电子科技大学

硕士学位论文

片上网络路由算法研究及路由节点的FPGA设计

姓名：黎黎

申请学位级别：硕士

专业：信息与通信工程

指导教师：李玉柏

20070501

摘要

技术的飞速发展把芯片设计带入了十亿级晶体管的领域，这意味着单个芯片中可集成更多的 IP 核。而随着芯片中 IP 核数量的增多，片上系统（SoC）的设计方法将不再满足设计要求。这时，片上网络（Network On Chip，NOC）做为一种新的解决途径被提出来了。片上网络的可升级性（scalable）和易扩展性很好的满足了新设计的要求。片上网络逐渐发展成为片上总线之外的一种新的通信结构。总的来说，NOC 网络的复杂程度（complexity）是由两方面的因素所表征，一个是网络拓扑结构，另外一个就是路由算法。

本文正是对这两方面进行了研究，文章综合了国内外已有文献，重点介绍了 NoC 直接型拓扑结构中的 Mesh 结构和 Torus 结构，以及常用的路由技术和算法，并针对 Mesh 结构中常用的防死锁路由算法——XY 路由算法，创立了适合于 Torus 结构的新型路由算法，建立了对应的两种拓扑结构、两种路由算法的模型，在 OPNET 上进行了仿真，并对仿真结果进行了分析。

为了在硬件上实现、验证并比较不同路由算法以及拓扑结构对网络性能的影响，我们建立了以 FPGA 为核心的硬件仿真测试平台，并在 FPGA 上初步完成了单个路由节点的设计和验证，并对节点进行了必要的分析，为将来整个网络的搭建奠定了基础。

关键词：片上网络，拓扑结构，路由技术，路由算法，仿真，FPGA 设计

Abstract

Rapid development of technology brings chip design into billion-transistor field. It is predicted that at the end of this decade, it will allow more than one billion of transistor integrated on a single chip. As the number of cores of system chip increased, traditional SoC design methodology will not fit the billion-transistor field design requirement. At this time NOC, as a new solution, is presented. The scalable ability of NOC well meets the challenge of new design requirement. NOC gradually becomes a new communication structure besides traditional on-chip bus. In general, the complexity of NOC is characterized by two main factors, one is the network topology, and the other one is routing algorithm.

This paper mainly introduced the Mesh topoplogy and Torus topology of NOC direct topology, and the popular routine techniques and algorithms. At the same time, a new dead-lock and live-lock free route algorithm to Torus topology is proposed. Moreover, we build the corresponding routing model for Mesh topology with XY routing algorithm and for Torus topology with self-designed routing algorithm, then simulate on OPNET and analyse the result.

In order to valuate the feasibility and practicability of the NOC structure, we designed efficient hardware emulation and testing platform primarily consisted of DSP and FPGA. Furthermore, a five-component-microarchitecture model for router based on FPGA is proposed, verified and analysed, which lay the foundation for futhre research on NOC.

Keywords: Network on Chip, Topology, Routing Technology, Routing Algorithm, Simulation, FPGA Design

图目录

表目录

缩略词表

英文缩写	英文全称	中文释义
SoC	System on Chip	片上系统
NOC	Network On Chip	片上网络
IP	Intellectual Property	知识产权
NI	Network Interface	网络接口
HOL	Head of Line	队头阻塞
LAB	Logic Array Blocks	逻辑阵列块
ALM	Adaptive Logic Module	自适应逻辑模块
DLL	Delay-Locked Loop	延时锁定环
PLL	Phase-Locked Loop	锁相环
IOE	Input/ Output Element	输入输出单元
VPP	Valueable Pulse Position	有效脉冲位置
PAL	Phase Alteration Line	正交平衡调幅逐行倒相制式
NTSC	National Television Systems Committee	全国电视系统委员会制式
EMIF	External Memory Interface	外部存储器接口
EDMA	Enhanced Direct Memory Access	增强型直接内存存取

独 创 性 声 明

本人声明所呈交的学位论文是本人在导师指导下进行的研究工作及取得的研究成果。据我所知，除了文中特别加以标注和致谢的地方外，论文中不包含其他人已经发表或撰写过的研究成果，也不包含为获得电子科技大学或其它教育机构的学位或证书而使用过的材料。与我一同工作的同志对本研究所做的任何贡献均已在论文中作了明确的说明并表示谢意。

签名：黎黎　　日期：2007年 5月23日

关于论文使用授权的说明

本学位论文作者完全了解电子科技大学有关保留、使用学位论文的规定，有权保留并向国家有关部门或机构送交论文的复印件和磁盘，允许论文被查阅和借阅。本人授权电子科技大学可以将学位论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存、汇编学位论文。

（保密的学位论文在解密后应遵守此规定）

签名：黎黎　　导师签名：

日期：2007 年 5 月 23日

第一章 绪论

1.1 课题背景

20 世纪 90 年代中期，随着半导体工艺技术的发展，IC 设计者能够将越来越复杂的功能集成到单硅片上，SoC 正是在集成电路向集成系统转变的大方向下产生的。由于 SoC 可以充分利用已有的设计积累，显著地提高 ASIC 的设计能力，因此发展非常迅速，引起了工业界和学术界的关注。

但是，随着深亚微米超大规模集成电路工艺技术的成熟及进一步发展，芯片设计业面临着严峻的问题：随着芯片功能和性能的需求发展，芯片规模越来越大，工作速度越来越高，开发周期越来越长，设计质量越来越难于控制，设计成本也越来越高。根据预测，半导体器件的特征尺寸在 2015 年左右将达到 25 纳米，时钟频率将达到 10GHz 以上。这些要求使得以总线结构为主要特征的 SoC 设计方法越来越难以满足要求：

(1) 可扩展性面临的问题

随着电路规模越来越大，片上集成的单元越来越多，数据处理量也越来越大，总线结构的可扩展性差的问题就越来越突出：虽然总线可以有效地连接多个通讯方，但地址资源总是有限的。有限的地址资源将成为扩大电路规模的瓶颈。另外虽说总线由多用户共享，但一条总线是无法支持一对以上的用户同时通讯的，传统总线结构的时间资源利用率是很低的。

(2) 单一时钟同步问题

总线结构要求全局同步，但是随着工艺特征尺寸越来越小，工作频率迅速上升，达到 10GHz 以后，互连线延时造成的影响将严重到无法设计全局时钟树的程度。而且由于时钟网络的的庞大，其功耗将占据芯片总功耗的大部分。由单一系统时钟同步全芯片的工作将极其困难。

所以在 1999 年，就有人提出了一种全新的集成电路体系结构——NOC (Network On Chip)，其核心思想是将计算机网络技术移植到芯片设计中来，从体系结构上彻底解决总线架构带来的问题[1]。

1.2 片上网络研究动态

NOC 是一个崭新的话题，在国际上自 2000 年才刚刚起步。随着技术的不断发展，越来越多的研究机构意识到 NOC 的潜力，纷纷投入到其中并推动着它的发展，使得 NOC 成为了一个十分活跃的学术前沿领域。据初步统计，国际上共有 30 多所大学、研究所以及工业界的研究单位正积极从事 NOC 研究工作[2]。其中影响较大的有瑞典皇家技术学院、斯坦福大学和荷兰菲利普研究实验室等。参与研究的国家还有法国、意大利、芬兰、希腊、以色列、巴西、印度，澳大利亚等。

2000 年和 2001 年处于 NOC 概念构思时期，因此有关的研究著述并不多；2002 年的成果有了稳步提高，步入研究的初步阶段；随着各种因素的成熟以及国外各大 NOC 专项的启动，2003 年、2004 年各出版了 1 部专著，标志着成规模、成系统的研究成果的出现。2005 年至今，各种有关 NOC 的研究论文大量涌现，深入到 NOC 的实现的细节研究，包括路由节点的实现方案，任务的映射以及延时、功耗、服务质量等性能优化方面的研究。

令人遗憾的是，在相关检索中，尚未发现国内在该领域的研究成果。也就是说，中国在 NOC 基础理论研究领域中至少滞后国际先进水平 3~5 年。从这个意义上看，目前国内立项研究 NOC 基础理论已经到了刻不容缓的地步。虽然中国集成电路产业的整体水平落后国外约 10 年时间，但自从 SoC 技术出现以来，国家的科技决策层敏锐地觉察出这是一个跟上国际发展水平的难得机遇，果断地设立了国家自然科学基金 SoC 重大专项、863 计划 SoC 重大专项等重大项目，使国内的 SoC 理论研究基本同步于国际水平。NOC 作为一个崭新的集成电路设计理论体系正处于初创阶段，如尽早立项支持 NOC 相关技术的理论研究工作，一定能够继续保持与国际前沿同步的局面。从这个层面看，NOC 的出现无疑为中国集成电路设计方法学的继续发展又提供了一次很好的机遇。中国若能够全面开展 NOC 学术领域的前沿工作，将为大量创新成果的出现提供一个空间。

1.3 论文结构及内容安排

本论文共分为七大部分。第一章介绍了课题背景、片上网络研究动态，说明论文结构及内容安排；在第二章总结 NOC 中几种常见的拓扑结构、常用的路由技术及路由算法；第三章对两种直接网络 2D Mesh 和 2D Torus 进行了深入研究，尤其对 2D Torus 网络进行了路由算法创新，建立了仿真模型；第四章介绍了片上网

络测试平台的硬件设计工作；第五章具体讨论了路由节点的设计以及在 FPGA 上的实现；第六章介绍了片上网络路由节点的验证和测试；第七章对本设计作了总结，分析了设计中存在不足，展望了未来应该开展的工作。

第二章　片上网络相关概念

自上个世纪90年代末片上网络的概念被提出以来，片上网络各方面的研究都在迅速进行中。片上网络的提出最早是借鉴并行计算机的互连网络，所以片上网络与并行计算机网络有很多的相同点。不同的是并行计算机的互连网络是一个板级的网络，而片上网络是一个芯片上的网络，特别是在路由算法方面，几乎所有的片上网络路由算法都在并行计算机网络中找得到它对应的算法。但是它们又有一些不同，主要表现在以下两个方面：

(1) 片上路由器结构简单，不宜采用较复杂的路由算法。

由于面积所限，片上路由器都是由较简单的逻辑元件组成的。所以，片上网络所采用的路由算法通常都为较简单。而较复杂的路由算法，类似于并行机中的静态维数翻转自适应算法和动态维数翻转自适应算法[3]，虽然可以取得较好的路由性能，但是由于片上网络的资源所限，一般都没有采用。此外，在片上网络中缓存是最宝贵的资源，片上路由器的缓存通常都很小，因此一般对缓存要求比较高的存储一转发机制，也仅仅是在理论分析时会用到，而在实际设计时很少采用。

(2) 片上网络的网络协议。

与并行机不同，在片上网络中没有专门的协议处理机，所有的协议都必须由硬件处理，这就要求片上网络的网络协议不能太复杂。这样，许多在并行机中的流控协议（如Go-Back-N等）在片上网络中都不能应用。所以通常都要求NOC中的路由算法将数据包完整、无损、顺序的投递。

可以看出，上面的两点差异都是由片上网络本身的特点造成的。也正是由于片上网络与并行机有着这些差异，所以有必要单独的整理和总结片上网络中的路由算法及路由技术。

2.1　片上网络的拓扑结构

NOC的拓扑结构就是指NOC中各个节点的连接方式。通常NOC拓扑结构分为两类，一类是直接型网络拓扑，另一类是间接型网络拓扑[4]。

2.1.1 直接型网络

在直接网络中，节点处理器（IP）直接地通过网络彼此连接。常见的直接型拓扑包括网状拓扑（Mesh）、花托拓扑（Torus）等。基于目前的技术特点和应用情况，本论文以直接型拓扑结构为主要的研究对象。下面将介绍本文研究的两种直接型拓扑结构：2D Mesh 结构[5]和 2D Torus 结构[6]。

2.1.1.1 2D Mesh 结构

2D Mesh 结构作为一种最简单、最直观的拓扑结构，如图 2-1。每个节点连接着一个资源和四个相邻的路由器，每个资源通过一个网络接口（NI）连接着一个路由器。其中的资源，可以是一个处理器核，内存，一个用户自定义硬件模块或者是其他任何可以插入插槽并且可以和网络接口相配的 IP（intellectual property）模块。路由器与路由器之间，路由器与资源之间是由一对输入和输出通道连接。通道是由两条单向的点对点总线组成。

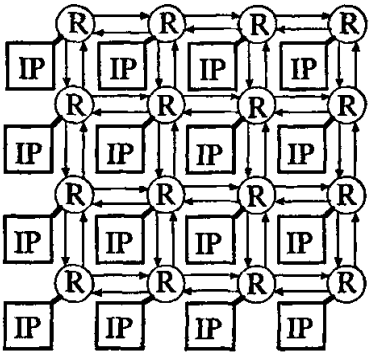


图2-1 4×4 的 Mesh 片上网络

2D Mesh 结构是目前在 NOC 中研究得最早的一种拓扑结构。它结构规则，简单易于实现，但是它边沿和顶点位置节点的相对闭塞性，会极大地影响网络性能。

2.1.1.2 2D Torus 结构

鉴于 2D Mesh 结构上述缺陷，本文还研究了另外一种更具有优势的直接型拓扑结构，这就是 2D Torus 拓扑结构，如图 2-2，是一个 4x4 的 2D Torus 花托结构。

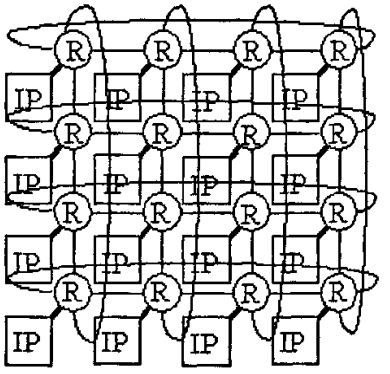


图2-2 4×4 的 2D Torus 结构

2D Torus 结构可以看成是对 2D Mesh 的一种扩展，即在边界的节点上增加了一条长的环路。因此，网络中的所有节点的度为 4，对于一个 m×n 的 Torus 网络，其中 m、n 为每个维度上的节点数，若 m＝n 则称之为规则的 Torus。

2D Torus 拓扑在物理形式上与 2D Mesh 相似，但由于其存在很多的环路，所以在路由算法和路由仲裁方面都要复杂许多。但是 2D Torus 结构在实际的应用中范围比 2D Mesh 广很多，其性能也有很大的提高，由于 2D Torus 拓扑的各个路由节点都是规则的，每个路由节点的结构都一样，所以扩展性也要比 2D Mesh 提高很多。因此，对 2D Torus 拓扑结构的研究，在今后的多处理器系统的具体应用中，有非常重要的意义，我们将在下一章对其进行着重研究。

2.1.2 间接型网络

在间接网络中，节点处理器通过一个(或更多)的中间开关节点相互连接。间接型拓扑包括蝶形拓扑、Banyan、Fat-Tree 拓扑（如图 2-3）等。

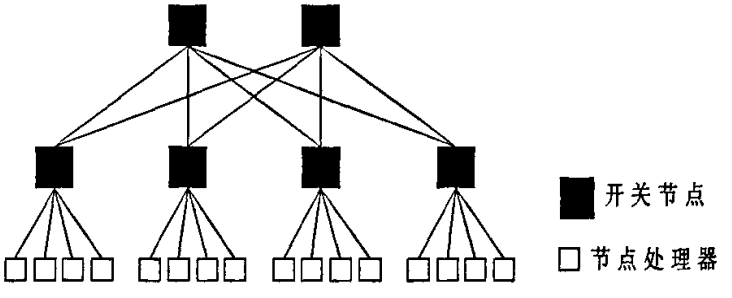


图2-3 Fat-Tree 拓扑

2 常用路由技术

本文所介绍的常用路由技术主要包括了三方面的内容，包交换技术、虚拟通道技术和死锁避免技术。包交换技术关注的是数据包是怎样从输入通道交换到输出通道的。采用的包交换技术不同，所产生的延迟就不同，网络的延迟与包交换技术直接相关。虚拟通道技术主要是结合虫洞路由一起使用，可以大大的降低阻塞发生的概率。最后的死锁避免技术，介绍了通过转弯模型避免死锁的一些内容。

.2.1 包交换技术

常用的包交换技术主要有四种：存储转发（Store-and-Forward）、虚拟直通（Virtual Cut-Though）、虫洞路由（Wormhole Routing）和偏转路由（Deflection outing）[7]。下面分别介绍它们的概念和一些相关问题。

.2.1.1 存储转发（Store-and-Foward）

存储转发方式是指的节点（路由器）在转发数据包的时候，要先将整条数据包存储在缓存中，然后再转发出去。由此可以看出，存储转发对路由器缓存要求比较高，特别是在数据包长度比较大的时候。

假设一个长度为 L 的数据包采用存储转发方式，经过 H 个节点到达目的地，则可以算出它的延迟为：

$T = (L/BW + R) \times H$。

其中，L 是数据包的长度，BW 是网络的带宽，R 是每个节点的寻径延迟，H 是所经过的节点数。

.2.1.2 虚拟直通（Virtual Cut-Though）

与存储转发方式缓存整个数据包不同，在虚拟直通中，数据包被分成了许多的片段（flits）。每个包的头片段控制路由，其余的片段跟在头片段之后，以“流水”的方式在网络中传输，如图 2-4。当阻塞发生时，整个数据包都存储在阻塞发生的路由器缓存里。这样，每个路由器只需要有限的缓存空间，即阻塞发生时所需缓存的数据包的长度大小就够了。

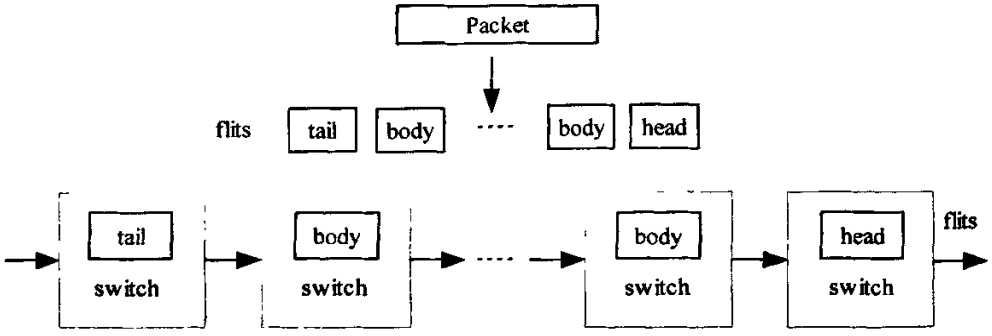


图2-4 数据包分片示意图

假设一个长度为 L 的数据包采用虚拟直通方式，经过 H 个节点到达目的地，则可以算出它的延迟为：

$T = L/BW + R \times H$。

其中，L 是数据包的长度，BW 是网络的带宽，R 是每个节点的寻径延迟，H 是所经过的节点数。

2.2.1.3 虫洞路由（Wormhole Routing）

虫洞路由与虚拟直通方式及其相似，也是将数据包分成许多片段（flit）；每个包的头片段控制路由，剩余的片段以流水的方式在网络中向前“蠕动”。每个片相当于虫的一个节，“蠕动”以节为单位顺序地向前爬行。当阻塞时，各个片段（flit）存储在阻塞发生时所在的各个节点里面。这样，每个路由器只需要很少的缓存空间，即只需缓存当前的一片以及阻塞发生时在刹车信号到来之前发进来的几个片段就可以满足了，而不需要缓存整个数据包。

在延迟方面，也与虚拟直通一样，都为：

$T = L/BW + R \times H$ 。其中，L 是数据包的长度，BW 是网络的带宽，R 是每个节点的寻径延迟，H 是所经过的节点数。存储转发和虫洞路由延迟比较如图 2-5。

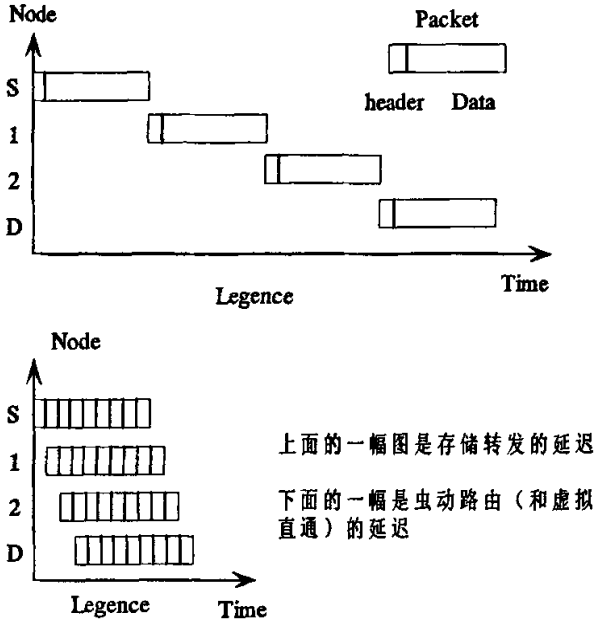


图2-5 虫洞路由与存储转发传输延迟比较

从上面的描述中可以看出，虚拟直通方式和虫洞路由方式是很相似的。事实上，在没有阻塞的情况下，虚拟直通和虫洞路由是完全一样的。它们两者的差别在于发生阻塞时对被阻塞的数据包的处理：虚拟直通将整个的数据包储存在发生阻塞的路由器中；而虫洞路由是将各个片段分散到各个节点中。如图 2-6 所示。

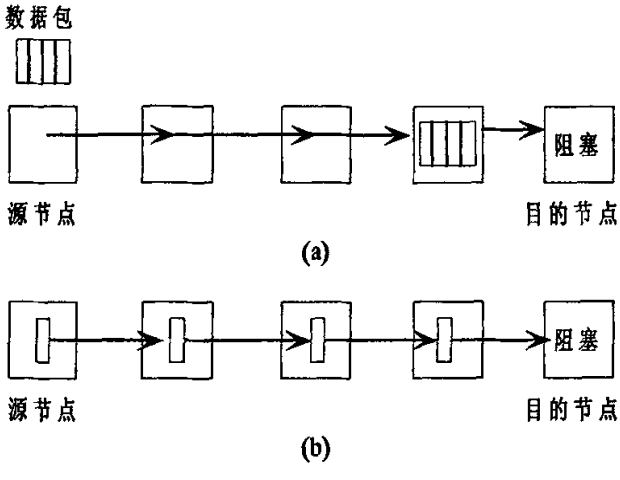


图2-6 虚拟直通(a)和虫洞路由(b)发生阻塞时的比较图

虫洞路由的优点：

(1) 相对的使网络延迟对路径长度不那么敏感。从公式 T = L/BW + R×H 可以看出来，延迟由两部分组成，前一项是发送数据包的时间，后一项是在各个节点寻径时间的总和。在实际应用中，一般数据包的长度比较大，远远大于各个节点寻径时间的总和，所以延迟主要由前一项决定。这个优点很好的适应了 NOC 可升级性（scalable）的要求。

(2) 虫洞路由只需要很少的路由器缓存。这一点也很好的迎合的 NOC 的特点。

虫洞路由带来的缺点：

(1) 只要一条通道开始传送包的第一个片段，它就被这条数据包所占用了，必须等到传完所有片段才能被释放，而被其它消息使用。也就是说，一个包发生阻塞时，可能会同时阻塞其他多个数据包，进一步的造成网络拥塞。

(2) 当死锁发生时，很难将发送来的消息集合在缓存中以晚点再传送。因为，虫洞路由的消息长度没有限制，而且缓存通常很小。

2.2.1.4 偏转路由（Deflection Routing）

偏转路由，也叫作烫手山芋（Hot-Potato）路由。正像它的名字“烫手山芋”一样，在这种极端的路由方式中，路由器并不缓存任何数据包，所有进来的数据包都被立即转发，所以数据包总是在运动的。偏转路由最大的好处是不需要任何缓存空间（事实上，仍然需要一个缓存单元来缓存当前的数据）。如果有多个数据包竞争一个输出的时候，它则按照偏移策略将最高优先级的数据包发送到正确的方向上，然后误传其他低优先级的数据包。

2.2.2 虚拟通道（Virtual Channel）

采用虚拟通道[8]可以避免死锁[9]的出现和对头阻塞 (Head of Line）[10]。

2.2.2.1 死锁问题

当多个数据包循环等待的时候，就会发生死锁，如图 2-7。

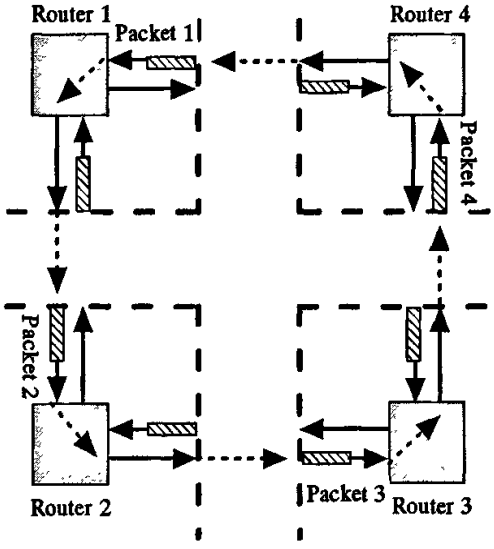


图2-7 四个数据包循环等待发生的死锁

数据包 1 占用着路由器 1 东面的入通道，它要路由到路由器 2 去。但是路由器 2 北面的入通道又被数据包 2 占用着，数据包 2 要路由到路由器 3 去。同时路由器 3 西面的入通道又被数据包 3 占用着，数据包 3 又要路由到路由器 4 去。但是路由器 4 南面的入通道又北数据包 4 占用着。最后，数据包 4 要路由到路由器 1 中，但是路由器 1 的东面入通道已经北数据包 1 占用了。这样就发生了循环等待，导致了死锁。在死锁中的数据包的延迟会无限制的增大，是永远没有办法到达目的地的，所以死锁也是路由算法要极力避免出现的一个问题。

2.2.2.2 队头阻塞（Head of Line）研究

如果输入端口等待发送的数据都使用一个队列排队，就会出现对头阻塞问题。即如果 FIFO 头部的数据要去的端口正忙，那么此数据队列只能在 FIFO 中等待，这时即使它后面的数据要去的端口是空闲的，也没有机会发送。这种阻塞会极大地降低交叉开关的流量。下面对其进行建模分析。

假定，输入队列与输出队列均是 FIFO，其处理包的速度相等，crossbar 是一个 $N*N$ 的交换结构，对某个指定的输入对列，有包到来的概率是 P，该包对输出端口的选择是任意的，即到达其中一个端口的概率为 $1/N$。在输入端，输入队列之间是独立的，包与包之间也是独立的。

把时间划分成若干个时隙，若在某一时隙内，输入队列所有 FIFO 的第一个包

均选择不同的出口，则无阻塞。这是一种理想的状态。若其中有 K 个包选择同一个出口，则随机选取一个包通过。即：选择该出口的包被选中的概率是 $1/K$。其余未被选中的包等到下一个时隙发送。

对阻塞的建模：

定义 1：A_m^i 是 m 时隙出现在 FIFO 头部并请求输出到 i 端口的包的个数。

定义 2：B_m^i 是 m 时隙请求输出到端口 i 但是被阻塞了的包的个数。

$B_m^i = \max(0, B_{m-1}^i + A_m^i - 1)$，其中 B_{m-1}^i 上一个时隙被阻塞的申请端口 i 的包的个数，A_m^i 这一时隙又进来的申请端口 i 的包的个数，1 是本时隙要处理一个包。

定义 3：F_{m-1} 是 m 时隙还允许进入 FIFO 头部的队列个数。则可以用 A_m^i 与 B_m^i 来表示它。

$$F_{m-1} = N - \sum_{i=1}^{N} B_{m-1}^i$$

或者：$$F_{m-1} = \sum_{i=1}^{N} A_m^i$$

定义 4：Crossbar 的吞吐量（输出端口利用率）为：$\rho_0 = \overline{\frac{F}{N}}$

下面对吞吐量进行极限值的计算：

根据 $\rho_0 = \overline{\frac{F}{N}}$ 和 $F_{m-1} = N - \sum_{i=1}^{N} B_{m-1}^i$

$$\rho_0 = \overline{\frac{F}{N}} = \frac{1}{N}\left(\frac{1}{M}\sum_{m=1}^{M}\left(N - \sum_{i=1}^{N} B_{m-1}^i\right)\right)$$

$$= \frac{1}{N}\left(N - \sum_{i=1}^{N} \overline{B^i}\right)$$

$$= 1 - \overline{B^i}$$

推出 $\overline{B^i} = 1 - \rho_0$。

这样，得到了 $\overline{B^i}$ 的一个表达式。

之后，利用 $B_m^i = \max(0, B_{m-1}^i + A_m^i - 1)$，得到 $\overline{B^i}$ 的另一个表达式：

$$\overline{B^i} = \frac{(N-1)}{N} \times \frac{\rho_0^2}{2(1-\rho_0)}$$

则有 K 个包要输出到 i 端口的概率为：

$$P(A_m^i = k) = C_{P_{m-1}}^k \left(\frac{1}{N}\right)^k \left(1-\frac{1}{N}\right)^{P_{m-1}-k}$$

求上式的概率发生函数（PGF）：

$$A(z) = \sum_{k=0}^{P_{m-1}} z^k P(A_m^i = k) \Rightarrow B(z) \Rightarrow \overline{B^i} = \lim_{z\to 1} B(z)$$

联立两式：

$$1-\rho_0 = \frac{(N-1)}{N} \times \frac{\rho_0^2}{2(1-\rho_0)} \Rightarrow \frac{N-1}{N} = \frac{2(1-\rho_0)^2}{\rho_0^2}$$

当 $N\to$inf 时$(N-1)/N$ 最大，此时 $\rho_0 = (2-\sqrt{2}) = 0.586$

所以，由于 HOL（Head of Line）阻塞问题，导致实际吞吐量只能达到总链路吞吐量的 58.6%。在 3.4.2.2 节“均匀流量下，虚拟通道对 Mesh 网络中数据包传输的影响”中将会提供相关的仿真图，可以证明这一结论。为了避免该阻塞问题，我们引入了虚拟通道概念。

2.2.2.3 虚拟通道

所谓的虚拟通道，就是指在两个相连的路由器中一对由共享物理信道连接的缓存。

通常，在采用了虫洞路由中的网络引入虚拟通道，即把一个缓存划分为多个虚拟通道，这些虚拟通道通过时分复用一条物理信道。如图 2-8 所示，一个 12 个 flits 的缓存被划分为 4 条虚拟通道，每条虚拟通道有 3 个 flits 的缓存。

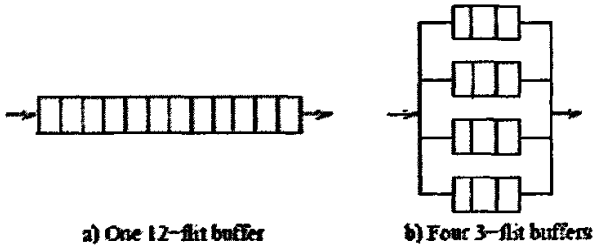


图2-8 虚拟通道示意图

在虫洞路由网络中引入虚拟通道技术，如图 2-9 所示，每个数据包就将占用一条虚拟通道，如果一个数据包被阻塞了，其他的数据包还可以通过剩下的虚拟通

道到达自己的目的地。这样，不仅可以避免冲突，改善连接的利用和网络吞吐量，还能有效的避免死锁。

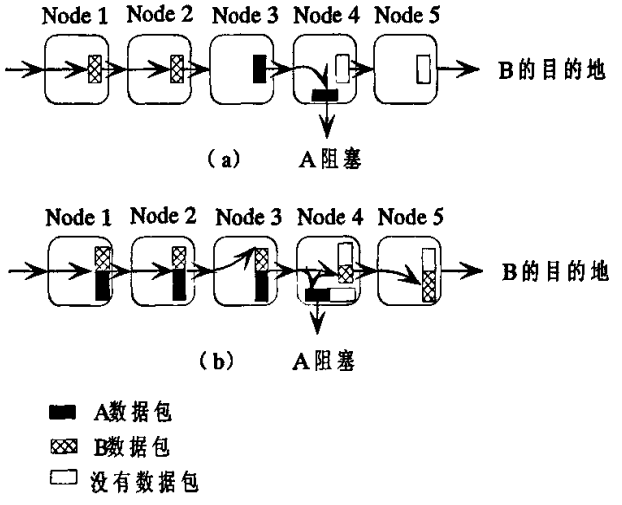


图2-9 虚拟通道的引入避免了死锁的发生

图（a）中没有采用虚拟通道，当A包在节点4被阻塞之后，它的最后一个片段被阻塞在了节点3向节点4方向发送的缓存中。后来的B包要通过节点4到达B的目的地，但是由于没有采用虚拟通道，B包被A包阻塞在节点3了。

图（b）引入虚拟通道后，A包同样在节点4发生阻塞，最后一个片段被阻塞在了节点3向节点4方向发送的缓存中。但是后来的B包就可以进入其他虚拟通道里面，绕过发生阻塞的A包，到达自己的目的地。

2.3 路由算法

NOC的拓扑结构必须保证每个节点可以发送数据包到其他节点。当没有完善的拓扑结构的时候，路由算法决定数据包从源地址开始选择哪一条路径到目的地址。所以，有效的路由算法对NOC网络性能的好坏是至关重要的。

路由算法可以按照不同的标准分为不同的几类。比如说源路由（Source Routing）和分布式路由（Distributed Routing），确定路由（Deterministic Routing）和自适应路由（Adaptive Routing）[11]。这里的讨论基于后一种分类方法。

2.3.1 确定路由（Deterministic Routing）

确定路由是一种常见的路由，它的路由路径只于起点地址和终点地址有关，给定起点和终点地址，路由路径就被确定了，与当前网络的状态无关。而在确定路由中，使用的最多的则是维序路由（Dimension-Ordered Routing），因为它有着非常简单易实现的路由逻辑。在维序路由中，每个数据包一次只在一个维上路由，当在这个维上到达了恰当的坐标之后，才按由低维到高维的顺序在另外的维上路由。因为数据包是按照着严格的单调的维数变化的顺序在通道内路由，所以维序路由也是没有死锁的。按照在不同拓扑结构的网络中路由，维序路由包括了 2D Mesh 中的 XY 路由和在超立方体（Hyper Cube）中的 E-cube 路由。

确定路由优点是，路由算法简单，在网络低拥塞环境下能获得较低延迟。但是由于其不能响应动态的网络状态变化，所以当网络拥塞增加时，性能迅速降低。

2.3.1.1 XY 路由

2D Mesh 中的 XY 路由是最简单的一种路由算法，它的路由过程是：先 X 方向将数据包包送至目的节点所在的列；再沿 Y 方向将信数据包送至目的地处理器所在的行，如图 2-10。

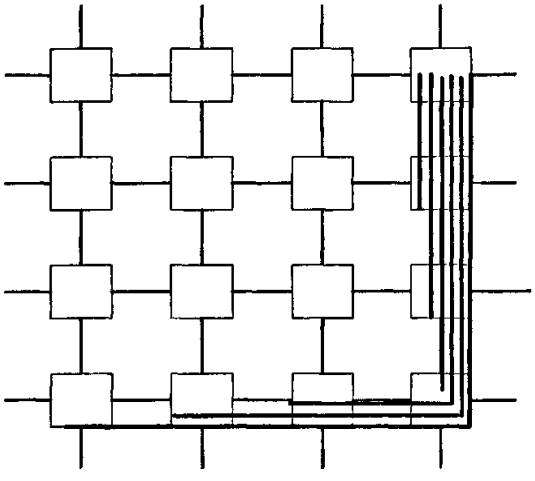


图2-10 XY 路由的过程

具体的处理流程如图 2-11。当路由器开始处理一个数据包的时候，因为采用的是 XY 路由，它先会判断 X 方向上是否到达了目的节点所在的列，如果没到达，

再比较目的 X 和当前 X，如果目的 X 比当前 X 大，则还应该向东路由；如果目的 X 比当前 X 小，则说明还应该向西路由。如果判断出来 X 方向上已经到达了目的节点所在的列，则再比较目的 Y 和当前 Y，如果相等，则说明已经该数据包已经到达了目的地，则收包，向 IP 路由。如果目的 Y 大于当前 Y，则还应该向南方路由；如果目的 Y 小于当前 Y，则说明还应该向北方路由。

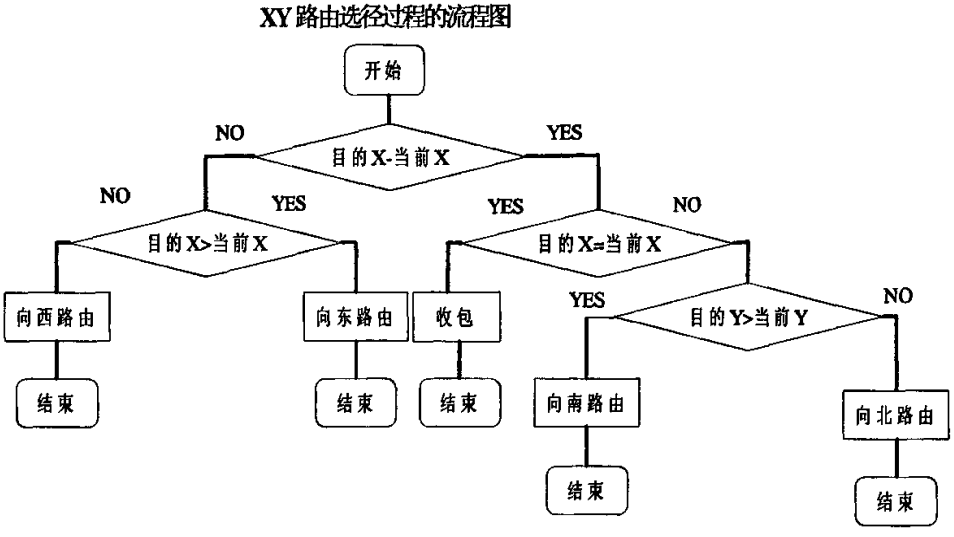


图2-11 XY 路由的选径流程

具体举例来说，一个从（1，2）发出的数据包，其目的地是（3，4），则它采用 XY 路由算法的路由路径是：

（1，2）—>（2，2）—>（3，2）—>（3，3）—>（3，4）。

2.3.1.2 E-cube 路由

E-cube 路由和 XY 路由很相似，都是先在一个方向（维）上路由，然后再在其它方向（维）上路由。具体来说，前面提到了在 n 维立方体中，每个节点是用一个 n bit 的二进制编号表示的。每个节点 n 条输出的通道，其中第 i 条通道就对应的第 i 维。在 E-cube 路由算法中，数据包的头部携带了目的节点的地址 d。当 n 维立方体中的一个节点 v 收到一个数据包时，E-cube 路由算法计算路由向量 c=d ⊕v，其中 ⊕ 是逻辑运算中的异或运算。如果 c=0，说明数据包已经到达了目的地，则传向 IP。否则，数据包将被送往第 k 维的输出通道，其中 k 是 c 里面最右边（或者最左边）的‘1’的那一维。

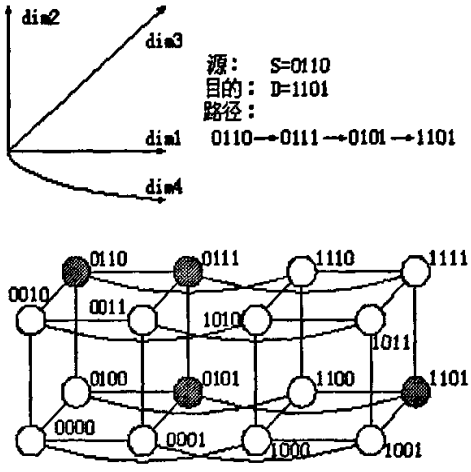


图2-12 E-cube 路由过程

举例来说，如图 2-12，一个由 s=0110 产生的数据包要传向目的地为 d=1101 的节点。首先计算路由向量 c1=d⊕s=1101⊕0110=1011。取最右边的不为零的一位为 k，则 k=1，说明这是的数据包将会被送往第 1 维上的通道。然后到达了 v1=0111，再计算路由向量 c2=d⊕v1=1101⊕0111=1010，得出 k=2，然后送到第 2 维的通道上。随即到达 v2=0101，再一次计算 c3=d⊕v2=1101⊕0101=1000，得出 k=4，接着把数据包送到第 4 维的通道上。最后到达了目的地 d=1101。最终，得出的路由过程是：

0110->0111->0101->1101。

从上面的路由过程可以看出，虽然 E-cube 路由与 XY 路由是在不同维数的网络中路由，但是它们都有很相似的共同点：即一次只在一个方向（维数）上路由，直到在该方向（维数）上当到达了与目的地址相同的节点，再按照单调的顺序改变方向在其它维上路由，XY 路由是由 X 方向改变到 Y 方向的顺序，E-cube 路由是从低维到高维（或者从高维到低维）的顺序。而这正是维序路由算法的典型特征。

2.3.2 自适应路由（Adaptive Routing）

确定路由优点是，路由算法简单，在网络低拥塞环境下能获得较低延迟。但是由于其不能响应动态的网络状态变化，所以当网络拥塞增加时，性能迅速降低。

所谓的自适应路由，就是指数据包的路由路径不只与起点地址和终点地址有

关，还要考虑网络的状态。也就是说，有同一对起/终点的地址的数据包，在不同的网络状态下，它们的路由路径也可能不同。

自适应路由的优点是采用自适应路由的路径，避免了网络拥塞，可以得到更高的网络带宽饱和值；但是它的路由逻辑较复杂，在网络低拥塞的情况下开销较大，而且还存在死锁问题。

在片上网络中，由于路由器结构所限，一般都采用的是比较简单的自适应算法，如带有一定自适应性的维序路由。所以，下面将着重介绍一下这种算法。

2.3.3 带有一定自适应性的维序路由

一般的维序路由是在维序路由中，每个数据包一次只在一个维上路由，当在这个维上到达了恰当的坐标之后，才按由低维到高维的顺序在另外的维上路由。而这里带有一定自适应性的维序路由则是，当数据包沿某一维（如 X 方向），从最低维到最高维路由的过程中发生阻塞的时候，即向另一维（Y 方向）发出传送请求，如果请求得到应答则向该方向传送数据，否则又转回 X 方向请求，如此循环，直到请求得到应答。同时规定，不允许数据向远离目的节点的方向运动，所以这种带有一定自适应性的维序路由也是没有死锁的。

下面以 4×4 的 2D Mesh 网络中的带有一定自适应性的 XY 路由为例，具体解释一下这种算法的路由过程。如图 2-13 所示，假设一个数据包路由的起点为(1，4)，终点为(4,1)。如果按照一般的 XY 路由的话，它的路由路径应该是(1,4)->(2,4)->(3,4)->(4,4)->(4,3)->(4,2)->(4,1)。这时如果假设数据包在节点(3,4)发生了阻塞，不能继续向(4,4)发送。如果按照原来的 XY 路由，数据包就不能在往下发送，被阻塞在了(3,4)中。这时如果采用的是带有一定适应性的 XY 路由，在向节点(4,4)发送传送请求没有被允许之后，则它就会向 Y 维方向上的(3,3)节点发送传送请求，被允许之后，数据包就被发往节点(3,3)了。到达(3,3)后，又会先向 X 维方向上的(4,3)发送请求。就这样最终的路由路径为(1,4)->(2,4)->(3,4)->(3,3)->(4,3)->(4,2)->(4,1)。

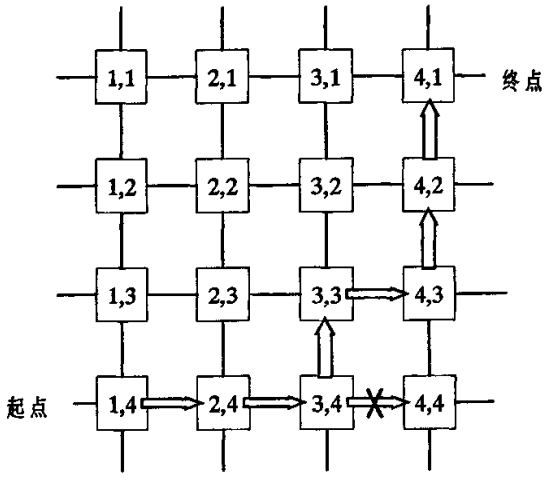


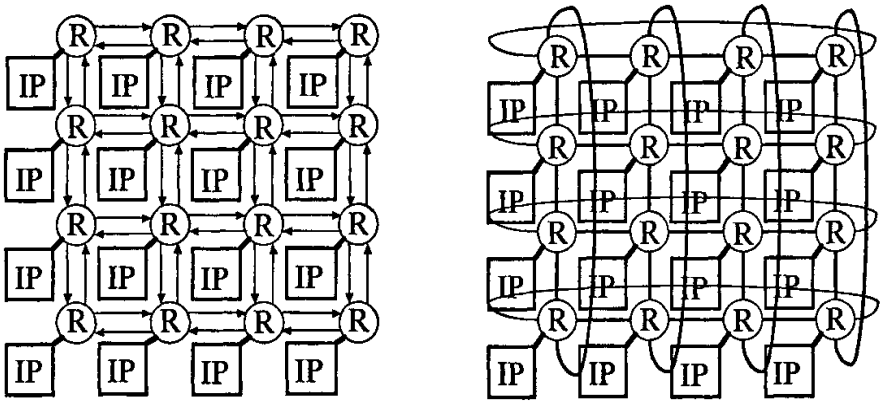
图2-13 带有一定自适应性的 XY 路由的路由过程

从上面的路由路径可以看出，带有一定自适应性的 XY 路由和一般的 XY 路由的差别就是在某个节点发生阻塞之后，它可以不按照先 X 维再 Y 维的顺序路由，而可以是向另一个维发送数据包。这样就可以从一定程度上缓解网络的拥塞。

第三章　NOC 拓扑结构及相应的路由算法的研究

上一章中已经讨论过片上网络的拓扑结构，这里对两种具有代表性的两种直连型拓扑结构——2D Mesh 结构和 2D Torus 结构，进行进一步深入研究。

3.1　2D Mesh 结构和 2D Torus 结构的拓扑结构比较



(a) 用于NOC的Mesh结构　　(b) 用于NOC的Torus结构

图3-1 NOC 的 Mesh 和 Torus 结构

图 3-1 显示了用于 NOC 的 2D Mesh 和 2D Torus 结构。其中，每一个路由节点 R（Router）都与一个 IP（或者是处理器核）通过一个本地的接口（Network Interface）相连接，IP 产生的数据由此进入网络，并按照网络的规则进行传送或者通信，而不再采用传统的总线方式，这样，极大的提高了并行处理的速度和性能。图 3-1（a）表示的是用于 NOC 的 2D Mesh 网络结构，它是目前在 NOC 中研究得最早的一种拓扑结构。它结构规则，简单易于实现，但是由于 Mesh 结构中每个节点并不完全相同，会影响网络的拓展性。图 3-1（b）表示的是用于 NOC 的 2D Torus 网络结构，它是一种完全对称的直连网络拓扑形式，其良好的特性可以使得 NOC 具有规则的对称性，路径的多样性，同时，能更方便 NOC 进行扩展。从图中可以清楚的看到，在一般的 2D Mesh 结构中，边缘节点与内部节点在物理链路的连接上就具有差别，例如，顶点与仅与另外两个节点连接，边上的节点与三个节点连接，而内部节点则直接连接了另外四个节点。所以，在对 2D Mesh 结构进行扩展的时候，就必须

考虑节点位置不同所带来的结构上的变化。但是，在2D Torus结构中，由于其结构是规则对称的，每个节点都直接与其他四个节点连接，所以具有更好的扩展性。

3.2 2D Mesh结构和2D Torus结构的路由算法比较

路由算法[12]是决定网络性能的另外一个重要方面，在NOC的片上环境中，路由节点应该尽量的设计得简单，不宜消耗过多得资源，因此，路由算法也要尽量简单化，具有尽可能小的路由表。另外，避免死锁和活锁[13]也是路由算法必须考虑的问题。在目前的NOC研究中，一般是在Mesh结构中采用XY路由来实现其拓扑和避免死锁活锁。这类确定性的路由算法实现方式简单，针对NOC的具体情况，能方便的进行软件和硬件仿真，因此也成为了目前NOC结构研究中一种被广泛采用的方法。

但是Torus结构中存在环绕信道，因此引入了很多环路。无法简单采用与Mesh结构中相类似的方法，而如果以Mesh中常用的XY路由为基础，进行一定改进，引入一定的虚通道，又会使得整个路由结构，判断机制变得比Mesh结构中复杂很多，同时也使片上路由节点的结构变得更加复杂，增加了设计和实现的开销。

Torus的路由算法主要的研究目的就是为了避免死锁和活锁，以及尽可能的发挥网络的最佳性能。在前面研究的基础上可以知道，在Mesh中使用的转向模型是网络设计无死锁的经典技术，它的基本思想是通过禁止网络中一定数量的转向来破除一些环路，从而使信道失去依赖关系。但是由于 Torus 网络中存在的环路比Mesh中多得多，所以传统的禁止转向技术已经不能适用。为了避免死锁，在有的研究中采用了虚网络的方法，该方法是将物理信道逻辑上划分为若干条虚信道，每条信道都有自己独立的缓存。由节点和虚信道组成的网络称为虚网络。对于NOC中，虚信道的方法在使用上会遇到一些比较大的困难，会使得增加很多附加的资源消耗率，而这种代价对NOC的环境来说很大，另外，由于虚网络的划分，会增加节点在仲裁和判断时的开销，使得路由节点更复杂。因此，我们设计了一种适用于NOC的2D Torus网络的路由方法。

3.3 2D Torus结构的路由算法

目前应用于2D Torus结构的路由算法，无论是自适应路由还是确定性路由，无论是采用转向模型还是虚网络，都无法摆脱基于二维空间选路的思想，我们设

计的路由算法的最大贡献，就是打破了这种思想禁锢，将平面内的各物理通道多维化，并将路由节点的标号与维度相对应，在选路时按低维到高维的顺序寻找路径。这种路由算法在 4x4 的 2D Torus 结构下可以得到完美的应用，虽然有些 mxn 的 2D Torus 结构下这种算法的应用还需有针对的进行改进，但这种多维化的思想却非常有意义。下面我们对这种路由算法在 4x4 的 2D Torus 结构的实现进行详细介绍。

3.3.1 节点标号与物理通道多维化

因为要将将平面内的各物理通道多维化，并将路由节点的标号与维度相对应，为了看起来更直观、更加便于理解，可将 4x4 2D Torus 网络先立体化。

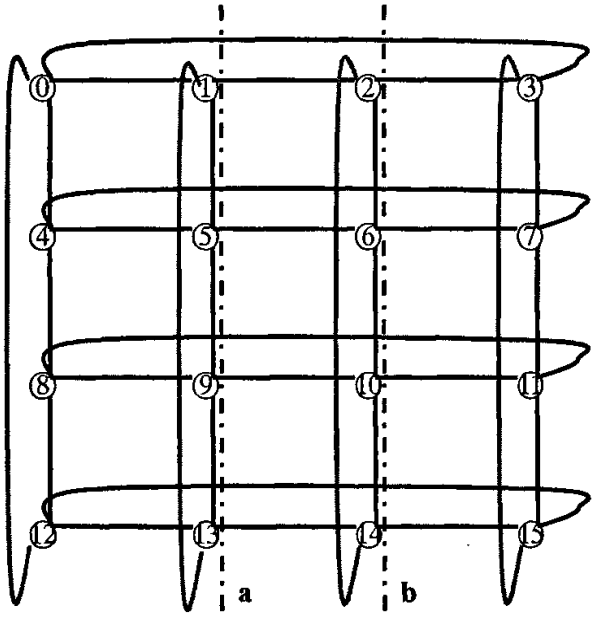


图3-2 原始 4x4 2D Torus 网络拓扑

首先，我们将 Torus 拓扑（如图 3-2）按图中的虚线 a 向里对折一次，再按图中虚线 b 向里对折一次，这样就形成了一个长方体，其中原来在 Torus 中处于同一行的四个节点，如：0、1、2、3；4、5、6、7；8、9、10、11，就构成了 4 个正方形，如图 3-3。

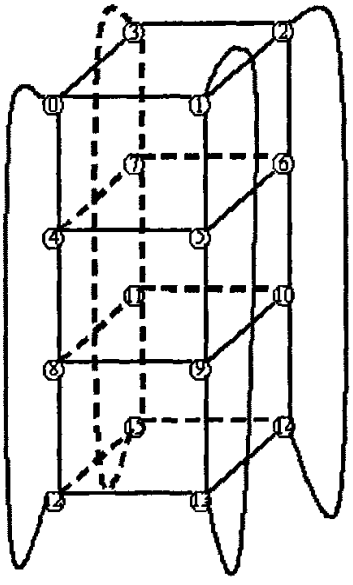


图3-3 立体化后的 4x4 2D Torus 网络

图 3-3 中的长方体可以看作是由上下两个正方体拼接而成，我们将下面的一个正方体（即以节点 8、9、10、11、12、13、14、15 为其顶点的正方体）的上下两个面，即 8-9-10-11 面与 12-13-14-15 面，交换位置，得到的图形如图 3-4。

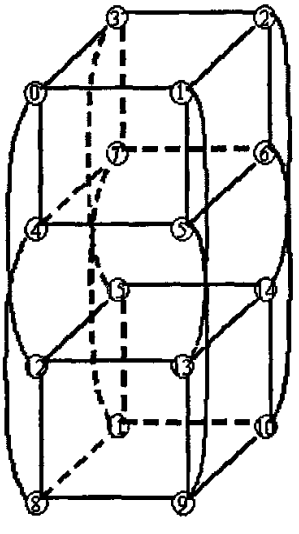


图3-4 立体化后的 4x4 2D Torus 网络再进行变换对

按图 3-5(a)的指示对图 3-4 中图形的节点重新进行标号。节点坐标的格式为 n3n2n1n0，其中 ni 代表第 i 维。在这里我们规定，从左向右为第一维的正方向，即 n0；从外向里为第二维的正方向，即 n1；从上向下为第三维的正方向，即 n2；从上面的一个正方体向下面一个正方体为第四维的正方向，即 n3。这样我们令节点 0 的坐标为 0000，就可以得出其他所有的坐标，编号结果如图 3-5(b)所示。

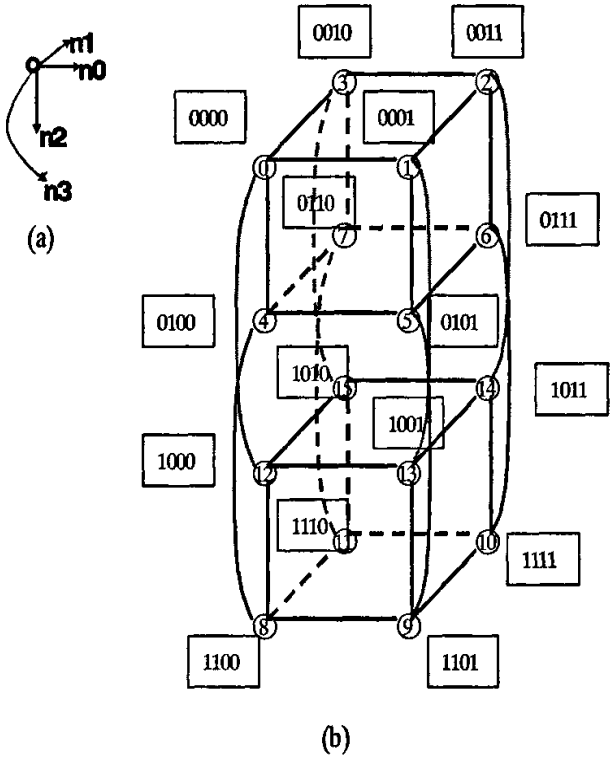


图3-5 4x4 2D Torus 网络节点重新标号

需要要说明的一点是，这里并不是真正的把 2D Torus 拓扑变换成立体拓扑，所以并不存在立体结构的布线问题。以上将 Torus 变换到立体拓扑的过程的目的有两个：一个是得到节点对应与 0~15 编号的四位坐标 n3n2n1n0；第二个是为了让后面将描述的路由过程更加直观化。接下来，带着得到与 0~15 编号的四位坐标回到 2D Torus 拓扑中，我们得到了如图 3-6 所示的拓扑图。

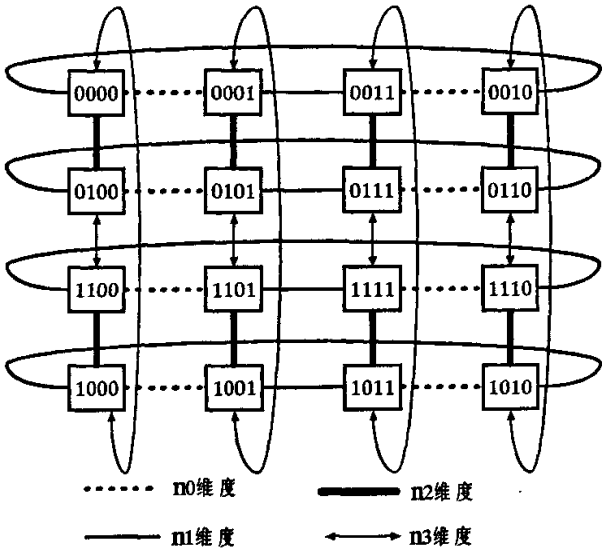


图3-6 网络络节点标识后的 Torus 结构图

可以看到，每一个节点都和其他四个节点相连，分别对应了四个维 n3、n2、n1、n0 变化的方向，因此以前 2D Mesh 网络中的东、南、西、北四个方向的输入输出端口，在 2D Torus 拓扑中就变为了 n3、n2、n1、n0 四个方向（但是和东、南、西、北四个方向并没有一一的对应关系，比如说 0 节点到 1 节点是向东路由，改变的是第 n0 维；1 节点向 2 节点也是向东路由，但是改变的是第 n1 维）。

在最后将节点连成网络的时候，就可以按照图 3-6，将两个节点对应的同维的‘输入输出口对’相连就可以了。例如，要连接 0 节点和 12 节点的时候，我们就只需将 0 节点的输出 n3 连接到 12 节点的输入 n3，将 0 节点的输入 n3 连接到 12 节点的输出 n3 就可以了。

3.3.2 路由算法和路由过程

根据上面论述的设计，下面将详细叙述所采用的路由算法和路由过程。

本文采用的路由算法就比虚网络要简单许多，由于数据在传输过程中，总是在不同的维度上传送，而对 4×4 的 2D Torus 结构来说，维度为四，所以一个数据包从源节点到目的节点最多仅需要四跳。对具体的路由算法描述如下。

假设源地址为 sn-1sn-2…s1s0，目的地址为 dn-1dn-2…d1d0。路由算法如下：

sn-1sn-2…s1s0 ⊕ dn-1dn-2…d1d0 = rn-1 rn-2 …r1 r0，其中，rn-1 rn-2 …r1 r0 就是具体的路由值。

路由过程的描述如下：

sn-1sn-2…s1s0 —> sn-1sn-2…s1s0 ⊕ r0 —> sn-1sn-2…s1s0 ⊕ r1 —>…—> dn-1dn-2…d1d0

简单的来说，其路由过程就是先比较当前节点和目的节点的最低维，如果不同则向最低维的方向路由，如果相同则再比较当前节点和目的节点的次低维，如果不同则向次低维方向路由，如果相同再比较下一维。直到路由到当前节点的所有维和目的节点的所有维都相等，则收包。在图 3-6 中，分别用 4 中不同的线段表示处于不同维度上的路径，可以清楚的看到，由于 4x4 2D Torus 网络中所有的路径均被分配到 4 个维度上，每一个节点都与 4 条在逻辑上属于不同的维度的链路信道相连接。

这种方法使得路由算法非常简单，假设数据包的目的地址为 d3 d2 d1 d0；当前节点的坐标为 p3 p2 p1 p0。路由算法可用如下的伪代码表示：

```
if   d0 ≠ p0
    then request for n0 direction;
else if d1 ≠ p1
        then request for n1 direction;
    else if d2 ≠ p2
            then request for n2 direction;
        else if d3 ≠ p3
                then request for n3 direction;
            else request for local;
end;
```

例如，假设有一个数据包要从节点 0000 发送至 1011，那么其路由过程为：

在节点 0000 中，首先比较最低维 n0，0000 与 1011 不同，则向 n0 方向路由至节点 0001；

在节点 0001 中，最低维相同 n0，比较次低维 n1，0001 与 1011 不同，则向 n1 方向路由至节点 0011；

在节点 0011 中，最低维 n0，次低维 n1，次高维 n2 都相同，比较最高维，0011 与 1011 不同，则向 n3 方向路由至节点 1011；

在节点 1011 中，所有维都相同，收包。如图 3-7。

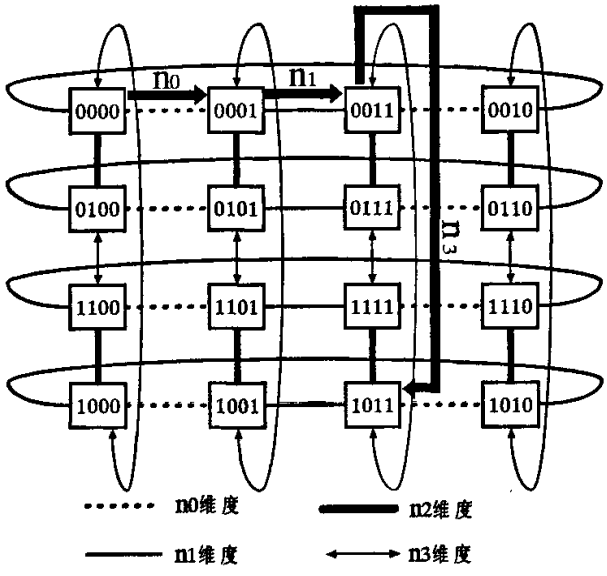


图3-7 0000 节点向 1011 节点传送数据包的选路过程

下面证明本文设计的 4 维度递增判断算法是无死锁和活锁的。

根据网络通信的理论，我们已经知道如下的定理。

定理 若直连网络 G 中的所有信道可按一定的规则标号，而路由算法 R 沿着信道的序号严格递增（或者递减）的顺序路由分组，那么该算法是无死锁的。

由于根据我们的划分规则，4×4 的 Torus 网络已经被划分成 4 个维度，所以，可以认为网络 G 被分成了 4 个子网络，Gs3 、Gs2 、Gs1 、Gs0，对于任意一个维度上的子网络来说，由于属于同一个子网络上的任意两条信道都没有连接在一起，并且在我们的路由算法中，由于在一个子网络上的路由只会发生一次，所以这一次路由一定是递增，或者是递减的。因此，该算法在 Gs0 中是严格按照序号递增或者递减选择路由的，由定理可得，该算法是无死锁的。同理，由于 Gs3 、Gs2 、Gs1 与 Gs0 在本文的 Torus 结构中是规则的对称的，所以在每一个子网络中都是无死锁的。

最后，由本文的路由算法可以知道，数据在进入网络后，只会在一个子网络中路由一次，所以进入第二个子网络中后，不会再回到前一个子网络中，即不会产生跨子网络的环路，所以，该算法在整个网络中都是无死锁的。

另外，该算法总是使得数据沿着最短的路径路由，因此不会发生活锁。所以，该算法在整个网络中都是无死锁和活锁的。

3.4 性能仿真

3.4.1 仿真模型的建立

使用 OPNET 进行仿真，这是由它的主要优点决定的：

(1) 网络仿真能够为网络的规划设计提供可靠的定量依据。网络仿真技术能够迅速地建立起现有网络的模型，并能够方便地修改模型并进行仿真，这使得网络仿真非常适用于预测网络的性能，回答"What…if…"这样的问题。

(2) 网络仿真能够验证实际方案或比较多个不同的设计方案。网络仿真能够通过为不同的设计方案建立模型，进行模拟，获取定量的网络性能预测数据，为方案的验证和比较提供可靠的依据。这里所指的设计方案可以是网络拓扑结构、路由设计、业务配置等等。

(3) OPNET 能够准确的分析复杂网络的性能和行为。得到的仿真输出可以以图形化显示、数字方式观察、或者输出到第三方的软件包去。此外，一系列仿真运行的结果被自动整理到一个单一的 OPNET 输出文件中，以便于比较分析（比如相对于网络负载的端对端延迟）。

图 3-8 是片上网络中的一个节点的模型，一个 4x4 的网络就是由 16 个这样的节点构成，它主要是由 5 个输入端口 I、集中式路由器 II 和 5 个输出端口组成 III 组成。

模型中节点内数据包发送过程（即节点内数据包发送的延迟组成）如下：

(1) 数据从节点外发送到输入端口 I。根据 3.3.1 节可知这里数据包的长度为 6 个 flit，所以所花的时间为 6 个 cycle。

(2) 输入端口 I 发送请求到 switcher II，所花时间为 1 个 cycle。

(3) Switch 将输入端口 I 的请求判断后转到所对应的输出端口 III 中，同时输出端口 III 向下一级的节点的输入端口发送请求，所花时间为 1 个 cycle。

(4) 下一级节点的输入端口向本级输出端口 III 回应请求。所花时间为 1 个 cycle。

(5) 输出端口 III 向 switcher II 发送允许信号，同时 switcher II 将允许发回原来发出请求的输出端口 I，所花时间为 1 个 cycle。

(6) 输入端口 I 将数据包发至输出端口 III，所花时间为 6 个 cycle。

(7) 到达下个节点后，循环执行上述步骤。

由此可知，数据包在无阻塞的情况下在节点中实际要经过 6+1×4+6=16 个时

钟周期。

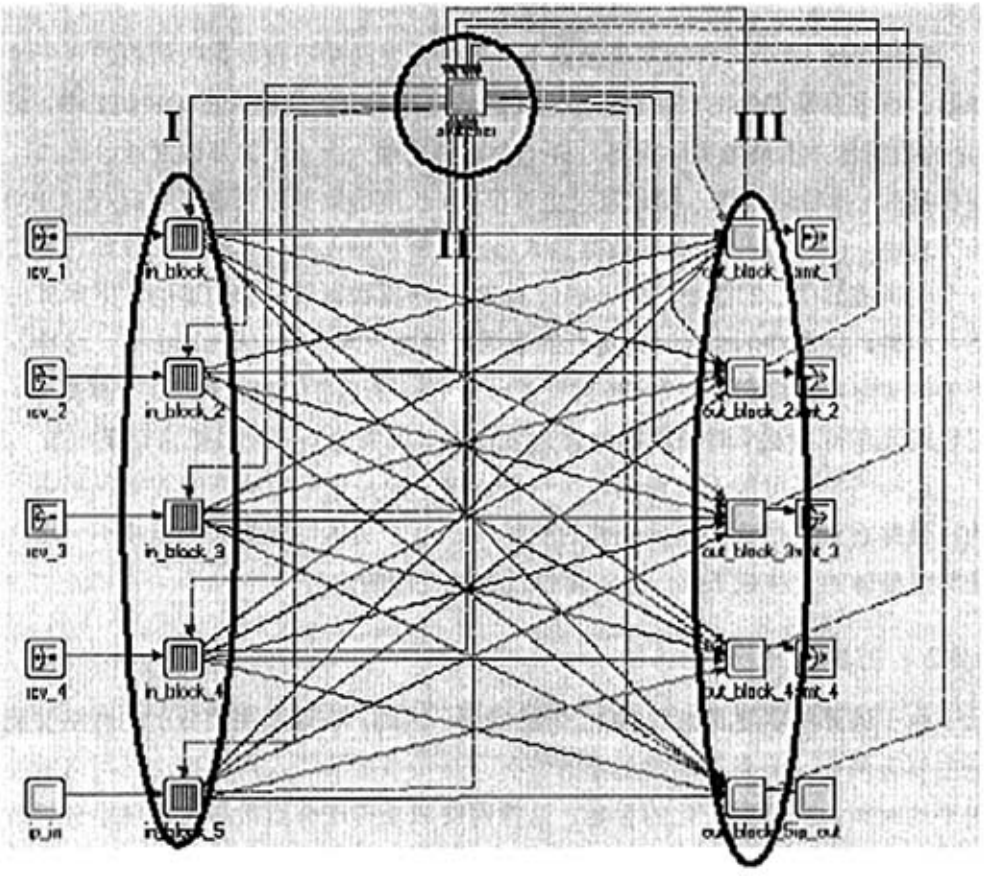


图3-8 OPNET中片上网络的节点的模型

这里必须要说明的一点是，其实OPNET并不是专门针对向我们这种要求周期精确、比较底层硬件的一种仿真工具，在它里面是没有时钟的概念的。它只有时间的概念，所有的事件都是按时间顺序加入到它的一个事件列表中，当仿真时间到了这个时刻的时候，该事件就发生。所以我在仿真的时候为了方便起见，假设了一个时钟周期为1秒。比如，要发送一个请求到switcher II，所做的就是将发送请求这个事件延迟1秒，加入到事件列表之中，然后1秒之后，请求这个事件就在switcher II中发生了。也就是说，这个OPNET模型里面的时间都是由自己来设定的，而不是实际的几个时钟周期跑出来的。

当然，收发数据包的时间除外，数据包收发的时间是系统根据数据包长度和信道的数据率（即信道的空、满程度）自动算的。这也就是说仿真所要观察的数

据包在网络中的延迟和网络的吞吐量是真实仿真结果的出现，而不是人为设定的结果。

我觉得这16个时钟周期里面最不合理的是发送端口收数据包所花的6个时钟周期。这主要是OPNET里面数据包发送的并行机制的问题。在OPNET中，6个flit的数据包并不是像实际电路一样一个时钟周期一个flit的那样发送。比如，要在仿真的t秒时刻从输入端口发一个6个flit长度的数据包到输出端口，在OPNET的仿真中，t秒时刻，所有的数据包就已经全部从输入端口读走了，放到它所谓的一个信道模型中，即图中的输入端口I，然后根据数据长度和信道的数据率算出需要8秒钟，接着在t+8秒时刻所有的数据包就被放入了输出端口III中。这样，就不能像实际电路那样来一个flit就发走一个flit。所以在OPNET的这个模型中，数据包就要多待上6个时钟周期。

这样的结果可能不太精确，会有些误差，但是这是对所有的数据包都是一样的，最终看到的仿真图是一个相对的结果，至于具体可能值大一点或小一点，只是将图整体的上移或下移一点，而并没有改变图的走势。

3.4.2 仿真结果分析

由于该算法主要是针对NOC中的应用，因此，在这里主要分析的网络性能指标主要有两个，一是平均端到端的延时，是指数据从注入网络到注出网络所经历的平均时间；一是归一化的吞吐，是指网络成功传送分组数与注入网络分组数之比。仿真采用了3种不同的流量模式，即均匀流量、热点流量和矩阵转置流量(transpose traffic pattern)模式。在均匀流量下，每一结点等概率发送分组到其他结点；热点流量模式下,一个或更多的结点被设为热点，它们将接收到比一般结点更多流量，对单个热点的情况进行仿真，其位置随机产生，它将比其他结点多获得一定百分比（如10%）的流量；矩阵转置流量模式下，结点(i,j)只发送消息给结点(j,i)。

3.4.2.1 均匀流量下，数据包在Mesh网络中的平均跳数

首先，研究一下均匀流量下，数据包在网络中的平均跳数，这是分析后面仿真图的基础。

根据节点在网络中所处的位置不同，Mesh中的节点可以分为3类，图3-9：

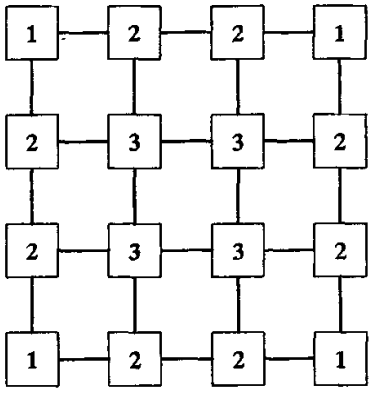


图3-9 4x4 Mesh 不同位置节点分类

第一类节点是处于顶点位置的节点，这样的节点有4个，如图3-10：

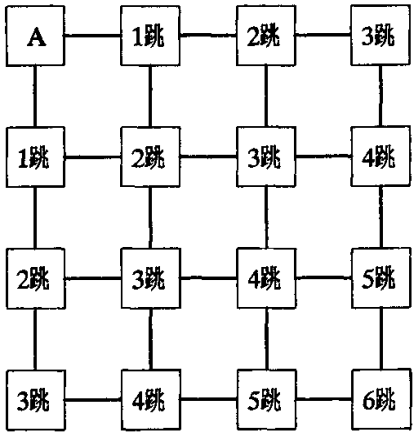


图3-10 顶点位置的节点到达其他节点跳数示意图

1跳可以到达的节点数：2；

2跳可以到达的节点数：3；

3跳可以到达的节点数：4；

4跳可以到达的节点数：3；

5跳可以到达的节点数：2；

6跳可以到达的节点数：1。

除自己外共15个节点。所以第一类节点发出的数据包到达目的地的平均跳数为：

$$\frac{2}{15}\times1+\frac{3}{15}\times2+\frac{4}{15}\times3+\frac{3}{15}\times4+\frac{2}{15}\times5+\frac{1}{15}\times6=\frac{48}{15}$$ 跳。

第二类节点是处于非顶点位置的边缘节点，这样的节点有 8 个，如图 3-11：

1 跳可以到达的节点数：3；

2 跳可以到达的节点数：4；

3 跳可以到达的节点数：4；

4 跳可以到达的节点数：3；

5 跳可以到达的节点数：1。

除自己外共 15 个节点。

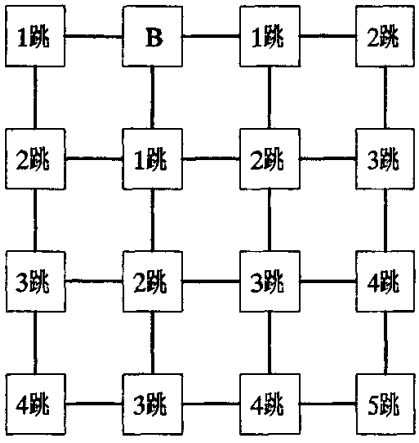


图3-11 非顶点位置的边缘节点到达其他节点跳数示意图

所以第二类节点发出的数据包到达目的地的平均跳数为：

$$\frac{3}{15}\times 1+\frac{4}{15}\times 2+\frac{4}{15}\times 3+\frac{3}{15}\times 4+\frac{1}{15}\times 5=\frac{40}{15}\text{ 跳}$$

第三类节点是网络内部节点，这样的节点有 4 个，如图 3-12：

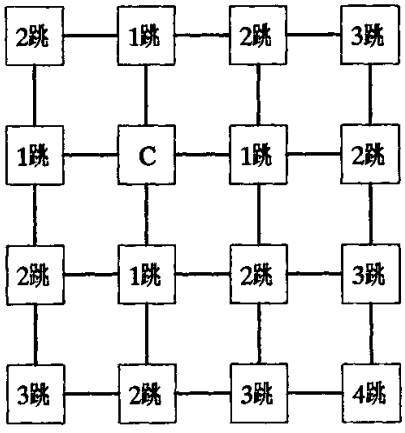


图3-12 网络内部节点到达其他节点跳数示意图

1跳可以到达的节点数：4；

2跳可以到达的节点数：6；

3跳可以到达的节点数：4；

4跳可以到达的节点数：1。

除自己外共15个节点。

所以第三类节点发出的数据包到达目的地的平均跳数为：

$$\frac{4}{15}\times 1+\frac{6}{15}\times 2+\frac{4}{15}\times 3+\frac{1}{15}\times 4=\frac{32}{15}\text{跳}$$

4×4 Mesh中有4个第一类节点，8个第二类节点，4个第三类节点。所以网络中任意的一个的数据包到达目的地的平均跳数为：

$$\frac{48}{15}\times\frac{4}{16}+\frac{40}{15}\times\frac{8}{16}+\frac{32}{15}\times\frac{4}{16}\approx 2.667\text{跳}$$

3.4.2.2 均匀流量下，虚拟通道对Mesh网络中数据包传输的影响

图3-13是均匀流量下，虚拟通道对Mesh网络中数据包传输延时的影响（采用的是具有一定自适应性的XY维序路由算法）。图3-13的横坐标是注入率（injection rate），即单位时间内向网络注入数据包的时间，例如，若注入率为0.3，则10个单位时间内，每个节点都有3个单位时间向网络注入数据包，而具体注入的时间则是随机的。纵坐标是数据包的延迟。

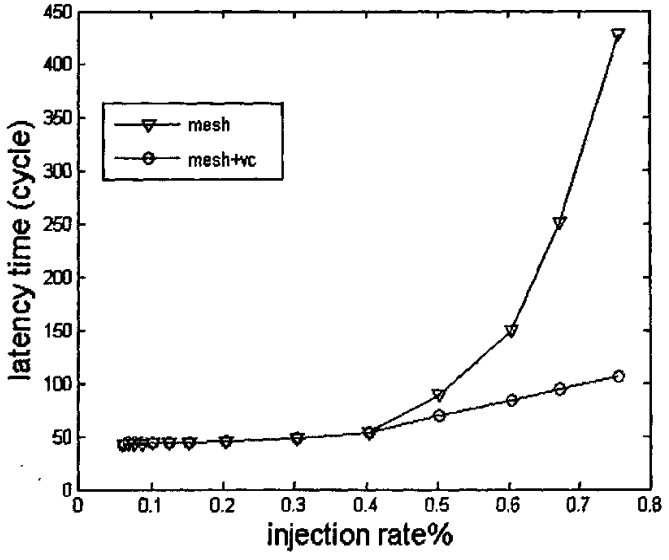


图3-13 均匀流量下，虚拟通道对 Mesh 网络数据包传输延时的影响

当注入率为 5%时，此时网络负载很低，数据包的延迟时间平均是：任意的一个的数据包到达目的地的平均跳数，再乘以一个平均每跳的所花费的时钟周期数 16，即可得平均延迟：2.667×16 = 42.667 个时钟周期，与仿真所得 43 点几的结果相吻合。

可以看到在注入率为 40%之前，即网络负载相对较低时，Mesh 网络中路由节点内是否有虚拟通道，对整个网络数据包的传输延迟影响不大，因为此时网络阻塞率较低，则对头阻塞（HOL）率也较低，所以虚拟通道对网络传输性能的影响体现不大。此时的网络数据包的平均延迟从 43 稳步上升到 68 左右。

当注入率大于 40%以后，网络负载不断增加，数据包阻塞率增高，对头阻塞（HOL）也开始出现并不断加剧，此时虚拟通道对网络传输性能的优化就明显的表现出来。

同时注意到，对于没有虚拟通道的 Mesh 网络，当数据包的注入率在 50%~60%之间时，网络数据包的平均延迟迅猛增加，这就验证了 2.2.2.2 节“队头阻塞（Head of Line）研究”中得到的“由于 HOL（Head of Line）阻塞问题，导致实际吞吐量只能达到总链路吞吐量的 58.6%”，这一结论。

3.4.2.3 无虚拟通道的Mesh网络中，各流量模式下的数据包延时仿真

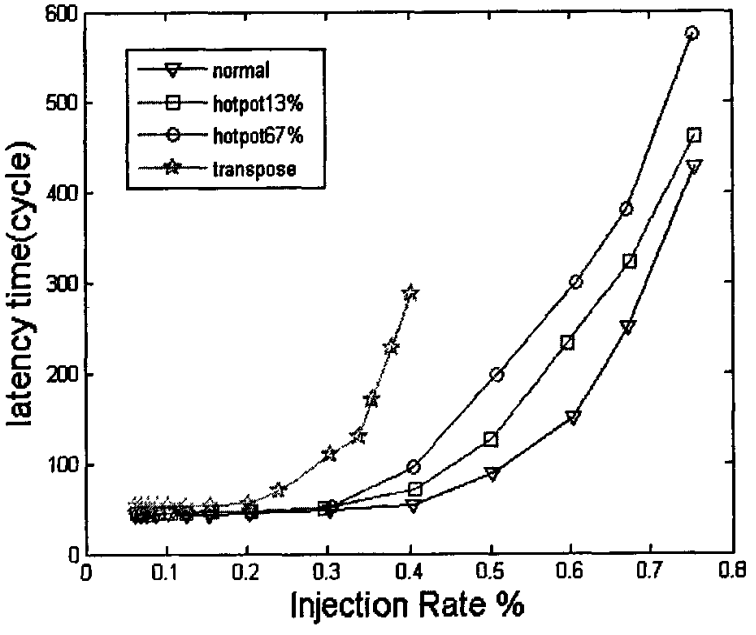


图3-14 Mesh网络中，不同流量模式下延时性能仿真

图3-14显示了在四种不同的流量模式下，无虚拟通道的Mesh网络中对数据包延时仿真的结果。其中，在热点流量模式下，设置了两种不同的热点流量，一种是热点比其他点多获得13%的数据，另外一种是热点比其它点多获得67%的数据流量。这里，仍然采用具有一定自适应性的XY维序路由算法。因此，在图中可以看到四根不同的曲线。

在NOC中，评价一个网络是否达到饱和，一般是看平均延时在注入率为多少的情况下到达最低延时的2倍。从图中可知，在均匀流量模式下，当数据注入率达到51.2%的时候，网络的延时性能趋于饱和（即平均延时数据包的平均延时为$42.667\times2\sim85$时）。如果再继续增加注入率，整个网络的性能会急剧的下降。在热点流量情况下，由于热点的存在，使得饱和点提前。在热点比非饱和点多13%和67%流量情况下，分别在网络负载达到43.3%和38.8%左右就饱和，分别比均匀模式下提前7.9%和12.4%。另外，在矩阵转置模式下，延时性能下降得很快，在数据注入率达到36.5%左右就已经达到了饱和点。

3.4.2.4 Mesh 网络中，各流量模式下的吞吐性能仿真

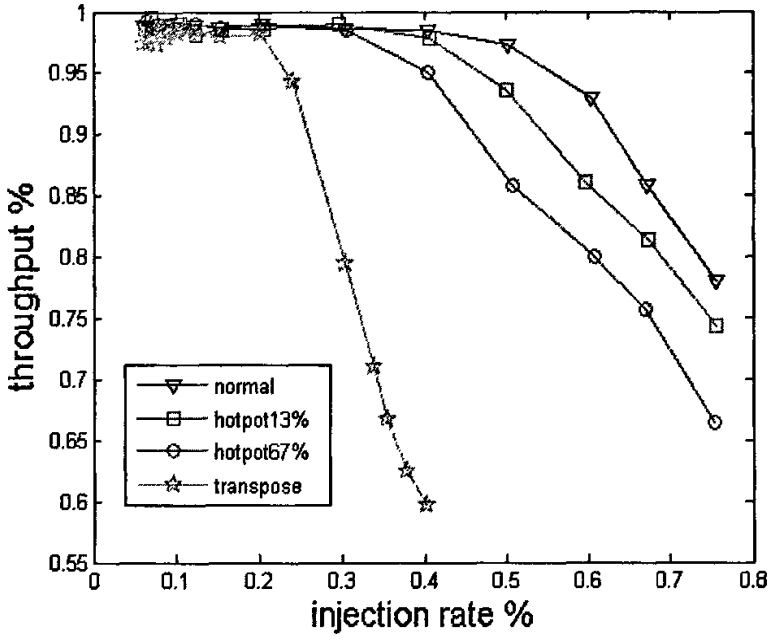


图3-15 Mesh 网络中，不同流量模式下吞吐性能仿真

图 3-15 显示了在四种不同的数据流量模式下对 Mesh 网络吞吐性能的仿真结果。在均匀模式、热点 13%和热点 67%模式下，吞吐性能在负载达到 40%以后都有显著下降，而在矩阵转置模式下，吞吐性能在负载达到 20%以后就很快下降。

3.4.2.5 均匀流量下，数据包在 Torus 网络中的平均跳数

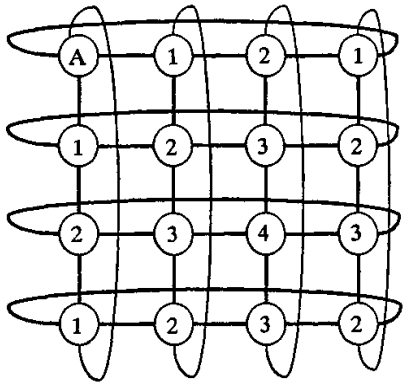


图3-16 Turos 网络中节点到达其他节点跳数示意图

不同于 Mesh 结构，Torus 结构中每个节点都与其他 4 个节点相连，因此看似处于不同位置的节点，但其发出的数据包到达目的地的平均跳数都相同，如图 3-16 所示：

1 跳可以到达的节点数：4；

2 跳可以到达的节点数：6；

3 跳可以到达的节点数：4；

4 跳可以到达的节点数：1。

除自己外共 15 个节点。所以 Turos 结构中节点发出的数据包到达目的地的平均跳数为：

$$\frac{4}{15}\times 1+\frac{6}{15}\times 2+\frac{4}{15}\times 3+\frac{1}{15}\times 4\approx 2.133\text{跳}<\text{Mesh 网络的 2.667 跳。}$$

3.4.2.6 矩阵转置模式下，数据包在 Torus 网络中的平均跳数

矩阵转置流量模式下，结点(i,j)只发送消息给结点(j,i)，i≠j，如图 3-17。

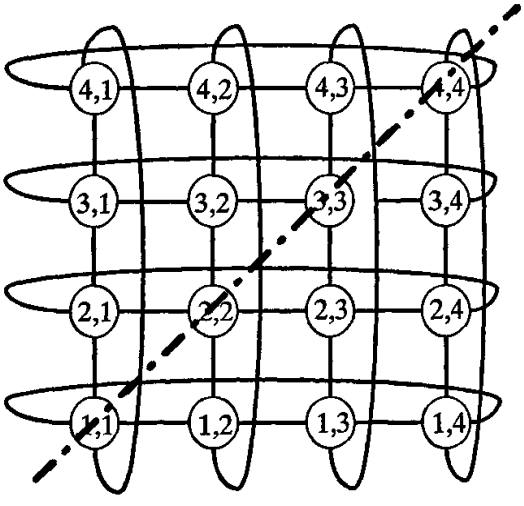


图3-17 矩阵专置模式下的 Torus 结构

2 跳可以到达对方的节点数：8；

4 跳可以到达的节点数：4。

除去对角线上的 4 个点，共 12 个点。所以 Turos 网络中，矩阵转置流量模式下，节点发出的数据包到达目的地的平均跳数为：

$$\frac{8}{12}\times 2+\frac{4}{12}\times 4\approx 2.667\text{跳}$$

3.4.2.7 Torus 网络中，各流量模式下的数据包延时仿真

众所周知，虚通道的加入对平均端到端的延时性能会有较大的提高，但是在 NOC 的应用中，加入虚通道会增加 NOC 中路由节点的复杂程度，因此，在的仿真中，为了测试网络的极限性能，并没有设计使用虚通道。

图 3-18 显示了在四种不同的流量模式下，对本文所设计的 Torus 结构和路由算法进行延时仿真的结果。

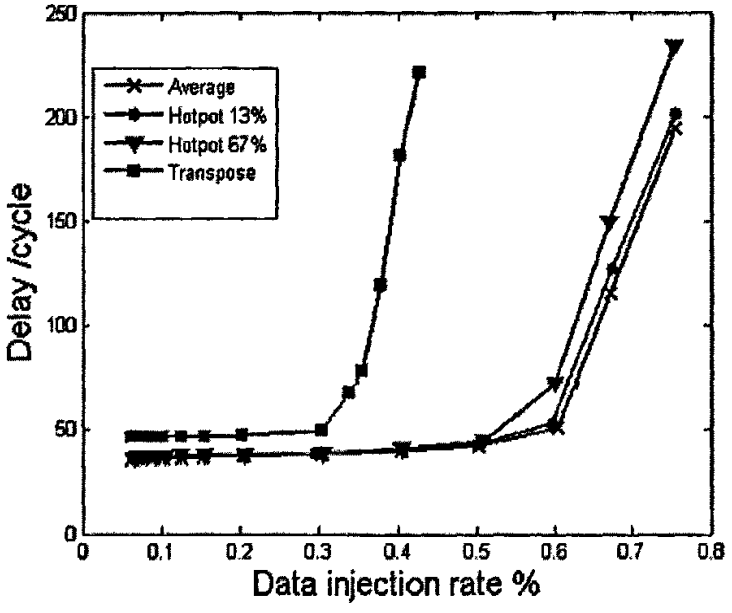


图3-18 Torus 网络中，不同流量模式下延时性能仿真

首先，观察初始时注入率为 5%的情况下，此时网络负载很低，数据包的延迟时间平均是：任意的一个的数据包到达目的地的平均跳数，再乘以一个平均每跳的所花费的时钟周期数 16。均匀流量模式下，Torus 结构中节点发出的数据包到达目的地的平均延迟为 $2.133\times16=34.133$ 个时钟周期，与仿真所得 36 的结果相吻合。热点模式下，由于只是个别节点对的流量增加，根据特殊节点对的跳数，对初始时的延迟略有影响，但影响不大。由于而矩阵转置流量模式下节点发出的数据包到达目的地的平均延迟为 $2.667\times16=42.667$ 个时钟周期，与仿真所得的结果 43 相吻合。

从图中还能知道，在均匀流量模式下，当数据注入率达到 62.7%的时候，网络的延时性能趋于饱和，如果再继续增加注入率，整个网络的性能会急剧的下降。

在热点流量情况下，由于热点的存在，使得饱和点提前。在热点比非饱和点多 13%和 67%流量情况下，分别在网络负载达到 61.4%和 59%左右就饱和，分别比均匀模式下提前 1.3%和 3.7%。从图中可以看出，本文所设计的 Torus 结构和路由算法，在热点存在的情况下延时性能并没有显著下降，在 13%模式下与均匀模式下几乎重合，因此，可以知道我们所设计的结构和路由算法的延时性能对热点并不敏感。另外，在矩阵转置模式下，延时性能下降得很快，在数据注入率达到 36%左右就已经达到了饱和点。

3.4.2.8 Torus 网络中，各流量模式下的吞吐性能仿真

图 3-19 显示了在四种不同的数据流量模式下对 Torus 网络吞吐性能的仿真结果。在均匀模式、热点 13%和热点 67%模式下，吞吐性能在负载达到 60%以后都有显著下降，而在矩阵转置模式下，吞吐性能在负载达到 36%以后就很快下降。因此，该结果也证明了，本文所述结构在在保证吞吐性能的时候对热点是不敏感的，但是矩阵转置形式则会带来巨大影响。

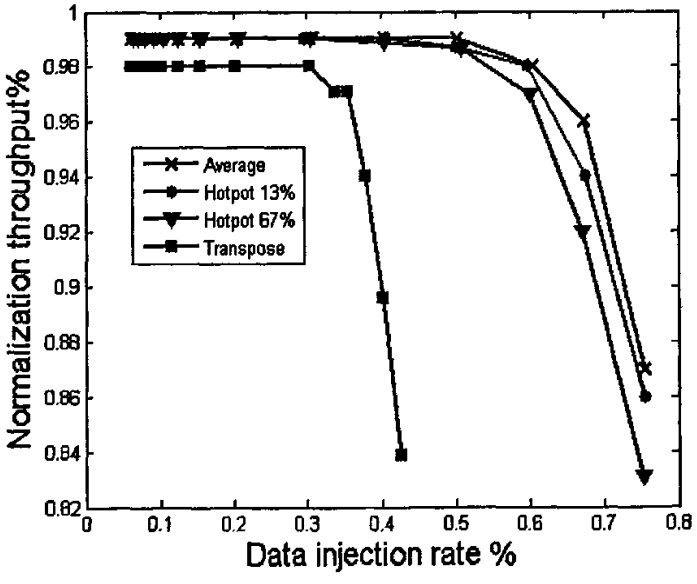


图3-19 Torus 网络中，不同流量模式下吞吐性能仿真

3.4.2.9 不同流量模式下 Mesh 网络和 Torus 网络的延时比较

图 3-20 是不同流量模式下，采用具有一定自适应性 XY 维序路由的 Mesh 网络和采用上文设计的路由算法的 Torus 网络的延时比较。图 3-20(1)是均匀流量模

式下两种网络拓扑中数据包延时比较，图 3-20(2)和(3)分别是热点 13%和热点 67%模式下两种网络中数据包延时比较，图 3-20(4)是矩阵转置模式下两种网络中数据包延时比较。可以看出这四种常见模式下，采用我们自己设计的路由算法的 Torus 网络的性能都明显优于采用具有一定自适应性 XY 维序路由的 Mesh 网络。

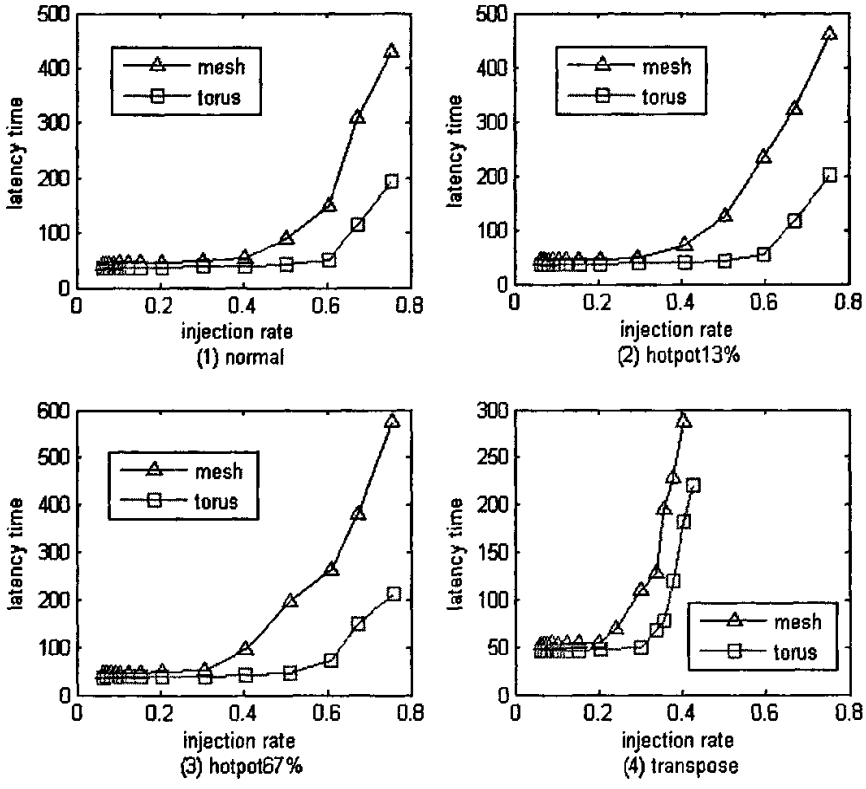


图3-20 不同流量模式下，Mesh 网络和 Torus 网络在延时性能仿真比较

3.4.2.10 不同流量模式下 Mesh 网络和 Torus 网络的吞吐性能比较

图 3-21 是不同流量模式下，采用具有一定自适应性 XY 维序路由的 Mesh 网络和采用上文设计的路由算法的 Torus 网络的吞吐性能比较。图 3-21(1)是均匀流量模式下两种网络吞吐性能的比较，图 3-21(2)和(3)分别是热点 13%和热点 67%模式下两种网络吞吐性能的比较，图 3-21(4)是矩阵转置模式下两种网络吞吐性能的比较。可以看出这四种常见模式下，采用我们自己设计的路由算法的 Torus 网络的吞吐性能都明显优于采用具有一定自适应性 XY 维序路由的 Mesh 网络。

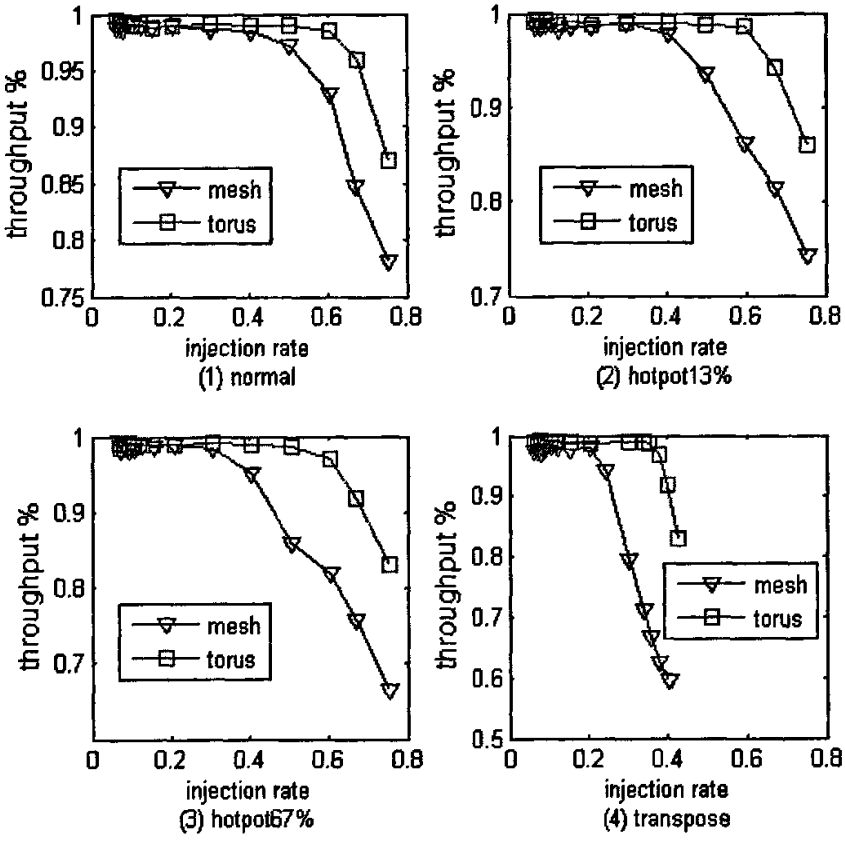


图3-21 不同流量模式下，Mesh 网络和 Torus 网络吞吐性能仿真比较

3.4.2.11 总结

通过上面的分析，能了解到，采用我们自行设计的路由算法的 Torus 网络具有良好的性能。但是，如果要将本结构用于 NOC 中，需要根据 NOC 的具体应用来选择本结构的使用或者进行改进[14][15]。首先，本结构能很好的缓解 NOC 中存在热点的情况，对于 NOC 中有较多的热点 IP 的情况，本结构是一种较好的选择，它能较大的降低热点数据流对整个网络性能的影响，包括平均延时和数据吞吐。但如果在 NOC 中的 IP 或者处理器核之间存在大量的成对互连通信，而对与对之间通信量较少时，使用本结构就会产生比较大的延时，如果要使用本结构，则应该避免将具有成对通信特点的 IP 放置在转置的位置上，如将 IP 映射到位于对角线上的点，或者有意打破转置映射，这都能降低由于结构和算法带来的对转置流量模式下延时和吞吐性能的影响。

第四章　片上网络硬件测试系统设计

片上网络是未来满足大数据量实时并行处理的主要方法，软件仿真固然具有一定的指导意义，但在硬件上实现、验证并比较不同路由算法以及拓扑结构对网络性能的影响也是必不可少、相当重要的。

4.1　片上网络硬件测试平台设计思想

由于硬件平台主要目的是为了验证路由节点和路由算法的设计，是为了满足大数据量实时并行处理，这决定了该硬件平台是以高性能的现场可编程序逻辑门阵列（Field Programmable Gate Array(FPGA)）为设计核心。FPGA 是近年在专用 ASIC 的基础上发展出现的，它拥有 ASIC 的稳定性、大容量、高度集成的优点，能够用程序语言对其进行功能编写，极大的提高了硬件设计的时间；并且因为 FPGA 中拥有大量时序资源、布线资源、存储资源，所以用户可以在其中实现复杂的逻辑设计和算法实现。同时，由于目前的 FPGA 已经可以达到千万等效门的规模，它也被用于 IC 设计的前期验证。

同时我们还需要一块芯片，能够模拟处理器节点（IP），可以和网络中各节点进行数据交换，并帮助控制和测试网络，所以 DSP 是一种很好的选择。而考虑到未来片上网络可能在音视频处理中的应用，因此我们选择了一款高性能的音视频 DSP 处理器为协处理器，并加入了 DSP 的外围电路，包括视频 AD，DA 和音频 Codec 芯片等，使整个平台同时拥有了强大的音视频处理能力。

4.2　测试平台的硬件构架

基于 4.1 节中关于处理系统硬件方面的设计思想，我们采用了 FPGA+DSP 的处理结构，如图 4-1 所示（另一片 FPGA 是用于“软硬件协同仿真”课题的需要，这里不详述）。该结构主要由 Altera 公司 StratixII 系列 FPGA 与 TI 公司 TMS320CDM642 构成。为了满足片上网络日后在音视频算法的应用，又选用了 TI 公司的 TLV320AIC23B 作为音频处理芯片，以及 Philips 公司的 SAA7113H 和 SAA7104 分别作为视频 A/D 和 D/A 芯片。

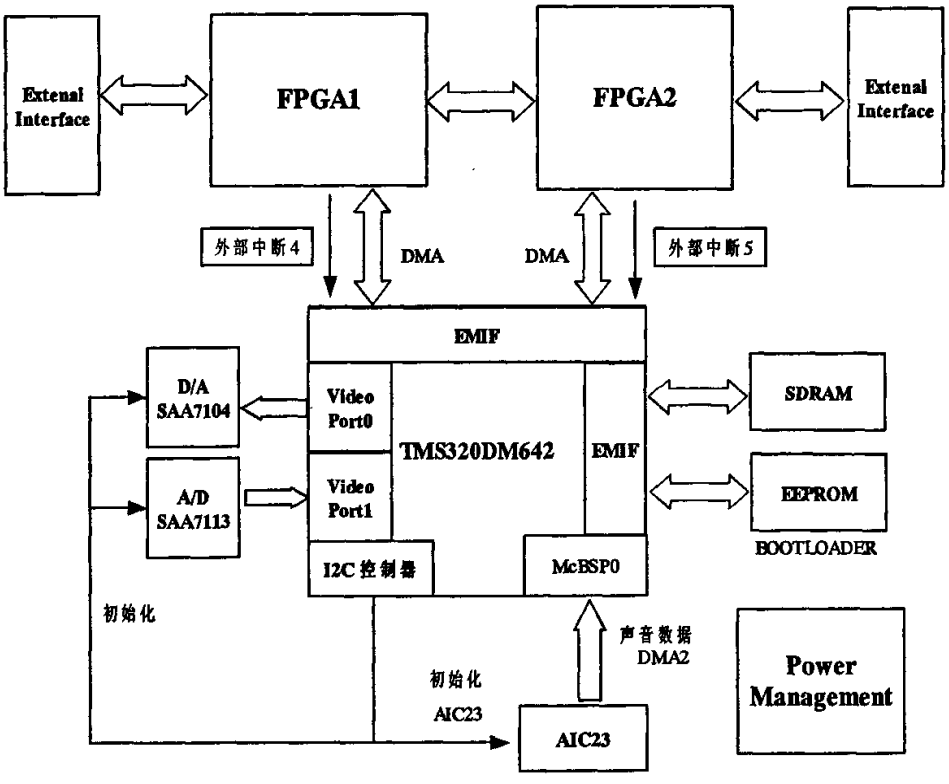


图4-1 硬件总体框图

图 4-2 是实际电路的正面图。

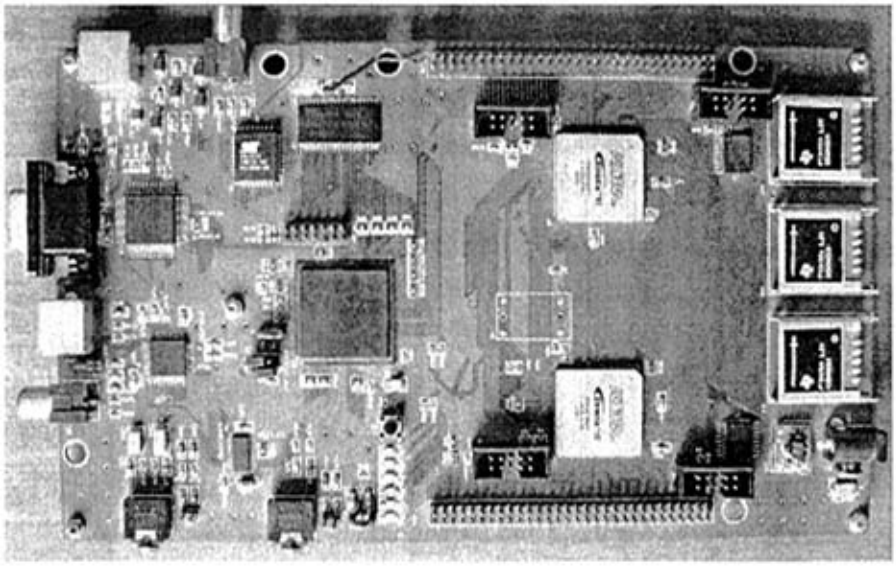


图4-2 实际电路板的正面图

4.3 系统各部分介绍

根据 4.2 节可以知道硬件平台主要分为五个部分，分别是：算法验证与实现部分、协处理器部分、音频模拟部分、视频模拟部分、电源管理和监视部分。

下面将在这章中分节对这五个部分进行详细的描述。

4.3.1 算法验证与实现部分

FPGA 是片上网络硬件实现的核心，这里选用了 Altera 公司 StratixII 系列 FPGA，其选型主要考虑芯片资源和芯片内部结构，同时要考虑其供电电压和配置方式。

(1) 估算芯片资源。

下图为 Mesh 网络的一个交换节点，可以看到一个完整的网络交换节点包括与东、南、西、北、处理器五个方向通信的输入输出控制，这五个方向的输入输出端口原理、结构都相同，所以只需要选其中一个进行仿真，估算可能占用的资源。

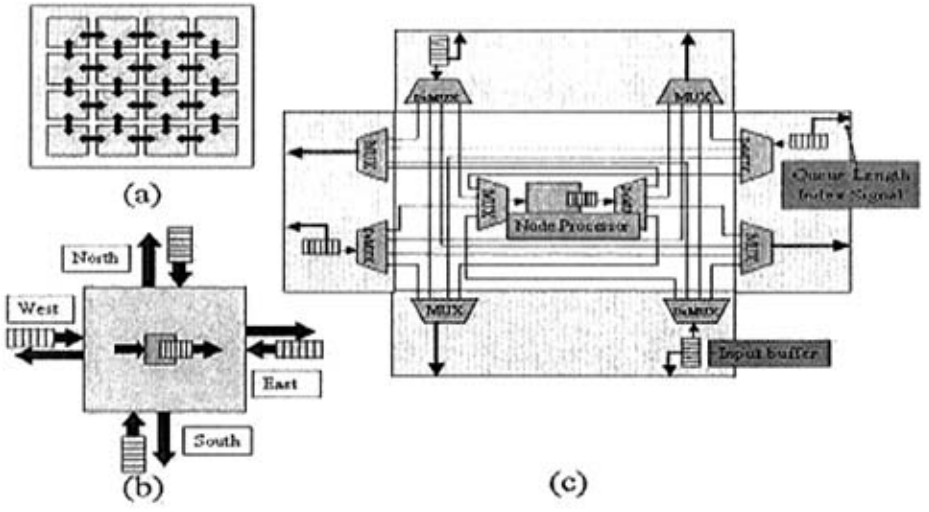


图4-3 路由节点框图

在 Quartus5.0 中对上图所示路由结点进行仿真，以东节点为例，节点设计参见第五章，编译、仿真可得到资源占用情况：

Flow Status: Successful - Fri Sep 23 11:58:20 2005

Quartus II Version: 5.0 Build 148 04/26/2005 SJ Full Version

Revision Name: crossbar

Top-level Entity Name: crossbar

Family: Stratix II

Device: EP2S30F484C5

Timing Models: Final

Met timing requirements: Yes

Total ALUTs: 275 / 27,104 (2 %)

Total registers: 119

Total pins: 95 / 343 (71 %)

Total virtual pins: 0

Total memory bits: 640 / 1,369,728 (1 %)

DSP block 9-bit elements: 0 / 128 (0 %)

Total PLLs: 0 / 6 (0 %)

Total DLLs: 0 / 2 (0 %)

LC: 588 (0)

一个 4*4 的 Mesh 网络，有 16 个这样的节点，初步估算，资源完全够用。

(2) 芯片内部结构研究。

Altera 公司的 StratixII 系列 FPGA，它有许多系统级的功能模块（如图 4-4），如用于时钟产生和管理的锁相环（PLL）、用于片内存储数据的 RAM 块、用于数字信号处理的 DSP 块等。除此之外，它采用了全新的逻辑结构——自适应逻辑模块（ALM），不仅显著地提高了性能和逻辑利用率，同时也降低了成本[16]。

传统的 FPGA 的逻辑单元（LE）的经典结构是由一个 4 输入查找表（LUT）和一个触发器组成的。而 ALM 中的组合逻辑块可以根据用户的需求由设计工具自动配置成需要的模式，可以配成 5 输入和 3 输入的 LUT，或 5 输入和 4 输入的 LUT 等等。ALM 结构，可以说是 FPGA 构架方面的革命。它能在为用户提高性能的同时，保持较低成本。另外 ALM 内部也有两个 3 输入加法器，和传统的 2 输入加法器相比，在实现算术运算时，显著减少了加法电路的级数，提高了计算性能[17]。

StratixII 的逻辑阵列块（LAB）中包含了 8 个 ALM。LAB 内部有局部布线资源，用于外部逻辑访问该 LAB 或 LAB 内 ALM 之间的连接。LAB 内部的 ALM 之间，也有触发器链和共享的算术链。这些丰富的互连资源，使得 StratixII FPGA 的性能和可布线性都大大提高。

片内存储器资源是指 FPGA 芯片自带的片上存储器资源，不同芯片的片上存储资源的种类和数目都有所不同。StratixII 器件各具不同应用的需求，设计了 3 种

内嵌的 RAM 块：M512RAM（512bitRAM）、M4K（4kbitRAM）、M-RAM（512kbitRAM）。M512 在 StratixII 器件中数量也比较多，通常用于分散的数据存储、浅 FIFO、移位寄存器、时钟域隔离等功能。M4k 的数量也比较多，通常用做芯片内部数据流的缓存、FIFO 接口以及 CPU 的程序存储器等。而 M-RAM 主要用于大数据包的缓存等。

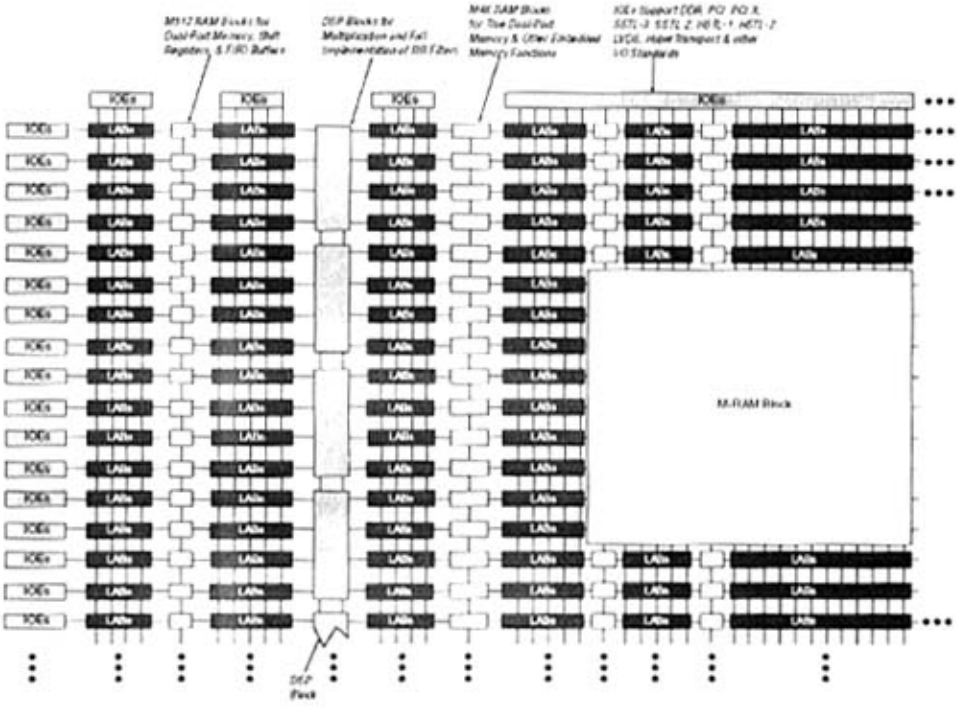


图4-4 StratixII 系列 FPGA 内部结构示意图

时钟资源主要指片内集成的 DLL 或 PLL，用以完成时钟的高精度、低抖动的倍频、分频、移相等操作。

输入输出单元（IOE）是芯片外部引脚和内部数据进行交换的接口电路，分布在芯片的周围。IOE 同样具有可编程序能力，可以把 IO 引脚配置成输入、输出、三态或者双向等不同的功能。

(3) 芯片供电和配置方式。

高性能 FPGA 一般都是内核和 I/O 分离供电，其中内核电压一般是固定的，而 I/O 电压又根据外接信号电平的不同而不同。所以在设计硬件时，要特别注意各种电压的值。

在该系统里，FPGA的内核电压是1.2V，由于设计只使用了LVCMOS电平的信号，因此I/O电压为单一的3.3V.

FPGA的配置方式大体分为串行，并行和在线配置三种方式，其中串行和并行配置是上电后FPGA从FLASH中读入程序并执行的过程，而在线配置是指通过JTAG端口对FPGA进行配置的过程，由于大多数时候都需要上电FPGA自动运行程序，因此需要选择外部FLASH芯片和对应的配置方式。

串行配置对比并行配置来说成本较低，布线简单，缺点是配置速度较慢。由于这里对配置速度没有要求，因此选用成本较低的串行配置方案，选用的芯片大小由FPGA的大小决定。

表4-1 StratixII系列FPGA串行配置需求

Stratix II Device	Raw Binary File Size (Bits)	Serial Configuration Device		
		EPCS4	EPCS16	EPCS64
EP2S15	4,721,544	v	v	v
EP2S30	9,640,672		v	v
EP2S60	16,951,824		v	v
EP2S90	25,699,104		v	v
EP2S130	37,325,760			v
EP2S180	49,814,760			v

从上表可以看出，EPCS16是我们的首选。选定配置ROM的大小后，按照图4-5的方式连接ROM和FPGA后，就能成功进行上电配置了。

4.3.2 协处理器

这里选用DSP作为本系统中的协处理器。在世界上众多的DSP厂商中，德州仪器（TI）的DSP始终占据着较大的市场份额（40%--50%）。目前得到广泛应用的TI的三个大的DSP处理器系列分别是TMS320C2000、TMS320C5000和TMS320C6000系列。本项目中使用的DSP是TI公司于2003年1月推出的数字媒体处理器TMS320DM642，它具备丰富的音视频接口以及强大的处理能力，能满足未来片上网络在音视频算法中的应用[18]。

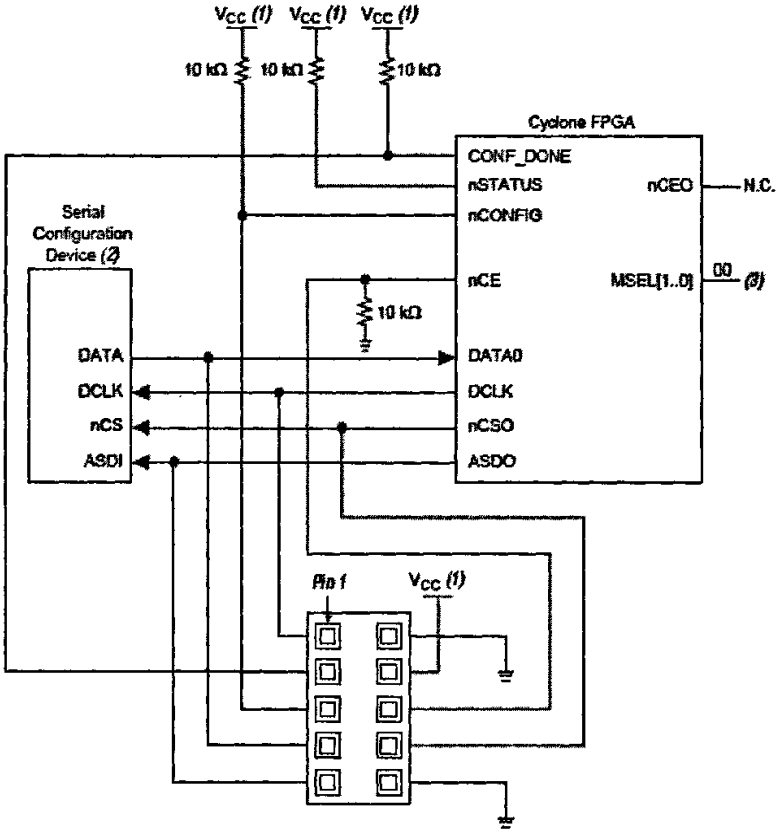


图4-5 FPGA与配置芯片的连接方式

4.3.3 FPGA和DSP之间的通信

目前阶段，DSP的主要作用是为FPGA的测试提供原始数据、读入路由节点的输出数据并进行验证，这样就涉及到一个DSP和FPGA进行通信的问题。

TI公司的TMS320DM642这款DSP，其EMIF（存储器扩展接口）结构[19]和EDMA（增强型直接内存访问）技术[20]为高速数据传输提供了一个可能和可靠的通道。它拥有64位的同步EMIF接口和6个独立的DMA通道，根据TI公司提供的C6000的数据手册，在片选信号CE有效时，在每个EMIF接口的时钟ECLKOUT的上升沿都可以读/写一次，这里使用本硬件平台对其进行了测试：

测试硬件环境：TMS320DM642+EP2S30F484C5

测试软件环境：CCS3.1+QuartusII 6.0 SP2

DSP设置：DSP运行在600MHZ下，ECLKOUT设置为CPU时钟的1/6，即100MHz，CE1使用32位同步通信方式与FPGA进行数据交换，读写均使用DMA，

DMA事件与外部中断4同步。

FPGA设置：在FPGA内部建一个RAM，DSP通过DMA通道写完数据后，使用ALTERA公司的In System Memory Editor工具来观察DSP写入的数据是否正确。DSP读时，先初始化FPGA里的RAM里的数值，然后DSP使用DMA通道读入RAM里的数据，通过DSP开发环境CCS看数据是否正确。FPGA内部RAM的结构如图4-6：

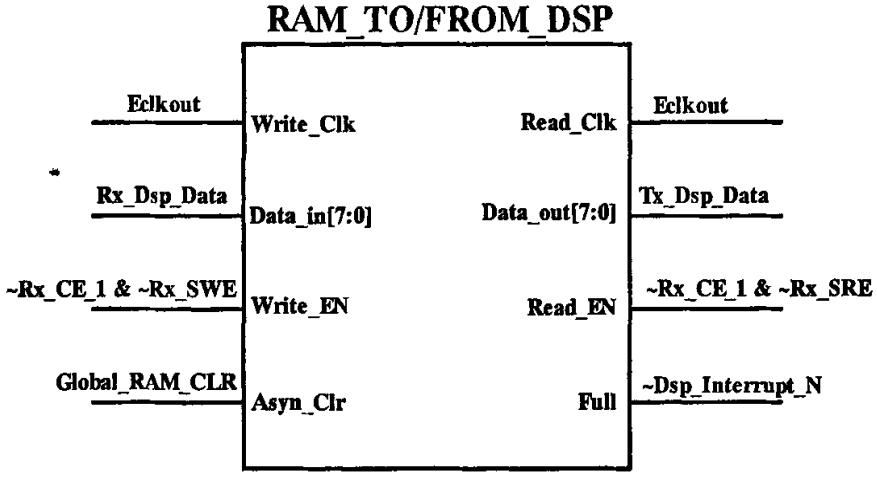


图4-6 FPGA内部RAM的结构

在观察读写信号的时候，我们采用了Altera公司FPGA内建的软件逻辑分析仪SignalTapII Logic Analyzer来对读写数据时的波形进行分析，以CE1的下降沿作为触发数据显示的时刻，得到DSP以DMA方式写入FPGA内部时的情况：

图4-7 DSP写入FPGA内部时的时序波形

从图4-7可以看到，当CE有效时，DSP确实能在每个ECLK的上升沿对FPGA进行写操作，这样，当DSP的ECLKOUT运行在133MHz时，64位EMIF的速度可以达到1Gbytes/s以上。

4.3.4 音频模拟部分

这里选用TI公司的TLV320AIC23B[21]作为音频处理芯片，它是集成了模拟功

能的高性能立体声音频编解码器 Codec，成本低，性能好。芯片中的 A /D 转换器和 D /A 转换器采用了多位 Sigma-Delta 技术，数据传输字长为 16、20、24、32 位，采样率为 8 kHz～96 kHz。A/D 转换器提供高达 90 dBA 的信噪比。D/A 转换器提供 100dB 信噪比，在播放中的功耗小于 23 mW。

采用 DM642 的 I2C 控制器来初始化 AIC23,使用 DM642 的 McBSP 口与 AIC23 进行音频数据交换。

4.3.5 视频模拟部分

视频 A/D 芯片这里选用 Philips 公司的 SAA7113H[22]，它是一款功能强大且操作简单的 9 位视频输入处理芯片，通过 IIC 总线和 VPO 接口与 DSP 相连构成应用系统。它内部包含两路模拟处理通道，能实现视频信号源选择、多制式解码、亮度 / 对比度 / 饱和度控制和多标准 VBI 数据解码等功能。

视频 D/A 芯片这里选用 Philips 公司的 SAA7104[23]，它功能强大，可接收 ITU-656 等多个视频标准，既可以直接输出 PAL、NTSC、SECAMS 视频格式,也可以以 Y/ C(S 端子) 及高清晰度电视输出，VGA 输出。

4.3.6 电源管理

本系统中电源较为复杂，需要：

(1) DSP 核心电压 1.4V；

(2) FPGA1 的核心电压 1.2V；

(3) FPGA2 的核心电压 1.2V；

(4) 系统的所有 I/O 电压以及其他数字芯片核电压 3.3V。

由于 FPGA 和 DSP 的功耗都比较大，因此，这里选用了三块 TI 公司的集成电源模块 PT5406[24]分别给三块处理器的核心供电。PT5406 是 TI 公司出品的单输出可调电源芯片，输入电压范围从 4.5V 到 5.5V，输出范围从 1.1V 到 1.65V 可调。输出可支持到 6A 电流。第一块 PT5406 输出 1.4V 给 DSP 的核心供电，第 2 块和第 3 块 PT5406 输出 1.2V 分别给两块 FPGA 的核心供电。对于系统的所有 I/O 部分，由于本系统中只使用到了 LVCOMS 电平的信号，因此对 I/O 部分的供电只需要 3.3V。这里使用 TI 公司出品的集成电源模块 PTH04070W[25]来提供高达 3A 电流的 3.3V 电压。

第五章 基于 FPGA 的路由节点设计和实现

作为一种芯片上的应用技术，NOC 最终是需要在硬件上具体实现的。路由节点是 NOC 的重要组成部分，它的设计是搭建整个网络的前提和基础，因此基于 FPGA 的路由节点的设计和实现非常重要。本章介绍的路由节点采用的流控机制为虫洞路由，使用的路由算法为带有一定自适应性的维序路由，整个 FPGA 设计在 Altera 集成开发环境 Quartus6.0 上完成。

本章首先简要介绍 Quartus6.0 的使用情况以及 FPGA 设计的一些常用设计思想，然后对路由节点的具体实现做一个详细的说明。

5.1 Quartus6.0 集成开发环境介绍

Quartus6.0 是 Altera 公司推出的针对 Altera 公司 FPGA 使用的全套 EDA 开发工具，它集成了 Altera 的 FPGA/CPLD 开发流程中所涉及的所有工具和第三方软件的接口。通过使用此综合开发工具，设计者可以创建、组织和管理自己的设计。

下面简要介绍用 Quartus6.0 进行 FPGA 开发的一般步骤。

(1) 设计输入：在 Quartus6.0 中进行 FPGA 设计时，可使用模块输入方式、文本输入方式、Core 输入方式和 EDA 设计输入工具来表达用户的电路构思，同时使用分配编辑器设定初始设计约束条件。

(2) 功能仿真：对模块或设计整体进行功能性验证，主要是检测设计能否实现设计目标，以及检查模块中是否有逻辑错误。

(3) 综合：对设计输入进行硬件的网表映射，将设计输入转换为与硬件相对应的连线网表文件。这个网表文件直接表现了设计的电路实现，网表的优化直接影响整个设计的性能。

(4) 布局和布线：布局布线过程将网表文件中指明的电路连接关系转化为 FPGA 芯片内电路的硬件连接关系。布局是指将各个功能模块及各种连线摆放到 FPGA 门阵列中的逻辑资源上。布线过程的任务是接照网表中的连接关系，利用 FPGA 上的布线资源连通各逻辑单元，实现整个设计。布局和布线的好坏关系到 FPGA 能否正常工作以及工作的性能好坏，一般可以采用对布局布线进行约束的方法，使其结果趋近于设计要求。

(5) 烧写文件生成以及 FPGA 编程：编程和配置是在全编译成功之后，将布局布线后的器件、逻辑单元和管脚分配转换为该器件的配置文件（目标器件的一个或多个 Programmer 对象文件（.pdf）或 SRAM 对象文件（.sof））写入芯片中便于测试。

5.2 FPGA 设计基本思想介绍

由于 FPGA 设计工程师在进行设计时直接面对电路级功能模块的实现，同时，FPGA 又拥有许多单处理器 ASIC 所没有的优势，所以在进行 FPGA 设计时，即有许多需要注意的地方（如毛刺、时序紊乱、数据位溢出）又有许多 FPGA 独有的设计技巧，本节简要介绍在进行 FPGA 设计时常见（也是最重要）的三种设计思想[17]。

5.2.1 同步时序设计思想

数字时序电路一般分为异步时序和同步时序。异步时序电路采用组合逻辑，它的输出与时钟信号没有任何关系。同步时序电路一般采用各种触发器（D、JK、RS 或者 T），它的输出逻辑被同一个时钟的上升沿（或者下降沿）控制。

异步逻辑的输出会由于不同信号到达最后一个组合逻辑单元传输延迟的不同而产生毛刺现象。时钟端口、清零和置位端口对毛刺信号十分敏感，任何一点毛刺都可能会使系统出错。

同步电路最大的好处就是能够清除电路中可能毛刺现象，由于同步电路信号的变化都发生在时钟沿，只要毛刺不出现在时钟的沿口并且不满足数据的建立和保持时间，就不会对系统造成危害（由于毛刺很短，多为几纳秒，基本上都不可能满足数据的建立和保持时间）。根据这个特性，我们应当在系统中尽可能采用同步电路。

5.2.2 空间换时间的设计思想

FPGA 中有大量的逻辑资源可供用户按自身要求进行电路定制，各个功能模块之间的操作可以互相关连，也可以完全独立进行；而一般的处理器都按指令译码单个执行，一个时钟周期处理的指令数最多为一个（对于单处理器核而言）。利用 FPGA 这一优势，对于需要大量运算处理模块，可以在 FPGA 中分解为多个处理单元的并行处理，用以大幅度提高处理速度（如图 5-1 所示）。这是 FPGA 中常用的

空间换时间思想，这一思想不仅仅用于处理核的大量复用，对于如多接口并行处理模块、串并转换模块等都是这一思想的应用。

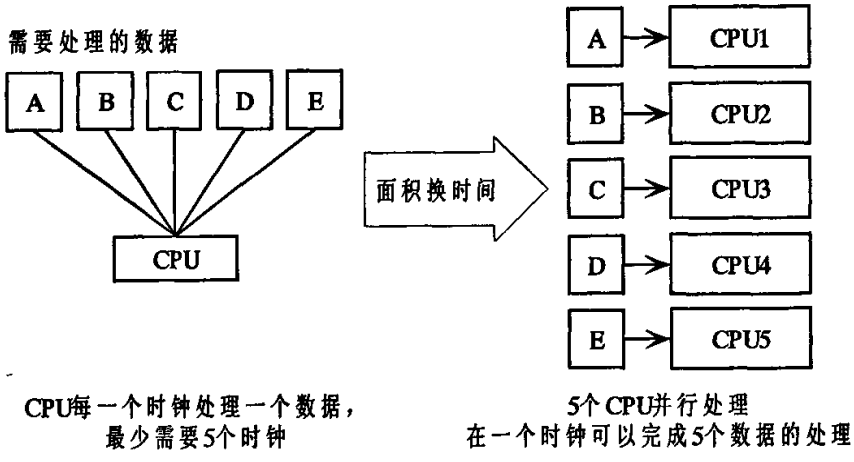


图5-1 空间换时间处理思想示意图

5.2.3 流水处理思想

流水线处理也是高速设计中的一个常用设计手段。如果某个设计的处理流程分为若干步骤，而且整个数据处理是 “单流向”的，即没有反馈或者迭代运算，前一个步骤的输出是下一个步骤的输入，则可以考虑采用流水线设计方法来提高系统的工作频率。

流水线设计的结构示意图如图 5-2 所示。其基本结构为：将适当划分的 n 个操作步骤单流向串联起来。流水线操作的最大特点和要求是，数据流在各个步骤的处理从时间上看是连续的，如果将每个操作步骤简化假设为通过一个 D 触发器（就是用寄存器打一个节拍），那么流水线操作就类似一个移位寄存器组，数据流依次流经 D 触发器，完成每个步骤的操作。

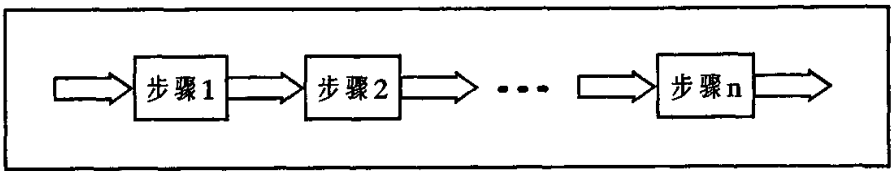


图5-2 流水处理思想示意图

流水线设计的一个关键在于整个设计时序的合理安排，要求每个操作步骤的划分合理。如果前级操作时间恰好等于后级的操作时间，设计最为简单，前级的

输出直接汇入后级的输入即可；如果前级操作时间大于后级的操作时间，则需要对前级的输出数据适当缓存才能汇入到后级输入端；如果前级操作时间恰好小于后级的操作时间，则必须通过复制逻辑，将数据流分流，或者在前级对数据采用存储、后处理方式，否则会造成后级数据溢出。

5.3 数据包格式

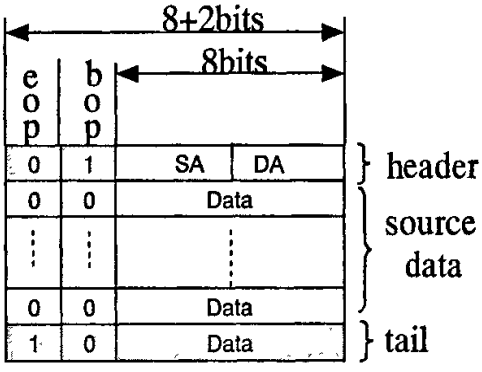


图5-3 数据包格式

数据包格式如图 5-3 所示，一共 6 个 flits，其中头尾各占一个 flit，原始数据占四个 flits。每个 flit 的长度是 10bits。其中，bop（begin of packet）和 eop（end of packet）各自占一个 bit，用来标志数据包的传输开始和结束。在 header 中，包括了源地址 SA（Source Address SA）和目的地址 DA（Destination Address DA）。tail 中的 data 部分用来存储一些需要最后统计的信息，这样，在数据包传输的过程中，就可以实时记录我们所关心的一些敏感量，保证最后统计结果的有效性。

5.4 路由节点的微结构介绍

一个完整的路由节点应该包括输入端口，输出端口，路由表，缓存空间，仲裁器等部分。这里的各个部分仅仅是逻辑上的一个划分，各个部分在物理上并非独立，比如路由表可能存在于输入端口中，那么这个时候路由表就与输入端口在结构构成上就是一个整体。又如，缓存可以采取的形式比较多样，其具体的结构位置也可以根据需求而有所不同。又比如，仲裁器可以分别存在于输入接口和输出接口中，也可以在两个接口中都具有，还可以独立于两个节点，与其他一些部

分组成独立的crossbar进行交换。可见，具体的微结构构成方式可以设定，但是这些部分是作为一个完整节点必不可少的部分。

需要特别说明的是，在有的文章和研究成果中提到了NI（Network Interface）接口，NI作为片上IP或者微处理器核心与NOC的接口，其主要作用是将IP中的原始数据流转换成可以被NOC所识别并传递的数据。但是，由于在目前的应用中各个IP或者处理器核具有不同的作用，所以其原始数据格式也不可能一样，所以针对不同的IP将会出现不同的NI。目前也没有任何研究机构或者个人提出了一种适用于各种IP的通用NI接口。所以，在此，我们将NI划分在IP中，作为IP的一个部分。不属于路由节点的构成部分。

对于平面的路由节点微结构来说，其标准形式中应该有5个输入和输出接口，分别代表东、南、西、北和本地5个方向。本地方向与本地的IP通过NI相连接。每一个输入口可以通过其他4个非本方向的输出口请求输出数据。在某些具体的应用中，接口的方向是可以扩展和缩减的，如在黑总线应用中，有的节点就仅仅需要两个出入口。而在[25]中提到的加入了长链路的节点，有的则需要6个输入输出口（当然，为了避免死锁，也有节点会减少一些出入口）。但是，这些应用都是一些具体特殊的应用，涉及到应用的细节问题，与我们提出一种路由节点的微结构标准定义并不矛盾。同时，我们设计的路由节点在微结构上也可以很方便的进行出入口的扩展。

数据缓存可以有不同形式的物理实现，位置可以位于输入接口中，也可以在输出接口中，同时，在我们设计的微结构中，虚拟通道是应该存在的。虚拟通道对路由节点甚至整个网络的性能有非常大的影响，但是存储器的耗费对整个片上资源的消耗来说是非常严重的，所以，对一种适用的NOC路由节点来说，必须能方便的调节缓存的特征值（数量和深度）。例如，在采用虫洞路由的NOC中，其需要的虚通道或者缓存就可以比采用存储转发的NOC要少。

输入接口和输出接口都应该包含各自的控制模块，用来接受请求，反馈响应信息，并根据一定的算法来仲裁选择。仲裁选择的算法可以根据具体应用不同来选择，目前比较常用的是RR（Round Robin）算法[26]，其他还有静态优先级（Static Priority）[27],时分复用机制（Time Division Multiplexing）[28]和彩票算法（Lottery）[29]。

5.5 路由节点微结构设计

5.5.1 分布式路由节点体系结构

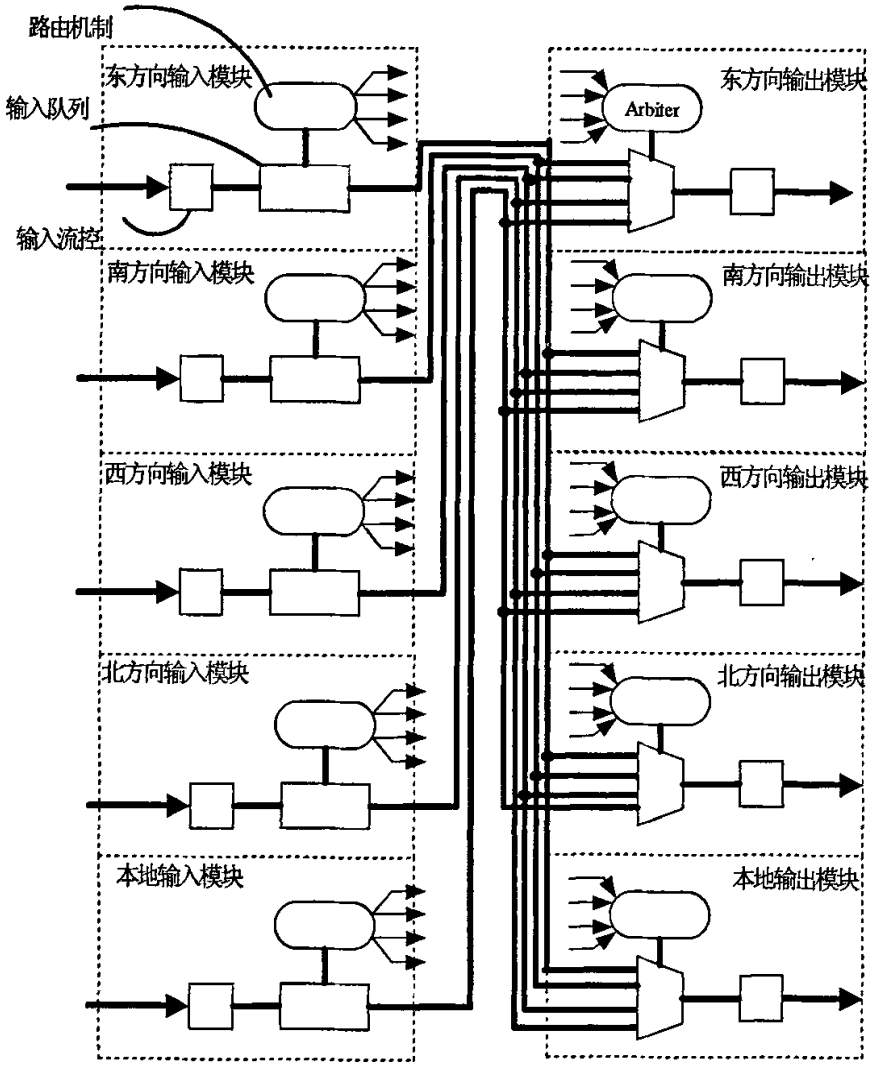


图5-4 分布式路由节点体系结构

一般说来，片上网络的路由节点可分为集中式和分布式。集中式路由主要由：（1）crossbar，（2）用于缓冲和流控的 FIFO 集和链路控制器集，（3）用于路由判优和交换的中央控制器。在分布式路由中，crossbar 和控制器都被分到各模块的端口输入输出通道之间。在我们的路由节点设计种就是采用分布式路由体系结构，

并且采用了输入缓冲的方式，当然，我们设计的结构也可以很方便的调整为输出缓冲的方式，这视具体应用而定。结构如图 5-4 所示。

5.5.2 输入模块

5.5.2.1 输入模块总体

输入模块包括三个部分，IFC（Input Flow Controller）输入流控模块，IB（Input Buffer）输入缓冲模块和 HD（Header Decoder）头部译码模块三个模块构成，其逻辑框图如图 5-5 所示。

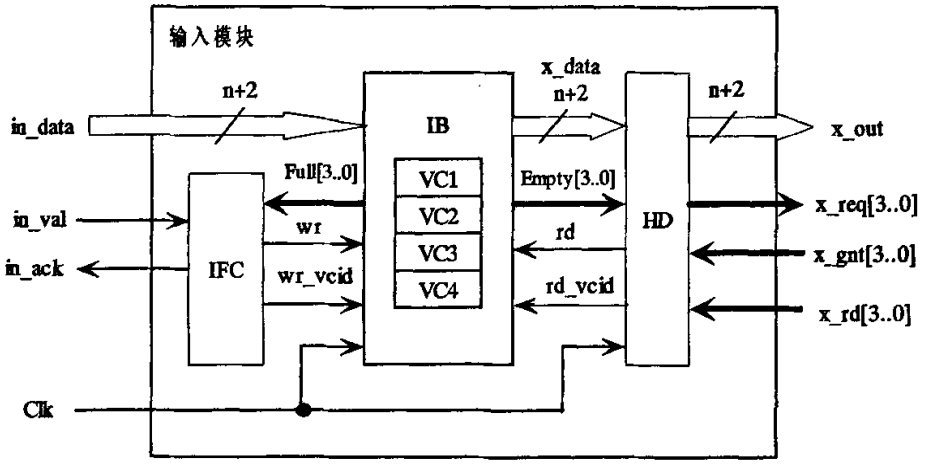


图5-5 输入模块逻辑框图

5.5.2.2 IFC（Input Flow Controller）输入流控模块

输入流控模块的作用是翻译握手协议和选择写入虚拟通道。in_val：表示上一级路由节点有有效数据来临，请求输入；当一个完整的数据包传输完毕，该信号放弃请求。in_ack 信号是输入确认信号，当来自 IB 模块的 Full 信号不全为高电平 1 时，说明至少有 1 条虚拟通道还有存储空间，可以接收数据包，此时 IFC 将 in_ack 响应信号置高电平，通知上级路由节点可以开始传输数据；若数据包传输过程中，所使用的虚拟通道达到满状态，停止响应，记录当前数据包与所使用的虚拟通道标号，以备下次使用。同时转换下一个不满的虚拟通道，准备开始新的数据包传输。

5.5.2.3 IB（Input Buffer）输入缓冲模块

数据缓存模块用于缓存数据，主要是由 N（N=1）个虚拟通道组成，在实现时

每个虚拟通道使用一个 FIFO，虚拟通道的宽度就是数据包的宽度，虚拟通道的深度可以由用户自己定义。所以在进行具体的设计实现时，应该注意实际可用资源和应用的要求，找到一个最佳的选择。这里，我们选择了 4 个虚拟通道，深度为 16。使用虚拟通道的优点是可以避免头部阻塞，所谓头部阻塞是如果某一缓存头部的包由于拥塞而不能交换到一个输出端口，那么该缓存中余下的包也会被线头包所阻塞，即使这些包的目的端口并没有拥塞。

图 5-6 中，wr 是来自 IFC 模块的表示 FIFO 写使能的输入信号；rd 是来自 IC 模块的表示 FIFO 读使能的输入信号；wr_vcid 是选择写入虚拟通道的标号；rd_vcid 是选择读出虚拟通道的标号，同时 IB 模块将 4 个虚拟通道的空、满状态通过 empty 和 full 信号分别输出给 IC 和 IFC 模块，需要说明一下，这里只要虚拟通道的剩余空间小于数据包长度即显示“满” 状态，这样可以保证数据包在传输的过程中不会断裂，不需要进行路径保持，减少了算法复杂度；clk 为时钟信号。

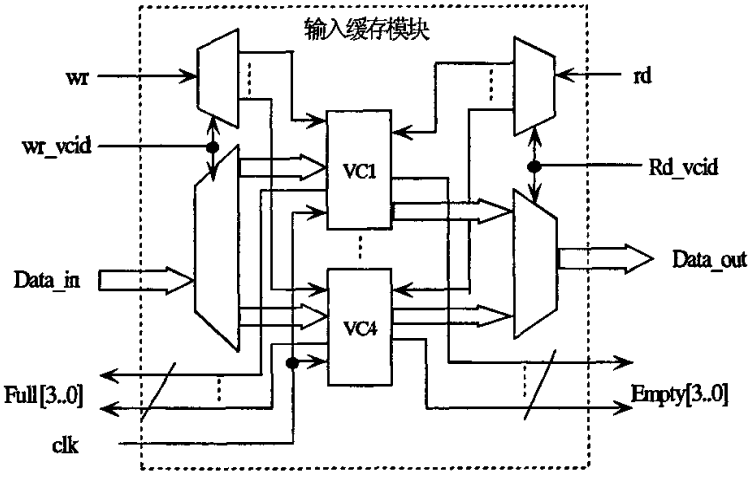


图5-6 IB（Input Buffer）输入缓冲模块

5.5.2.4 HD (Header Decoder) 头部译码模块

头部译码模块（图 5-7）是输入模块中最重要和复杂的一个部分。它负责执行带有一定自适应性的维序路由算法。它首先通过虚拟通道选择器选择一个非空通道输入数据包，对包头进行分析，向相应方向的输出端口发送 x_req 请求信号，并接收 x_gnt 允许信号和 x_rd 读使能信号。一旦连接建立，头部译码器便开始从虚拟通道向输出单口传送数据包。当 1）输入虚拟通道状态变为空，或 2）输出端口阻塞，头部译码器主动放弃连接。然后选择下一个非空虚拟通道进行进行上述操

下面分别仔细讨论虚拟通道选择器，数据流控制器和路由选择器。

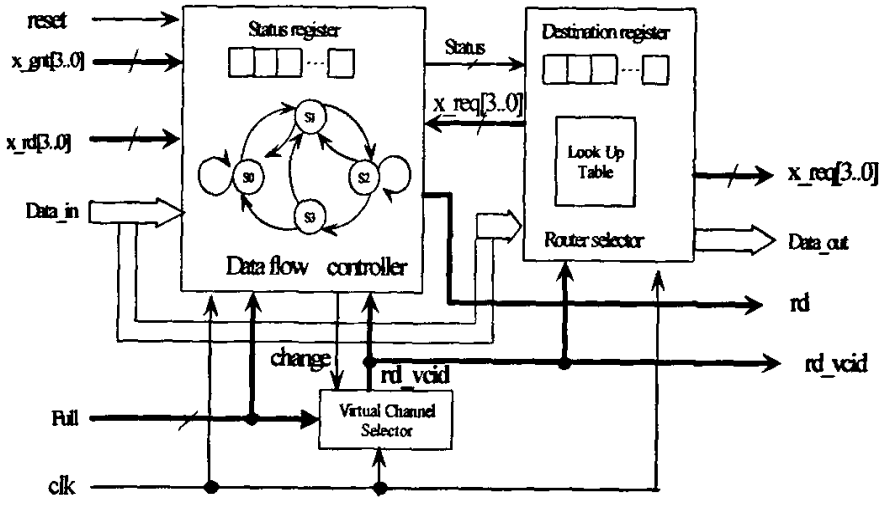


图5-7 头部译码模块逻辑框图

- 虚拟通道选择器（Virtual Channel Selector）

对于虚拟通道的选择借助了 Round Robin 调度算法的思想。

Round Robin 调度算法，又名时间片轮循法，是一种经典的调度算法。在分时系统中，算法将 CPU 的处理时间分为时间片（Time slice），一个时间片从几毫秒到几百毫秒不等。系统给每一个进程分配若干个时间片，被调度选中的进程进行完了系统分配的时间片后，系统就会发生调度，如果该进程还未完成要求的任务，则调度程序暂时停止该进程的执行，将它排到就绪对列的末尾，等待下一次调度。同时，调度程序会调度当前就绪队列中的下一个进程。这样就可以保证就绪对列中的所有进程在一个给定的时间均能得到执行。

将 Round Robin 调度算法稍作修改，就可以将它用在虚拟通道的选择中。只是这里时间片比较固定，一般为传输一个完整数据包所需要的时间。而各个数据包所在的不同虚拟通道相当于分时系统的进程，调度过程也类似，如图 5-8 所示。

在图 5-8 中，假设系统有 8 个需调度的通道，调度的起点在 A 通道，通道中的数据包数目各不相同。首先，由于 A 通道有数据包可发送，故第 1 轮调度出 A 通道，然后在第 2 轮调度中，先检查 B 通道，由于 B 通道没有数据包可发送，所以跳过 B 通道，检查 C 通道，发现 C 通道有数据包，因而 C 被选出。第 3 轮调度，先检查 D 通道，从而调度出 D 通道，以此类推，其中第 6 轮调度完成了对所有通

道的检查，重新从 A 通道开始检查；第 8 轮调度出于 D 通道在第 3 轮调度中把一个数据包发送出去后处于无数据包可发状态，从而调度出了 E 通道。从图中可以看出要实现 Round Robin 算法，关键是从起始的通道开始，找到第一个“准备好”（有数据包）的通道。

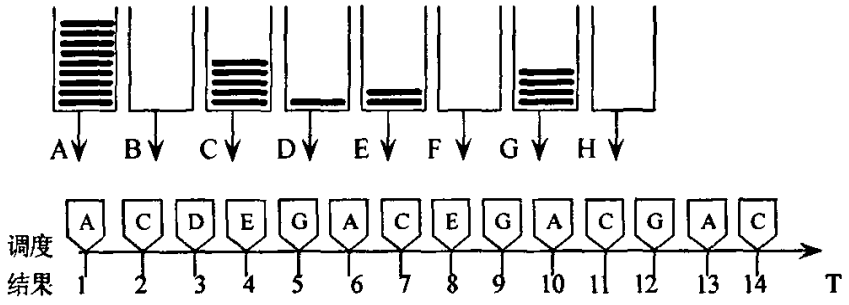


图5-8 Round Robin 调度算法示意图

实现 Round Robin 调度算法的逻辑结构图如图 5-9 所示。通道状态寄存器用来存放需调度通道是否准备好（即该虚拟通道内是否有数据包）的状态，它是 Round Robin 调度的基础，准备好的通道才可以被调度上，没准备好的（即没有数据包的虚拟通道）则不可能被调度出来。以 4 虚拟通道为例，通道状态寄存器就是 4 位的寄存器，通道状态由前一级 FIFO 的~empty 信号给出，哪一位为“1”表示哪一个通道非空，有数据包可传输。

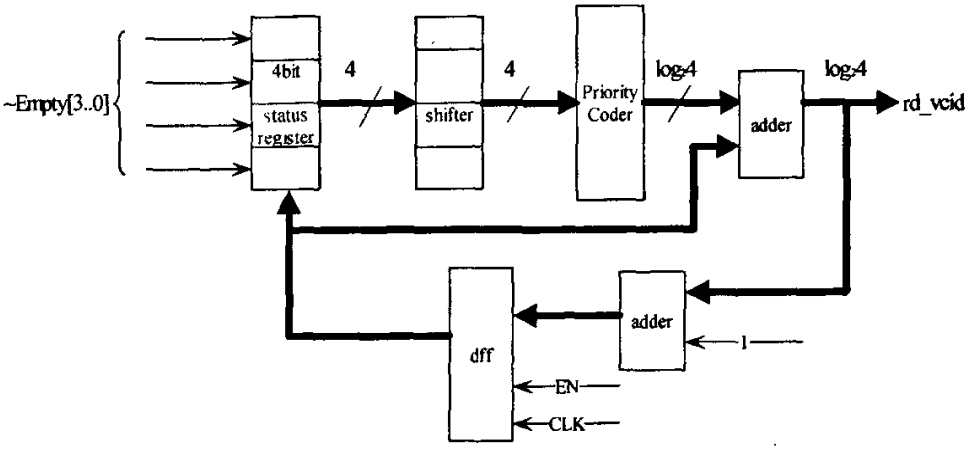


图5-9 算法实现逻辑结构图

存器完成移位操作。桶式移位器是现代告诉 RISC 微处理芯片中采用的结构，它能使各种位数的移位操作都能在单周期内完成，这对于算法的实现是很关键的。

桶式移位器完成移位操作后，将所得的结果送到优先编码器进行优先编码。

调整逻辑主要是一个累加器，它把优先级编码器的编码结果加 1 后进行累加，这就相当于把每次 Round Robin 调度的开始点给保存在累加器中。累加器的位数由调度通道数决定，4 个通道需要 2 位的累加器。

- 数据流控制器（Data Flow Controller）

数据流控制模块用于控制从虚拟通道到输出端口的数据流。如图 5-10 所示，控制器有 4 个状态，S0，S1，S2 和 S3。S0 为空闲状态，当发现有非空的虚拟通道时便转换为状态 S1。状态 S1 为头部译码状态，在该状态中控制器等待输出端口的 x_gnt 允许信号。控制器先向 X 方向发送请求，若得到允许信号则跳到 S2 状态，否则向 Y 方向发送请求，若得到允许信号则跳到 S2，进行数据包传输；若依然没有得到相应，则放弃请求，保存当前路由信息后，转向下一个非空虚拟通道进行操作。S2 为数据传输状态，由于下级节点为传输保留了至少一个数据包的空间则一定可以保证数据包顺利传输到包尾，即由 S2 状态跳到 S3 结束状态。S3 状态中，对传输过程中暂存的信息进行清零，同时检测是否有非空虚拟通道，有则跳到 S1 状态，否则回到 S0 状态。

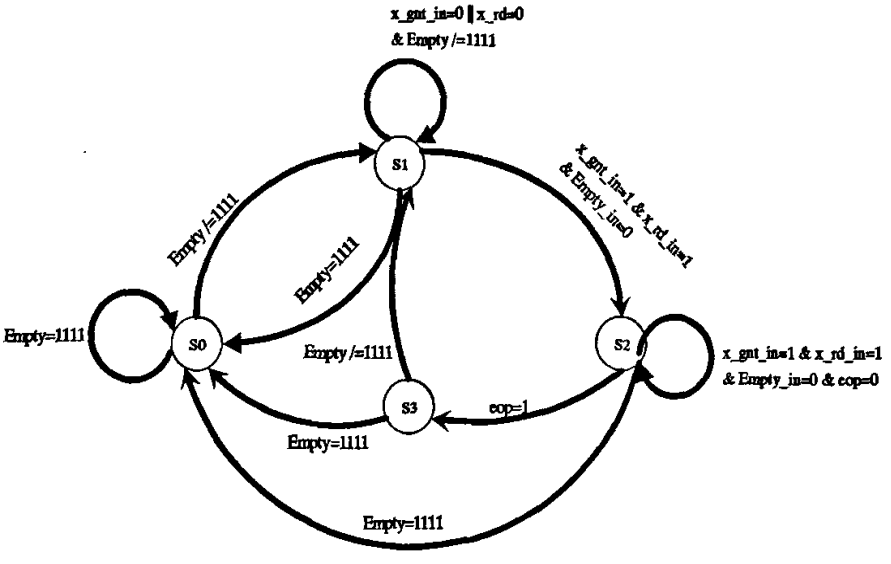


图5-10 IC 模块状态转移图

- 路由选择器（Router Selector）

在 Mesh 结构下，路由选择器执行带有一定自适应性的维序路由，在状态 S1 下，接收包头，根据查找表查找路由信息，向相应输出端口发送 x_req 请求信号。在 Torus 结构下，路由选择器将包头中目的地址的标号（a3a2a1a0）和当前节点的标号（b3b2b1b0）从最低维开始比较，当发现标号不同时，向相应的维序发送 x_req 请求信号。

5.5.3 输出模块

输出模块中由于并不含有 FIFO，所以其结构相对输入模块来说要简单一些，主要包括 Arbiter 仲裁器、输出流控制器 OFC 两个部分。

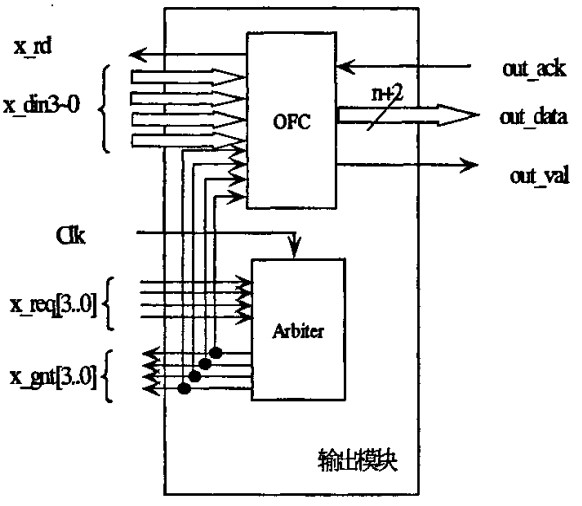


图5-11 输出模块逻辑结构图

- Arbiter 仲裁器

Arbiter 仲裁器用来对请求同一输出端口的不同输入模块进行仲裁。这里依然使用 RR(Round Robin)轮循算法来对请求进行调度。选中一个请求，将相应的 x_gnt 信号拉高，通知该输入端口可以进行数据传输。同时将调度结果通知 OFC 输出流控制器。

- OFC (Output Flow Controller)输出流控制器

输出流控制器根据 x_gnt 允许信号选择一路数据作为输出数据，同时将 out_val 信号拉高，通知下级路由节点有有效数据来临。将下级路由节点反馈回来的 ack 信号作为 x_rd 读使能信号传输到输入模块，这样当输入模块同时接收到 x_gnt 和

x_rd有效信号时，输入输出模块间连接建立，数据包开始传输。

第六章　路由节点的验证与测试

在本章中，我们先对 FPGA 设计的测试思想、测试方法和测试技巧作一定的总结，再对基于 FPGA 设计的路由节点进行仿真验证，然后对测试环境作简要介绍，并对其进行测试，最后给出的测试结果。

6.1　FPGA 测试思想和方法介绍

在 FPGA 内部，所有逻辑单元和存储块的连接使用情况可以由用户通过硬件描述语言进行配置。其内部不存在固定的逻辑或物理关系，因此，FPGA 不能像 DSP 那样，通过标准的 JTAG 测试口监测内部寄存器的状态。在 FPGA 设计完成后，整个 FPGA 对于测试人员来讲是一个“黑盒子”，对其工作状态的监测和结果的验证是 FPGA 电路设计时的一大问题。

6.1.1　FPGA 可测试性设计

在进行 FPGA 设计时，应该使 FPGA 在设计阶段就使 FPGA 具有可测试性（testabililty）。这包括两个方面，一是可控制性(controlability)，即 FPGA 的测试能够很好的得到操控；另一方面是可观测性(viewability)，即测试输出能够容易被观察并且测试结果的正确性容易得到验证。

FPGA 的可测试性设计[30]是指在 FPGA 设计初期，对每个单独子模块编写时，即要求保留出对该模块的测试接口。这些测试口的输入能够在测试使能的情况下，反映该模块中数据处理每一步的运行情况，并且通过这些测试输出，能够方便的观测到模块运行是否正确，并且能够对出现的故障进行快速准确的定位。

由于对 FPGA 模块的测试工作越来越成为 FPGA 设计和 IC 设计中最重要的一步，大约能够占到整个开发流程的 60%的时间。所以好的设计模块在开发时一定要注意测试的方便性。

6.1.2　软件仿真测试

在 FPGA 设计中，一般要求所设计的模块在逻辑上没有明显的逻辑错误，对暗藏的逻辑问题需要通过仿真的办法进行查找，尤其是在 IC 设计中，由于流片成

本太高，需要在流片之前，对 IC 中模块及系统的运行情况进行全面的仿真过程。

软件仿真测试是指在软件环境下虚拟出设计出一个测试平台（testbench），该平台能够产生需测试的模块的各种输出情况，并且，该测试平台能够接收模块在这些激励下的输出，并对这些输出进行自动分析，得出测试结果报表。软件仿真测的关建在于仿真测试平台和测试算法的代码覆盖率。

这种仿真测试一般分为前仿真和后仿真，前仿真是指测试模块的逻辑关系是否达到设计要求；后仿真要求在对模块进行布局布线后提取线路延迟参数，测试模块能否在实际芯片中达到设计要求。

6.1.3 外部激励测试

一个好的软件测试平台和测试算法能够做到 90%以上的代码覆盖率，但往往测试平台软件相当昂贵，同时测试算法设计复杂度往往会超过 RTL 代码的编写。另一种常用于 FPGA 设计的测试方法是进行外部激励测试。

这种测试方法是用外部的信号发生系统（如可编程信号源）产生实际的输入激励源，并将 FPGA 内部需要测试的信号在布线时连接到测试点上，用示波器或逻辑分析仪进行分析。这种方法的好处是可以很好的模拟实际的输入，并且简单易行。但是它的缺点也是明显的：

(1) 不能做到对所有可能情况进行测试，测试覆盖率差；

(2) 由于测试不同的输出需要不同的布线过程，会花费大量的时间；

(3) 测试点需要占用 FPGA 一定的 I/O 资源，并且影响 PCB 布线；

另一种改进的方法是采用由 FPGA 厂商提供的观测软件(如 XILINX 的 ChipScope)[31]，这种测试软件通过在综合时，插入 ChipScope 测试所需要的 IP 核，这些 IP 核使用 FPGA 中未用的资源，将用户所设定的测试信号通过 JTAG 口输入 PC 机，并进行实时的显示。

6.1.4 实际运用测试

这主要在 FPGA 设计的后期，在整个产品系统基本完成的基础上，在整机系统的环境下，测试 FPGA 在实际运用中是否会出现问题，这些问题包括外部突然的冲击、实际温度、湿度对 FPGA 工作的影响等。

同时，实际运用的测试还有一个重要目标是最后检测 FPGA 及整个系统中的算法能否达到项目的技术要求，比如信号输入的动态范围等等。

6.2 仿真验证

按照上述设计思想进行设计输入，并进行仿真验证。下面是 4x4 的 2D Mesh 结构，对 16 个路由节点按顺序进行编号（16 进制），如图 6-1。我们知道 Torus 结构与 Mesh 结构的路由节点，在 FPGA 实现中，只有路由选择器中的路由算法不同，其他实现都相同，例如虚通道选择，请求输出，仲裁响应等。所以这里以 Mesh 结构的 5 号路由节点为例，进行仿真验证。

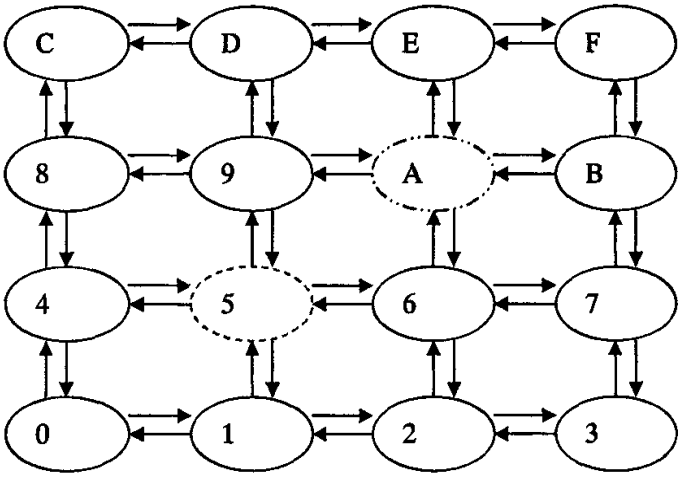


图6-1 节点已标号的 4x4 Mesh 网络

首先，验证单个方向的数据输入输出。如图 6-2，看来自东方向的数据包 Ein_data，根据数据包的各式（5.4 节），可知道 16 进制的输入数据“249”，2 表示 bop = ‘1’，‘4’ 为源地址——4 号节点，‘9’ 为目的地址。所以这是一个 4 号节点通过 5 号节点，向 9 号节点传输数据包的过程。

如图 6-2，输入信号为：clk，reset，Ein_data（来自东方向的数据），Ein_val（输入数据有效），Nout_ack（来自北输出端口的响应信号）；输出信号：Ndata_out，Nout_val，Ein_ack（向东边上级端口输出的响应信号）。可以看到整个输入输出过程经过了 4 个时钟周期：第 1 拍，5 号节点的东边输入口检测到 Ein_val 和 bop=1 的信息，检查是否有空闲虚拟通道；第 2 拍，将数据写入 FIFO，同时头部译码器将读使能置为 1；第 3 拍，头部译码器读出包头，分析路由，并向相应的方向发送 x_req 请求信号，同时将读使能置为 0；第 4 拍，输出节点响应请求；第 5 拍，继续读 FIFO，顺利将数据包送到输出端口，Nout_val=’1’。

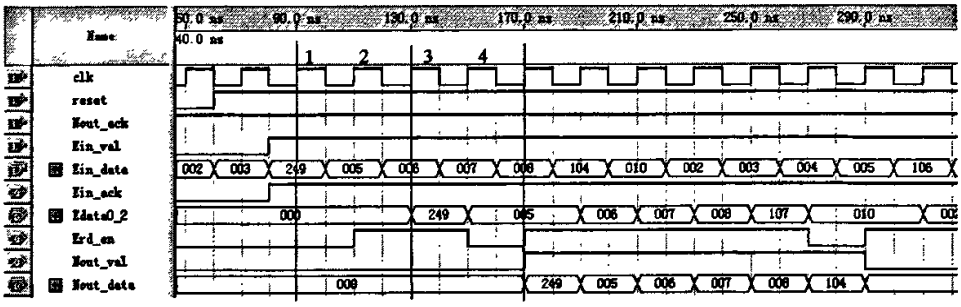


图6-2 单个方向的数据输入输出下，节点的功能仿真波形

图 6-2，是数据传输的理想情况——没有冲突。下面给出两个输入端口请求同一输出端口的仿真例子。看图 6-3，来自 9 号节点的数据包通过 5 号节点的北口输入，请求东输出端口，将数据送到 7 号节点。在这一过程中，来自 0 号节点的数据包通过 5 号节点的南口输入，请求东输出端口，将数据送到 6 号节点。可以看到，按东输出端口按顺序，将两个数据包都输出。从图中可以看到，两个数据包传输之间隔了一个时钟周期，这是头部译码器中的，数据流控制器的“S3”（End）状态，该状态用来释放暂存的路由信息。

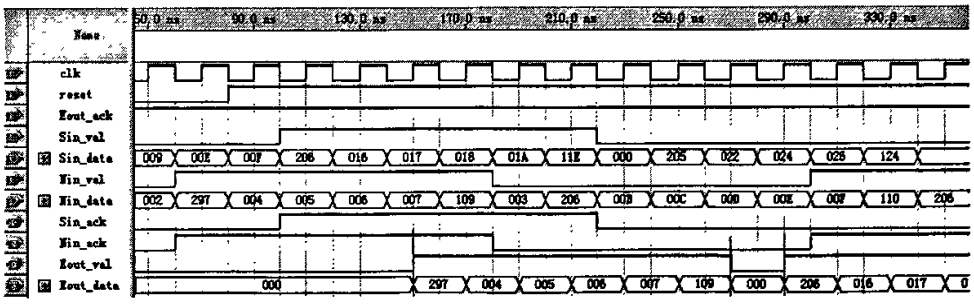


图6-3 数据传输无冲突时，节点的功能仿真波形

同理可以得到其他方向的仿真验证结果。

但是，对于路由节点这种设计，想要给出完备的测试向量，即想要模拟出所有可能出现的冲突情况是不太容易的，所以这里采用对路由节点各端口随机的输入数据包，将输出端口输出的数据包进行存储，然后进行比较来验证。

6.3 路由节点测试流程

根据上一节所述的路由节点测试思想，我们设计了如下的节点测试流程，如

图 6-4 所示。

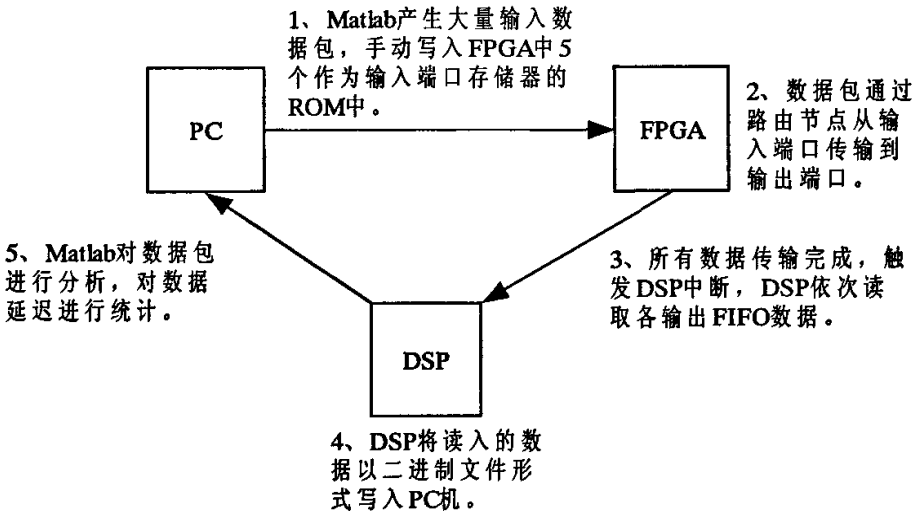


图6-4 路由节点测试流程

这里需要注意两点，第一是由 Matlab 产生符合要求的数据包。所谓“符合要求”的数据包是指，首先，数据包的格式要满足 5.3 节中的要求。其次，由于路由节点的设计是采用维序路由，所以路由节点不支持数据包的回传，设计中所使用的是 5 号路由节点，其南方向的输入端口不支持目的地址为 0，1，2，3 的数据包，如图 6-5。同理，其西方向的输入端口不支持目的地址为 0，4，8，C 的数据包；其北方向的输入端口不支持目的地址为 8，9，A，B，C，D，E，F 的数据包；其东方向的输入端口不支持目的地址为 2，3，6，7，A，B，E，F 的数据包。将数

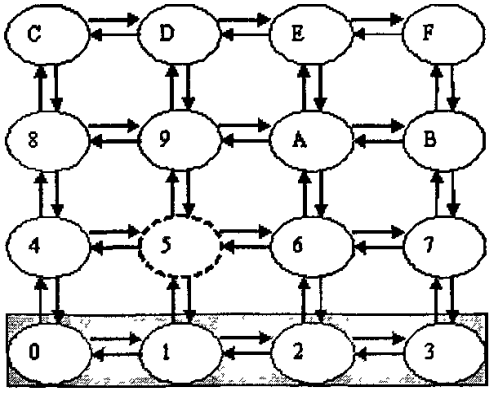


图6-5 5 号路由节点南方向输入数据包所含目的地址的要求示意图

据包保存为存储器数据文件.mif 文件，如图 6-6，之后直接调用 FPGA 内嵌的 ROM 资源作为数据包存储器，将刚才生成的.mif 文件作为 ROM 的初始化文件，如图 6-7。

Addr	+0	+1	+2	+3	+4	+5	+6	+7
0	532	132	125	209	136	256	540	253
8	226	239	69	256	529	46	235	207
16	178	256	536	152	164	127	202	256
24	529	43	129	121	241	256	528	48
32	100	213	204	256	532	136	53	45
40	206	256	536	183	17	192	132	256
48	537	5	200	149	100	256	541	43
56	176	94	128	256	536	186	202	134
64	124	256	532	252	254	129	128	256

图6-6 将 Matlab 产生符合要求的数据包写入.mif 初始化文件

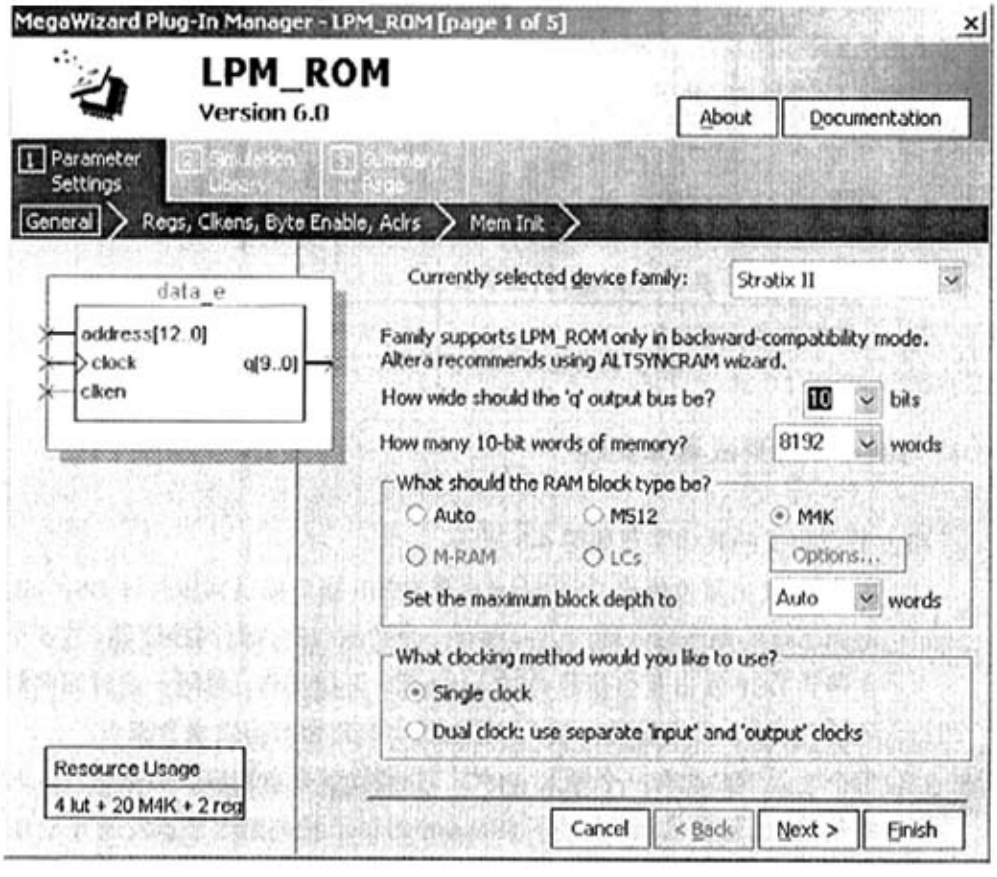


图6-7 直接调用 FPGA 内嵌的 ROM 资源

第二是数据包注入路由节点的时间。数据包注入路由节点的时间可以服从多

种分布，这里选用均匀分布。我们实现的具体方法是，在节点的每个输入方向建立一个触发时间存储器，用于存储数据包送入路由节点的时间间隔。使用 Matlab 产生一系列服从均匀分布的随机数，将这些随机数保存分别保存在五个触发时间存储器中。在每个输入方向，计数器每从触发时间存储器中读出一个随机数就对其进行计数，计到后，向相应方向的输入 FIFO 发送脉冲，输入 FIFO 向路由节点发送数据包，如图 6-8。由此可知，这些随机数其实影响的是网络的注入率。随机数越大，计数器产生脉冲的时间间隔越大，向路由节点送入数据包的时间间隔就越大，数据注入率就越低；反之，随机数越小，计数器产生脉冲的时间间隔越短，向路由节点送入数据包的时间间隔就越短，数据注入率就高。

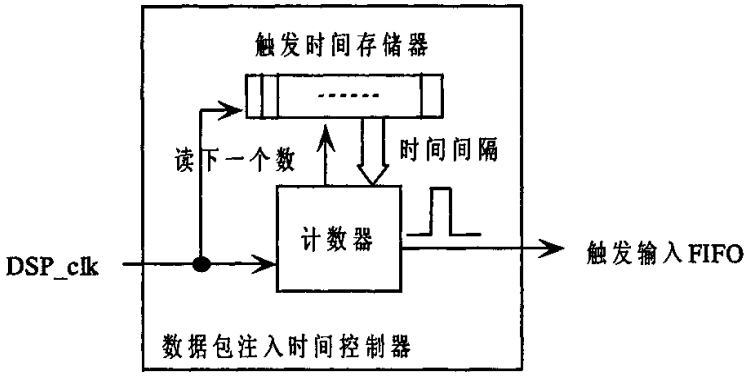


图6-8 数据包注入路由节点的时间控制

6.4 路由节点测试具体实现

路由节点测试的具体实现如图 6-9 所示：

(1) 由 DSP 发出复位信号 DSP_reset，其 EMIF 接口提供同步时钟 DSP_clk；

(2) 路由节点的每个输入端口方向都有一个数据包注入时间控制器（共 5 个），在得到 DSP 发出复位信号 DSP_reset 后，开始工作，每隔一定时间向对应的输入 FIFO 发送脉冲，输入 FIFO 就会向路由节点发送数据包。

(3) 每个输入端口都有一个输入 FIFO，分别收到来自相应“数据包注入时间控制器”的触发信号，在接下来的 6 个时钟节拍向路由节点发送 6 个 flits，即一个数据包。

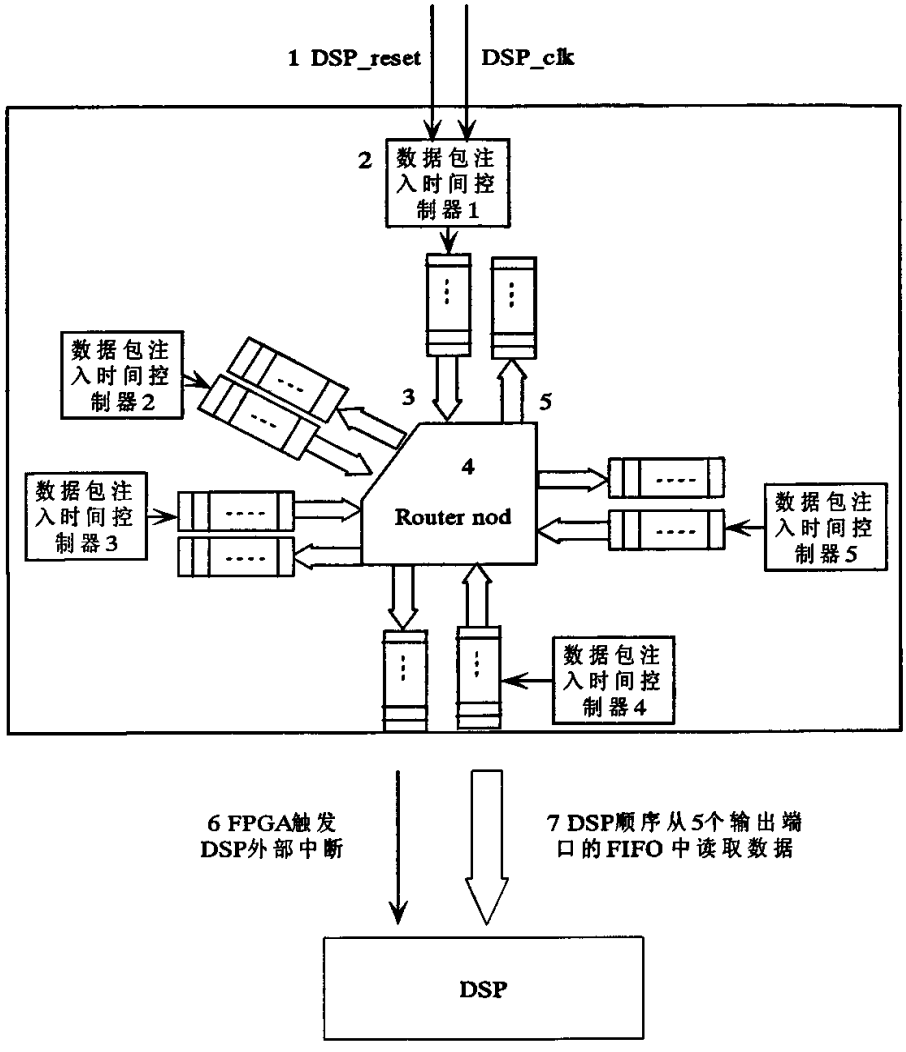


图6-9 路由节点测试方案

(4) 路由节点将数据包包头中的目的地址信息将数据包送到相应的输出端口。

(5) 每个输出端口将数据包分别写入自己的输出 FIFO。

(6) 全部数据传输结束，FPGA 向 DSP 发送外部中断，DSP 进入中断程序。

(7) DSP 在中断程序中按顺序从 5 个输出 FIFO 中巴数据读出，并写入文件。

(8) 在 Matlab 里将输入输出数据进行比较、分析。

通过将输入输出数据进行文件比较，可以发现设计的路由节点即使在随机冲突下也可以正确传输数据包。

6.5 基于 FPGA 设计的路由节点的其他结论

6.5.1 路由节点资源评估

首先，对该路由节点微结构所占用的资源进行分析。随着 NOC 技术的发展，需求的趋势将使得人们越来越希望使用更少的资源，实现更多的功能。因此，作为 NOC 中最基本结构的路由节点，其资源占用量决定了整个 NOC 网络的资源占用率。很容易知道，在该路由节点微结构中，由于各个部分功能不同，承担的任务也不一样，因此，它们会具有不同的资源占用率。资源占用率如表 6-1 所示。

表6-1 微结构在 stratix-II FPGA 上的实现

FPGA Unit / Component	ALUTs	ALMs	Memory (bits)
IFC	29	19	0
HD	124	76	0
V_Ch	132	84	640
OC	14	8	0
OFC	26	24	0

6.5.2 单节点数据延时统计

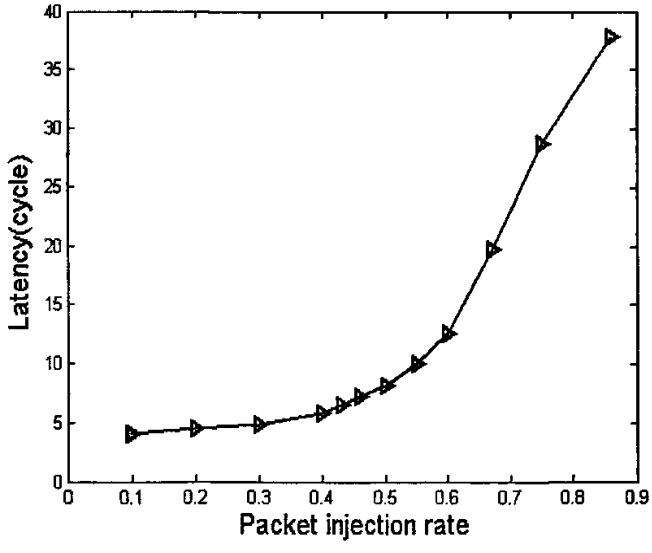


图6-10 数据包通过单个路由节点的平均延时

时间延迟是NOC性能的一个重要方面，我们希望看到路由节点在不同数据输入率下，传送一个数据包所需要的时间。所以在 FPGA 中设计一个全局计数器，当数据包进入路由节点时，在包尾对记录当前计数器的值，当数据包离开路由节点时，再次记录计数器值，两者之差就为该数据包经过路由节点的延迟。最后在不同负载下对所有数据包尾进行求和平均，则得到我们所设计的路由节点的时间延迟，如图 6-10。

可以看到，当注入率为 10%左右，网络负载很低，几乎没有数据冲突，数据包的平均延迟为 4.3 左右，与理论值 4 相符合。注入率在 40%之前节点性能都良好，数据包的平均延迟缓慢增长为 6.8 左右。注入率在 40%到 60%之间，数据包的平均延迟开始较快增长。注入率在 60%之后，数据包的平均延迟超过了 12 并急剧增加。

第七章　结论与展望

7.1　结论

本论文对片上网络的路由方面的技术进行了较深入的讨论，主要内容如下：

(1) 介绍了片上网络技术的研究背景和当前发展状况；

(2) 对片上网络拓扑结构、包交换技术、虚通道技术以及路由算法这些关键概念进行了讨论；

(3) 深入分析了 Mesh 结构和 Torus 结构，设计了一种全新的、适用于 Torus 拓扑结构的路由算法，该算法不仅可以防止死锁、活锁，而且简单易实现。同时，论文中还给出了该算法下，Torus 网络延时和归一化吞吐量的仿真结果；

(4) 在分析了算法的基础上提出了 FPGA+DSP 的硬件测试平台构架，并对其进行了实现和调试；

(5) 介绍了一种采用的虫洞路由作为流控机制、使用带有一定自适应性的维序路由作为路由算法的路由节点的 FPGA 设计，整个 FPGA 设计在 Altera 集成开发环境 Quartus6.0 上完成。

(6) 对节点的仿真验证、测试和得到的结论进行了分析。

本论文中所讨论的 Torus 路由算法具有创新意义，设计的硬件和基于 FPGA 的路由节点的实现则具有现实指导意义。

7.2　展望

本课题对片上网络的两种最具代表性的直接型网络——2D Mesh 网络和 2D Torus 网络进行了深入研究，而对间接型网络未有涉及。间接型网络拓扑的研究具有一定复杂性，但随着人们对片上网络的深入研究、网络传输的多样化需求，间接型网络拓扑结构将逐渐显示出它不可替代的作用。

目前，我们基于 FPGA 的路由节点的设计，只是针对具有一定自适应性的 XY 维序路由算法的、采用虫洞路由技术并含有虚通道的路由节点，对于采用其他路由算法和流控机制的路由节点的设计，也是以后研究中非常必要的。而且，路由

节点的 FPGA 设计只是迈出了片上网络硬件实现的第一步，连结成真正的网络所得出来的结论（如网络的数据包延迟、网络的吞吐量），才更具有现实意义。

最后，片上网络的最终目的是走向应用，IP 核的设计、网络接口 NI 的设计以及片上网络功耗的评估等等，都是我们还未涉及但最终不可避免的研究领域。

总之，需求的增长、越来越高的芯片集成度和越来越复杂的片上结构，使得 NOC 取代总线结构已经成为了必然趋势。随着对其研究的不断深入，NOC 技术一定会得到越来越快速的发展。

致谢

本课题的研究经历了漫长而艰辛的过程。我要衷心地感谢许多教育我、帮助我和关心我的人。

首先要向我的指导教师李玉柏教授致以我最真诚的感谢和无比的敬意！他为研究室创造了优越的硬件条件，为我们提供了良好的学习气氛和实验环境。他严谨的科研作风、深厚的专业功底和不倦的工作精神，潜移默化地影响着我，使我受益匪浅。他给与我的不仅仅是学习上的指导，更重要的是思想上的启迪和鼓舞。

在此，我还要对李桓老师表示真心的感谢！他在信号处理领域有丰富的经验，为本课题的研究提出了许多宝贵意见。同时，李桓老师也教给我许多系统设计与工程实践的经验。

感谢和我同在一个教研室的许多优秀的同学。在本课题的设计和实现过程中，黄争、彭洪、武畅、朱芸、刘凌桥给与了我极大的帮助，他们总是耐心的解答我的疑问，热情的与我讨论学习上的心得体会。本文的顺利完成和他们的支持和鼓励密不可分。感谢艾真、王坚、柴松、杨中明等同学，在学习上给了我很多的帮助。感谢所有在我攻读硕士学位期间给与我帮助的老师和朋友。

此外，感谢我的父母给予我的无私关爱和鼓励，是他们使我保持对生活的热爱和追求生命真谛的信心！感谢我的家人给我的温暖支持，给我坚强面对困难的勇气！

最后，衷心感谢为评阅本论文而付出辛勤劳动的各位专家和学者。

参考文献

[1] Axel Jantsch, Hannu Tenhunen. Network on Chip [M]. Academic Publishers. June 2003: 6.

[2] Axel Jantsch, Johnny Qberg, Hannu Tenhune, Special issue on networks on chip. Journal of Systems Architecture, Volume S0, Issues 2-3. February 2004: 61-63.

[3] L. M. Ni, P. K. McKinley. A survey of wormhole routing techniques in direct networks. IEEE Tran. On Computers, Feb 1993.

[4] Axel Jantsch. NOC Architecture. ESSCIRC. September 2001.

[5] Shashi Kumar, Axel Jantsch, Juha-Pekka Soininen, Martti Forsell, Mikael Millberg, Johnny Öberg, Kari Tiensyrjä, Ahmed Hemani. A Network on Chip Architecture and Design Methodology. In Proceedings of IEEE Computer Society Annual Symposium on VLSI. April 2002.

[6] Resve Saleh, Steve Wilton, Mirabbasi. System-on-chip: reuse and integration. Proceedings of the IEEE, Vol. 94, No. 6. June 2006.

[7] Zhonghai Lu. Using Wormhole Switching for Networks on Chip: Feasibility Analysis and Microarchitecture Adaptation. IEEE, June 2004.

[8] Duato J, Robles A, Silla F.; Beivide R. A comparison of router architectures for virtual cut-through and wormhole switching in a NOW environment. Parallel and Distributed Processing, IPPS/SPDP. April 1999: 240 - 247

[9] Franck Binard, Deadlock Preventive. Adaptive Wormhole Routing on k-ary n-cube Interconnection Networks. University of Ottawa. December 2003.

[10] 华云，苏德富. 基于 Wormhole 的流控制新技术研究[C]. 计算机工程. 2001, 10 (10): 7-9.

[11] L. M. Ni, P. K. McKinley. A survey of wormhole routing techniques in direct networks. IEEE Tran. On Computers, 26:62.76, Feb. 1993.

[12] W. J. Dally, B. Towles. Router packets not wires: on-chip interconnection networks. DAC, 2001.

[13] Dally, W.J Aoki. Deadlock -free adaptive routing in multi-computer networks using virtual channels. IEEE Trans on Parallel and Distributed Systems. April 1993.

[14] Murali, S. Coenen, M. Radulescu, Goossens, De Micheli G. Mapping and configuration methods for multi-use-case networks on chips. Design Automation, 2006.

[15] G. Ascia, V. Catania, M. Palesi. Multi-objective mapping for mesh-based NoC architectu res. ISSS-CODES. 2004: 182–187.

[16] Altera Inc. Stratix II Device Handbook, www.altera.com, March 2005.

[17] EDA 先锋工作室，吴继华，王诚编著. Altera FPGA/CPLD 设计（高级篇）[M]. 北京：人民邮电出版社. 2005.

[18] Texas Instruments Inc. TMS320DM642 Datasheet. www.TI.com, August 2005.

[19] Texas Instruments Inc. TMS320C6000 DSP External Memory Interface (EMIF) Reference Guide. www.TI.com, April 2005.

[20] Texas Instruments Inc. TMS320C6000 DSP Enhanced Direct Memory Access (EDMA) Controller Reference Guide. www.TI.com, March 2005.

[21] Texas Instruments Inc. TLV320AIC23B Datasheet. www.TI.com, March 2003.

[22] Philips Inc. SAA7113H Datasheet. www.philips.com.cn, July 1999.

[23] Philips Inc. SAA7104 Datasheet. www.philips.com.cn, March 2004.

[24] Texas Instruments Inc. PT5400 series Datasheet. www.TI.com, May 2002.

[25] Texas Instruments Inc. PTH04070W Datasheet. www.TI.com, September 2004.

[26] Gao Xiaopeng, Zhang Zhe, Long Xiang. Round Robin Arbiters for Virtual Channel Router. Computational Engineering in Systems Applications, IMACS Multiconference on Oct. 2006: 1610 – 1614.

[27] Gaughan, P.T., Dao, B.V., Yalamanchili, S.; Schimmel, D.E. Distributed, deadlock-free routing in faulty, pipelined, direct interconnection networks. Computers, IEEE Transactions on Volume 45, Issue 6. June 1996: 651 – 665.

[28] Barry, L.P.; Guignard, P.; Debeau, J.; Boittin, R.; Bernard, M. A high-speed optical star network using TDMA and all-optical demultiplexing techniques. Selected Areas in Communications. IEEE Journal on Volume 14, Issue 5. June 1996: 1030 – 1038.

[29] Xufan Wu, Jun Yang, Longxing Shi. Bus Buffer Evaluation of Different Arbitration Algorithms. SOC Conference, 2005. Proceedings. IEEE International 25-28 Sept 2005: 261 - 264

[30] Alfred L.Crouch. Design-for-test for digital IC's and embedded core systems.USA: Pearson Education. 2004.

[31] 钟信潮，王诚，薛小钢.实用 FPGA 调试工具-ChipScope Pro.Http;//www.edacn.net.

个人简历

出生年月：1981 年 10 月；

2000 年 9 月—2004 年 7 月：　电子科技大学通信学院，获得学士学位；

2002 年 9 月—2004 年 7 月：　电子科技大学外国语学院，获二学位；

2004 年 9 月—2007 年 6 月：　电子科技大学通信学院，攻读硕士；

攻读硕士学位期间的研究成果

参加的科研项目：

(1) 国家自然科学基金项目——复杂 SoCs 片上通信关键技术研究。

(2) 安捷伦研究基金——Multi-DSP infrastructure for measuring device based on NOC

发表、录用的论文与申请专利：

[S1] 黎黎，李玉柏，片上网络概述，2004 年中国西部青年通信学术会议论文集增刊，2004. 799.