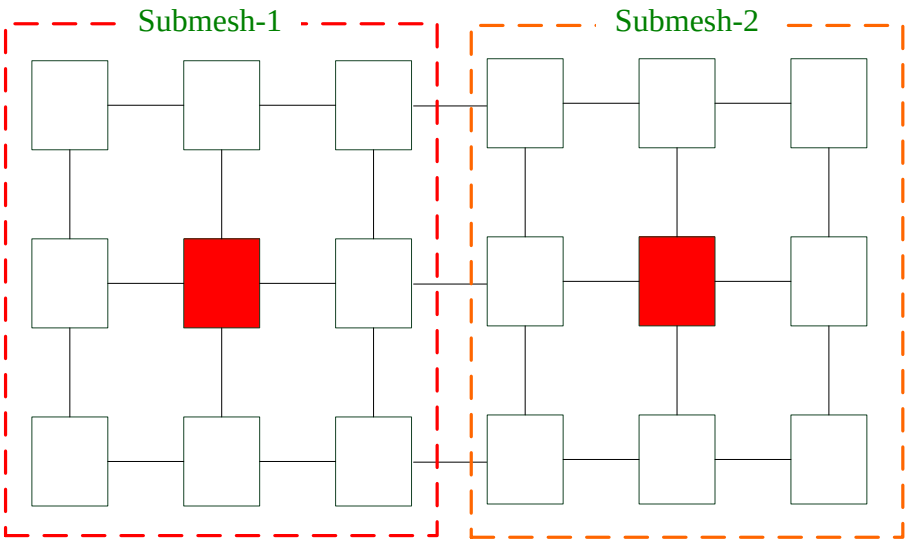


圖 3-7　Case 2 之子網格分佈

接下來我們提出一個適用於case 2，建造第兩層網格拓樸架構的方法。首先，我們考慮一個N×M mesh網路拓樸架構，N×M可以分為下列六種型態：

1. Type 2A：N = M，M、N皆為奇數，如9×9
2. Type 2B：N = M，M、N皆為偶數，如8×8
3. Type 2C：N≠M，M、N皆為奇數，如11×17
4. Type 2D：N≠M，M、N皆為偶數，如20×8
5. Type 2E：N≠M，N為奇數、M為偶數，如9×14
6. Type 2F：N≠M，N為偶數、M為奇數，如14×9

由於mesh網路規模不同，第二層mesh的建構方式也會有所差異。因此，我們必須將Type 2B、Type 2D、Type 2E、Type 2F再細分為各種不同型態，其中Type 2E的結構與Type 2F相同，僅是將其mesh的邊長互換，因此在之後的文章中，我們將Type 2E及Type 2F視為相同的型態，不再額外獨立討論。

1. Type 2B:
   - Type 2B1: {N×M | N=M=(2+6t), t=0,1,2…}
   - Type 2B2: {N×M | N=M=(4+6t), t=0,1,2…}
   - Type 2B3: {N×M | N=M=(6+6t), t=0,1,2…}
2. Type D:
   - Type 2D1: {N×M | N≠M； N=(2+6t) , t=0,1,2…； M=(2+6s), s=0,1,2…}
   - Type 2D2: {N×M | N≠M； N=(2+6t) , t=0,1,2…； M=(4+6s), s=0,1,2…}
   - Type 2D3: {N×M | N≠M； N=(4+6t) , t=0,1,2…； M=(4+6s), s=0,1,2…}
   - Type 2D4: {N×M | N≠M； N=(4+6t) , t=0,1,2…； M=(6+6s), s=0,1,2…}
   - Type 2D5: {N×M | N≠M； N=(6+6t) , t=0,1,2…； M=(6+6s), s=0,1,2…}
   - Type 2D6: {N×M | N≠M； N=(6+6t) , t=0,1,2…； M=(4+6s), s=0,1,2…}
   - Type 2D7: {N×M | N≠M； N=(6+6t) , t=0,1,2…； M=(2+6s), s=0,1,2…}
3. Type E:
   - Type 2E1: {N×M | N≠M； N=1,3,5…； M =(2+6t), t=0,1,2…}
   - Type 2E2: {N×M | N≠M； N=1,3,5…； M =(4+6t), t=0,1,2…}
   - Type 2E3: {N×M | N≠M； N=1,3,5…； M =(6+6t), t=0,1,2…}
4. Type F:
   - Type 2F1: {N×M | N≠M； N=(2+6t), t=0,1,2…； M =1,3,5…}
   - Type 2F2: {N×M | N≠M； N=(4+6t), t=0,1,2…； M =1,3,5…}
   - Type 2F3: {N×M | N≠M； N=(6+6t), t=0,1,2…； M =1,3,5…}

以下針對各種子型態進行網格建構介紹：

1. Type 2A: 在N=M且為奇數的情況下，可以找到整個mesh最中心的節點，如圖3-8所示。將此中心節點作為第二層網格的其中一點，再從最中心節點開始拓展子網格建構出整個網格。

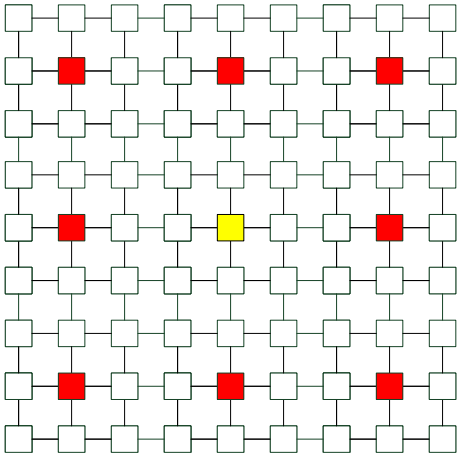


圖 3-8　由子網格拓展成 9×9 Type 2A 的網格

2. **Type 2B2:** 在N=M且皆為偶數，M=N=4+6t時網格角落的4個頂點必為中心節點，如圖3-9所示，藉由這4個中心節點可以拓展出整個第二層網格。

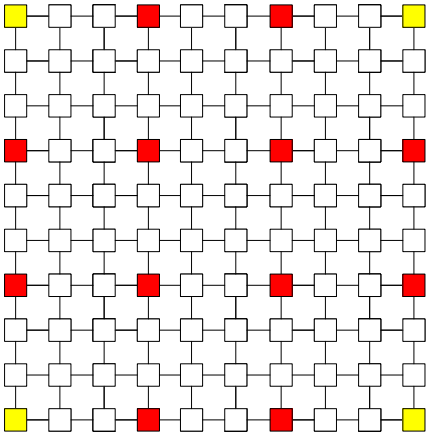


圖 3-9　由子網格拓展成 10×10 Type2B2 的網格

3. **Type 2B3:** 在N=M且皆為偶數，M=N=6+6t時，如圖3-10所示，不考慮最外圍的路由器，便可用Type 2B2的方式建構第二層網格。

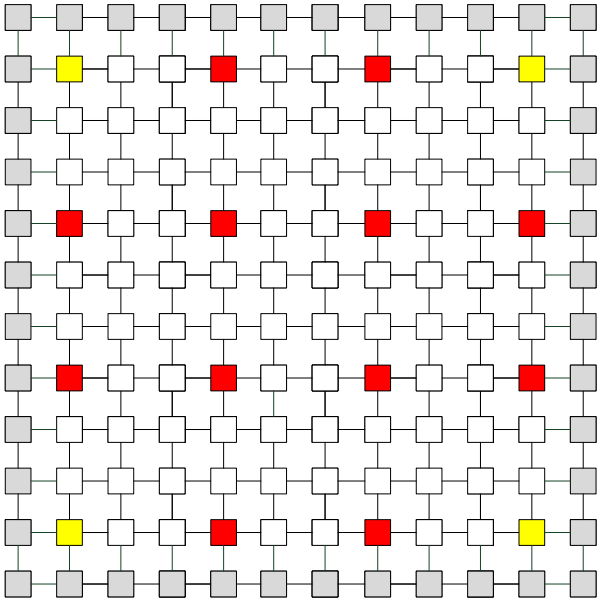


圖 3-10　由子網格拓展成 12×12 Type 2B3 的網格

4. **Type 2B1:** 在N=M且皆為偶數，M=N=2+6t時，如圖3-11示可以找到mesh最中心的4個路由器，然後如圖可將整個網格分割成4個Type 2B2或Type 2A，再利用對應之Type建構第二層網格。

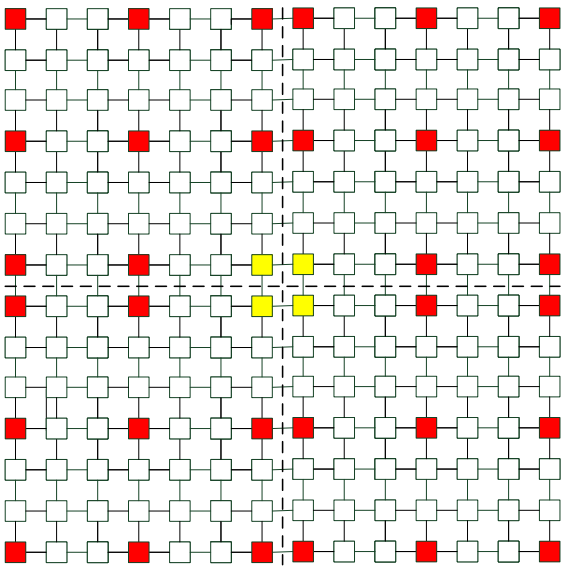


圖 3-11　由子網格拓展成 14×14 Type 2B1 的網格

5. **Type 2C:** 在N≠M且N和M皆為奇數，如圖3-12所示，如Type 2A的方式，先找出mesh中最中心的節點，再以此為中心節點拓展子網格建構出第二層網格架構。

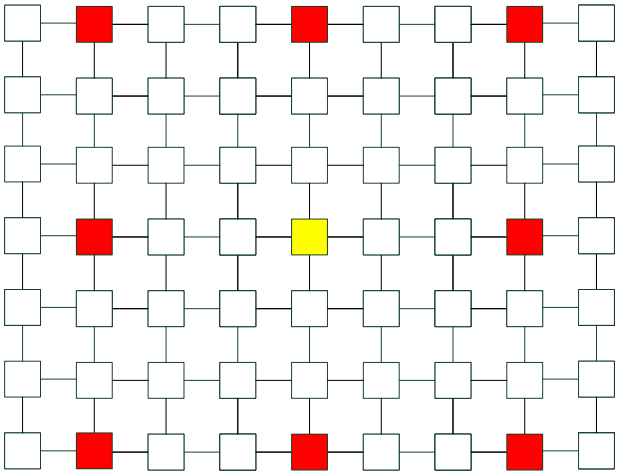


圖 3-12　由子網格拓展成 7×9 Type 2C 的網格

6. **Type 2E2:** 在N≠M且N為奇數M為偶數，M=4+6t的情況下，如圖3-13所示，在奇數邊的中心點必為中心節點，再由這2個中心節點拓展成整個網格。Type 2F2的型態亦可依此方式建構。

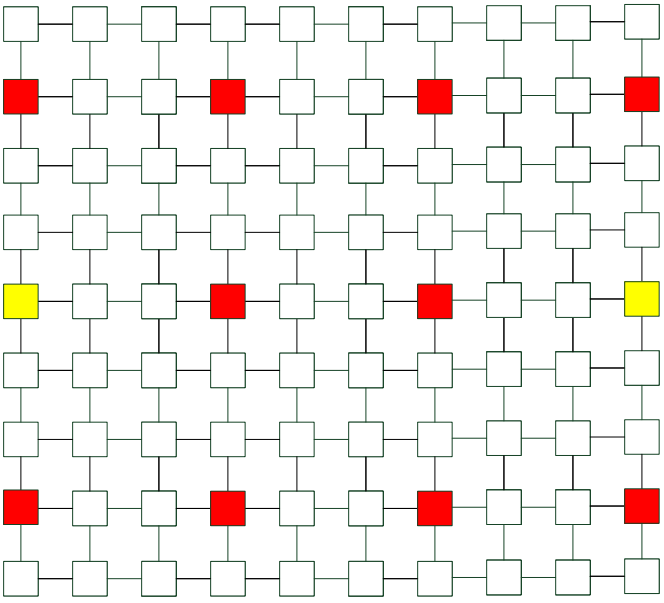


圖 3-13　由子網格拓展成 9×10 Type 2E2 的網格

7. **Type 2E3:** 在N≠M且N為奇數M為偶數，M=6+6t時，如圖3-14所示，不考慮左右兩排之路由器，並使用相同的t代入Type 2E2後，便可得到第二層網格之建構。Type 2F3型態可將M,N對調後，依此方式建構。

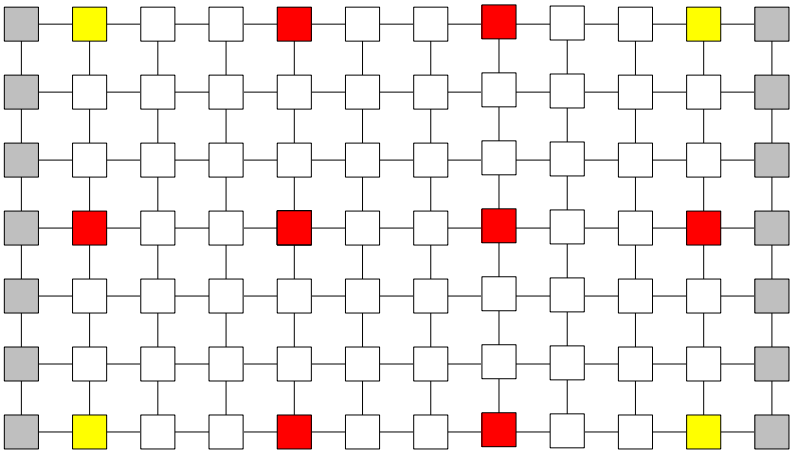


圖 3-14　由子網格拓展成 7×12 Type 2E3 的網格

8. **Type 2E1:** 在N≠M且N為奇數M為偶數，M=2+6t的情況下，如圖3-15所示，可以找到網格最中心的2個路由器，然後如圖可將整個網格分割成2個Type 2E2或Type 2C，再依照Type 2E2或Type 2C的方式建構第二層網格。Type 2F1型態可將M,N對調後，依此方式建構。

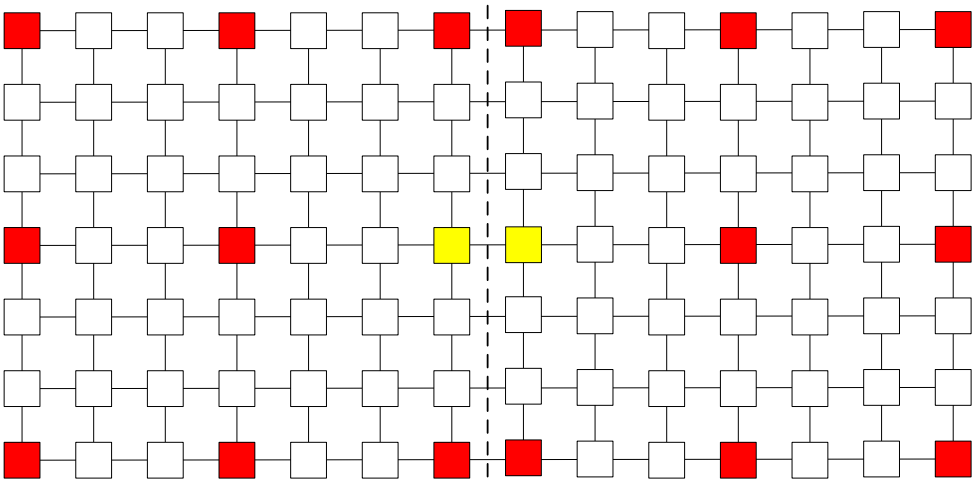


圖 3-15　由子網格拓展成 7×14 Type 2E1 的網格

9. **Type 2D3:** 在N≠M且N和M皆為偶數，N=4+6t、M=4+6s的情況下，如圖3-16所示，整個網格角落的4個頂點必為中心節點，再由這4個中心節點拓展成第二層網格。

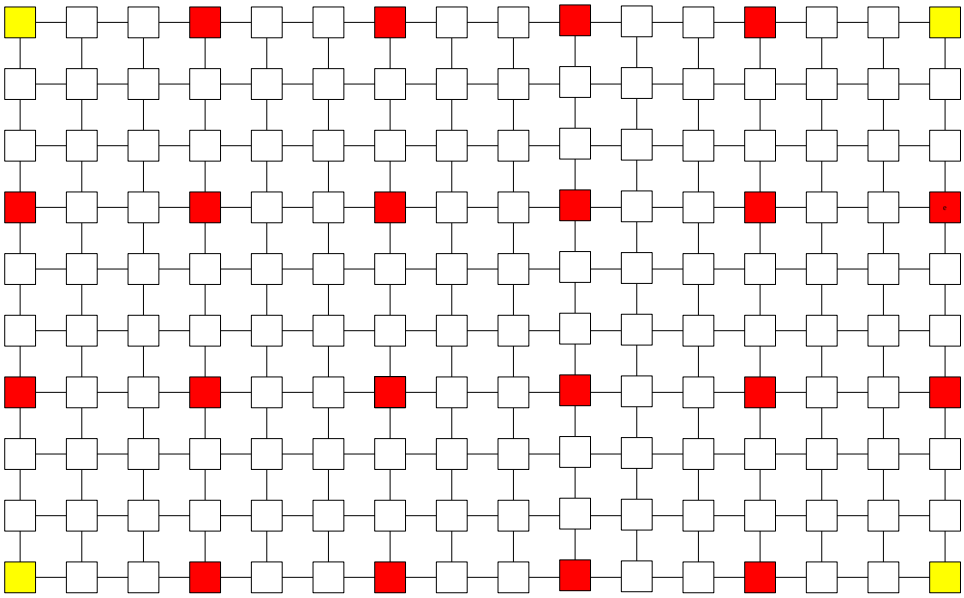


圖 3-16　由子網格拓展成 10×16 Type 2D3 的網格

10. **Type 2D4:** 在N≠M且N和M皆為偶數，N=4+6t、M=6+6s的型態下，如圖3-17所示，不考慮左右兩排節點，便能以Type 2D3的方式建構第二層網格。

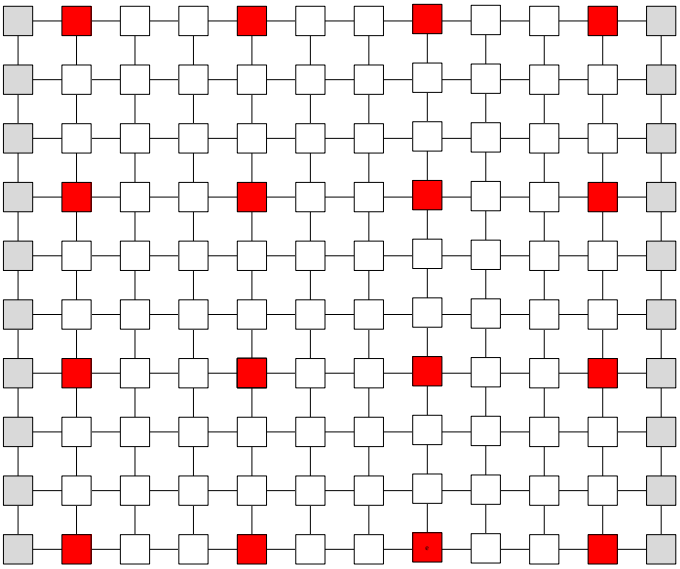


圖 3-17　由子網格拓展成 10×12 Type 2D4 的網格

11.Type 2D1: 在N≠M且N和M皆為偶數，N=2+6t、M=2+6s的情況下，如圖3-18示，可以找到mesh最中心的4個路由器，然後將整個網格成4個Type 2D3、Type 2E2或Type 2F2，再利用Type 2D3、Type 2E2或Type 2F2的方式建構，便可拓展出整個第二層網格架構。

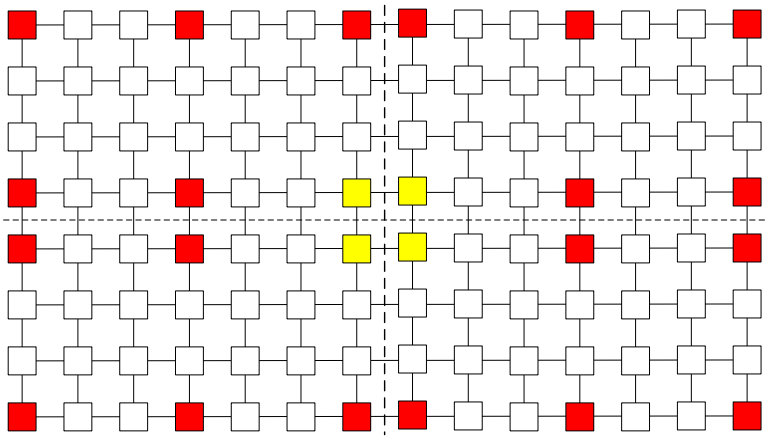


圖 3-18　由子網格拓展成 8×14 Type 2D1 的網格

12.Type 2D2: 在N≠M且N和M皆為偶數，N=2+6t、M=4+6s的條件下，如圖3-19所示，在N=2+6t的一邊可以找到中間的兩個點，可將整個網格分割成2個N=4+6t或奇數、M=4+6s的網格，再依照Type 2D3或Type 2E2的方式建構出整個網格架構。

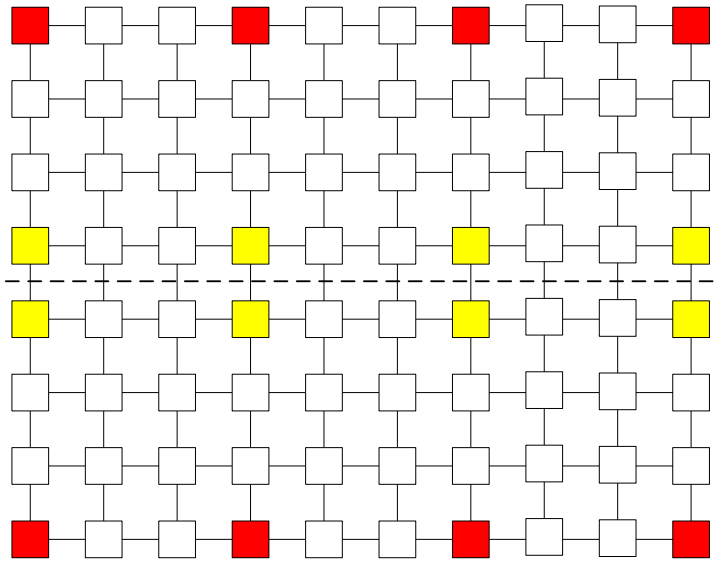


圖 3-19　由子網格拓展成 8×10 Type 2D2 的網格

13. **Type 2D5:** 在N≠M且N和M皆為偶數，N=6+6t、M=6+6s時，如圖3-20所示，不考慮最外圈的節點，以Type 2D3的方式便能建構出第二層網格結構。

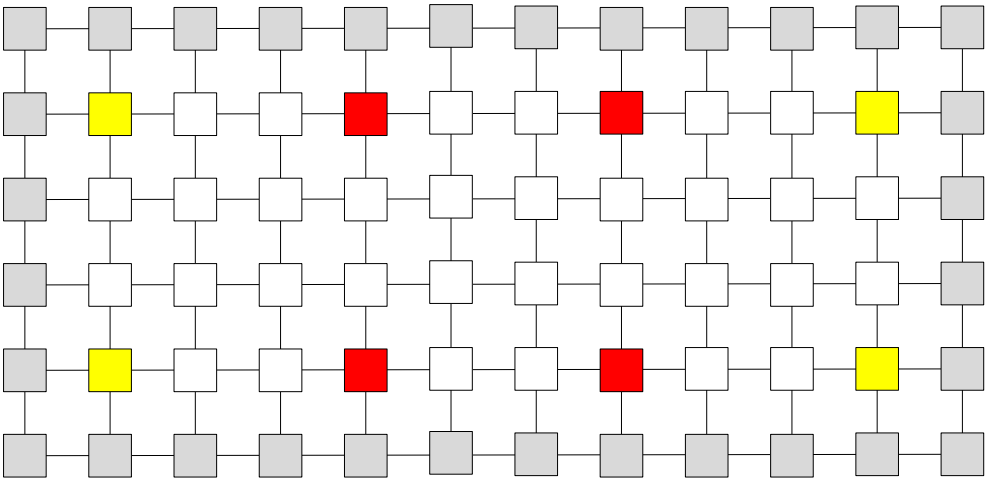


圖 3-20　由子網格拓展成 6×12 Type 2D5 的網格

14. **Type 2D6:** 在 N≠M 且 N 和 M 皆為偶數，N=6+6t、M=2+6s 時，如圖 3-21 所示，在 M=2+6s 可以找到中間的兩個點，可將整個網格分割成左右 2 個網格，若 s 為奇數，則利用 Type 2D4 的方式建構，若 s 為偶數，則利用 Type 2F3 的方式建構第二層網格。圖 3-20 是以 Type 2F3 為建構方式之例子。

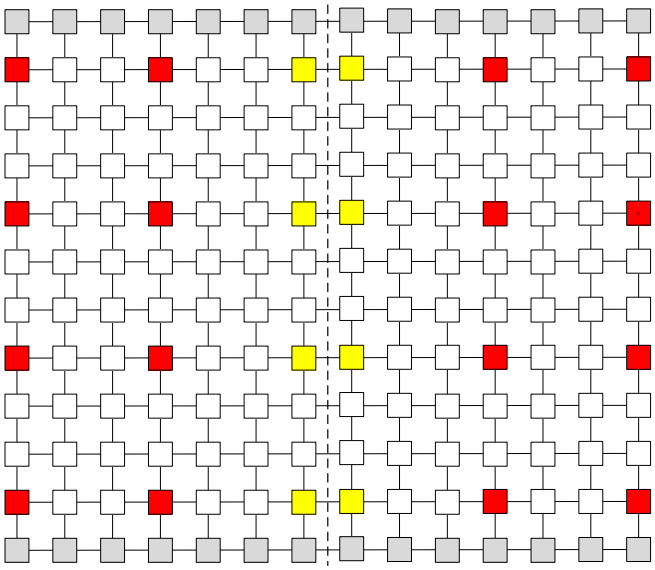


圖 3-21　由子網格拓展成 12×14 Type 2D6 的網格

3.4 Case 3 建構方式與分類

Case 3為兩個相鄰的子網格相隔一排路由器，如圖3-22所示。圖3-22為一個簡單的7×7網格，在此圖中分別有四個子網格，名為Submesh-1、Submesh-2、 Submesh-3與Submesh-4。此四個子網格皆以綠色之路由器區塊相隔，圖中紅色區塊為中心節點路由器。在Case 3中，我們針對各種不同的網格型態加以討論，共可分為四種子類型。

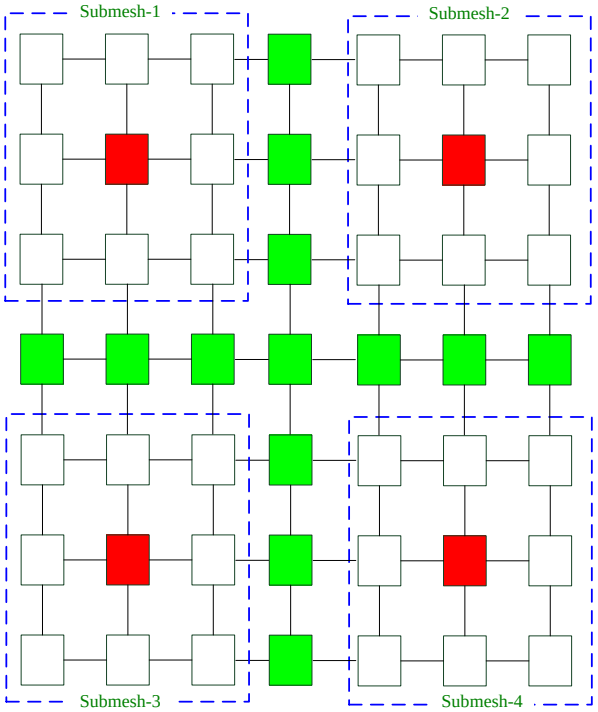


圖 3-22　Case 3 之子網格分佈

接下來我們提出一個適用於 case 3，建造第兩層網格拓樸架構的方法。首先，我們考慮一個 N×M mesh 網路拓樸架構，N×M 可以分為下列四種型態：

1. **Type 3A:** { N mod 4 =3 or 0 & M mod 4 =3 or 0 }
2. **Type 3B:** { N mod 4 =1 or 2 & M mod 4 =1 or 2 }
3. **Type 3C:** { N mod 4 =1 & M mod 4 =3 }
   or { N mod 4 =2 & M mod 4 =3 }
   or { N mod 4 =1 & M mod 4 =0 }
   or{ N mod 4 =2 & M mod 4 = 0 }

4. **Type 3D:** { N mod 4 =3 & M mod 4 =1 }
   or { N mod 4 =0 & M mod 4 =1 }
   or { N mod 4 =3 & M mod 4 =2 }
   or { N mod 4 =0 & M mod 4 = 2 }

由於 mesh 網路規模不同，第二層 mesh 的建構方式也會有所差異。因此，我們必須將 Type 3A、Type 3B、Type 3C、Type 3D 再細分為各種不同型態。

5. **Type 3A:**
   - **Type 3A1:** {N=M | N mod 4 = 3 =M mod 4 }
   - **Type 3A2:** {N≠M | N mod 4 = 0 & M mod 4 = 3 }
   - **Type 3A3:** {N≠M | N mod 4 = 3 & M mod 4 = 0 }
   - **Type 3A4:** {N=M | N mod 4 = 0 =M mod 4 }
6. **Type 3B:**
   - **Type 3B1:** {N=M | N mod 4 = 1 =M mod 4 }
   - **Type 3B2:** {N≠M | N mod 4 = 2 & M mod 4 = 1 }
   - **Type 3B3:** {N≠M | N mod 4 = 1 & M mod 4 = 2 }
   - **Type 3B4:** {N=M | N mod 4= 2 =M mod 4 }
7. **Type 3C:**
   - **Type 3C1:** {N≠M | N mod 4 = 1 & M mod 4 = 3 }
   - **Type 3C2:** {N≠M | N mod 4 = 2 & M mod 4 = 3 }
   - **Type 3C3:** {N≠M | N mod 4 = 1 & M mod 4 = 0 }
   - **Type 3C4:** {N≠M | N mod 4 = 2 & M mod 4 = 0 }
8. **Type 3D:**
   - **Type 3D1:** {N≠M | N mod 4 = 3 & M mod 4 = 1 }
   - **Type 3D2:** {N≠M | N mod 4 = 0 & M mod 4 = 1 }
   - **Type 3D3:** {N≠M | N mod 4 = 3 & M mod 4 = 2 }
   - **Type 3D4:** {N≠M | N mod 4 = 0 & M mod 4 = 2 }

以下針對各種子型態進行網格建構介紹：

1. **Type 3A1:** 在 N=M 且 mod 4 皆為 3 的情況下，可以找到整個 mesh 最中心的節點，如圖 3-23 所示。將此中心節點作為第二層網格的其中一點，再從最中心節點開始拓展子網格建構出整個網格。

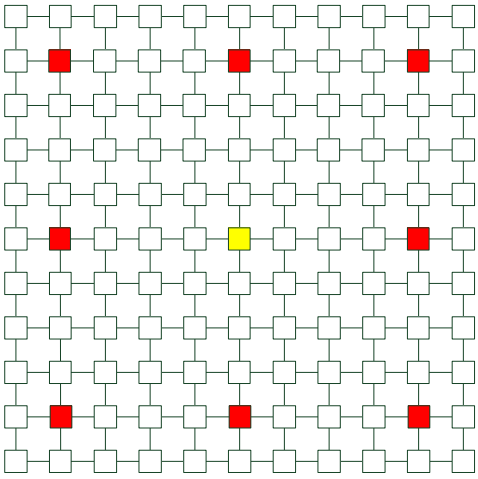


圖 3-23　由子網格拓展成 11×11 Type 3A1 的網格

2. **Type 3A2:** 在N≠M且N mod 4為0及M mod 4為3時，如圖3-24所示，不考慮最上側的節點，以Type 3A1的方式便能建構出第二層網格結構。

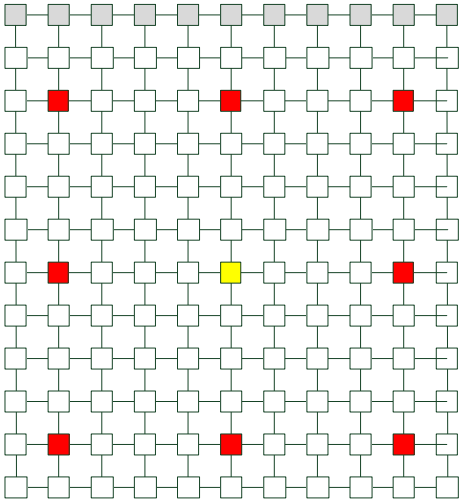


圖 3-24　由子網格拓展成 12×11 Type 3A2 的網格

3. **Type 3A3:** 在 N≠M 且 N mod 4 為 3 及 M mod 4 為 0 時，如圖 3-25 所示，不考慮最右側的節點，以 Type 3A1 的方式便能建構出第二層網格結構。

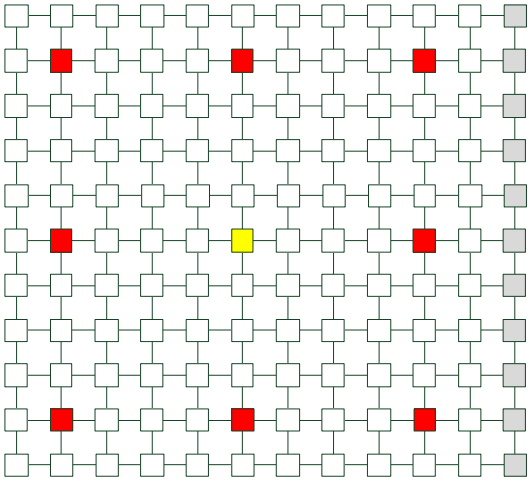


圖 3-25　由子網格拓展成 11×12 Type 3A3 的網格

4. **Type 3A4:** 在 N=M 且 mod 4 皆為 0 時，如圖 3-26 所示，不考慮最上側及最右側的節點，以 Type 3A1 的方式便能建構出第二層網格結構。

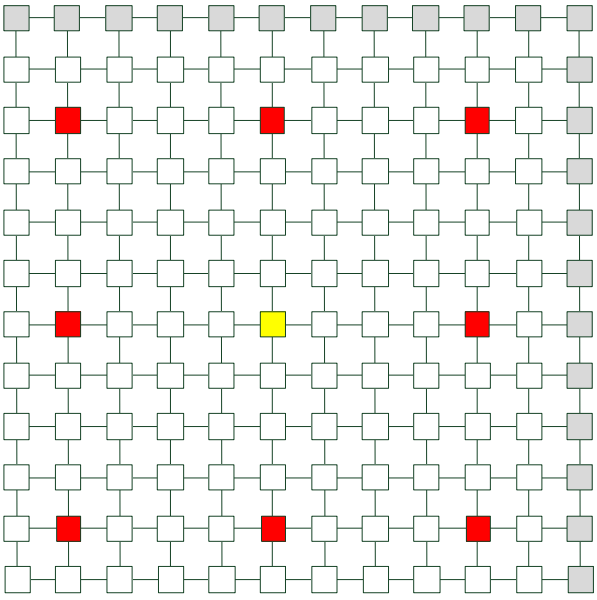


圖 3-26　由子網格拓展成 12×12 Type 3A4 的網格

5. **Type 3B1:** 在 N=M 且 mod 4 皆為 1 時，網格角落的 4 個頂點必為中心節點，如圖 3-27 所示，藉由這 4 個中心節點可以拓展出整個第二層網格。

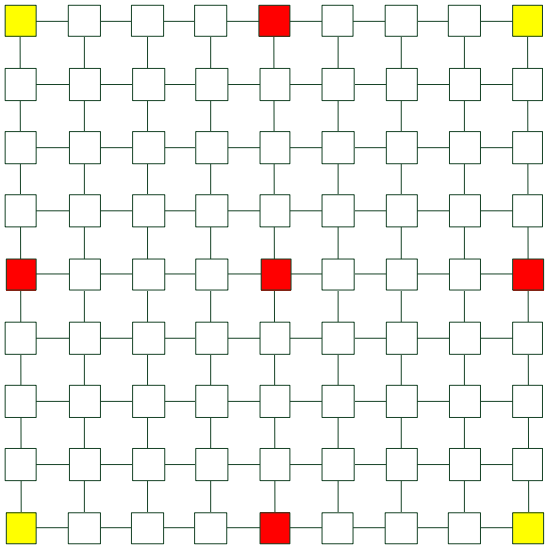


圖 3-27　由子網格拓展成 9×9 Type 3B1 的網格

6. **Type 3B2:** 在 N≠M 且 N mod 4 為 2 及 M mod 4 為 1 時，如圖 3-28 所示，不考慮最上側的節點，以 Type 3B1 的方式便能建構出第二層網格結構。

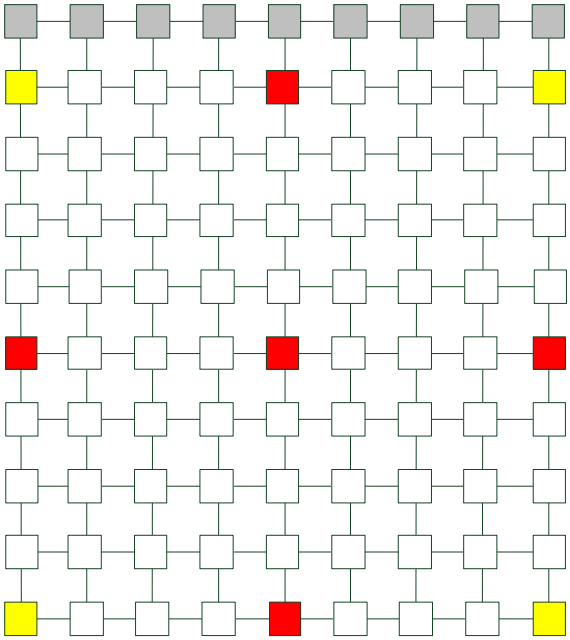


圖 3-28　由子網格拓展成 10×9 Type 3B2 的網格

7. **Type 3B3:** 在 N≠M 且 N mod 4 為 1 及 M mod 4 為 2 時，如圖 3-29 所示，不考慮最右側的節點，以 Type 3B1 的方式便能建構出第二層網格結構。

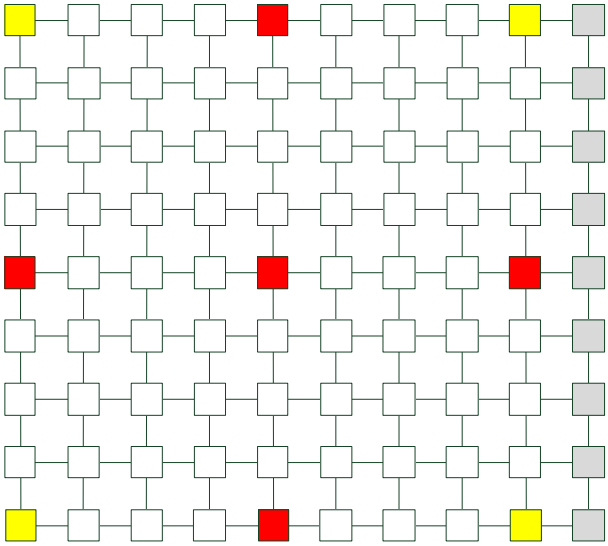


圖 3-29　由子網格拓展成 9×10 Type 3B3 的網格

8. **Type 3B4:** 在 N=M 且 mod 4 皆為 2 時，如圖 3-30 所示，不考慮最上側及最右側的節點，以 Type 3B1 的方式便能建構出第二層網格結構。

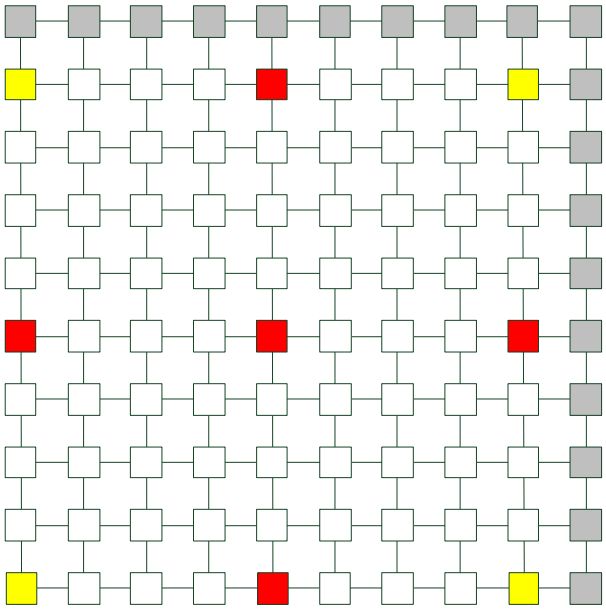


圖 3-30　由子網格拓展成 10×10 Type 3B4 的網格

9. **Type 3C1:** 在 N≠M 且 N mod 4 為 1 及 M mod 4 為 3 時，如圖 3-31 所示，在最上下兩側的中心點必為中心節點，再由這 2 個中心節點拓展成整個網格。

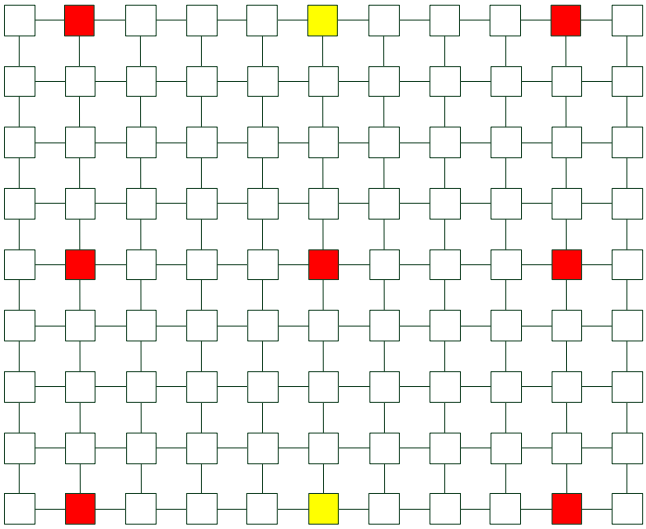


圖 3-31　由子網格拓展成 9×11 Type 3C1 的網格

10. **Type 3C2:** 在 N≠M 且 N mod 4 為 2 及 M mod 4 為 3 時，如圖 3-32 所示，不考慮最上側的節點，以 Type 3C1 的方式便能建構出第二層網格結構。

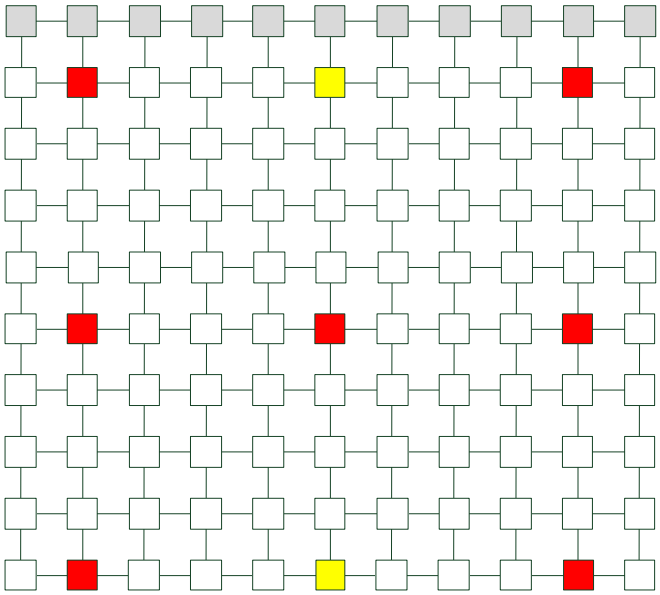


圖 3-32　由子網格拓展成 10×11 Type 3C2 的網格

11. Type 3C3: 在 N≠M 且 N mod 4 為 1 及 M mod 4 為 0 時，如圖 3-33 所示，不考慮最右側的節點，以 Type 3C1 的方式便能建構出第二層網格結構。

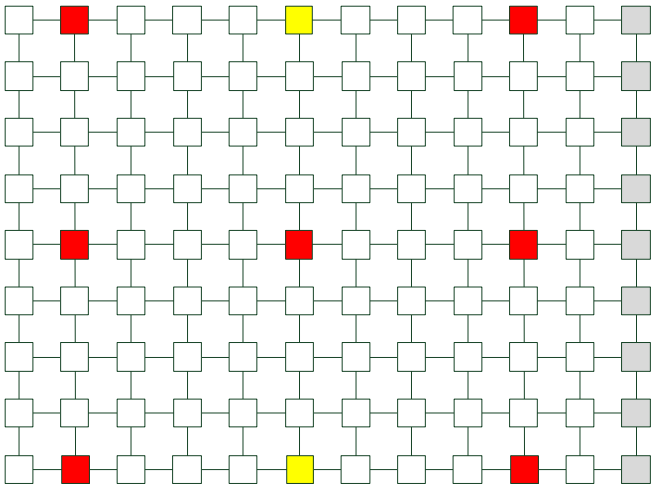


圖 3-33　由子網格拓展成 9×12 Type 3C3 的網格

12. **Type 3C4:** 在 N≠M 且 N mod 4 為 2 及 M mod 4 為 0 時，如圖 3-34 所示，不考慮最上側及最右側的節點，以 Type 3C1 的方式便能建構出第二層網格結構。

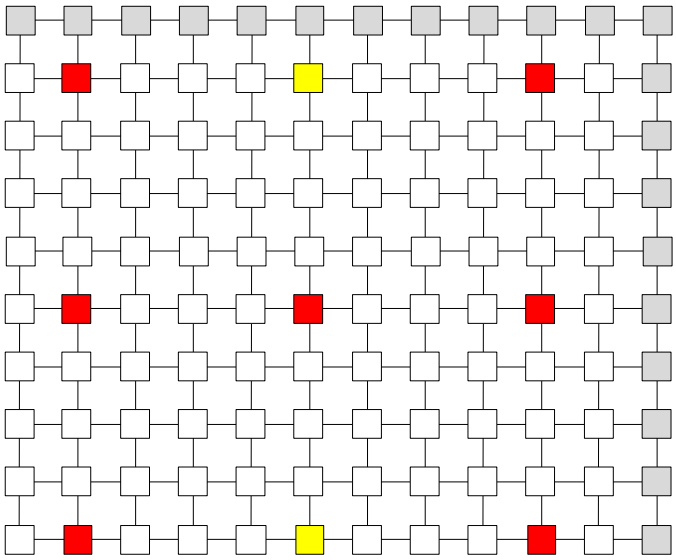


圖 3-34　由子網格拓展成 10×12 Type 3C4 的網格

13. Type 3D1: 在 N≠M 且 N mod 4 為 3 及 M mod 4 為 1 時，如圖 3-35 所示，在最左右兩側的中心點必為中心節點，再由這 2 個中心節點拓展成整個網格。

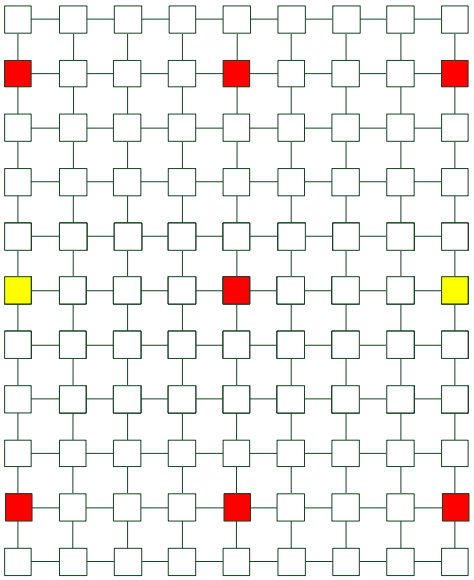


圖 3-35　由子網格拓展成 11×9 Type 3D1 的網格

14. Type 3D2: 在 N≠M 且 N mod 4 為 0 及 M mod 4 為 1 時，如圖 3-36 所示，不考慮最上側的節點，以 Type 3D1 的方式便能建構出第二層網格結構。

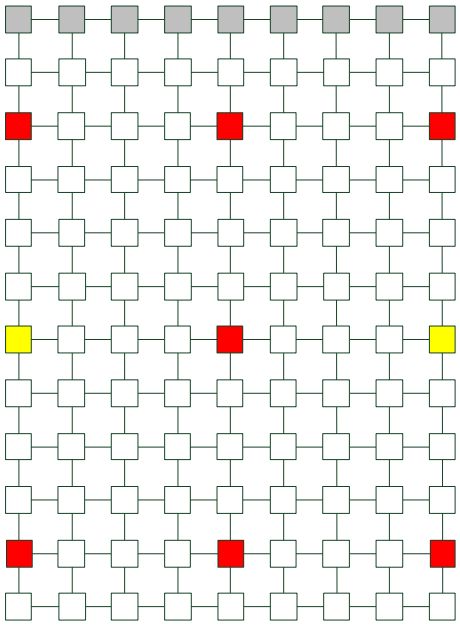


圖 3-36　由子網格拓展成 12×9 Type 3D2 的網格

15. Type 3D3: 在 N≠M 且 N mod 4 為 3 及 M mod 4 為 2 時，如圖 3-37 所示，不考慮最右側的節點，以 Type 3D1 的方式便能建構出第二層網格結構。

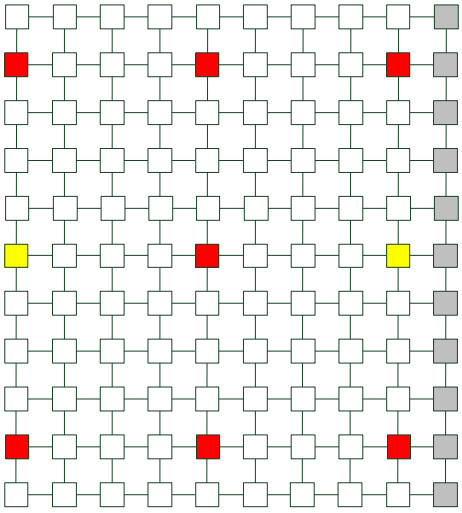


圖 3-37　由子網格拓展成 11×10 Type 3D3 的網格

16. Type 3D4: 在 N≠M 且 N mod 4 為 0 及 M mod 4 為 2 時，如圖 3-38 所示，不考慮最上側及最右側的節點，以 Type 3D1 的方式便能建構出第二層網格結構。

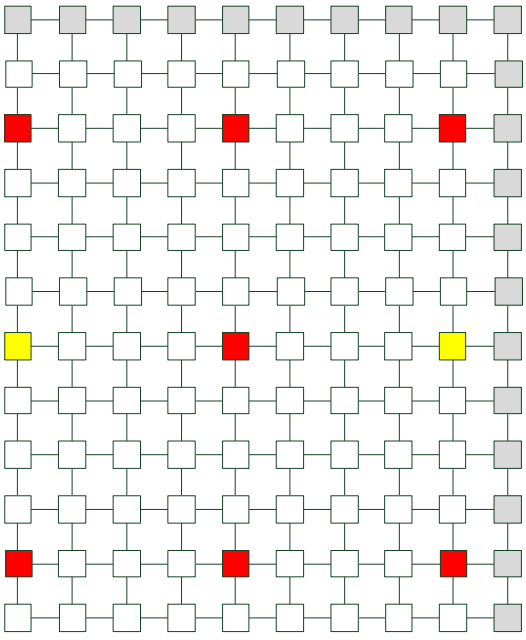


圖 3-38　由子網格拓展成 12×10 Type 3D4 的網格

3.5 經過路由器數量之比較

比較使用Case 1二層網格結構(圖3-39)、Case 2二層網格結構(圖3-40)、Case 3二層網格結構(圖3-41)與原始只有一層mesh結構(圖3-42)的封包平均經過路由器數量比較，圖中Y軸AVE hop為平均經過路由器數量、X軸與Z軸為網格大小(N×M)、最右側為某顏色時個數量範圍。可以明顯的看出當網格規模越大，使用二層網格結構可以大幅減少資料封包所經過的路由器個數(hop數)。在case 1、case 2、case 3及單層mesh結構的mesh為7×7時，平均經過路由器數量分別為3.985544、3.551020、3.32568及8.166667，資料封包路徑平均減少4.1個至4.8個hop數，而case1至單層mesh結構的mesh為50×50時，平均經過路由器數量分別13.775766、14.749772、18.086134及51.020408，平均所經過的路由器個數比單層mesh結構減少32.9個至37.2個hop數。而case 2網格的hop數在邊長有2+6t這種情況時，會經過較多的router個數，因此圖3-40中會有些點較周圍其他點的hop數高，改善的情形較不明顯。

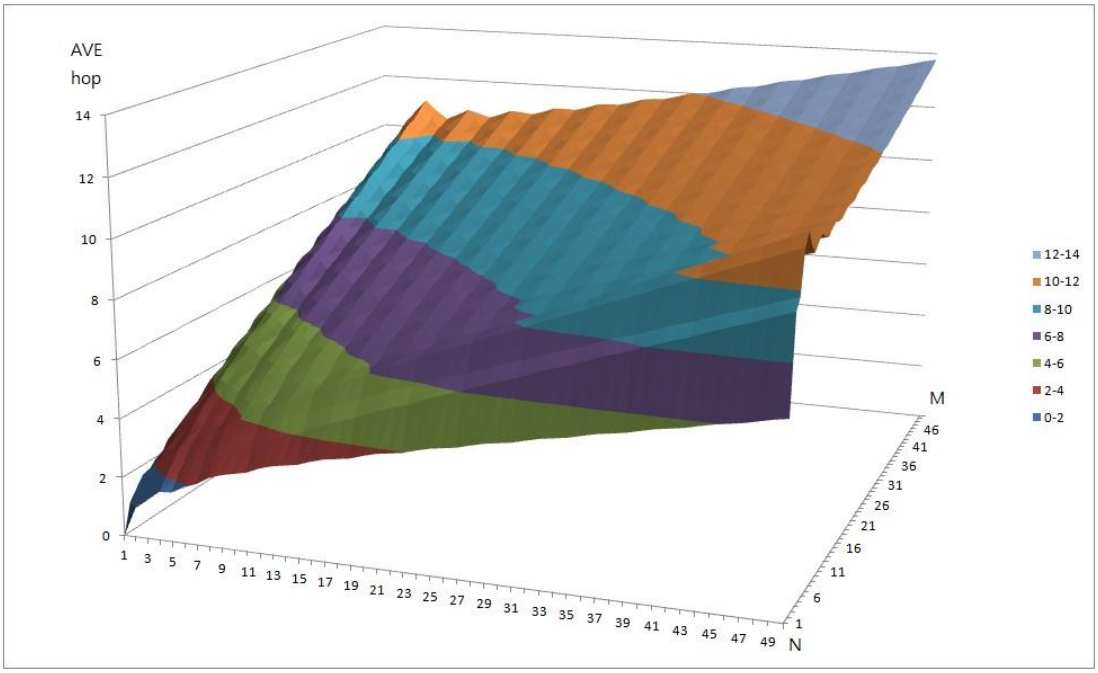


圖 3-39 Case 1 多維網格之封包路徑平均 hop 數

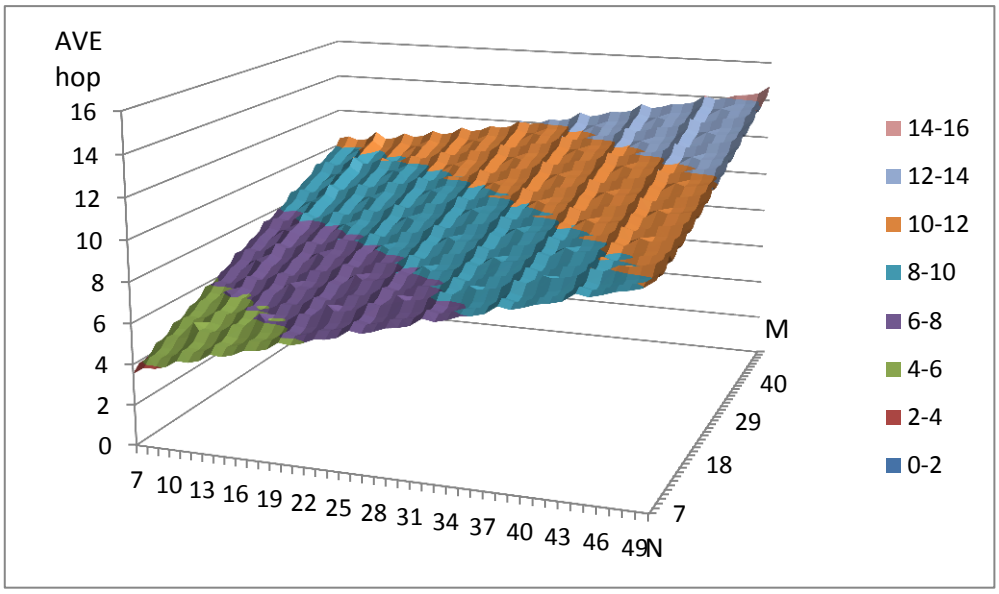


圖 3-40　Case 2 多維網格之封包路徑平均 hop 數

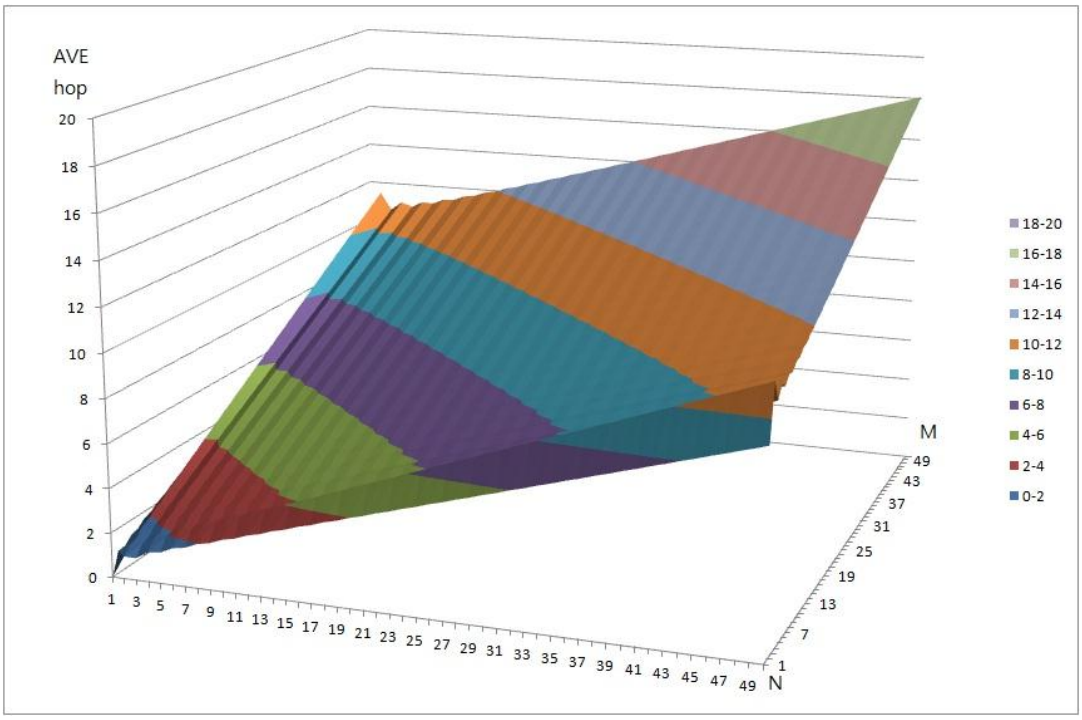


圖 3-41　Case 3 多維網格之封包路徑平均 hop 數

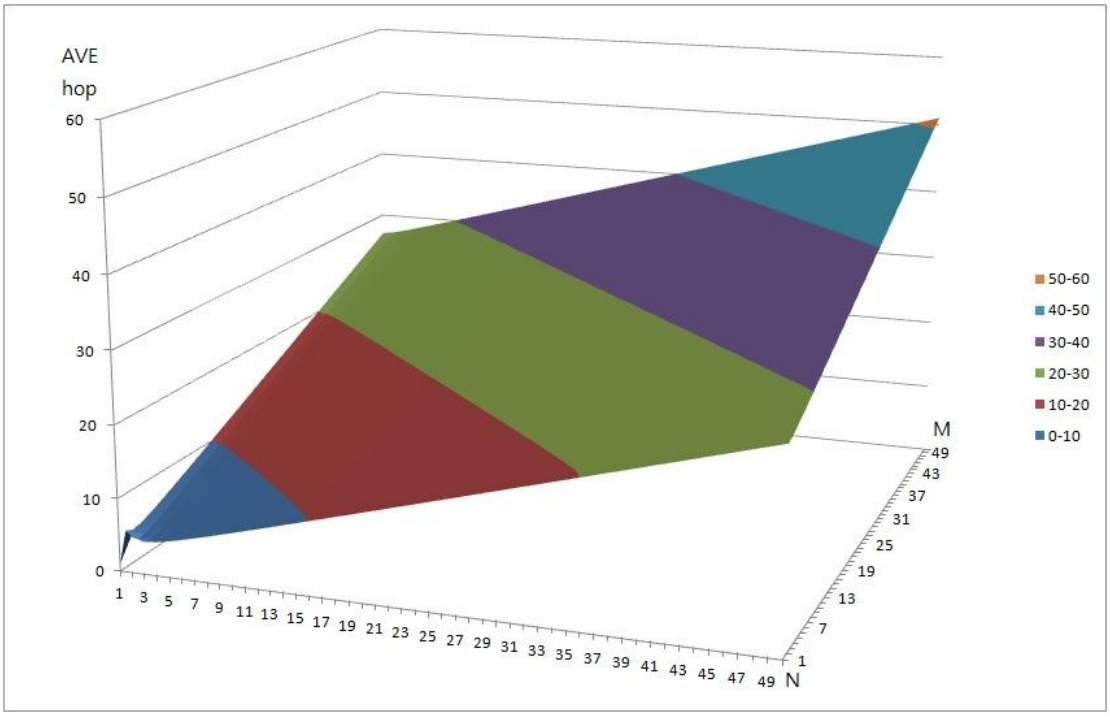


圖 3-42　一般網格之封包路徑平均 hop 數

3.6 使用於多層次網格拓樸之尋徑演算法

在我們所提出來的多層次二維網格設計中是採用XY尋徑演算法決定資料的傳輸路徑，使用XY routing的主要原因是此演算法具有死結避免(deadlock-free)與容易實施的特點。此外，XY routing是一種路徑確定(deterministic)及最短路徑(minimal path)的尋徑演算法。先將封包以X軸方向傳送，等到封包到達與目的地X座標相同時在改變方向以Y軸方向傳送到達目的地。對於中型或大型的NoCs，XY尋徑優於複雜的調適尋徑(adaptive routing)方法。以下所使用的例子都為case 2之網格架構。

3.6.1 交通管制路由器

為了使長距離傳輸的封包可以利用第二層網格進行傳輸，我們在每個子網格中修改了兩個路由器的尋徑電路，使這些被修改的路由器能夠依據封包的需求將其傳送到子網格的中心節點，並利用第二層網格縮短傳輸的時間及所需要經過的router數。此一修改後的路由器，我們稱之為交通管制路由器(Traffic Control Router; TC Router)。

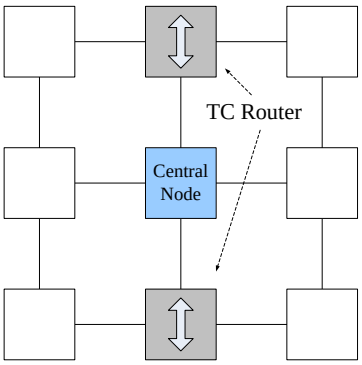


圖 3-43　子網格中的交通管制路由器

圖3-43表示交通管制路由器的特定位置，交通管制路由器協助長距離旅程的封包傳送到中心節點，圖中灰色的區塊即為交通管制路由器，箭頭符號則表示交通管制路由器傳送封包到中心節點的方向。當封包走過交通管制路由器時，封包剩餘的傳輸路徑將會被重新計算，接著交通管制路由器會決定封包是否需要藉由第二層mesh傳輸，若需要，則交通管制路由器會將這個封包原來的路徑做改變，並傳輸給中心節點。我們選擇這些位於沿相同中心節點Y方向的特定路由器作為交通管制路由器，是因為在XY尋徑的所有封包都必須先沿著X方向進行傳送，當封包有很長距離旅程它們有更多的機會經過交通管制路由器。

交通管制路由器判斷封包是否需要藉由第二層架構傳輸的方式如圖3-44所示。圖3-44在每個子網格中，當封包經過位於中心節點上方的交通管制路由器而向上(正Y方向)傳送才會抵達目的節點時，交通管制路由器會計算封包到目標節點是否還要經過大於等於5個hop，假如是的話，交通管制路由器會將封包往上(正Y方向)傳送，如圖S1到D1。當封包經過位於中心節點上方交通管制路由器而向下(負Y方向)傳送才會抵達目的節點時，交通管制路由器會計算封包到目標節點是否還要經過大於等於4個hop，假如是的話交通管制路由器會將

封包往下(負Y方向)傳送，如圖S2到D2。當封包經過位於中心節點下方交通管制路由器而向上(正Y方向)傳送才會抵達目的節點時，交通管制路由器會計算封包到目標節點是否還要經過大於等於4個hop，假如是的話交通管制路由器會將封包往上(正Y方向)傳送，如圖S3到D3。當封包經過位於中心節點上方交通管制路由器而向下(負Y方向)傳送才會抵達目的節點時，交通管制路由器會計算封包到目標節點是否還要經過大於等於5個hop，假如是的話交通管制路由器會將封包往上(負Y方向)傳送，如圖S4到D4。

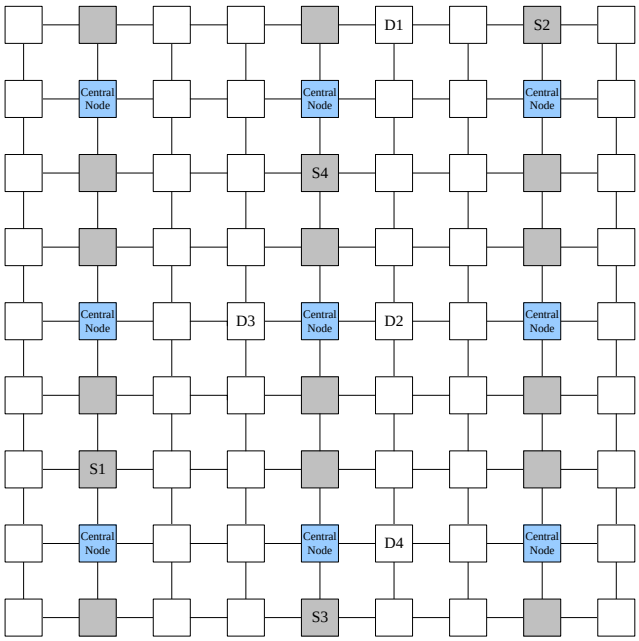


圖 3-44　以 case 2 為例子之交通管制路由器決定封包的傳輸路徑

3.6.2 第二層網格尋徑法

第二層mesh的router相互之間的封包傳遞與第一層mesh使用相同的尋徑方法，但封包於第二層mesh傳輸時，還需要額外考慮封包需要被轉向或停止。在我們的設計下，第二層的尋徑演算法是XY 尋徑，但是在單一方向下，若封包所在的目前節點與封包的目的節點之間的距離小於3個hop時，封包將會被轉入另一個方向。

在每個子網格中，中心節點以YX尋徑，當封包傳送到中心節點時，中心節點會判斷封

包距離目的節點的距離是否大於等於3，若是，封包將會經由第二層mesh被傳送。而當Y方向剩下的hop數小於3時，會由X方向剩餘的hop數做為是否回到第一層mesh的判斷，當Y方向剩餘hop數小於3且X方向hop數大於3時，封包會在第二層mesh轉向X方向；當Y方向剩餘hop數小於3且X方向hop數小於3時，封包會向第一層mesh做傳送。

3.6.3 長距離封包

考慮到上一節的尋徑條件，我們可以簡單地定義長距離封包。封包到目的節點的餘距離單一方向大於等於3即為長距離傳輸的封包，這表示這個封包至少將會以XY尋徑在第二層mesh移動一個hop。

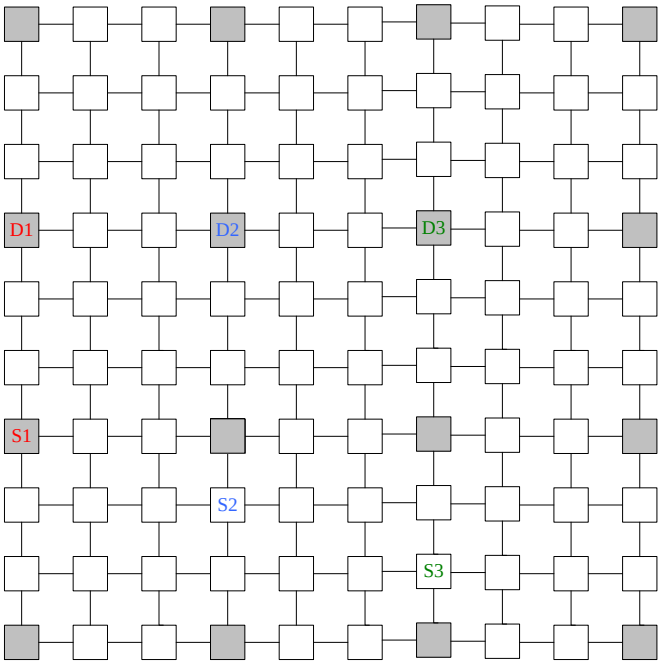


圖 3-45　以 case2 為例子之長距離封包的例子

如圖3-45的例子，我們可以根據封包從目前節點到目的節點剩下的距離分類，起始節點S1到目的節點D1剩下的距離為3、起始節點S2到目的節點D2剩下的距離為4、起始節點S3到目的節點D3剩下的距離為5。他們最少都會經由第二層mesh移動一個hop。

3.6.4 尋徑選擇

在我們的架構下，只有中心節點路由器與交通管制路由器需要特殊的尋徑策略，其它的路由器則使用正常XY尋徑。如前所述，交通管制路由器可以指派長距離封包到中心節點，交通管制路由器使用與原路徑不同的XY尋徑法。另外，因為第二層mesh是額外增加的結構，當我們結合這個額外的結構到原始mesh上時，它提供了封包第二種路徑到達目標節點。因此，當長距離封包到達中心節點路由器或交通管制路由器時，這個封包有兩種可能的路徑：

(1) 封包經由原始第一層 mesh 傳輸資料。

(2) 位於中心節點路由器時，封包經由第二層 mesh 傳輸資料，或位於交通管制路由器時，向中心節點方向傳送。

為了防止中心節點因為傳送太多的封包到第二層mesh而造成瓶頸，我們訂定一個特殊的尋徑策略給交通管制路由器和中心節點路由器：

1. 封包不能取得虛擬通道時，封包須從第二層 mesh 轉回第一層 mesh

2. 當封包尋徑在第二層 mesh 時發生阻塞，必須將封包傳送回第一層 mesh

圖3-46顯示一個可能的路徑封包從節點S到節點D，紅色虛線表示封包位於第一層mesh，藍色實線則表示封包尋徑在第二層mesh，值得注意的是，長距離封包未必皆於第二層mesh傳輸。

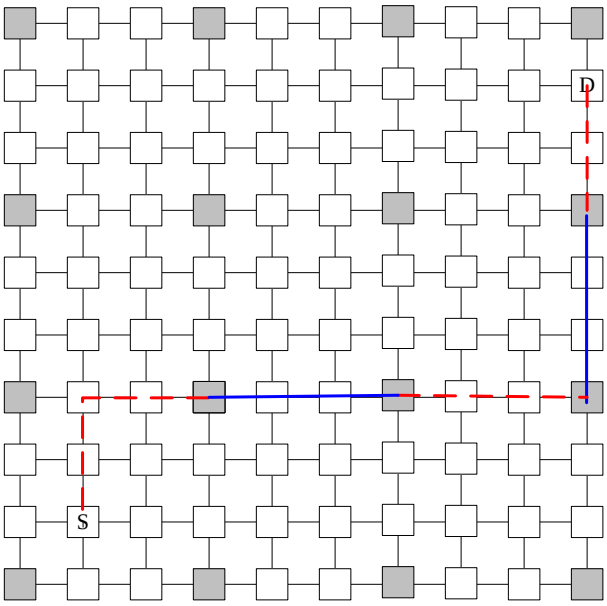


圖 3-46　以 case 2 為例子之封包尋徑在第二層 mesh 的例子

3.6.5 死結避免

XY尋徑可以避免死結的產生，如圖3-47所示，它禁止所有來自Y方向的轉向(虛線)，所以它能夠避免循環等待(circular-wait configuration)。此外，XY尋徑是最短路徑尋徑，封包不能取得與input port同方向的output，因此不存在180度轉向。

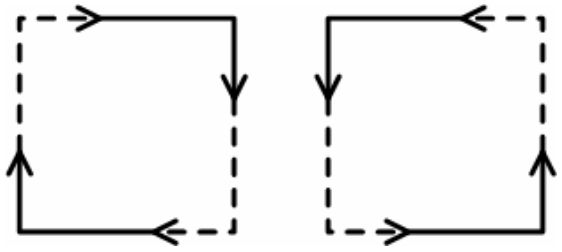


圖 3-47　XY 尋徑允許的轉向

在我們所提出的多層次二維網格架構下，第一層mesh是以XY進行傳輸，而第二層是以YX 方向進行傳輸。雖然傳輸方向有所改變，但因為是在不同層次，且封包不被允許增加路徑長度(hop數)，因此並不會有deadlock的問題。圖3-48顯示當長距離旅程的封包朝著不同路徑尋徑到中心節點，但他們由於遇到阻塞而轉回到第一層網格使用正常的XY尋徑，

在這種情況下可以看出，允許把封包從Y方向到X方向。

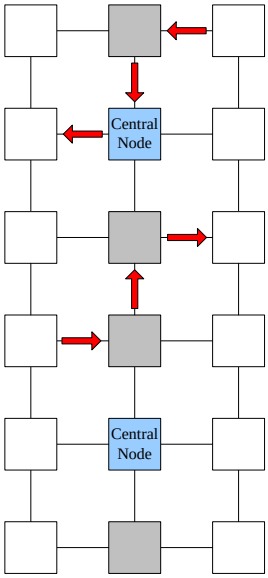


圖 3-48　以 case 2 為例子之封包 YX 轉向

3.7 使用於多層次網格拓樸之路由器設計

為了使我們提出來的架構能順利運作，並確知架構對效能與功率的影響，我們實作了三種不同的路由器。第一種路由器是一般路由器，其內部結構已於2.6節介紹過，如圖2-24所示。在一般路由器中，主要包含輸入緩衝器、虛擬通道控制器、交換控制器與crossbar等。第二種路由器為中央節點路由器，如圖下3-49所示，中央節點路由器負責將封包由第一層網格送往第二層網格，或反過來由第二層網格送到第一層網格，此外，中央節點路由器還必須負責第二層網格之間的封包傳遞。因為中央節點路由器的功能較多，因此必須區分為第一層網格專用的輸入緩衝器及第二層網格專用的輸入緩衝器。不論是第一層或第二層網格所使用的輸入緩衝器內部架構皆與一般緩衝器一樣。唯一的差別是在於第一層網格所使用的緩衝器有五個(由東、南、西、北方向路由器及本地IP來的封包)，第二層緩衝器則有四個，主要負責連接鄰近的第二層中央節點路由器。除了增加輸入緩衝器之外，我們也修改了allocator的控制內容及crossbar的大小，使其傳輸功能正確無誤。第三種路由器為交通控制路由器，除了具有一般路由器的功能之外，還必須判斷長距離封包是否需要往鄰

近的中央節點路由器傳送。交通控制路由器的架構與一般路由器相同，唯一的差別在於其內部尋徑邏輯(routing logic)及switch allocator對於長距離封包可以做不同方向的調整，並將其指定到特定的輸出通道。

我們進一步比較了三種路由器的面積、功率消耗及電路延遲，如表3-1所示。我們先利用Verilog HDL進行路由器硬體電路設計，再經過Design Compiler以TSMC 0.18 μm製程進行合成。從表3-1中可以觀察得到，由於中心節點路由器比一般路由器多了四組緩衝器，因此不論是面積、功率消耗或電路延遲都比一般路由器來得大，而由於交通管制路由器僅修改了部分控制電路，因此增加的幅度不大。以面積來看，中心節點路由器比一般路由器大182.5%，而交通管制路由器則比一般路由器大2.3%。以動態功率消耗來看，中央節點路由器比一般路由器消耗多130.5%，交通管制路由器則比一般路由器多14.3%，靜態功率消耗上，交通管制路由器與一般路由器接近，而中心節點路由器則為一般路由器的2.83倍。在電路延遲上，中心節點路由器比一般路由器約多13%。

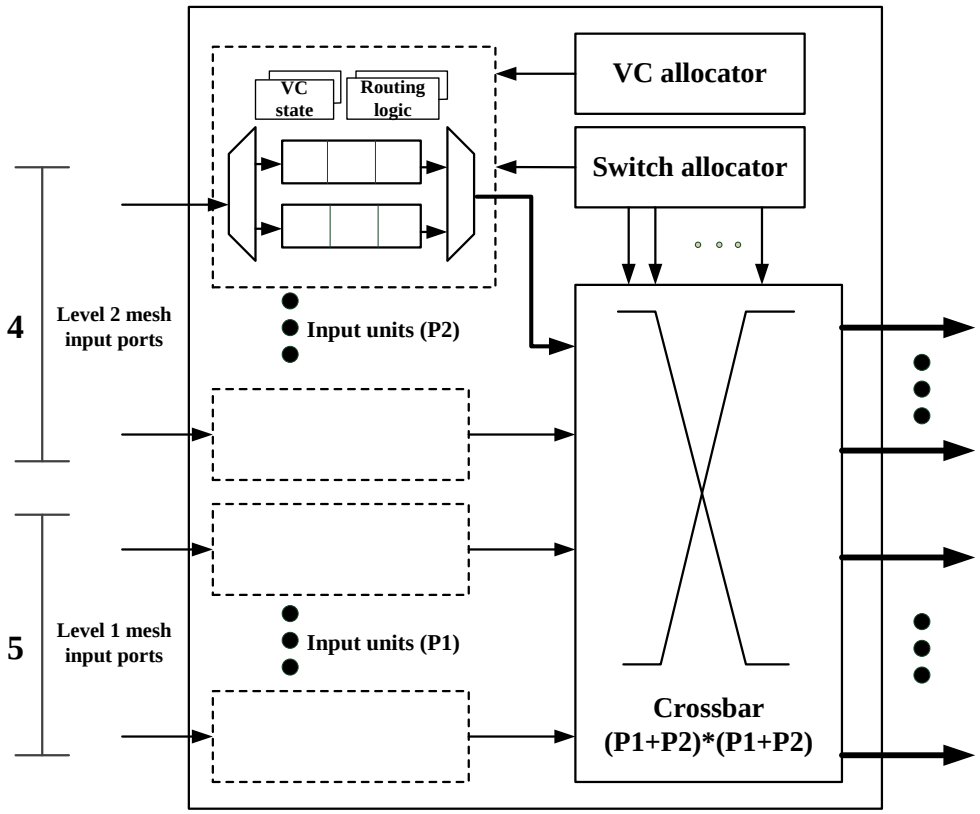


圖 3-49　具有第二層網格功能之中心節點路由器

表 3-1　三種路由器的比較

	一般路由器	中心節點路由器	交通管制路由器
Total cell area (um^2)	283761.88	801539.33	290284.95
Dynamic Power (mW)	15.28	35.22	17.47
Leakage Power (uW)	7.47	21.17	7.32
Delay	4.25	4.83	4.25

3.8 最佳化

為了整合我們所提出的網格拓樸類型，故提出了多層次網格最佳化演算(Multi-level Mesh Optimization Algorithm)，此演算法的功能在於因應面積、功率以及效能的需求做適當的網格拓樸選擇。

```
Multi-level Mesh Optimization Algorithm

Case 1= A , Case 2= B , Case 3= C
Reduce area =X, Save Power=Y, Improve Performance= Z

If sum(X, Y )=100% then
{
  If  in “short-distance” and “(Z= X , don’t care Y) ” then use A;
   //for performance
  else if  in “long-distance” and “(Z= Y, don’t care X) ” then use C;
   //for performance
  else if  in “short-distance” and “(X, don’t care Y& Z) ” then use C;
   //for area

  End if;
}
Else  Others use B;
End if;
```

圖 3-50　多層次網格最佳化演算

圖 3-50 為多層次網格最佳化演算，如果在”短距離傳輸”我們重視效能與面積且忽略功率時，則使用 Case 1， 因為在短距離封包傳輸時，Case 1 本身進入第二層的機會高，且面積較小，換言之通過的路由器也較少。若在”長距離傳輸”我們重視效能與功率且忽略面積時，則使用 Case 3，因為在長距離封包傳輸時，Case 3 能進入第二層的機會較高，並且

可大量減少使用路由器的數量。如果在”短距離傳輸”我們重視面積且忽略功率與效能時，則使用 Case 3，因為 Case 3 所用到之路由器較少，但進入第二層的機會較低所以常傳輸需要經過較多的路由器以造成傳輸之延遲與功率之消耗。

Case 2 的表現較為平均，不會特別著重於效能或功率，故上述 Case 1 與 Case 3 表現較差的部分都使用 Case 2。

第四章 模擬與實驗結果

在本章，我們對三種多層次二維網格架構進行模擬實驗，並分別對面積、功率消耗及效能改善做探討。

4.1 模擬環境與條件

為了驗證我們所提出之多層次二維網格架構的可行性，我們使用Verilog HDL進行router設計，並利用TSMC 0.18 μm製程做電路合成，最後再以硬體模擬的方式來測量NoC的效能。在router設計過程中，我們先實作了給晶片網路使用的傳統虛擬通道路由器(Virtual Channel Router)[30]。再依照此路由器設計3.7節所介紹之中心節點路由器及交通管制路由器。這些路由器皆擁有4級管線結構(four pipeline stages)，一個封包通過路由器需要經過4個時脈週期。另外，兩個路由器之間的通道延遲為一個時脈週期。

表 4-1 模擬環境設定及組態

Configuration	Experimental sets
Network topology	Ideal 2D mesh and 2-level mesh
Flow control	Credit-based
Virtual channels	2 VCs per physical channel
Routing Algorithm	XY routing
Scheduling algorithm	Round robin (least recently served)
Buffer depth	4 flits
Packet size	8 flits
Flit size	34 bits
Buffer type	Input buffer
Switch architecture	Wormhole switch
Delay	Router: 4 ; Channel:1 (cycles)

上表4-1顯示模擬環境的設定及組態。路由器使用輸入緩衝器，每個實體通道擁有2個虛擬通道，每個虛擬通道具有4個flit的深度。我們假設一個封包有固定大小為8 flits，通道頻寬為一個flit的大小。模擬器以wormhole routing和XY尋徑方式設計。路由器使用Credit-based flow control管理緩衝器，此方法為利用一個記憶體空間記錄下一個路由器虛擬通道的空緩衝區。另外一個NoC擁有許多仲裁器，為了公平我們使用矩陣仲裁器[33][34]，它是一個round-robin仲裁器，並且可以提供「least recently served」優先策略。

圖4-1為流量產生與ejection元件的內部模組[33]。封包被產生在進入source buffer或網路之前會馬上被標記時間資訊。在我們的設計中，封包被分成flits，所以我們標記封包的每個flit測量延遲。source buffer假設成無限大，當網路是full且沒有loss時，所有的flits皆存於緩衝區中。output count & timing module可以解碼並記錄收到flits的延遲並計算throughput。

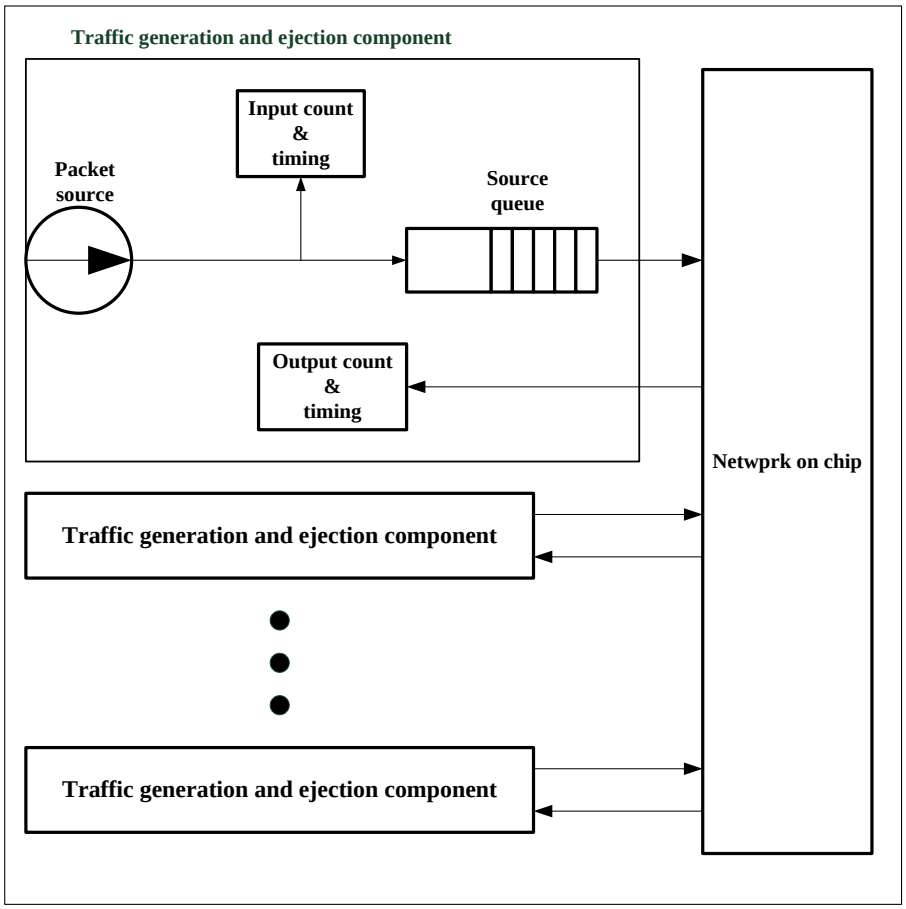


圖 4-1　流量產生與 ejection 元件

Flit的延遲主要來自於兩個部分，一個是在source queue的queuing time，而另外一個是網路的傳送時間。Throughput被定義為每個節點每個時脈週期平均收到的flits數，它代表著網路可接受的最大流量。

4.2 面積評估

為了評估多層次網格晶片網路設計的影響，我們分別將case 1至case 3實作了各種不同型態的mesh，最小的mesh為6x6，最大的mesh為50x50。5x5以下的mesh架構規模較小，若建構第二層網格無法獲得任何好處。在設計當中，我們使用200MHz的0.18 μm CMOS technology library進行電路合成，然後測量我們設計的面積。由於中心節點路由器有9個輸出入埠，而每個輸出入埠皆須設置buffer，因此它的面積為一般路由器的2.8倍。交通管制路由器只增加些許判斷電路，因此面積比起一般路由器僅多出1%。若case 1及case 2單純考慮子網格(3x3)之面積，case 1及case 2網格架構比一般二維網格架構的面積增加約20%。但由於不同的型態具有不同的第二層網格節點，因此我們分別裂在case 1至case 3列舉了五個不同規模mesh進行比較，如表4-2至表4-4所示。從表4-2至表4-4中，我們可以觀察到case 1至case 3的面積增加分別約在case 1 25%至32%之間、case 2 20%至30%之間及case 2 18%至25%之間。

由於Case 1第二層之網格節點較多，故每個輸出入埠皆須設置buffer與中心路由器節點有9個輸出入埠以及方向路由器的數量增加，所以較Case 2的面積還大。反之Case 3第二層之網格節點較少，故所需面積也會隨之減少。

表 4-2 一般網格架構與 case 1 網格架構面積比較

	原始mesh (mm^2)	改良後mesh (mm^2)	增加面積百分比
10x10	28.38	37.22	31.15%
20x20	113.51	149.43	31.64%
30x30	255.39	319.85	25.24%
40x40	454.02	590.72	27.91%
50x50	709.41	902.38	27.20%

表 4-3 一般網格架構與 case 2 網格架構面積比較

	原始mesh (mm^2)	改良後mesh (mm^2)	增加面積百分比
10×10	28.38	36.88	29.95%
20×20	113.51	147.27	29.74%
30×30	255.39	308.47	20.78%
40×40	454.02	557.88	22.87%
50×50	709.41	880.93	24.17%

表 4-4 一般網格架構與 case 3 網格架構面積比較

	原始mesh (mm^2)	改良後mesh (mm^2)	增加面積百分比
10×10	28.38	35.52	25.15%
20×20	113.51	141.71	24.84%
30×30	255.39	299.58	17.30%
40×40	454.02	542.11	19.41%
50×50	709.41	868.64	22.45%

4.3 功率消耗及效能改善分析

為了完整呈現功率消耗及效能改善的情形，我們將實驗結果繪製成 3D 圖。圖 4-2 及圖 4-3 分別呈現 Case 1 的功率消耗及效能改善情形，圖 4-4 及圖 4-5 分別呈現 Case 2 的功率消耗及效能改善情形，圖 4-6 及圖 4-7 分別呈現 Case 3 的功率消耗及效能改善情形。在這六個圖中，可以發覺網格規模越大，所能夠節省的功率消耗及效能改善的幅度也越大。圖 4-4 及圖 4-5 中會看得到不規則的下凹處，主要來自於 mesh 的邊出現有 2+6t 的情況時由於中心路由器會排列在一起而造成功率消耗較大或電路延遲較大。當而在 mesh 的邊出現有奇數、4+6t 和 6+6t 的情況時由於中心路由器會被排放的較平均，所以功率消耗及傳輸延遲的程度較小，改善的幅度也較多。

從實驗結果中，可以清楚地觀察到，不論是何種型態，當網格規模越大時，其功率節省及效能改善的幅度也越大。最主要的原因是網格規模較大時，長距離封包較能有效利用第二層網格進行傳輸，因此所需經過的路由器個數會大幅度減少。雖然單一個中央節點路由器或交通管制路由器的功率消耗及電路延遲均比一般路由器來得大，但整體而言，此多層次架構能夠節省的功率消耗及改善的傳輸延遲遠高於單一個路由器的影響。

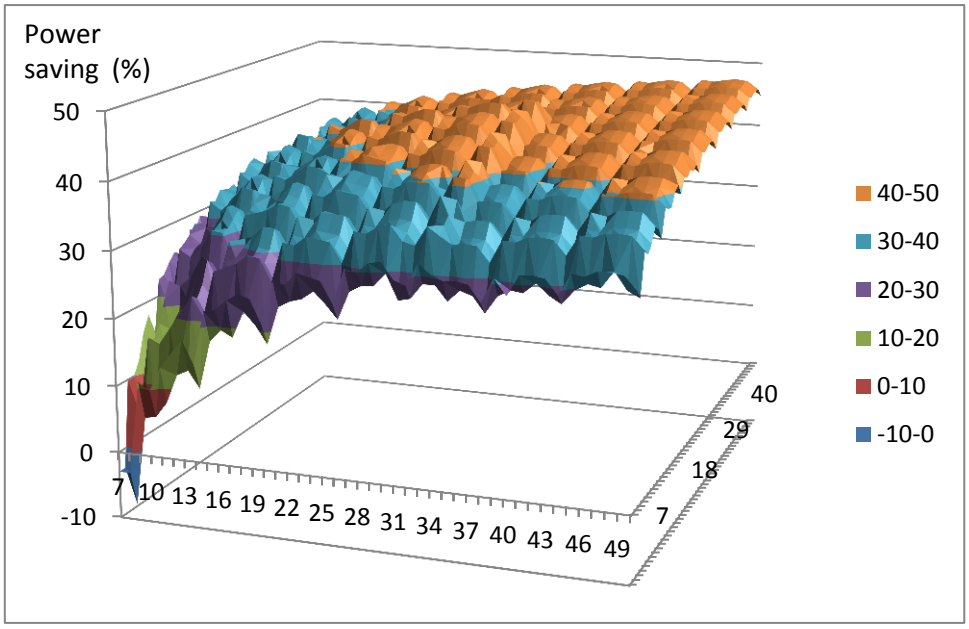


圖 4-2　Case 1 功率消耗改善情形

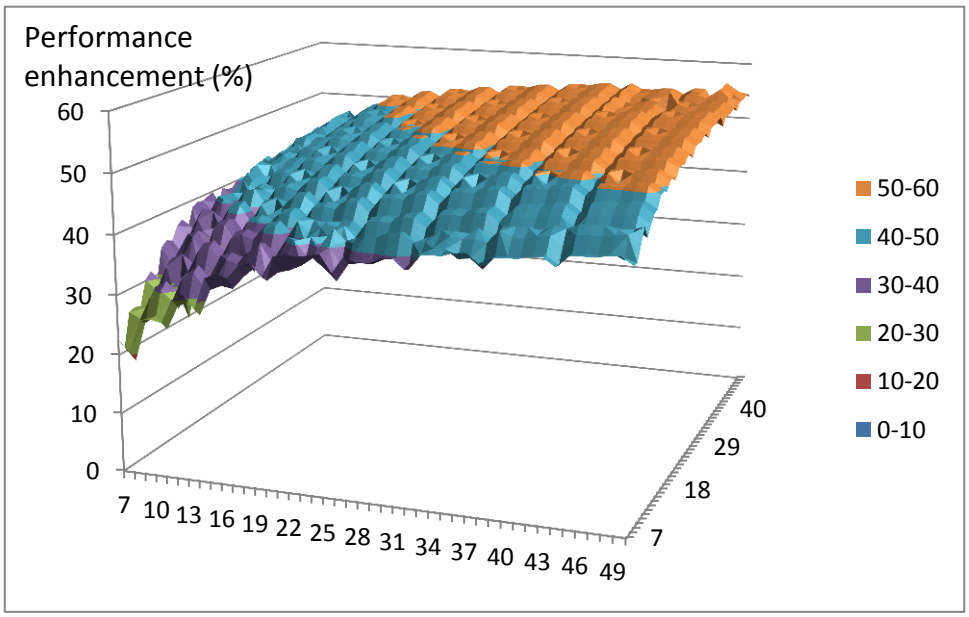


圖 4-3　Case 1 效能改善情形

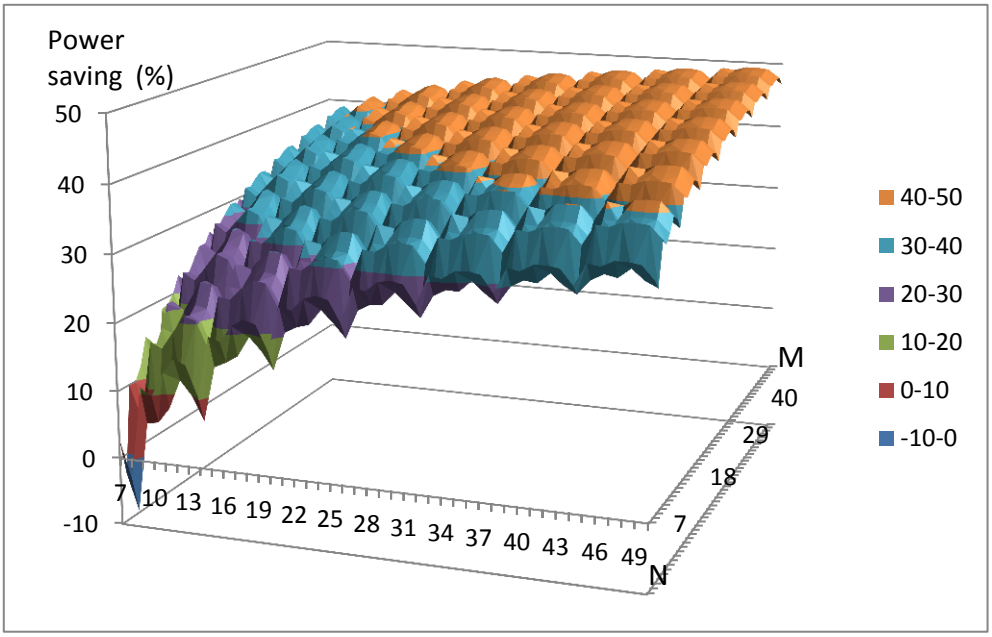


圖 4-4　Case 2 功率消耗改善情形

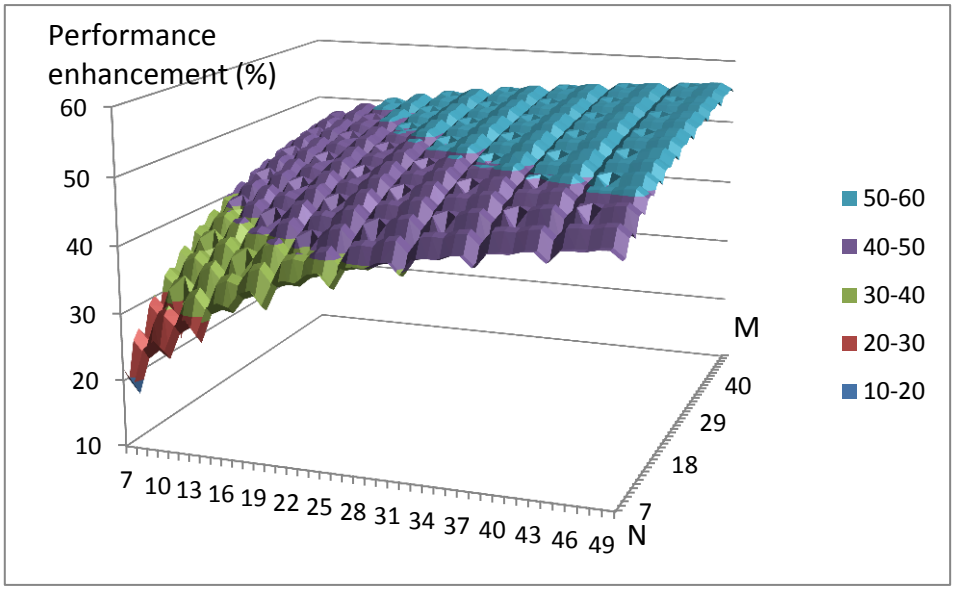


圖 4-5　Case 2 效能改善情形

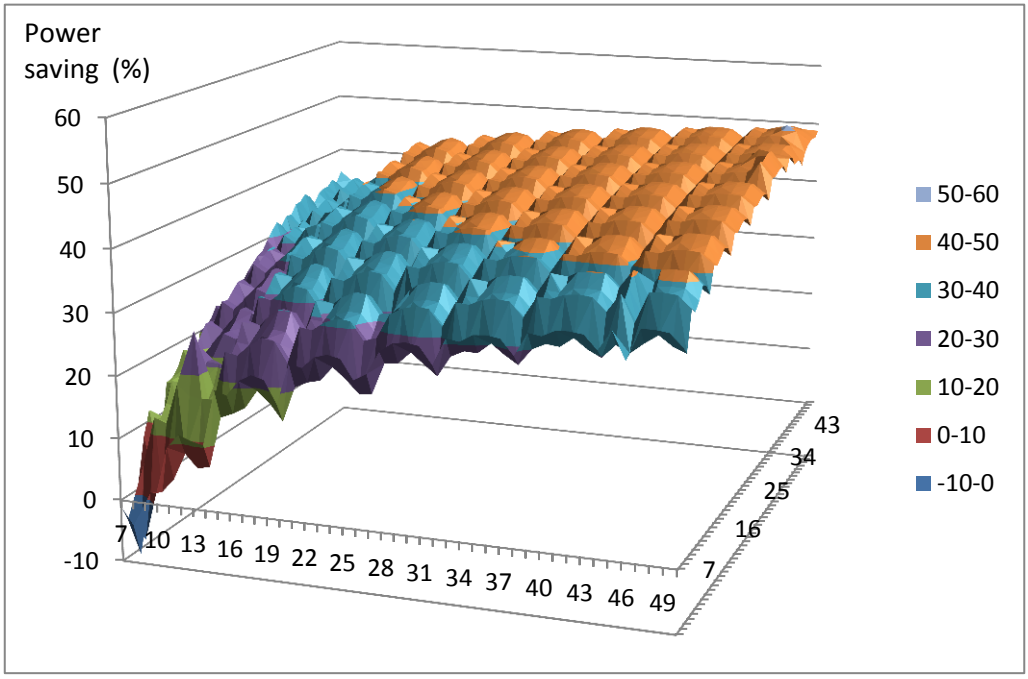


圖 4-6　Case 3 功率消耗改善情形

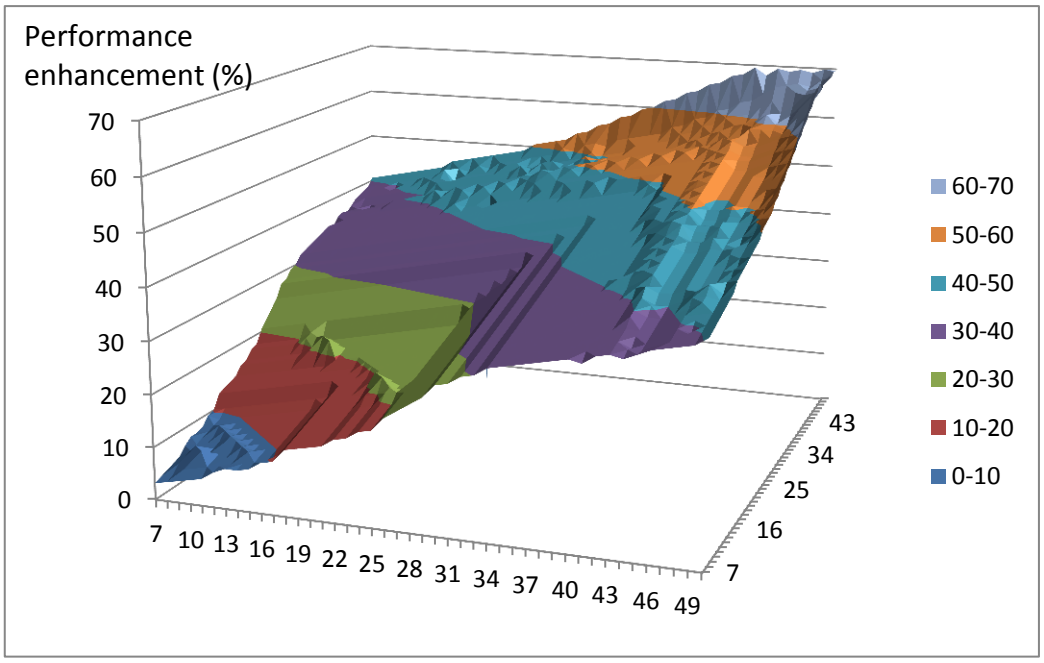


圖 4-7　Case 3 效能改善情形

4.4 最佳化之功率消耗及效能改善分析

根據上節我們所得到 Case 1 至 Case 3 之數據加以做分析後並套入多層次網格最佳化演算法，晶片網路將可以因應各種條件得到最佳的改善。如功率、面積以及效能之改善。由於封包傳輸距離的長短會影響最佳化的選擇，網格規模 7×7 至 25×25 視為短距離封包傳輸而 26×26 至 50×50 視為長距離封包傳輸，並且我們對功率與效能給了些比重，以下我們將分成「短距離封包傳輸」以及「長距離封包傳輸」兩個部分來介紹最佳的節省功率與改善效能。

4.4.1 短距離封包傳輸

規模為 7×7 短距離傳輸時進入第二層網格為 Case 1 的機會最大，原因在於第二層之間的距離只間隔一個路由器，故封包在傳輸時經過交通管制路由器的機會變大而得以順利進入第二層網格。反之 Case 3 的間隔較大，故進入第二層的機會較低且造成的功率消耗也較高在短距離封包傳輸時不適用。不過網格規模愈來愈大(長距離封包傳輸)時 Case 1 的效能並無太大之改善，並且功率的消耗會來的比 Case 2 與 Case 3 來要大。

4.4.2 長距離封包傳輸

規模為 26×26 長距離傳輸時進入第二層網格依然為 Case 1 的機會最大，但由上節所說明，我們可以知道雖然進入第二層的機會大，但是隨著網格規模的加大造成功率的增加且效能的改善也較低。

所以在規模較大的網格所適用的類型為 Case 3，雖然 Case 3 進入第二層的機會較低不過在長距離時進入第二層的機會提升，且在第二層的傳輸效能比 Case 1 與 Case 2 都要來的好，原因在於封包進入第二層後所經過第二層路由器的數量大量的減少並且減少了許多功率之消耗。在以上條件我們得到了使用最佳化演算法去選擇網格類型與一般二維網格比較可提升 70%效能與減少 55%功率消耗

第五章 總結

由於二維網格(2D mesh)拓樸有相對較大的網路半徑，造成有些長距離的封包傳送有較大的傳輸延遲。故此篇論文的研究重點在晶片網路網格拓樸的改良方法，用以取代傳統上所使用的二維網格網路拓樸。主要目的是讓長距離資料傳送可以進行快速傳輸。為了達成目的，我們著重於改良網路網格拓樸的架構型態，並且提出了多層次的設計方法，換句話說是讓長距離資料傳送可以利用另一層網格拓樸進行快速傳輸。為了能讓封包傳輸路徑正確並且可以死結避免(deadlock-free)我們修改了路由器架構，並將它取名為交通管制路由器(Direction Control Router; DC Router)，此一路由器搭配 XY 尋徑演算法，此演算法具有死結避免與容易實施的特點，且此演算法為一種路徑確定(deterministic)及最短路徑(minimal path)的尋徑演算法。

實驗的結果顯示，雖然增加第二層網格會帶來額外的電路面積增加，不過當網格規模愈來愈大時，傳輸延遲及功率消耗改善的幅度都會增大。其改善幅度為可以節省 50%的功率消耗，並改善 70%的效能。因此，我們可以對改良過後的二維網格下一個結論，多層次網格架構可以在增加少量面積條件下，得到較佳的傳輸延遲及功率消耗，當晶片網路規模越大時，愈適合使用論文所提出來的多層次網格架構。

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/ , 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), Apr. 2004, pp. 192－195.

[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*, 2002, pp. 105–112.

[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, Sept. 2003.

[9] M. Horowitz and B. Dally, "How Scaling Will Change Processor Architecture," in IEEE Proceeding of *International Solid State Circuits Conference* (ISSCC), Feb. 2004, pp. 132-133.

[10] P. Guerrier and A. Greiner, "A Generic Architecture for On-Chip Packet-Switched Interconnections," in Proceeding of *Design, Automation and Test in Europe* (DATE), Mar. 2000, pp. 250-256.

[11] 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) ,

May 2003, vol. 5, pp. V-217－V-220.

[12] W. J. Dally and B. Towles, “Route Packets, Not Wires: On-Chip Interconnection Networks,” in Proceeding of *Design Automation Conference* (DAC), 2001, pp. 683-689.

[13] F. Karim, A. Nguyen, and S. Dey, “An Interconnect Architecture for Networking Systems on Chips,” IEEE *Micro*, vol. 22, no. 5, pp. 36-45, Sept./Oct. 2002.

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

[15] M. Coppola, R. Locatelli, G. Maruccia, L. Pieralisi, and A. Scandurra, “Spidergon: a novel on-chip communication network,” Proceedings. *System-on-Chip*, 2004, pp. 15.

[16] A. V. de Mello, L. C. Ost, F. G. Moraes, and N. L. V.r Calazans, *Evaluation of Routing Algorithms on Mesh Based NoCs*, Technical Report Series, PUCRS, Brazil, May, 2004.

[17] L. Benini, and G.D. Micheli, *Networks on Chips: Technology and Tools*, Morgan Kaufmann, 2006.

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

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

[20] 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), July 1995, pp. 275－287.

[21] 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), June 2009, pp. 208－219.

[22] 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.

[23] 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), Aug. 2009, pp. 57－62.

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

[25] 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.

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

[27] K. Goossens, J. Dielissen, and A. Radulescu, "AEthereal network on chip: concepts, architectures, and implementations," IEEE *Design & Test of Computers*, vol.22, no.5, pp. 414-421, Sept.-Oct. 2005.

[28] 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.

[29] A. Mello, L. Tedesco, N. Calazans, and F. Moraes, "Virtual Channels in Networks on Chip: Implementation and Evaluation on Hermes NoC," in Proceeding of *Symposium on Integrated Circuits and Systems Design*, Sept. 2005, pp.178-183.

[30] L. S. Peh and W. J. Dally, "A delay model and speculative architecture for pipelined routers," in Proceeding of *International Symposium on High-Performance Computer Architecture*, 2001, pp.255-266.

[31] J. W. Joyner, R. Venkatesan, Z.-H. Payman, J. A. Davis, and J. D. Meindl, "Impact of three-dimensional architectures on interconnects in gigascale integration," IEEE *Transaction on Very Large Scale Integration (VLSI) System*, vol. 9, no. 6, pp. 922–927, Dec. 2000.

[32] 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), May 2003, vol. 5, pp. 217-220.

[33] W. J. Dally and B. Towles. *Principles and Practices of Interconnection Networks*, Morgan Kaufmann, 2004.

[34] 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*, June 2004, pp. 188-197.