

NoC 路由算法及仿真模型的设计与研究

摘 要

随着半导体技术以及集成电路技术的飞速发展,单个芯片中 IP(Intellectual Property)核数量越来越多。当单个芯片上集成的 IP 核数目达到成百上千的时候,基于片上总线的 SoC(System-on-a-Chip)在设计上遇到了全局时钟难以同步、地址空间有限、无法支持多节点并行通讯与系统扩展不够灵活等问题,严重制约了集成在单一芯片上的 IP 核规模及系统性能。将计算机网络技术引入 SoC 设计领域,以片上网络的形式从体系结构上彻底解决上述问题并成为该领域的研究热点。目前 NoC(Network on chips)上的研究大部分集中在对拓扑结构、路由器和路由算法等各个部件的性能优化方面。

论文的主要工作正是对以上几个方面进行研究的,重点介绍了 NoC 常用的拓扑结构,以及常用的路由技术和路由算法。在研究 Turn Model 模型的基础上,提出一种基于 2D Mesh 结构的 XY-YX 路由算法。该算法是一种确定性的无死锁的最短路径路由算法,并且给出无死锁的证明,最后通过 NoC 模拟仿真实验平台 NIRGAM(NoC Interconnect Routing and Application Modeling),将该算法在一个 4×4 的 2D Mesh 网络中进行了仿真,并对仿真结果进行了分析。

为了将来能在硬件上实现、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,并提出了一个基于 E-cube 路由算法的路由节点模型。该路由节点模型采用了规则二维 Torus 拓扑结构、虚拟通道技术、基于虫孔的数据包交换方式和 GALS (Globally Asynchronous Locally Synchronous) 系统架构。在此基础上给出了片上网络路由节点的功能定义、设计与验证方案、数据包格式定义、子模块划分以及电路设计与验证过程,并对路由节点进行了必要的分析,为将来整个片上网络模型的搭建奠定了基础。

关键词: 片上网络; 路由算法; 死锁; 仿真

The Design and Research of Routing algorithm and Simulation Model on Network on chip

ABSTRACT

As the growing development of semiconductor technology and integrated circuits, more and more IP cores integrated on one single chip. Some problems have become difficult to be solved in designing SoC based on chip bus. Firstly, synchronization of global clock is impossible. Secondly, address space is limited. Thirdly, chip bus can not support multi-node parallel communications. Lastly, the bus system is not flexible enough to be expanded. The quantity of IP cores and system performance are restricted due to the problems above. As a consequence, the technology of computer networks was transplanted into SoC design to solve systematic problems of chip bus and this has been the hot topic in research field. At present most of researches about NoC focused on topology structure, router and routing algorithm.

The main works of this thesis focused on the aspects of above. This thesis mainly introduced the common NoC topology, and the popular routine techniques and algorithms. At the same time, a XY-YX Routing algorithm of a 2D mesh structure is presented, which is based on researching the turn model. The algorithm is deterministic, minimal and deadlock-free. Proof of deadlock freedom is presented. Finally, the algorithm is simulated on 4×4 2D Mesh topology network to evaluate and analyzed the performance through NIGAM experiment platform.

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, we proposed a NoC routing node based on 2D Torus topology, wormhole Exchange mechanism, E-cube routing algorithm, GALS structure and virtual channels. After that, we gave functional definition of the router module, design and verification programs, data packet format definitions, sub-modules division scheme and circuit details. And then we analyzed the routing node, which lay the foundation for future research on NoC.

Keywords: Network on chips; routing algorithm; deadlock; simulation

插图清单

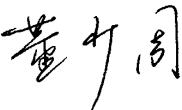
| | |
|--------------------------------------|----|
| 图 1-1 NoC 结构示意图 | 5 |
| 图 2-1 NoC 及通讯节点示意图 | 10 |
| 图 2-2 (A) 规则二维网格结构 (B) 二维折叠环结构 | 11 |
| 图 2-3 二维折叠环结构 | 12 |
| 图 2-4 超级立方体结构 | 12 |
| 图 2-5 BUTTERFLY FAT TREE 拓扑结构 | 12 |
| 图 2-6 基本延时计算示意图 | 15 |
| 图 2-7 (A) 虚拟直通 (B) 虫孔交换 | 17 |
| 图 2-8 虚通道示意图 | 17 |
| 图 3-1 死锁描述 | 21 |
| 图 3-2 2D MESH 节点的通道 | 22 |
| 图 3-3 TURN MODEL 模型 | 23 |
| 图 3-4 XY 路由对应 TURN MODEL 模型 | 23 |
| 图 3-5 XY-YX 算法对应 TURN MODEL 模型 | 24 |
| 图 3-6 节点通道编码 | 24 |
| 图 3-7 4×4 2D MESH 通道编码 | 25 |
| 图 3-8 数据包注入模型 | 26 |
| 图 3-9 一般模式 | 28 |
| 图 3-10 转置模式 | 28 |
| 图 3-11 热点模式 | 29 |
| 图 4-1 4X4 的 TORUS 拓扑结构的 NoC 模型 | 30 |
| 图 4-2 请求应答协议时序图 | 31 |
| 图 4-3 路由器结构 | 33 |
| 图 4-4 X 维子路由内部结构 | 34 |
| 图 4-5 数据流控制状态机 | 38 |
| 图 4-6 输出通道内部结构 | 40 |
| 图 4-7 数据包格式 | 41 |
| 图 4-8 FPGA 设计流程图 | 43 |
| 图 5-1 子模块验证流程图 | 47 |
| 图 5-2 握手协议仿真波形 | 49 |
| 图 5-3 子路由模块数据传输仿真波形 | 49 |
| 图 5-4 芯片面积开销 | 51 |

表格清单

| | |
|--------------------------------|----|
| 表 2-1 几种拓扑结构参数比较 | 13 |
| 表 2-2 四种包交换机制比较 | 14 |
| 表 2-3 片上网络协议堆栈 | 18 |
| 表 4-1 子路由模块部分输入输出信号线功能描述 | 34 |

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志和致谢的地方外，论文中不包含其他人已经发表或撰写过的研究成果，也不包含为获得合肥工业大学或其他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。

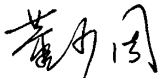
学位论文作者签字：

签字日期：2009年3月20日

学位论文版权使用授权书

本学位论文作者完全了解合肥工业大学有关保留、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，允许论文被查阅或借阅。本人授权合肥工业大学可以将学位论文的全部或部分论文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适用本授权书)

学位论文作者签名：

签字日期：2009年4月20日

学位论文作者毕业后去向：

工作单位：

通讯地址：

导师签名：

签字日期：2009年4月20日

电话：

邮编：

致 谢

在我攻读硕士学位期间，首先我要真诚感谢我的导师欧阳一鸣副教授！欧阳老师在学术研究上给予了我许多悉心的指导和帮助，他教会了我进行研究的方法，帮助我形成了严谨、创新的思维方式，使我从一开始对学术研究的一无所知，到现在的具备独立的学术研究能力。他严谨的治学态度以及勤奋的工作作风将是我一直学习的榜样，他循循善诱的教导和不拘一格的思路将给予我无尽的启迪！在生活上，欧阳老师也给予了我们无微不至的关怀，热情帮助解决遇到的各种困难。在此，我谨向欧阳老师致以崇高的敬意和由衷的感谢！

同时，我还要特别感谢梁华国教授以及系统结构实验室的所有成员！在我的学术研究过程中，他们都给予了我许多宝贵的意见和建议，更快更好的发现自己研究工作中的不足，使我在研究课题上受到了颇多启发。

感谢曾经给予过我帮助的所有老师和同学们！

感谢我的父母这么多年给予我的关爱、支持、鼓励和无私的付出，你们是我一路走来的精神动力，在以后的人生中我也会因此而加倍努力。在此，祝愿你们永远健康快乐！

最后，感谢在百忙之中评阅本论文的专家、教授，感谢你们对本论文的宝贵意见和建议。

作者：董少周

2009年3月

第一章 绪论

1.1. SoC 设计方法学

1.1.1. SoC 的发展概况

20 世纪 90 年代中期,随着半导体工艺技术的发展,集成电路(Integrated Circuits, IC)产业的发展可以用日新月异来形容。在集成电路发展初期,电路设计都是从器件的物理版图入手,后来出现了集成电路单元库(Cell-lib),使得集成电路设计从器件级进入逻辑级,这样的设计思路使得大批电路和逻辑设计师可以直接参与集成电路设计,极大地推动了 IC 产业的发展。但集成电路仅仅是一种半成品,只有装入整机系统才能发挥它的作用。IC 芯片是通过印刷电路板(Printed Circuit Board, PCB)等技术实现整机系统的。尽管 IC 速度可以很高,功耗可以很小,但由于 PCB 中 IC 芯片之间的连线时延、PCB 可靠性以及重量等因素的限制,使整机系统的性能受到很大制约。随着系统向高速度、低功耗、低电压、多媒体、网络化、移动化的发展,系统对电路的要求越来越高,传统的 IC 设计技术已无法满足性能日益提高的整机系统的要求。同时由于 IC 设计与工艺水平的提高,集成电路规模越来越大,复杂程度也越来越高。正是在需求牵引和技术推动的双重作用下,出现了系统芯片设计方法。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。

从广义的角度讲,SoC 是一个微电子小型计算机系统。该系统应包含两个基本部分:硬件部分和软件部分。硬件部分包括 CPU, BUS, ROM/RAM, I/O Port 等计算机系统的基本部件;软件部分主要指操作系统,也可以包括重要的应用软件。从狭义的角度讲,SoC 是电子信息系统的集成;SoC 是指在单一硅芯片上实现信号采集、转换、存储、处理和 I/O 等功能,从而实现一个系统的功能,它是在专用集成电路(Application Specific Integrated Circuits, ASIC)的基础上发展起来的,具有很多不同于 ASIC 的独特优点:SoC 实现了一个系统的功能,速度快、集成度高、功耗低,同时由于 SoC 集成了多个功能,使整机成本和体积都大大降低,加快了整机系统更新换代的速度。SoC 的这些优点正好顺应了通信产品、微型计算机、消费类电子产品向体积小、重量轻和低功耗发展的趋势,对于移动通信、掌上电脑和多媒体产品的生产厂商有很大的吸引力,因此市场对 SoC 产品有强烈的需求。与传统 IC 设计技术和方法相比,SoC 的设计有以下五个不同的特点:

(1) SoC 具有数百万门乃至上亿个元器件的设计规模,而且电路结构包括微处理器、静态存储器、动态存储器、模/数转换器、闪存存储器以及其他模拟和射频电路,要求起点比普通 ASIC 高,不能依靠基本逻辑电路单元作为基础单元,而是采用复杂的知识产权模块。在验证方法上要采用数模混合验证方法,

为了对各模块特别是 IP 模块进行有效的测试，必须进行可靠性测试。

(2) SoC 具有高达数百兆的系统时钟频率，而且各模块内和模块间错综复杂的时序关系，给设计带来了很多问题，如时序验证、低功耗设计、电磁干扰和信号串扰等。

(3) SoC 多采用深亚微米工艺加工技术，总线延迟和门延迟相比变得不可忽视并成为主要因素，再加上复杂的时序关系，增加了电路中时序匹配的困难。深亚微米工艺具有非常小的线间距和层间距，使得线间和层间的信号耦合作用增强，再加上高频效应下的电磁干扰和信号串扰等现象，给设计验证带来很大困难。

(4) 由于芯片上要实现的功能繁多，而且要分别用硬件和软件实现，功能之间的关系也特别复杂，这就需要在更高的设计层次上来进行 IC 设计。

(5) 在 SoC 设计中，硬件和软件是并行设计的。在设计初期，系统中的任何一个功能模块是由硬件还是软件来实现并没有确定，在完成系统功能的定义后，才对系统进行软硬件划分，这时需要运用具有评估技术的软硬件划分工具，分析相关的评估数据，已完成系统硬件和软件的划分工作。

1.1.2. SoC 相关研究技术

在 SoC 的设计过程中，最具特色的是 IP 核复用技术，即选择所需功能的 IP 核集成到一个芯片中。由于 IP 核的设计千差万别，所以 IP 核的互连技术就成为 SoC 设计的关键技术。片上总线是实现 SoC 中 IP 核互连的主要技术方式，它以总线方式实现 IP 核之间的数据通信。片上总线的规范一般需要定义各个模块之间初始化、仲裁、请求传输、响应、发送接收等过程中驱动、时序、策略等关系。与板上总线不同，片上总线不用驱动底板上的信号和连接器，所以使用更简单，速度更快。SoC 设计中 IP 核互连采用片上总线具有如下优点：提供灵活的集成方案，可以针对应用定制；提供不同的总线周期和数据通道宽度；降低设计难度，加快设计周期；允许在系统设计时选择不同的 IP 核，用以降低成本，提高性能和可靠性。

(1) IP 复用技术

在强大的商业压力下，设计公司想要大幅度提高 SoC 的设计效率，就必须尽可能多地使用现有的电路模块，甚至从其他公司那里获得需要的 IP 核，然后再将这些合适的 IP 核通过某种方式“拼装”成符合一定功能需求的 SoC。这就是 IP 复用技术。

IP 核是具有知识产权的专用集成电路，这里将其定义为：经过预先设计、预先验证，具有相对独立的功能，可以重复使用在 SoC 和复杂 ASIC 中的电路模块。在工业界，其又常被称为 SIP (Silicon IP)。通常按照其在设计流程中的位置，IP 核可以分为三种：软 IP 核、硬 IP 核和固 IP 核。

软 IP 核(Soft IP): 可综合 HDL 模、HDL 代码、网表、测试文档 (TESTBENCH), 通常以 HDL 文本形式提交给用户, 它已经过 RTL 级设计优化和功能验证, 但其中不含有任何具体的物理信息。据此, 用户可以综合出正确的门电路网表, 并可以进行后续的结构设计, 具有很大的灵活性。

硬 IP 核(Hard IP): 电路结构掩膜, 布局与工艺确定, 基于半导体工艺的物理设计, 已有固定的拓扑布局和具体工艺, 并通过工艺验证, 具有可保证的性能。提供给用户的形式是电路物理结构掩模版图和全套工艺文件。

固 IP 核(Firm IP): FPGA 电路结构编码文件, 介于软硬核之间, 可按用户要求修改设计程度介于硬核和软核之间, 除了完成软核的所有设计之外, 还完成了门级电路综合和时序仿真等涉及环节, 一般以门级网表的形式提供给用户。固 IP 核提供了介于软 IP 核和硬 IP 核之间的一个折中方案, 比硬 IP 核具有更好的灵活性和可移植性, 比软 IP 核又在性能和面积上有更好的可预知性。

建立在 IP 核重用技术基础上的 SoC 设计使设计方法从电路设计转向系统设计, 设计重心将从今天的逻辑综合、布局布线、后模拟转向系统级模拟, 软硬件联合验证, 以及若干个 IP 核组合在一起的物理设计。IP 核重用技术迫使设计业向两极分化, 一是转向系统, 利用 IP 核设计高性能、高复杂的专用系统。另一方面是设计深亚微米下的 IP 核, 步入物理层设计, 使 IP 核的性能更好并可预测。

(2) IP 核互连技术

在 SoC 设计中, 一个重要的课题就是 IP 核的互连问题, 即当一个 SoC 中需集成几十个, 甚至更多的 IP 核时, 如此多的 IP 核以怎样的方式进行数据交换。IP 核互连的不同形式会影响到 SoC 芯片的数据带宽、时延、数据吞吐率及功耗等指标。为了使 IP 核集成更快速、更方便, 缩短进入市场的时间, 迫切需要一种标准的互联方案。在这一背景下产生了片上总线 OCB (on-chip bus) 技术, 该技术是基于 IP 核互联标准技术发展起来的。片上总线是目前 SoC 设计中广为使用的 IP 核互连方式, 它是通过仲裁和译码的方式来完成不同主、从部件的互连及总线复用, 常用的总线有 ARM 的 AMBA(Advanced Microcontroller Bus Architecture) 总线^[28]、IBM 的 Core Connect 总线^[27]、Silicore 公司的 Wishbone 总线^[29]、Altera 的 Avalon 总线^[31]、Plamch IP 的 Core Frame 总线^[30]等。

片上总线有两种实现方案, 一是选用国际上公开通用的总线结构, 包括 IBM 公司的 Core Connect、ARM 公司的 AMBA、Silicore 公司的 Wishbone 等; 二是根据特定领域自主开发片上总线。由于 Open Core 国际组织和其他致力于开放知识产权组织的大力推广(开发设计大量基于标准化芯片上总线的免费模块), SoC 的设计者在总线的选择上更倾向于采用标准化、开放化方案。采用片上总线的 SoC 设计技术研究重点包括: ①总线结构及互连技术, 直接影响芯片

总体性能发挥；②软、硬件的协同设计技术，其主要解决硬件开发与软件开发同步进行的问题；③IP 核可重用技术，解决如何对 SoC 进行测试和验证的问题；④低功耗设计技术，主要研究多电压技术、功耗管理技术以及软件低功耗利用技术等；⑤可测性设计方法学，研究 EJTAG (enhanced joint test action group) 设计技术、批量生产测试问题；⑥超深亚微米实现技术，其研究时序收敛、信号完整性和天线效应等。

当前 SoC 芯片设计显出两种发展趋势：一是结合自顶向下与自底向上的系统设计方式，二是基于设计平台的设计方式。前一种设计方式在芯片性能、产品成本方面有较大的优势，但设计时间较长，设计成本与风险也都较高，在系统设计中还需用少量的全定制模块，在高性能、低功耗领域有较大的优势；后一种基于平台设计方式最大限度地利用了 IP 核的特点，芯片设计时间较短，风险也低，但可能无法达到最佳的芯片设计技术，因而这种设计方式在芯片上市时间紧迫的情况下非常合适。在未来的 3~5 年内，SoC 芯片的这两种设计方式还将同时并存，而且将在相应的应用领域内取得进展。

1.1.3. SoC 面临的问题

总线结构是 SoC 的主要特征，由于可以提供高性能的互连而被广泛应用。然而随着集成电路制造工艺的持续发展，进入 21 世纪以后，45nm 和 32nm 工艺相继投产，电子元件的特征尺寸步入了纳米时代。在未来若干年内，特征尺寸仍将呈指数性迅速缩小，并且，集成电路的规模将超过几十亿晶体管。从而，在一颗芯片上会出现数十甚至数百个 IP 核联合工作的超复杂电路系统。届时，基于共享总线互连机制的传统 SoC 通信体系结构将遇到无法逾越的障碍。主要表现在以下几方面：

(1)可扩展性差。SoC 系统设计是从系统需求分析开始，确定硬件系统中的各功能模块。为了使系统能够正确工作，SoC 中各物理模块在芯片上的相对位置是一定的，总线和时钟网络以及其他全局信号走线都是针对这一具体需求进行专家级的设计，从而才使得 SoC 芯片能够在深亚微米效应下正确工作。一旦在物理设计完毕后，个别错误或不合理设计的纠正错误过程，就可能是再一次的重新设计的过程。另外，如果设计需要增加某个功能模块时，可能会打乱原来的物理布局，新的物理布局可能导致必须重头设计新的系统，延缓了投放市场的时间。

(2)平均通信效率低。SoC 中多采用总线结构，在一个时间段内只能有一个功能模块独占总线，其他各功能模块只能在获得总线控制权后才能和系统中其他模块进行通信。从通信的局部看，这种结构的通信带宽也很高，实时性很好。但从系统整体考虑，这种通信结构的平均通信带宽是非常低的，因为一个模块控制总线通信时，系统中其他模块必须等待，直到得到总线控制权。

(3)单一时钟同步问题。总线结构要求全局同步，但随着工艺特征尺寸越来越小，工作频率迅速上升，达到 10 GHz 以后，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。

1.2. NoC 的产生背景

鉴于上述原因，在 90 年代末，一些研究机构借鉴和吸收了计算机通信网络中的一些思想，提出了一种全新的互连结构——片上网络(Networks on Chips, NoC)。NoC 技术从体系结构上彻底解决了 SoC 的总线结构所固有的三大问题：由于地址空间有限而引起的可扩展性问题，由于分时通讯而引起的通讯效率问题，以及由于全局同步而引起的功耗和面积问题。NoC 采用全局异步-局部同步 GALS 的通讯机制，提供了良好的并行通讯能力，使得 NoC 成为面向纳米工艺的新型体系结构。

NoC 可以定义为在单一芯片上实现的基于网络通信的多处理器系统。图 1-1 为典型的 NoC 结构示意图。NoC 包括计算和通信两类节点，计算节点(又称资源节点)完成广义的计算任务，它们可以是 SoC，也可以是各种单一功能的 IP；通信节点负责计算节点之间的数据通信。通信节点及其之间的网络称为 OCN，它借鉴了分布式计算机系统的通信方式，用路由和分组交换技术替代传统的总线技术完成通信任务。

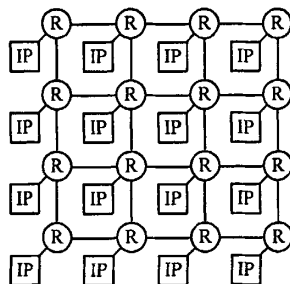


图 1-1 NoC 结构示意图

NoC 与 SoC 有两点本质的不同，即处理器数量和通信机制，前者是多处理器系统，而后者是单一处理器系统；前者使用网络通信，而后者使用总线通信。片上网络体系结构克服了 SoC 所面临的种种问题，它是一个支持芯片上内核间互连通信的结构，即定义了资源的路由通信及点到点通信互连的结构集合。NoC 具有良好的地址空间可扩展性，理论上可集成的资源节点的数目不受限制；其次提供良好的并行通信能力，从而提高数据吞吐率及整体性能；NoC 体系结构采用了消息传递的通信模型，通过发送请求消息和接收响应消息将资源连接到网络通信中。

NoC 之所以可以在很大程度上提高设计能力，主要是因为它采用全局异步局部同步的工作方式、通信框架可复用以及可复用的 IP 核技术。全局异步局部

同步设计方式可以在一定程度上降低由于全局同步所面临的困难。全局异步局部同步就是把芯片人为的划分成很多个域，每一个资源节点都工作在特定的时钟域，域和域之间采用异步通信方式，这种全局异步局部同步的工作方式很好地解决了总线结构的单一时钟同步问题，从而彻底解决了 SoC 中总线架构庞大的时钟树所带来的功耗和面积问题。NoC 和 SoC 均是以 IP 核复用为基础的芯片设计方法。即在芯片上采用可重用的通信网络框架，同时利用可复用的 IP 核来实现可扩展性的一种硬件系统，并且这样的系统是以通信为基础，其通信框架可跨工艺。但是 NoC 的可扩展性、可测试性与 SoC 相比要更强。

1.3. NoC 的国内外研究现状

随着 NoC 体系结构和设计方法的提出，NoC 的相关学术研究已经逐渐展开。如拓扑结构、路由算法、路由器设计、NoC 测试和 NoC 功耗研究等等。

1.3.1. 国外研究概况

在学术界，NoC 从 1999 年第一次被提出，经过十年多的发展，理论研究进入高峰阶段，并已有了相当的成果。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。其中影响较大的有瑞典皇家技术学院的 NOCARC 项目、斯坦福大学的 Netchip^[1]项目以及 Manchester 大学的 Marble 项目等。2000 年，Guerrier 等人率先开展了片上多核系统研究，他提出了一种基于“胖树”（fat-tree）拓扑结构的多核片上互连网络架构 SPIN，并建立了周期精确的路由器性能模型。次年，Sgroi 等人在讨论基于平台的 SoC 的设计方法学时，提出了基于网络通讯的多核系统的理念^[2]。同年，Benni 等人发表概念性论文，提出基于报文（packet）的片上互连网络结构。Kumar 等人则提出一种基于二维网格的片上网络结构，这种结构允许继承大型资源，如大面积存储单元、FPGA 器件以及高性能多处理器等^[3]。此后，该领域中的研究成果日益增多，已经初步形成一个理论体系。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、测试等性能优化、可靠性方面的研究；同时从 2007 开始，IEEE 就组织了一个专门进行 NoC 学术研究的国际性论坛 NoCS（The IEEE International Symposium on Networks-on-Chip），供国际上所有的学者一起来探讨和研究 NoC，该论坛每年举办一次，目前已经举办了两届。

1.3.2. 国内研究概况

在国内方面，从相关的搜索中我们可以看到，从 2005 年开始国内就有一些院校的研究单位开始对 NoC 有了初步的研究，但大多都处于初步阶段，成果数

量之少，和国外的研究成果相比较还有很大的差距。但是目前的研究形势还是好的，而且国内已经立项研究 NoC 相关的理论技术了。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。值得庆幸的是，从搜索的大量文献中我们发现，国外在 NoC 的研究方面也处于起步阶段，所以这正是我们与国际上 NoC 研究保持同等起点的好机遇，另外国内在多核集成电路制造方面也有了比较大的进步，国内的研究人员正在全力打造 Godson（龙芯）的第一个多核版本，它将成为中国首个自行研发的多核微处理器，具有四到八个核，预计在未来的几个月里完成流片。中国希望在 2010 年完成基于 Godson CPU 每秒钟可完成千万亿次浮点运算的高性能计算机。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，随着国家对 NoC 相关技术的理论研究工作的立项支持，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。

NoC 不仅在学术研究中很受关注，从产业界来看，自 2001 年 IBM 发布其首个双核产品 Power4 开始，世界各大巨头纷纷把目光聚集到多核技术，陆续推出其多核产品。如 Intel 推出的 Core 2 Duo，AMD 的 AMD Athlon64 FX 均为双核处理器，SUN 推出了含有 8 个处理器核的 Niagara，STI(SONY、TOSHIBA 和 IBM)联合开发的 CELL 则包含 9 个处理器核^[4]，ARM 公司的 MPCORE 也包含 4 个 ARM11 处理器。NoC 将是集成电路未来发展的必然趋势。

1.4. 课题来源、创新点及文章结构安排

1.4.1. 课题来源及创新点

本论文的研究工作受到以下项目的资助：

国家自然科学基金重大研究计划，（项目编号：90407008）

国家自然科学基金重点项目（项目编号：60633060）

安徽省自然科学基金的资助（项目编号：050420103）

本论文在对 NoC 的路由算法问题以及路由器设计等问题展开研究和分析，本课题的研究内容主要分为两个方面，第一方面是在基于 2D Mesh 拓扑结构的基础上，寻找一种高性能的确定性无死锁的路由算法。它是通过在研究 Turn Model 模型的基础上，寻找一种无死锁的路由算法，并且在 NIRGAM 仿真平台上进行了仿真，与 XY 路由算法，和 OE 算法进行比较，结果显示了良好的性能。第二个方面是基于硬件描述语言 Verilog HDL 的 NoC 的硬件平台的路由节点设计，为将来的 NoC 硬件仿真平台的搭建以及完善奠定基础。该仿真模型的设计主要是采用 wormhole 的交换机制，E-cube 路由算法，最终的仿真是在

Altera Stratix III 器件上进行的。

在本论文研究的方法中，主要的创新点如下：

(1)基于 Turn Model 模型设计的思想设计一种无死锁的确定性路由算法，该算法可以避免传统 XY 路由算法在同一个方向上传输时存在堵塞的情况，从而提高了通信的吞吐量。该算法的思想是数据在到达每个节点时，并不是总是沿着先 X 方向传输，再 Y 方向传输（这种情况容易在 X 方向产生堵塞），而是采取判断的方式，当目的节点在在当前节点的北边时采用 XY 路由算法，当目的节点在当前节点的南边时，采用 YX 路由算法，通过证明，该算法是无死锁的。

(2)设计 NoC 路由节点的具体实现如下：首先拓扑结构是采用 2D Torus 结构，方案初期准备设计二到四个 IP 核的芯片，其次数据交换机制采用虫孔交换技术，路由算法采用 E-cube 路由算法，节点标识采用 2 进制格雷码的编码方式进行表示，这样更有利于硬件的快速实现。

1.4.2. 论文结构及内容安排

在对国内外研究成果的分析过程中，我们发现 NoC 具有很广阔的发展前景，本论文拟在对 NoC 研究中比较热点的两个问题，路由算法问题以及路由器设计等问题展开分析和研究，论文分为五章，具体安排如下：

第一章绪论。简要介绍了集成电路发展设计相关知识，从系统芯片设计 IC 设计技术到 SoC 技术的相关背景知识介绍以及 SoC 发展所面临的问题，NoC 的产生背景等等。同时，简单介绍了 NoC 的国内外研究概况，并对本论文的课题来源、创新点和内容结构安排做了介绍。

第二章 NoC 体系结构概述。该部分内容主要介绍了 NoC 的基本定义以及拓扑结构、路由算法以及 NoC 的相关研究技术问题等。

第三章基于 2Dmesh 结构的 NoC 路由算法设计与仿真。本章在基于 Turn Model 模型研究的基础上，提出了一种基于 XY 路由算法的改进的 XY-YX 路由算法，并将该算法通过 NIRGAM 平台进行仿真，最后将仿真结果与其他常用的路由算法进行了详细的分析和比较。

第四章基于 FPGA 的 NoC 路由节点设计。主要介绍基于 Verilog HDL 语言的 NoC 的路由节点的设计，详细介绍路由节点组成模块，以及各模块的功能和各模块的具体实施方案等。

第五章路由节点的仿真验证及性能分析。该章节是对前一章节设计的路由节点的测试和验证，首先提出测试方案，然后给出测试验证结果，最后在速度和面积开销上给出参考分析。最后是总结与展望部分，对本文进行了简要总结，并提出了今后要进行的进一步工作。

第二章 NoC 体系结构概述

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。NoC 技术虽然移植了计算机网络中的关键技术,但是由于通信媒介存在着根本差异以及纳米级工艺条件下芯片设计的特定需求,使得 NoC 在以下几个方面与传统计算机网络之间存在着明显的不同,即连线资源远较计算机网络丰富;流量分布函数的差别(传统计算机网络的流量服从泊松分布);资源节点的异类性(NoC 中的计算节点的功能可以从单一功能的 IP 核到整个 SoC);显著的低功耗需求(纳米级工艺条件下任何芯片都无法回避的最重要的问题之一)。另外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^[44]。而较复杂的路由算法,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-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制协议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. NoC 的定义

NoC 目前处于初创阶段,不同的研究机构和研究人员对 NoC 虽然有不同的定义和理解,但是有一点是相同的:借鉴并移植计算机网络通讯中的概念和方法,用于多个子系统(可以理解为现有规模的 NoC)或 IP 核集成。

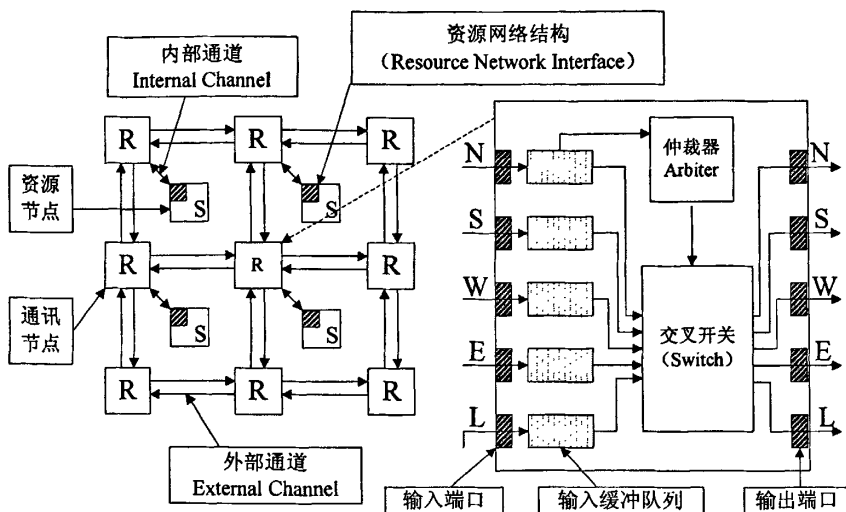


图 2-1 NoC 及通讯节点示意图

图 2-1 是 NoC 的概念示意图, 从图中可以看到, NoC 由资源节点(Resource Node)^[5], 通讯节点(Communication Node)、通道(Channel)和资源网络接口(RNI)组成:

(1)资源节点: 执行计算任务的节点。资源节点既可以是 SoC, 也可以是各种特定功能的 IP 模块。典型的资源节点可以是带缓冲的嵌入式微处理器和 DSP 核、专用硬件资源, 可重构硬件资源, 或者是上述各种硬件的组合。

(2)通讯节点: (又称路由节点, Router) 负责计算节点之间的数据通讯, 通讯节点及其之上的网络称为片上通讯网络 (On-chip Communication Network, OCN), 其核心包括链路控制器、仲裁器、交叉开关、缓冲器、输入输出通道等^[40]。它借鉴了分布式计算机系统的通讯方式, 用路由和分组交换技术替代传统的总线技术完成通讯任务。通讯节点的功能就是将信息从它的输入端口传输到其中的一个或多个输出端口。

(3)通道: 指资源节点和通讯节点之间, 通讯节点和通讯节点之间的连线, 包括外部通道和内部通道。外部通道指通讯节点和通讯节点之间的连线, 内部通道指资源节点和通讯节点之间的连线。

(4)资源网络接口 (RNI): 指资源节点和通讯节点之间的接口, 只有配置了资源网络接口的资源节点才能连接到网络上与其他资源节点进行通讯。

2.2. NoC 的拓扑结构

拓扑结构关心的是节点的布局 and 互连。NoC 拓扑结构的选择对系统性能和芯片面积具有明显的影响。由于通信需求、节点模块的尺寸、芯片封装等因素的不同, 片上网络需要根据实际情况采用不同类型的拓扑结构。主要有 Mesh^[6]、Torus^[7]、二维折叠环^[8]、超立方体、k 元 n 维立方体^[9]、Fat-Tree^[10]、Octagon^[11]、Spidergon^[12]、SPIN^[13]等等。Mesh 和 Torus 是规则的 NoC 拓扑结构, 具备了

硬件实现简单、网络扩展性好等方面的优点，因此它们一般做为 NoC 最为常用拓扑结构。Torus 结构是 Mesh 结构的一种改进，将 Mesh 结构中每个行、列的首、尾节点进行互连，从而消除了 Mesh 结构的边界效应。二维折叠 Torus 拓扑结构是由斯坦福大学 W.J.Dally 提出的片上互连网络的一种结构，它的任意节点之间的连线距离相等，使得各节点在网络中的特性相同，更易于在片上硬件实现。

以下具体介绍几个在当前 NoC 研究领域中经常被采用的拓扑结构：

(1) 规则的二维网格结构(Regular 2-D Mesh)

规则的二维网格是片上互连网络研究中最常用的拓扑结构，如图 2-2 (a) 所示，在这种结构中每个资源节点和一个通信节点相连，而每个通信节点和四个通信节点（边界节点除外）以及一个资源节点相连，通信节点实现路由功能。二维网格结构简单，具有良好的规则性，扩展性好，但是其带宽、延时等性能上却不是最优的。

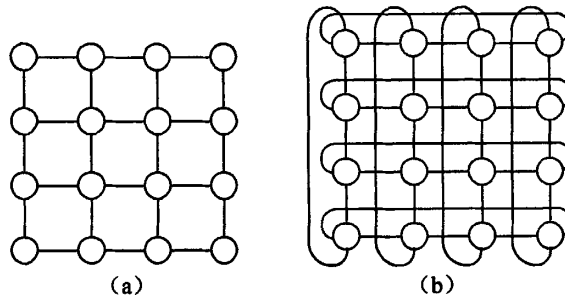


图 2-2 (a) 规则二维网格结构 (b) 二维折叠环结构

(2) 二维环绕网络结构(2-D Torus)

二维环绕拓扑结构，如图 2-2 (b) 所示，与规则二维网格结构相比，它将处于边界的通信节点也连接起来，使所有的通信节点成为一个环路，从而增加了通信路径，降低了网络出现阻塞的概率。但是从图中也可以看出，环路和环路之间有交叉的地方，这样不可避免地在物理实现时需要更多的版图布线资源。

(3) 二维折叠环结构(2-D Folded Torus)

为了解决 2D Torus 中由于过长环形信道而产生额外延迟的问题，W.J. Dally 和 C.L. Seitz 提出 2-D Folded Torus^[14]结构，如图 2-3 所示。从图中我们可以看到，与 2D Torus 不同的是：在 Torus 中，边缘节点通过环形链路连接在一起，而在 Folded Torus 中，将环形链路进行扩展，缩短了链路的长度，该结构通过改变交换结点的连接方式，解决了由于过长的环形信道产生额外的延迟，提高了系统的整体性能。

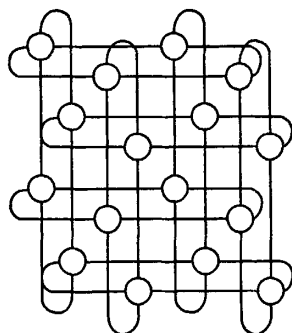


图 2-3 二维折叠环结构

(4)超立方体结构(Hypercube)

超立方体结构是 n 维网格与 k 元 n 立方的特例。如图 2-4 所示的 4 维超立方体结构，在互连网络的拓扑结构研究的初期就出现了超立方体结构。该结构已经应用与很多方面的应用，尤其是在早期的并行机中。比如 Cosmic cube^[15], Ncube1/2/3^[16]。这一结构具有比较小的网络直径和简单的路由算法。它支持很多置换拓展，例如带环网格、树和其它不同网络结构可以嵌入到 Hypercube 中。

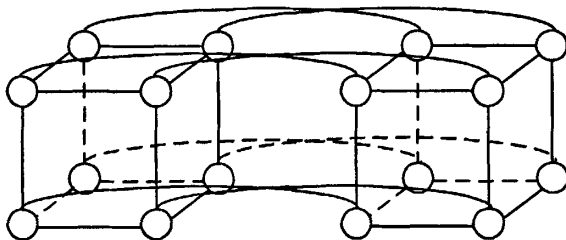


图 2-4 超级立方体结构

(5)BFT 结构 (Butterfly Fat Tree) ^[17]

BFT 用于分级的 NoC 通信系统中。相对于规则拓扑，不规则拓扑结构虽然能提高性能、降低功耗、减小面积，但同时可能产生版图设计、不均匀的线长等设计问题。BFT 结构在源节点到目的节点间提供了多条数据路径，它可被看作是一个扩展的多个根节点的 N 列树型网络，其网络延时依赖于树的深度。随着 BFT 节点的增加，其节点的复杂性也随着增大，对引脚的需求也增加。从某一方面理解，胖树结构已经超出了片上网络对物理的限制。如图 2-5 所示。

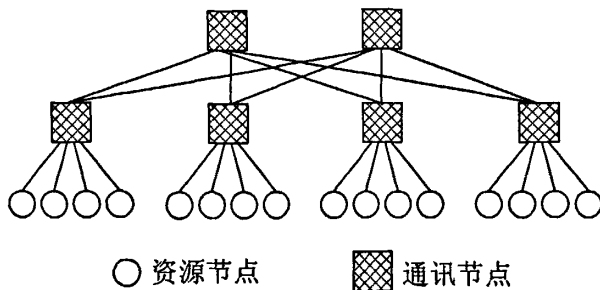


图 2-5 Butterfly Fat Tree 拓扑结构

拓扑结构的衡量标准通常是以理论上影响路由成本和性能为基础的，除了要考虑普通网络中所关心的节点数量、边的数量、网络维度、网络直径、平均距离、对分宽度之外，还要考虑通信模式的嵌入属性，例如消息吞吐量、传输延迟、功耗、芯片面积等。表 2-1 给出上述几种常用 NoC 拓扑结构的性能参数。

表 2-1 几种拓扑结构参数比较

| 网络拓扑 | IP 核数 | 交换节点数 | 交换节点度 | 网络直径 | 链路数 |
|------------|---------|-------|-------|-----------------------------|-----------------|
| Mesh | N | N | 5 | $2(\sqrt{N}-1)$ | $2(N-\sqrt{N})$ |
| Torus | N | N | 5 | $2\lfloor\sqrt{N}/2\rfloor$ | 2N |
| Fold Torus | N | N | 5 | $2\lfloor\sqrt{N}/2\rfloor$ | 2N |
| Hypercube | $N=2^n$ | N | n | n | nN/2 |
| BFT | $N=2^n$ | N/2 | 6 | N/8 | 2N |

在众多拓扑结构中，从特征参数看，规则的二维网格结构并不是最优的，但是因为其结构简单、易于实现和良好的重用性，大多数片上网络研究项目都采用二维网格作为基本的拓扑结构。当然在实际设计片上网络的过程中，拓扑结构还会受到通信事务的不均匀分布等因素的影响，所以对拓扑结构的研究是 NoC 的主要研究内容之一。

2.3. NoC 的路由算法

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。当具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，路由算法对评价 NoC 网络性能的好坏起着至关重要的作用。评估和优化的过程中除了考虑路由算法的连通性、自适应性、死锁和活锁的发生率、容错能力等基本性能外，更加关注路由算法在时间开销、硬件实现的复杂度以及资源开销方面的情况。路由算法可以按照不同的标准分为不同的几类。比如确定性路由（Deterministic Routing）和自适应性路由（Adaptive Routing）、部分自适应性路由算法等。

2.3.1. 确定性路由（Deterministic Routing）

确定性路由是一种常见的路由算法，它的路由路径只与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。而在确定性路由算法中，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，每个数据包每次只在一个维上路由，当在该维上到达了目的坐标之后，才按由低维到高维的顺序在其他维上路由。因为数据包是按照严格单调的维数变化的顺序在通

道内路由，所以维序路由在维序间是没有死锁的。按照在不同拓扑结构，维序路由包括了 2D Mesh 中的 XY 路由和超立方体(Hyper Cube)中的 E-cube 路由。确定性路由算法的优点是，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。

2.3.2. 自适应性路由 (Adaptive Routing)

所谓的自适应路由，是指数据包的路由路径选择不仅与起点地址和终点地址有关，还与整个网络的通信状态有关。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，它们的路由路径可能是不同的。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；能很好地适应网络状态的变化，是一种具有容错功能的网络路由算法。例如，网络中某节点不可用时(可能发生故障了)，包就绕开该节点，沿着其它的路径传输。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路由算法，如带有一定自适应性的维序路由算法。

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。而这里提到的带有一定自适应性的维序路由算法指的是，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。

2.4. NoC 包交换机制

与路由相关的一个问题是数据交换技术，数据交换技术决定了内部开关连接路由器的输入和输出端口的时机和方式，以及消息可以沿着这些路径传输的时间。不同的交换技术会对芯片设计面积和性能有着不同的影响。常用的交换技术包括电路交换、报文交换、虚拟直通交换、虫孔交换^[35]等。表 2-2 是几种交换机制的参数比较。

表 2-2 四种包交换机制比较

| 交换方式 | 电路交换 | 报文交换 | 虚拟直通交换 | 虫孔交换 |
|-------|------|------|--------|------|
| 面向连接否 | 是 | 否 | 否 | 否 |

| | | | | |
|----------|-----|-------|-------|--------|
| 提供服务类型 | 有保证 | 尽最大努力 | 尽最大努力 | 尽最大努力 |
| 分组乱序否 | 否 | 是 | 是 | 是 |
| 信道利用率 | 低 | 高 | 高 | 高 |
| 容错性 | 低 | 高 | 高 | 高 |
| 缓冲器大小 | 无 | 分组 | 分组 | 微片 |
| 拥塞时分组的缓冲 | 无 | 本地节点 | 本地节点 | 相关若干节点 |

为便于比较，我们对每种交换技术计算基本延时，它指 L 位消息在没有任何堵塞条件下通过网络所需的延时。假设 (flit, flow control unit) 微片，与物理通道宽度相同，都为 W 位，网络一次传输一个微片。路由头为 1 个微片，故消息总大小为 L + W 位。工作频率为 B 赫兹，即带宽为每秒 BW 位。假设连线很短，忽略连线延时，则物理通道的传播延时为 $t_w=1/B$ 。路由器路由时间为 t_r ，路由仅对路由头进行，之后的数据跟着路由头开辟的通道前进，数据交换时间为 t_s 。假设源处理器和目的处理器间相隔 D 段链路。图 2-6 给出了各个参数之间的关系，R 表示路由器。

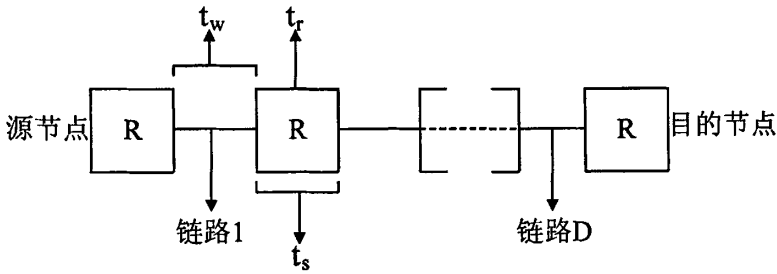


图 2-6 基本延时计算示意图

电路交换。它指在消息传输之前要在源节点和目的节点之间预订一条物理路径，这是通过向网络中注入路由头微片实现。当路径建立后，消息就可以沿着这条路径进行传输。路径是独占使用的，即路径建立后在释放前不能被其它微片使用，这使得其通道利用率不是很高。它的基本延时为：

$$t_{\text{circuit}} = D[t_r + 2(t_s + t_w)] + \frac{1}{B} \left\lceil \frac{L}{W} \right\rceil$$

报文交换。也叫存储转发交换，报文交换将消息划分为固定长度的报文，每个报文的头部包含路由和控制信息。每个报文从源节点到目的节点独立路由，报文在向下一个节点转发之前缓存在中间节点里。报文交换不需要在传输中独占一条物理路径，因此相对电路交换能提高链路利用率。报文交换每个中间节点都需要较大的缓存来存储整个报文，其资源需求量较大。它的基本延时为：

$$t_{\text{packet}} = D[t_r + (t_s + t_w) \left\lceil \frac{L+W}{W} \right\rceil]$$

虚拟直通交换。虚拟直通交换同样将消息分为报文。与报文交换不同，虚

拟直通交换并不等待整个报文接收到后才转发数据，而是一旦接收完数据头完成路由决策且输出通道空闲就直接跨接到下一级路由器的输入端。当没有阻塞时报文流水地通过每个路由器，而当网络负载很大时就等同于报文交换。整个数据包都存储在阻塞发生的节点缓存里。这样，每个节点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。它的基本延时为：

$$t_{\text{vct}} = D(t_r + t_s + t_w) + \max(t_s, t_w) \left\lceil \frac{L}{W} \right\rceil$$

虫孔交换。报文交换与虚拟直通交换都要缓存整个报文，使路由器需要很大的缓存空间。虫孔交换重点解决空间问题，它将消息分割为微片，每个路由器仅缓存几个微片，每个包的头微片包含控制路由信息，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，各个微片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。但当消息被阻塞时消息可能占据多个路由器的缓存器，使网络死锁问题更复杂。它的基本延时为：

$$t_{\text{wormhole}} = D(t_r + t_s + t_w) + \max(t_s, t_w) \left\lceil \frac{L}{W} \right\rceil$$

虫孔交换方式中，一个包可能占有几个中间节点开关。虫孔交换减少了包传输中的存储转发等待时间。当头微片到达一个开关，如果目的端口可用，头微片将保存这个端口，接下来的微片将通过保存的端口流过，直到尾微片，最后释放端口。虫孔交换的一个主要优点是它在等待头微片路由到下一阶段时不要求在开关中存储整个分组。它不仅减少了每个开关的存储转发延迟，而且要求更少的缓冲空间。在交换技术中，虫孔交换对典型的 NoC 应用来说是最有前途的技术，因为 NoC 有有限的缓冲资源和严格的延迟要求。

从上面的描述中可以看出，虚拟直通方式和虫孔交换方式是很相似的。事实上，在没有阻塞的情况下，虚拟直通方式和虫孔交换方式是完全一样的。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；而虫孔路由是将各个片段分散到各个节点中，如图 2-7 所示。

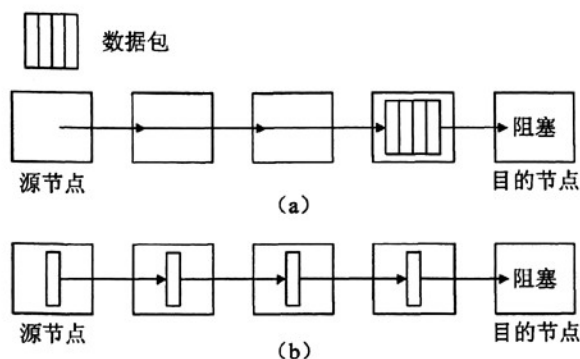


图 2-7 (a) 虚拟直通 (b) 虫孔交换

对片上互连网络而言，很重要的特性就是资源非常有限。考虑到虫孔交换需要最少的缓存空间，同时具有较好的基本延时特性，非常迎合 NoC 的可升级性要求和特点，在很多文献中多选择虫孔交换作为基本交换方式^[36]。

2.5. 虚通道技术

虚拟通道 (Virtual Channel) 技术的出现原本用于防止计算机网络堵塞造成的死锁，也可以用于降低通信延迟，提高网络吞吐率。我们将其引入片上网络，提高片上网络路由单元的性能。

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，在采用了虫孔路由的网络中引入虚拟通道，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。

在虫孔路由网络中引入虚拟通道技术，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。

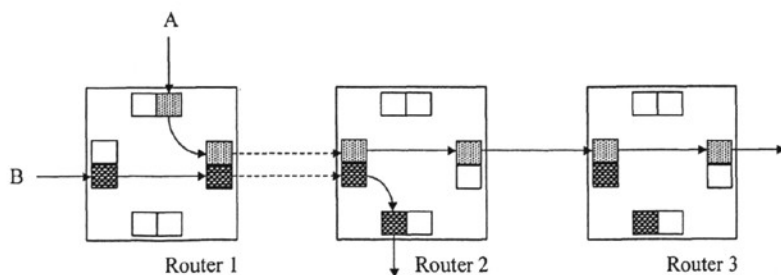


图 2-8 虚通道示意图

多个虚拟通道中的数据包可以共享一个物理通道，如图 2-8 所示，如果没有虚拟通道，数据包 A 堵塞会阻止数据包 B 前进，直到数据包 A 传输完成。采用虚拟通道技术后，两个数据包分时复用物理通道，从而使两个数据包都得

以前进，实现多路发送。两条虚拟通道所能提供的带宽理论上是不共享物理通道时的一半。这样使得数据包在转发过程中被阻塞的时间减少，每个消息的总延迟降低。共享物理通道带宽在以下两种情况下特别有利。第一种情况是数据包 A 在当前节点被阻塞，数据包 B 仍然可以完成传输，如果没有虚拟通道则两条消息均被阻塞；另一种情况是数据包 A 比数据包 B 长很多（如数据包 B 为系统消息包），消息 B 仍然可以利用 1/2 带宽传输，然后 A 利用全带宽传输。可以想象，增加虚拟通道的数量，应该能进一步减少消息堵塞的概率，提高片上网络数据吞吐率。但是由于片上网络和计算机网路在系统开销成本上的容忍度是不一样的。进一步增加虚拟通道的数量会消耗更多的片上资源，同时增加虚拟通道控制电路的复杂度，降低每条虚拟通道实际分配到的带宽。在我们对基于规则二维网格拓扑结构的片上网络研究过程中，发现虚拟通道的数量超过 4 以后，其对片上网络性能提高不明显。

2.6. 网络协议

在片上网络中，通信协议比总线协议要复杂得多，为了便于扩展，同计算机网络一样，NoC 也采取分层协议的设计思想借，借鉴通用网络的 OSI 协议堆栈模型，协议的每一层提供特定的功能和接口。OSI 模型只是定义了抽象的网络层次，并没有限制如何实现，而且很多已有网络协议也没有完全遵守该模型，比如 WAN、LAN、SAN。Benini.L 和 De Micheli.G 提出了一种片上网络协议堆栈模型^[18]，如表 2-3 所示，最低层是物理层，中间三层定义片上网络系统结构，其它层定义软件协议。中间三层组成了基础结构并向终端节点提供通信服务，软件层包括系统程序和应用程序。

表 2-3 片上网络协议堆栈

| | |
|----------|-------|
| 软件层 | 应用程序 |
| | 系统程序 |
| 片上网络系统结构 | 传输层 |
| | 网络层 |
| | 数据链路层 |
| 物理层 | 物理层 |

现有的片上网络研究通常只涉及到物理层、数据链路层和网络层协议，分别描述如下：

(1) 物理层

定义连接资源节点和通讯节点的连线的数目和长度、电压级别、信号时序等。全局连线是 NoC 中信道的物理实现。物理层的设计在提供一个令人满意的品质和为其他协议层提供一个清晰的、完整的信道属性的抽象之间达到平衡。

(2) 数据链路层

定义资源节点和通讯节点以及两个通讯节点之间的传输协议，保证物理连

接之间可靠的信息传输。数据链路层协议必须提供一定程度的可靠性。对数据打包可以有效的解决通信错误，包边界包含检错信息，可以以包为单位进行错误恢复。数据包在逻辑上由三部分组成：包头、有效负载和包尾。包头包含路由和控制信息，当数据包到来时，路由器和网络接口能根据它决定对数据包做什么操作。有效负载包含网络上传输的数据。包尾包含消息进入链路时生成的差错校验码。

(3) 网络层

在多通信通道的网络结构中，该层负责实现端对端的传输控制。主要研究数据包如何在网络中传输，分为交换算法和路由算法，前者决定建立连接的类型，后者决定数据传输的路径。

第三章 基于 2D mesh 结构的 NoC 路由算法设计与仿真

NoC 不仅具有良好的空间可扩展性, 还提供了很好的并行通讯能力, 从而提高数据吞吐率及网络整体性能。目前拓扑结构和相关的路由算法的研究是 NoC 相关研究当中的比较重要的两个方面。在 NoC 中路由的选择策略取决于 NoC 特殊的网络结构以及应用需求。

3.1. 问题的提出

拓扑结构, 总体说来 NoC 的结构比较整齐, 通讯节点与资源节点或者通讯节点之间的通讯带有明显的本地相邻特征; 作为片上系统, NoC 有严格的资源限制、延迟约束和功耗要求, 在这些方面确定性路由相比自适应路由有着巨大的优势。但是确定性路由没有考虑网络实际负载情况, 不能根据网络实时变化对路由做出调整, 当网络在一段时间内出现拥塞, 引起通道竞争时, 就会增加数据包的传输时间, 浪费带宽和系统资源。

应用需求与计算机网络或并行系统不同, NoC 通常只支持一个应用或是同一类型的应用。因此设计者可以在很好地在了解应用的通讯特性的基础上, 结合拓扑结构较好地分配传输路径, 从而最大程度地避免通讯拥堵。二维规则 Mesh 网络结构是 NoC 中最基本也是最重要的拓扑结构, 针对这种结构下 NoC 的路由问题, 已经发展出很多相应的路由算法, 可以分为确定性路由和自适应路由。最简单的路由算法是确定性路由 XY 路由算法。XY 路由是最短路径路由, 每一个数据包首先在 X 维上接近目的节点, 然后从相对的另一维上路由到目的节点。在自适应路由中主要有静态限制类算法, 其中最具有代表性的为基于 Turn Model 模型。

本文即是在研究 Turn Model^[19]模型的基础上, 提出一种基于 2D Mesh 结构的 XY-YX 路由算法。该算法是一种确定性的无死锁的最短路径路由算法, 并且给出无死锁的证明, 最后通过 NoC 模拟仿真实验平台 NIRGAM^[20], 将该算法在一个 4×4 的 2D Mesh 网络中进行了仿真, 并与 XY 路由算法以及 minimal OE (odd-even)^[21]路由算法进行了比较, 结果显示在转置模式和热点模式下具有良好的性能。

3.2. NoC 相关研究

对于确定拓扑结构的 NoC 网络, IP 核之间的数据通信方式会极大的影响网络性能, 因此路由算法是 NoC 网络通信研究中的关键问题。评估和优化的过程中除了考虑路由算法的连通性、自适应性、死锁和活锁的发生率、容错能力等基本性能外, 更加关注路由算法在时间开销、硬件实现的复杂度以及资源开销方面的情况。

根据不同特征，路由算法的分类也各不相同。在前面章节中曾提到路由算法可以是确定性的或自适应性的。确定性路由算法是一种静态路由算法，它在源和目的结点间定义了唯一的一条路由通路，如 XY 路由算法和 e-cube^[22]路由算法，这些算法简单且容易实现，而自适应性路由算法则在源和目的结点间提供了多条路径。自适应性是一种良好的特性，它增大了数据包绕开热点 (hot spot) 及故障结点的概率，但是硬件开销很大。目前大多数基于 FPGA 或 ASIC 的 NoC 模型都采用确定性路由算法。

路由算法也可以分为最短路径路由算法和非最短路径路由算法，最短路径路由算法只选择最短路径，因而可以降低时延。而在一些特殊情况下，如存在故障结点时，就有必要采用非最短路径路由策略，但是容易产生活锁。活锁和死锁是设计路由算法时需要考虑的两个关键因素。

活锁是许多自适应路由方案中潜伏的问题。当一个分组以循环的方式围着它的目的地一直运行下去，就产生了活锁。它是指数据包一直在网络中传输总也到达不了目的结点的情况，通过采用最短路径算法即可避免活锁的产生。

死锁是指网络中数据包 A 在结点 1 的缓冲区申请结点 2 的缓冲区，而此时结点 2 的缓冲区被数据包 B 所占用，而数据包 B 又在同时申请结点 1 中数据包 A 所占用的缓冲区，两者都不释放自己所占用的资源，且相互等待对方释放资源，结果产生资源依赖环，AB 都被阻塞，即产生了死锁，如图 3-1 所示。

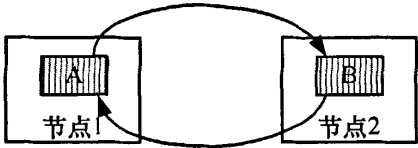


图 3-1 死锁描述

我们在前一章节根据片上网络的特性，选择了相对优秀的数据交换方式——虫孔交换方式。因为往往片上网络中的存储资源非常有限，如果采用报文交换或者虚拟直通都不符合片上网络的特征。但是虫孔交换方式中死锁的问题更容易发生，一个分组可能同时占有几个中间开关，分组可能以循环的形式相互阻塞从而所有的包都不能前进，产生死锁。

打破死锁的方法一个是在设计路由算法的时候就破坏分组之间的循环依赖关系，另一个是使用虚拟通道。维序路由是解决死锁的一个办法，分组总是首先在一维传输(如列优先)，根据到达目的节点的行或列，然后交换到其它维，直到最后到达目的节点。维序路由是确定性的路由，分组对同一源目的节点的路由，总是遵循同一路由。因此不能避免竞争，当发生竞争时，分组需要等待通道空闲。另一个解决死锁的方法是使用虚拟通道。在这种方法中，物理通道被划分为几个虚拟通道。虚拟通道可以解决死锁问题，同时获得高性能。这种

方法的缺点是每个虚拟通道的等待队列需要大的缓冲空间，另外，虚拟通道仲裁也会使电路设计变得复杂，增加系统功耗。而还有一种则是通过检测，发现死锁的产生，再强制破坏资源依赖环。目前大多数研究人员都采用前两种方法，或者两者结合的方式来避免死锁，本文即是采用第一种方法来设计路由算法从而避免死锁。在 NoC 的设计中，路由算法可以用不同的方法实现，目前提出的最有效的方法是查找路由表或由软件或硬件根据有限状态机执行路由算法。

3.3. Turn Model 模型

基于虫孔交换机制的路由算法容易产生死锁，产生死锁的原因是虫孔交换机制在传输过程中不释放节点资源很容易形成环路，Christopher J. Glass 和 Lionel M. Ni 提出了一种 Turn Model 模型，它的思想是分析数据包在网络传输中存在的转向以及由于转向而可能形成的环路，然后禁止网络中特定的转向来消除网络传输中的环路，从而避免死锁，最终在该模型上实现各种确定性或者自适应性的路由算法。

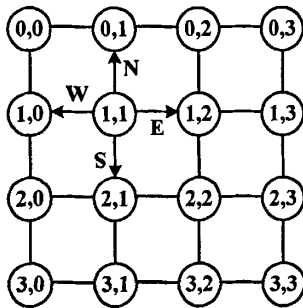


图 3-2 2D mesh 节点的通道

Turn Model 是把与每个路由节点相连接的四个通道分别用 N(北), S(南), W(西), E(东) 进行标识(边缘节点按照实际情况进行标识)。如图 3-2 所示。如果没有明确的区分,在此规定基于 2D Mesh 结构的 Turn Model 模型的默认转向指的是 90 度的转向。根据数据在网络中的传输的所有可能情况有八种转向,规定数据自东向北传输的转向标识为 EN,以此类推,自南向西的转向表示为 SW,剩下的六种转向分别为:ES, WN, WS, NE, NW, SE。这八个转向可以构成两个基本环,ES, SW, WN, NE 构成一个基本环,WS, SE, EN, NW 构成一个基本环,如图 3-3 所示,基于 2D mesh 结构的 Turn Model 模型就是限制这两个基本环中的特定的转向从而破坏环路来避免死锁。

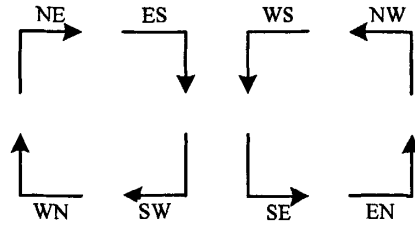


图 3-3 Turn Model 模型

下面以 XY 路由算法来具体说明，XY 路由算法属于维序路由算法，先 X 维，后 Y 维，所以是无死锁的，如果从 Turn Model 的角度来分析的话，XY 路由算法只用到了 Turn Model 中八个转向中的 WS, EN, WN, ES。数据始终按照这四个转向来传送数据永远也不会造成死锁，如图 3-4 所示，图中实线表示 XY 路由算法允许走的转向，虚线表示 XY 路由算法不允许走的转向。

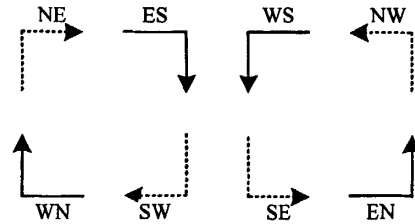


图 3-4 XY 路由对应 Turn Model 模型

3.4. XY-YX 路由算法

XY 路由算法的思想首先是在 X 方向进行传输，然后在 Y 方向进行传输到达目的地。它是一种无死锁、确定性和基于源地址的路由算法。这种路由能在负载较轻和负载均匀的条件下比较好的工作，但在负载加重时性能变差。虽然是无死锁的，但是容易造成阻塞，因为数据在到达任何一个目的节点前都要先从 X 方向开始，要么是 E 通道，要么是 W 通道，如果多个数据同时处在同一 X 方向上传输时，对 X 方向的 E, W 通道容易造成阻塞。本文提出的 XY-YX 路由算法思想是数据并不总是先 X 方向，后 Y 方向，而是根据目的节点和当前节点的位置来选择，当目的节点的 Y 方向的值小于当前节点的 Y 方向的值时，即目的节点在当前节点的北边，选择先 Y 后 X；当目的节点的 Y 方向的值大于当前节点对应的值时，即目的节点在当前节点的南边，选择先 X 后 Y，从而减轻 XY 路由算法在 X 方向产生的阻塞。具体实现如下：

本算法的路由转向是选择 Turn Model 模型中的 NE, ES, WS, NW。即路由过程只允许上述四个转向，如图 3-5 所示，根据 Turn Model 模型，该算法所允许的转向没有依赖环的存在，所以是防死锁的。实线表示路由算法允许的转向通道。

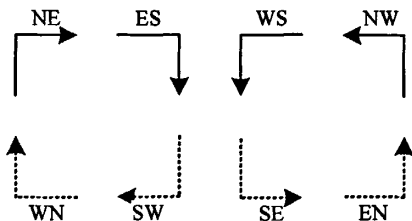


图 3-5 XY-YX 算法对应 Turn Model 模型

对应模型的确定性路由算法如下(mesh 结构节点编码参考图 3-2):

当前节点标识 (xco, yco), 目的节点标识 ($dest_xco, dest_yco$)

```

if(dest_xco == xco && dest_yco == yco)
return C; //将数据发向本地节点
else if(dest_xco < xco) //目的节点在北
return N; //将数据发向 N 通道
else if (dest_xco >= xco)//目的节点在南
{
    if (dest_yco == yco)
    return S; //将数据发向 S 通道
    else if (dest_yco > yco)
    return E; //将数据发向 E 通道
    else if (dest_yco < yco)
    return W; //将数据发向 W 通道
}

```

证明算法的无死锁特性, 我们使用 Dally 和 Seitz 提出的方法^[22]。该方法是通过网络中的通道进行特定编号, 如果能使得路由算法所经过的路径通道编码是严格递减或者递增的话, 那么该路由算法是无死锁的。

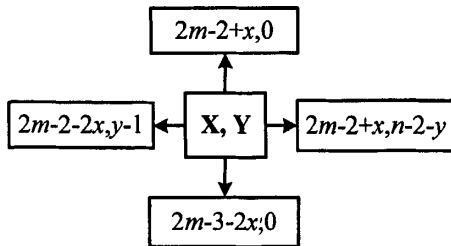


图 3-6 节点通道编码

定理 1: XY-YX 路由算法是无死锁的。

证明: 假设 $m \times n$ 的 2D Mesh 网络的每个通道用一对数来标识, 图 3-6 显示的是根据本文提出的算法得到的 $m \times n$ 的 2D Mesh 网络的节点通道编码。图 3-7 是 4×4 2D Mesh 通道编码图, 我们可以通过该通道编码图来验证 XY-YX 路由算法的无死锁特性, 使用本文提出的路由算法可以发现任何一个节点的输出通

道编码都比其输入通道编码小，也就是说数据包的路由路径是严格按照递减的顺序进行的，根据 Dally 和 Seitz 的理论，可得出该算法是无死锁的。

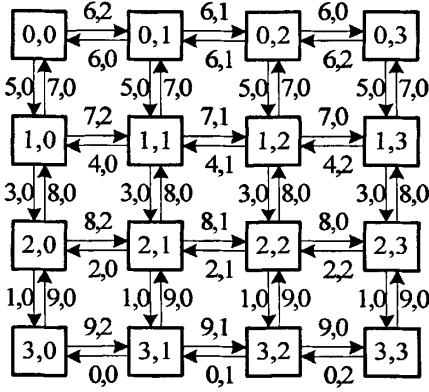


图 3-7 4×4 2D Mesh 通道编码

3.5. 性能仿真分析

评价网络的性能，一般从三个方面进行评测，包括吞吐率，链路利用率，平均网络延迟等^[23]。其中吞吐率表征的是网络能够承受的流量负载能力；链路利用率表示网络中的物理链路的使用状况；平均网络延迟则反映了数据在网络中的传输速度。不同的实验环境可以得到不同的性能值，只有归一到相同的实验环境下的性能指标的值才有意义，因此对性能的评价就需要一些参数来表明实验环境。在片上网络的性能评价中，注入率是一个表明网络所处的实验环境的重要参数。下面分别对网络注入率，网络吞吐率，网络链路利用率，平均网络延迟等进行阐述。

(1) 网络注入率

在片上网络的性能评价的实验环境的描述中，除了网络规模，硬件资源等一些表明硬件本身的配置因素之外，网络注入率是用来表征网络所处的实验环境的重要参数。它表征的是网络外部向网络内部注入流量的速度。在以微片 (flit) 为基本传输单位的片上网络中，使用每个节点每个时钟周期向网络中注入多少微片来衡量，单位是 flit/cycle/node。该值最大值为 1，表示每个节点每时每刻都在向网络中注入数据。由于网络拥塞现象的存在，节点向网络中注入的数据并不是全部可以实际注入到网络中，有相当一部分数据在其试图注入的时刻，网络发生拥塞，不能注入。对待这些不能即时注入的流量，一般采用数据包丢弃的办法，即采用如图 3-8 所示的注入机制。从图中可以看出，外部向网络注入的数据包分成两个方向，一个是通过数据包缓存真正注入网络中，另一个则由于缓存空间满，不能即时注入网络，这里采用丢弃的机制。显然，随着注入率的升高，网络拥塞情况会随着增加，缓存处于满的状态增多，因此会有更加多的数据被丢弃。

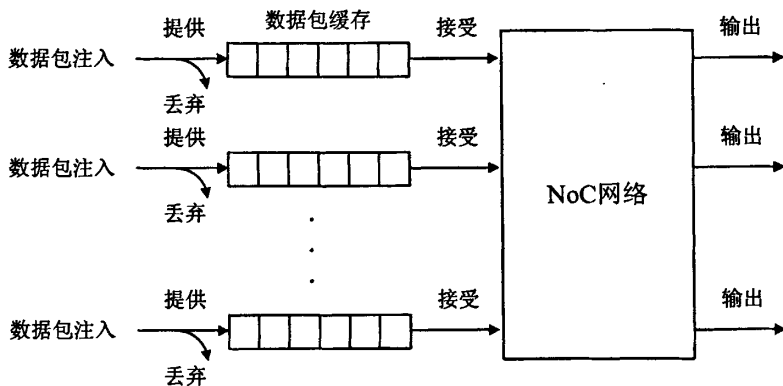


图 3-8 数据包注入模型

(2) 网络吞吐率

评价数字通信网络的性能一般都使用带宽来描述，但是在片上网络中，更加关心的是可以通过网络发送的消息的比率。因此吞吐率就是非常恰当的参数，是衡量网络整体性能的一个重要指标。吞吐率是指单位时间网络接收或者发送的消息量。网络达到饱和时，该指标的值约大，说明网络性能越好。在以微片 (flit) 为基本传输单位的片上网络中，当数据通道宽度等于一个微片时，可以用每节点每个时钟周期传输多少微片来度量。它的理论计算公式如下：

$$TP = \frac{\sum_{i=1}^S LEN_i}{Nnode \times Ttime}$$

这里 TP 指的是吞吐率；Ttime 指的是总的模拟时间，单位是时钟周期；S 指的是 Ttime 时间段内从网络中接收到或者发送出的消息的总数；LEN_i 指的是该段时间内从网络中接收到的第 i 个消息的长度(消息包含的微片数)；Nnode 指的是网络中节点的数量。因此，这里吞吐率的单位是 flit/cycle/node。理想情况，吞吐率的最大值为 1 flit/cycle/node，表示每个节点每个时钟周期都发送或者接收到一个 flit，实际情况小于 1。当网络达到饱和状态时，测得的吞吐率即为该网络可以提供的最大吞吐率。

(3) 网络链路利用率

链路利用率是除网络吞吐率以外网络输出端的另一个重要的性能参数，用它来衡量网络中链路被使用的情况。与网络吞吐率的定义类似，链路利用率是指单位时间网络链路所能传输的最大消息量。在以微片为基本传输单位的片上网络中，它指的是单位时间每条链路上传输的微片数。当网络达到饱和时，该指标越高，表示网络的链路资源越能够被充分利用。理想情况下，链路利用率值为 1，表示每时刻每条链路上都在传输数据，或者说每时刻每条链路都被使用。可以使用如下公式来计算：

$$LU = \frac{\sum_{i=1}^s LEN_i \times Dmin_i}{Clink \times Ttime}$$

其中，LU 指的是链路利用率；Dmin_i 指的是数据包 i 根据路由算法在网络中需要跳跃的最小步数；Clink 代表网络所包含的物理链路的数量，对于 K×K 的 2D-Mesh 的拓扑结构，其值为 4K×(K-1)。其它变量定义与网络吞吐率公式相同，不予赘述。

(4) 平均网络延迟

延迟是衡量网络整体性能的又一个重要指标。网络延迟是指从消息在源节点注入网络到目的节点接收到该消息的最后一个信息单位所经历的时间。在以微片为基本传输单位的片上网络中，消息的网络延迟可以定义为从消息的头微片在源节点处注入网络到该消息的尾微片被目标节点接收所经历的时间。网络延迟包含等待延迟和传输延迟两个部分，其中，等待延迟指的是消息在源节点处注入网络的过程中，由于网络的吞吐率限制，在头微片注入网络之后，后续的微片需要在网络外部等待，直到网络能够接收所经历的时间。传输延迟指的是消息在网络中实际传输所需要的时间。在网络传输中，每个消息的网络传输延迟各不相同，即使是相同的源/目标节点对之间，由于传输过程中的网络拥塞情况不同，消息的传输延迟也会随时间动态变化。这样，考察单个消息的网络延迟没有实际意义，需要使用平均网络延迟来衡量片上网络性能。在其它性能指标值相同的情况下，平均延迟越小说明网络性能越好。假设 P 代表网络传输的消息数目，L_i 代表第 i 个消息的网络延迟，其中 i 的取值范围为从 1 到 P，则平均的网络延迟(Lat)可按照如下公式计算：

$$Lat = \frac{\sum_{i=1}^P L_i}{P}$$

在本文的 NoC 路由算法设计中，我们采用网络注入率以及平均网络延迟作为评价网络性能标准。本设计的仿真验证是基于英国南安普顿大学开发的 NIRGAM 仿真实验平台上进行的，NIRGAM 是一种基于 SystemC 语言的专门面向 NoC 研究的仿真平台。SystemC 是一种开源的基于 C++ 语言的硬件设计和验证语言，传统的硬件描述语言不适合复杂的基于片上网络的 SoC 系统级建模。作为 IEEE 标准的 SystemC 语言，比已有的 HDL 语言在系统级建模与软硬件协同设计方面具有优势，更适合于片上系统的系统级建模。SystemC 使得设计者可以采用成熟的软件设计技术比如面向对象的设计来进行系统仿真和验证。用户可以通过 NIRGAM 来设置仿真的各种参数来达到仿真效果，NIRGAM 本身也是开源的，用户可以自己定制拓扑结构和路由规则来实现仿真目的。

本设计的路由算法的仿真采用 4×4 的 Mesh 结构，交换机制采用虫孔交换机制，仿真环境中数据的最小单元是 Flit，每个数据包由 6 个 Flit 组成。仿真

频率为 1GHz，每个 Flit 之间的时间间隔为 2ns，包之间的时间间隔由数据注入率确定。实验仿真过程是通过三种模式对本文提出的算法和 XY 路由算法以及 minimal OE 路由算法分别进行比较的。三种模式分别是一般模式，转置模式，热点模式。

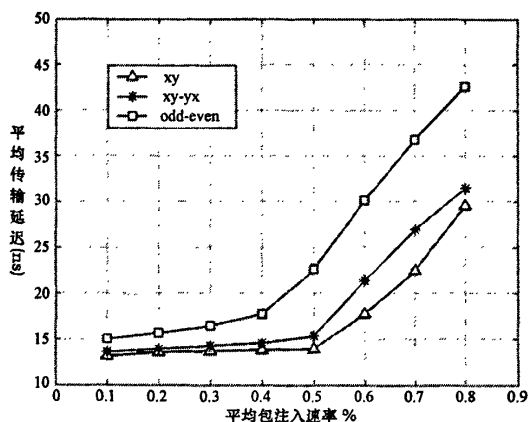


图 3-9 一般模式

一般模式又叫随机模式，即网络中的每个源节点以同样的几率随机的向其他节点发送数据，每次发送的目标节点都是随机产生的。图 3-9 显示的是一般模式下的仿真实验结果，从仿真结果来看，XY 路由算法具有最低的延迟，本文算法和 XY 路由算法比较接近，而 OE 算法显示比较高的延迟，且饱和点来的比较早。

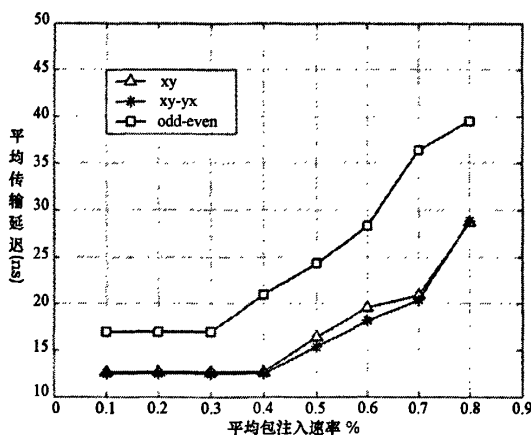


图 3-10 转置模式

转置模式是指节点 (i, j) 和节点 (j, i) 互为源节点和目标节点，相互向对方发送数据，图 3-10 显示的是转置模式下的仿真实验结果，从图中可以看到本文设计的算法与 XY 路由算法相比较，饱和点基本一致，但平均延时要略微比 XY 路由算法小，在饱和点之前，我们可以发现本文算法和 XY 路由算法没

有任何区别，随着包注入率的不断增加，超过饱和点时，XY 路由算法的延时就超过本文算法，因为本文算法相比较 XY 路由算法减少了 X 方向上堵塞。minimal OE 算法显示比较高的延迟，而且饱和点来的比较早，当数据注入率达到 30% 的时候平均延时就开始迅速增长了。

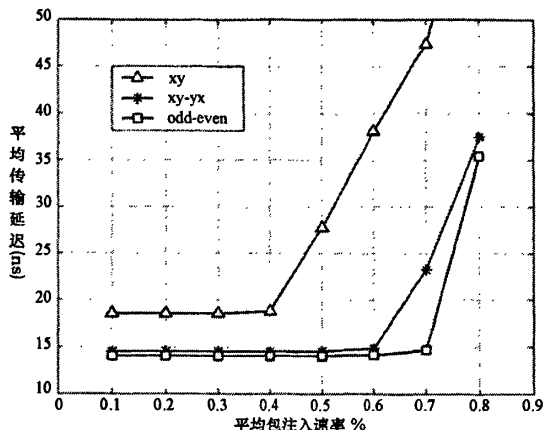


图 3-11 热点模式

热点模式即让某一个或几个节点成为热点，然后其他节点都向该节点发送数据。本仿真中选中了三个热点，然后选中其他节点向热点发送数据。图 3-11 显示的是热点模式下的实验结果，结果显示本文算法要好于 XY 路由算法，XY 路由在热点模式下当数据注入率达到 40% 的时候就进入饱和状态了，而本文算法达到 60% 才进入饱和状态，在热点模式下，目标节点固定，所以针对确定性路由算法来说，路径是固定的，XY 路由算法总是从 X 方向开始，则更容易在 X 方向产生堵塞，而本文的算法正好是在该方面的改进，所以相比较 XY 路由算法，具有更低的延迟，OE 算法在热点模式下要略优于本文算法，因为 OE 算法是具有一定的自适应性。

3.6. 结束语

本文是在 Turn Model 模型的基础上提出一种无死锁的确定性 XY-YX 路由算法，并通过 NIRGAM 仿真平台对本文算法与 XY 路由算法、minimal OE 路由算法进行了性能方面的比较，结果显示本文算法和 XY 路由算法相比较在热点模式和转置模式下具有更少的延迟。本文的路由算法可以很好的解决 XY 路由算法在 X 方向上的堵塞问题，而且算法具有很小的复杂度，几乎和 XY 路由算法一样简单，相比于自适应的 OE 算法，该算法的硬件开销更小。

第四章 基于 FPGA 的 NoC 路由节点实现

片上网络以网络互连结构代替传统总线结构，很好地解决了片上高性能计算资源之间的通信瓶颈问题。路由节点是实现 NoC 的重要基础部件，本文在分析国内外相关技术的基础上，拟设计一个可以在 4X4 的 Torus 网络结构上进行扩展的路由器模型和资源网络接口模型。路由器的设计和实现过程遵循 FPGA 设计规范，分为系统设计流程和验证流程两个部分。本章首先给出片上网络路由器的功能定义和子模块划分方案，然后采用 Verilog 硬件描述语言对各子模块的电路进行设计与建模。在完成电路设计后，利用 Altera 的 FPGA 对子模块电路模型进行综合，设计优化和布局布线。最后进行子模块的验证以及系统组装和系统验证。

4.1. NoC 路由节点总体设计

本文设计并实现了一个基于虫孔交换的路由器。该路由器采用 E-cube 路由算法，提供 3 组输入输出端口，分别是本地端口，X 维端口和 Y 维端口。初步设计拟打算设计一个 4X4 的 Torus 拓扑结构的 NoC 网络模型，设计的路由器可以在该 Torus 结构上进行扩展，为将来 NoC 的平台设计打下基础。如图 4-1 所示。在 Torus 拓扑结构中，一个网络节点分别在二维的空间上和它的相邻节点进行连接，在网络的边缘节点上，每个边缘节点连线绕回到对应维数的第一个节点，与其相连，每个路由节点连接一个本地资源节点。

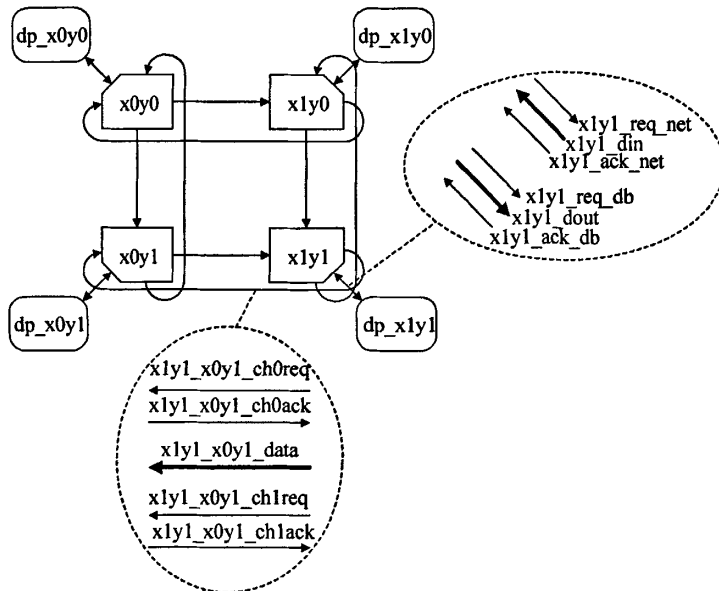


图 4-1 4X4 的 Torus 拓扑结构的 NoC 模型

另外，为了考虑硬件开销以及设计简单的因素，该网络路由节点之间的数

据传输采用单工的方式，即数据只在一个方向上传输；数据节点编码采用 2 进制编码，这样便于硬件实现；每个路由节点分别与对应维数的下一节点相连，同时也和本地节点相连，与本地节点相连的是两组支持单工的数据线和对应的控制总线，请求控制线（req）和应答控制线（ack）。

数据的通信是基于异步的请求/应答传输控制协议^[41]。该协议是应用于异步电路中的一种较为成熟的数据传输协议，协议中引入两个控制传输的信号，它们分别是请求信号（request），记作 req 和应答信号（acknowledge），记作 ack。根据协议规定，完成一次异步传输 req 信号和 ack 信号的电平必须依次经过四个阶段，如图 4-2 中 A、B、C、D 所示。

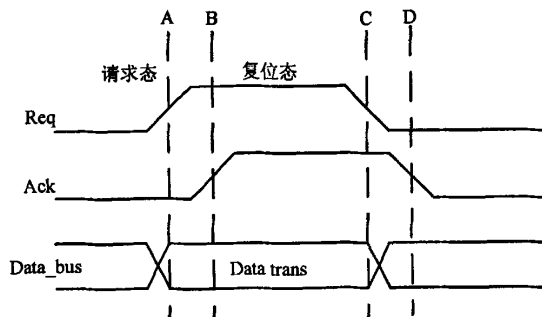


图 4-2 请求应答协议时序图

阶段 A，通信发起端在判断 ack 信号为 0 时，将需要传输的数据送到数据总线，并且将 req 信号置 1，进入请求态；

阶段 B，通信接收端侦测到 req 信号有效后，如果具备接受条件，则将 ack 置 1，同时接收数据，至此请求态结束。必须注意的是在 ack 信号有效之前，必须确保数据已经达到稳定，否则会引发数据错误；

阶段 C，通信发起端在侦测到 ack 信号有效后，确认接收成功，立刻将 req 信号置 0，进入复位态；

阶段 D，通信接收端在侦测到 req 信号无效后，将 ack 信号置 0，复位态结束，可以进入下一个的传输周期。

请求应答传输控制协议能够保证数据传输的可靠性，因此设计路由节点模块的重点之一是用于完成该协议控制的接口电路。

为了防止数据在同一维数内部通信时死锁和堵塞情况的发生，我们采用虚通道技术来解决，即采用两个虚拟通道共享一个物理链路，分别标记为 0 和 1，我们规定每个数据微片总是从通道 0 来传送数据直到到达目的节点，如果虚拟通道 0 发生堵塞，则数据微片则通过通道 1 进行传输，这种交换机制打破了网络中的资源依赖环从而防止死锁，同时也减少了网络中的堵塞情况的发生。所以在路由节点的连线组中包含了两组控制总线 and 一组数据总线，对应的控制总线分别用来控制两组虚通道的通信。

4.2. NoC 路由器

片上网络的路由器也被称作片上网络适配器，它是片上网络节点之间的连接桥梁，是实现包交换方式的基础，是片上网络的关键组件，也可以被认为是一个执行通信任务的特殊 IP 核。它主要负责根据源节点和目标节点的网络地址，按照一定的路由算法对路径进行选择，实现源节点与目标节点之间的相互通讯。路由器的核心部件就是仲裁器，它是根据一定的路由算法，为输入通道选择合适的输出通道，并通过交叉开关来进行通路选择。当同时有多条输入通道请求同一条链路时，则需要先对输入通道进行仲裁，选择相应允许的输入通道。当请求的链路处于繁忙状态，即正在传送报文时，输入报文先被缓存在输入端的缓存器中，直到链路中的报文传送完毕，同时当该输入通道经过仲裁成功获得链路时，输入报文才能经由此链路继续传输。

本设计中采用的是 E-cube 的维序路由算法，E-cube 路由算法是一种确定性的基于源地址、目的地址的路由算法。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，在 n 维立方体中，每个节点是用一个 n 位的二进制编码表示的。每个节点有 n 条输出的通道，其中第 i 条通道就对应第 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信息 Dest。当 n 维立方体中的当前节点 Current 收到一个数据包时，E-cube 路由算法计算路由向量 $Offset := Current \oplus Dest$ ；其中， \oplus 是逻辑运算中的异或运算。如果 $Offset = 0$ ，说明数据包已经到达了目的地，则向本地节点转发。否则，数据包将被送往第 k 维的输出通道，其中 k 是函数 $FirstOne(Offset)$ 的返回值：Offset 中从右边起(或者从左边起)第一个置为 '1' 的那一维。

在整个系统中，该算法给全部节点定义了绝对地址。消息的发送遵从 E-cube 路由算法的规定，先沿 X 维方向传递，当消息在到达 X 维目标节点时再转向 Y 维进行传递，直到到达 Y 维目标节点。从前面的介绍得知，由于 NoC 的链路是单向的，所以数据包只能按照地址严格递增或者严格递增的方向进行传输，在必要的情况下需要通过边缘节点的环路绕回。

根据 E-cube 的算法思想，本文的路由器设计在 X 维和 Y 维分别用 2 个子路由器进行路由，分别为 xrouter 和 yrouter，xrouter 负责 X 维的数据路由，yrouter 负责 Y 维的数据路由，如图 4-3 所示，具体的子路由器的详细设计在后面介绍。由于 E-cube 路由算法总是从 X 维开始，所以本地资源节点通过 din 数据链路和 xrouter 相连，数据由本地资源节点通过资源网络接口传输给 xrouter，同时 xrouter 和相邻的 X 维的前一个节点通过 xin 数据链路相连。数据由前一个节点传输进来。xrouter 主要作用是判断 X 维数据的路由，当数据到达该节点时（从本地节点或者前一个路由节点），首先进入 xrouter，由 xrouter 进行判断，如果到达的节点在 X 维是目的节点，则数据通过 xydata 数据线直接传输给 yrouter，

否则数据通过 xout 数据线传输给 X 维的下一个路由节点。

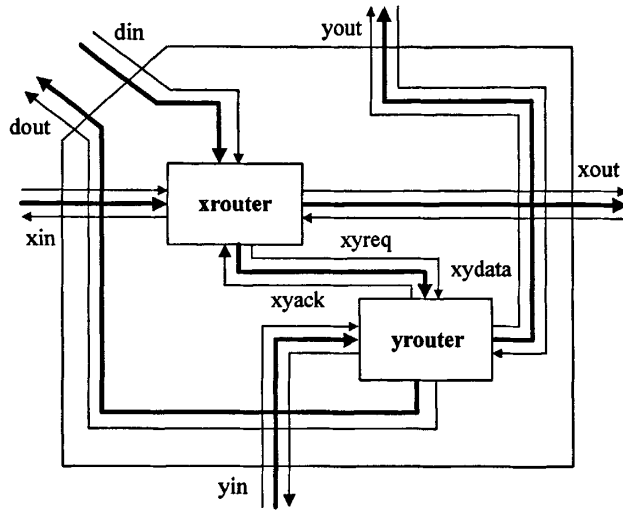


图 4-3 路由器结构

同理，yrouter 的输入链路也有 2 个，一个 xrouter 的输出链路 xydata 进来的数据，另一个是前一个 yroute 的 yout 输出链路即本节点的 yin 输入链路。进入 yrouter 的数据包在 X 维上都已经到达目的节点了，所以数据进入 yrouter 只需判断消息的路由信息是否在 Y 维达到目的节点即可，如果消息到达的节点不是目的节点，则数据直接通过 yout 数据链路传给下一个路由节点的 yrouter，否则数据通过 dout 数据链路传给本地资源网络接口，再由本地资源网络接口传送到本地资源节点，这样整个一次路由过程就结束了。

4.3. NoC 子路由器设计

本文设计实现的 NoC 路由器主要应用于 2D Torus 拓扑结构的 NoC。每个路由器有 3 组 I/O 接口，有 2 组分别连接其它两维方向的路由器，还有一组 I/O 接口用于连接本地计算资源节点。为了提高可重用性，采用相同的 I/O 接口连接路由器和计算资源节点。计算资源节点到路由器的数据转换功能由专用资源网络接口来实现。前面提到 NoC 的路由器在不同维数有独立的子路由器来进行控制对应维数的数据路由，子路由再通过指定的连接方式进行数据通信。本设计中是基于 2 维的 E-cube 算法，所以只用了两个子路由器，分别是 xrouter 和 yrouter。这两个路由器内部结构都是一样的，只不过一个是负责 X 维，一个是负责 Y 维的数据路由控制，下面将以一个子路由器 xrouter 来进行详细介绍。

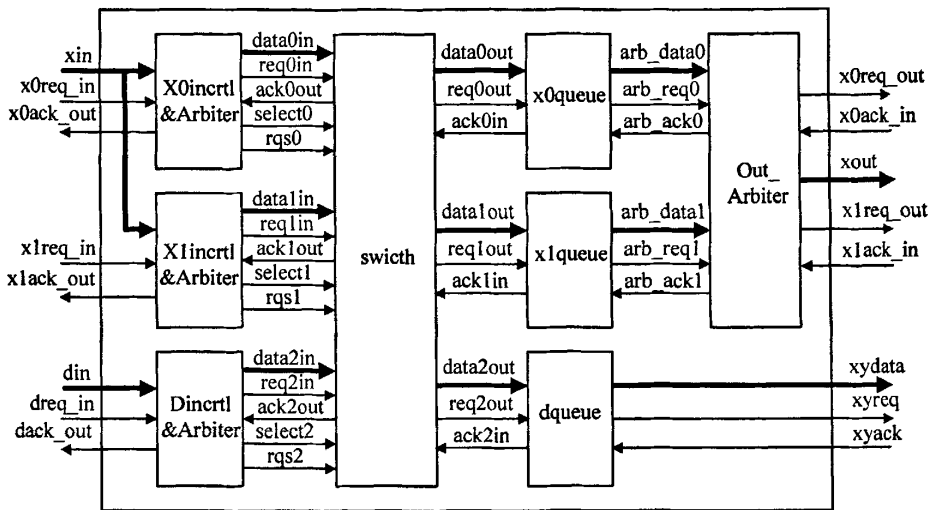


图 4-4 X 维子路由内部结构

在前面提到，一个基本的路由器一般至少包括以下几个部分，输入输出控制器，缓冲队列，路由仲裁器和交叉开关等。如图 4-4 所示。X0inctrl&Arbiter, X1inctrl&Arbiter, Dinctrl&Arbiter 主要负责输入通道控制，对数据包交换方式的控制，以及路由算法的实现，即包含输入控制器和路由仲裁器，其中 X0inctrl&Arbiter, X1inctrl&Arbiter 表示虚通道 0, 1 输入数据的控制，Dinctrl&Arbiter 表示本地资源节点输入数据的控制，专门负责本地资源节点与通讯节点的通信；switch 表示交叉开关；x0queue, x1queue, dqueue 表示输出缓冲队列；Out_Arbiter 表示输出仲裁控制器，同时根据指定的逻辑来选择对应的虚拟通道有效。下面以虚拟通道 0 为例描述各信号线的功能，如表 4-1 所示，其他通道信号线的功能类似。

表 4-1 子路由模块部分输入输出信号线功能描述

| 信号名 | 输入输出 | 功能描述 |
|-----------|------|----------------------------------|
| xin | 输入 | 表示上一节点向输入通道输入的 18 位的数据信号线 |
| x0req_in | 输入 | 0 虚拟通道的请求信号线 |
| x0ack_out | 输出 | 0 虚拟通道的应答信号线 |
| data0in | 输入 | 输入控制器到交叉开关的数据信号线 |
| req0in | 输入 | 0 通道想交叉开关的请求信号线 |
| ack0out | 输出 | 交叉开关向虚拟通道 0 的应答信号线 |
| select0 | 输入 | 输入控制器向交叉开关的路由选择信号线,该信号线是 2 位二进带宽 |
| rqs0 | 输入 | 输入控制器向交叉开关发出的路由选择请求信号线 |
| data0out | 输出 | 交叉开关给输出缓冲区发出的数据信号线 |
| req0out | 输入 | 交叉开关给输入缓冲区发出的请求信号线 |

| | | |
|-----------|----|----------------------------|
| ack0in | 输出 | 输出缓冲区给交叉开关发出的应答信号线 |
| arb_data0 | 输入 | 虚拟通道 0 输出缓冲区给输出仲裁器发出的数据信号线 |
| arb_req0 | 输入 | 虚拟通道 0 输出缓冲区给输出仲裁器发出的请求信号线 |
| arb_ack0 | 输出 | 输出仲裁器给虚拟通道 0 输出缓冲区发出的应答信号线 |
| x0req_out | 输出 | 输出仲裁器向下一节点发出 0 虚拟通道输入请求信号线 |
| x0ack_in | 输入 | 下一节点的 0 虚拟通道给输出仲裁器的应答信号线 |
| xout | 输出 | 输出仲裁器向下一个节点输出的数据信号线 |

从子路由器的结构图来看，每个子路由器有 2 组数据输入链路和三个独立的输入控制器，一个数据输入链路是用来连接本地资源节点，另一个是用来连接前一个路由节点。我们可以看到，每个虚拟通道都有各自独立的输入通道控制逻辑，当输入控制器检测到有数据传输请求信号时，输入控制器将微片寄存并开始对微片进行检测。如果检测到的微片是头微片即包含路由信息，输入控制器将向交叉开关发出请求信号来选择路由进入合适的输出队列，被其请求的输出缓冲队列的选择依赖于数据包的地址信息，输入交叉开关通过请求应答信号来选择输出缓冲通道。如果当前缓冲区为空，它将存储数据并将输入控制器发出应答信号，之后的数据微片将直接通过头微片建立的路由通路进行数据传输，直到尾微片才释放该通路，当尾微片达到输出缓冲区时，输出缓冲区给出应答信号，输入控制器在收到应答信号时撤销请求信号，释放交叉开关选择的路由通路。输出缓冲队列只要从输入控制器得到微片就将其转发给输出控制器。当微片被传送完后，缓冲队列被清空，准备接收下一个微片。其中 dqueue 输出缓冲队列直接连接到输出通路，而另外两个与虚通道相关的输出缓冲队列相互竞争一个物理通路。

4.3.1. 输入控制器和路由仲裁

在前面提到输入控制模块主要功能是对输入数据的控制，微片分析控制以及路由选择控制。下面以其中一个输入通道控制器来进行详细介绍。由前面的介绍可知，根据模块的逻辑功能进行划分，该模块主要有三大部分组成：

第一部分是输入流控模块，它的作用是翻译握手协议和微片分析处理为后续模块的操作奠定基础，握手协议在前面已经介绍，主要是通过 req 和 ack 信号来进行数据的异步传输。微片的分析处理操作主要是将数据放入锁存器，然后对高 16, 17 位数据进行微片的分类并给出对应的信号线。同时如果是头微片则将对地址信息进行分析并计算得到进入下一个节点的相对地址。具体的微片

分析处理代码如下所示：

```
always @(data_latch or end_of_route)
  begin :
    bFT = data_latch[17:16];
    bXA = data_latch[15:8];
    bYA = data_latch[7:0];
    flit_type = bFT;
    xaddr = bXA;
    yaddr = bYA;
    if (xaddr == 8'b00000000)
      begin
        end_of_route = 1;
        xaddr_out = yaddr;
        yaddr_out = 8'b00000000;
      end
    else
      begin
        end_of_route = 0;
        xaddr_out = xaddr - 1;
        yaddr_out = yaddr;
      end
    case (flit_type) //synopsys parallel_case
    1: begin
        start_of_packet = 1;
        end_of_packet = 0;
      end
    2: begin
        start_of_packet = 0;
        end_of_packet = 1;
      end
    default : begin
        start_of_packet = 0;
        end_of_packet = 0;
      end
    endcase
    bXAout = xaddr_out;
```

```

        bYAout = yaddr_out;
        hout[17:16] = bFT;
        hout[15:8] = bXAout;
        hout[7:0] = bYAout;
        header = hout;
    end

```

第二部分是基于虫孔交换机制的数据传输控制功能，它的主要作用是负责执行头微片，数据微片，尾微片传输，由于头微片包含路由信息，在传输头微片的时候需要进行路由通路选择，而数据微片和尾微片直接通过头微片选择的路由通路来进行数据传输，同时尾微片的传输表示传输结束，需要释放头微片选择的路由通路，三种类型的数据传输要按照一定的顺序进行。该部分主要由状态机来实现，详见后面的介绍。

第三部分是路由选择功能，该功能主要是通过对头微片的高 8 位地址信息进行分析处理，然后得到 select 信号来选择下一个输出端口。然后向相应方向的输出端口发送 req 请求信号和数据微片，并在接收到目标端口的 ack 应答信号后撤销请求信号。具体的路由端口选择编码如下所示。

```

    always @(data or data_latch or latch or my_address or ch0 or ch1)
        begin : route_select
            my_addr = my_address;
            if (latch)
                __tmp61 = data;
            else
                __tmp61 = data_latch;
            d = __tmp61;
            bvaddr = d[15:8];
            still_to_go = bvaddr;
            out_select_n = 2'b00;
            if (still_to_go == 8'b00000000)
                out_select_n = 2'b11;
            else
                begin
                    if (my_addr == 8'b00000000 && ch0 || ch1)
                        out_select_n = 2'b10;
                    else
                        out_select_n = 2'b01;
                end
            end
        end

```

由于采用的是虫孔交换机制，网络中存在三种类型的微片，头微片，数据微片和尾微片，微片的顺序不能有任何错位，所以在数据传输的过程中要对微片数据流进行控制，本设计采用有限状态机的方式对微片的传输进行控制，具

体如图 4-5 所示。其中 req 表示请求信号，out_ack 表示输出应答信号，end_of_packet 表示尾微片结束标志。

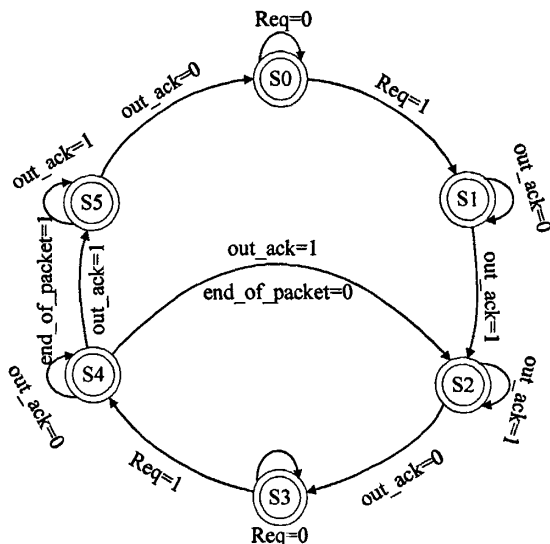


图 4-5 数据流控制状态机

数据流控制状态机有 6 个状态，分别是 S0, S1, S2, S3, S4 和 S5。其中，S0, S1 负责头微片的传输，S2, S3, S4 负责数据微片，尾微片的传输，S5 判断尾微片结束传输。S0 为初始状态，设置锁存器和头微片有效，当 req 请求信号有效时进入 S1 状态。S1 状态表示头微片的请求传输并确认收到 ack 应答信号，S2 状态是等待 ack 信号结束，确认数据传输结束。S3 状态表示数据微片的请求传输，并等待 ack 应答信号，S4 等待 ack 信号结束。并判断是否是尾微片，如果是尾微片则进入 S5 状态，否则继续进入 S2 状态继续数据微片的传输，S5 表示尾微片传输结束，等待确认 ack 信号结束进入 S0 状态，否则继续等待。

4.3.2. 交叉开关和缓冲区

在一个路由节点内部，输入和输出通道之间的互连采用交叉开关技术。交叉开关 switch 由多路选择器来实现，负责将各输入通道中的数据连接到输出通道的缓存器。由图 4-4 所示，当输入通道接收到数据微片后，就向交叉开发送 req_in 信号，交叉开关经过处理再向指定的输出通道发出请求信号 req_out，如果该输出通道响应该请求信号，则会向请求传输的输入通道返回相应的应答信号 ack_out，同时将该输出链路连通到请求传输的输入链路，将输入通道的数据保持到输出缓冲中。交叉开关的端口选择是由路由仲裁器来控制的，主要是通过输入通道的 select 信号来进行选择。如果节点中有 N 组通道的话，那么对每组输出通道而言，均与 N-1 条输入通道通过多路选择器相连，即整个通讯节点的路由功能、仲裁功能以及开关功能分布于各个链路的通道之中，它们之间

互不影响。

缓冲区的设计放在输出通道，当数据进入路由器，由路由器选择控制直接进入输出缓冲区，然后由输出仲裁选择虚通道输出。由于采用的是虫孔交换机制，所以缓存器的大小就是一个微片的大小，采用寄存器来存储实现，它在不能立即向输出通道发送数据微片时，负责存储输入数据微片。缓冲区的宽度等于数据微片的宽度，缓冲区也是通过状态机来实现数据微片的存储和转发功能。该状态机包括两个状态，S0 和 S1 状态，S0 表示对请求数据写入缓冲区，S2 表示根据 ack 信号将缓冲区数据输出到 Output_Arbiter (输出仲裁模块)，它主要通过 buffer_write 信号和 buffer_clear 信号来判断进行写入和清空操作。具体代码如下所示。

```
always @(current_state or req_in or ack)
    begin : control_logic
        req_v = 0;
        ack_in_v = 0;
        buffer_write_v = 0;
        buffer_clear_v = 0;
        next_state = 1'b0;
        case (current_state)
            0: if (req_in)
                begin
                    buffer_write_v = 1;
                    next_state = 1'b1;
                    ack_in_v = 1;
                end
            else
                next_state = 1'b0;
            1: begin
                req_v = 1;
                if (ack)
                    next_state = 1'b0;
                else
                    next_state = 1'b1;
            end
        endcase
        req = req_v;
        ack_in = ack_in_v;
        buffer_write = buffer_write_v;
        buffer_clear = buffer_clear_v;
    end
```

4.3.3. 输出控制器

Output_Arbiter 模块，它主要负责输出通路的选择，当虚拟通路控制器需要

选择一个链路时，它将向输出控制器发送一个请求信号，等待请求允许信号 grant，0 请求信号比 1 请求信号具有更高的优先级，但是在每一次传输结束，正在请求的虚拟通道将必须撤销请求信号，其他的等待通道队列才可以选择输出链路，这样可以使虚拟通道 1 不至于出现饿死的情况。

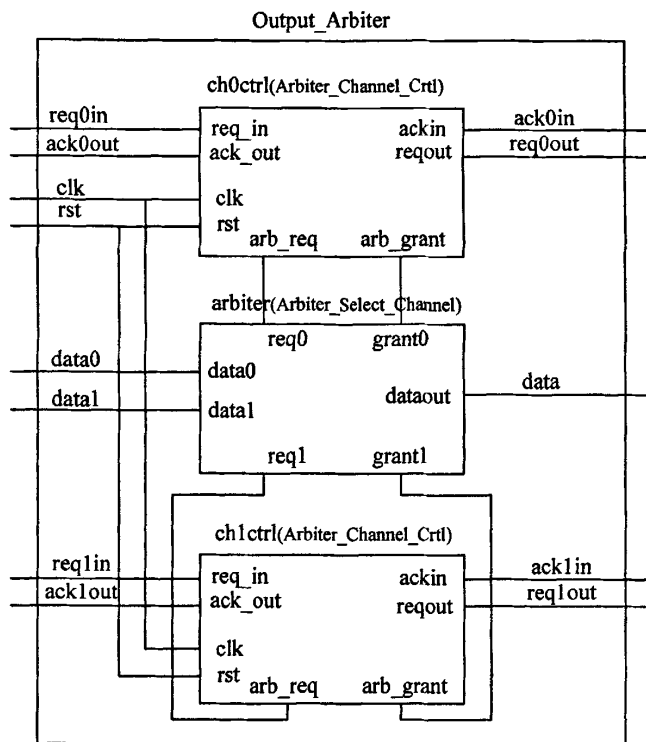


图 4-6 输出通道内部结构

Output_Arbiter 模块包括三个子模块，如图 4-6 所示，分别是 2 个虚拟通道仲裁控制模块 Arbiter_Channel_Ctrl 和一个通道仲裁选择模块 Arbiter_Select_Channel。其中 Arbiter_Channel_Ctrl 模块主要负责对以虚拟通道的数据传输控制，并负责向 Arbiter_Select_Channel 提交申请来选择数据通路。而 Arbiter_Select_Channel 模块则主要负责两个虚拟通道的优先级控制选择，具体实现代码如下所示。

```

always @(posedge clk)
begin : select
    r0 = req0;
    r1 = req1;
    ch_bsy = channel_busy;
    if (ch_bsy)
    begin
        if (selected_channel && !r1 || !selected_channel && !r0)
            ch_bsy = 0;
    end
end
    
```

```

if (!ch_bsy)
begin
    if (r0)
    begin
        selected_channel <= 0;
        ch_bsy = 1;
    end
    else
    begin
        if (r1)
        begin
            selected_channel <= 1;
            ch_bsy = 1;
        end
    end
    end
    channel_busy <= ch_bsy;
end
always @(req0 or req1 or data0 or data1 or channel_busy or selected_channel)
begin : output__17
    if (selected_channel)
        __tmp63 = data1;
    else
        __tmp63 = data0;
    dataout = __tmp63;
    grant0 = channel_busy && !selected_channel;
    grant1 = channel_busy && selected_channel;
end
end

```

4.4. 数据包格式

物理链路的数据带宽为 18 位，其中 16 位的数据位和 2 位的微片类型标识位，主要用来区分微片的类型，地址格式如图 4-7 所示。微片标识位 01 表示的是头微片，00 表示数据微片，10 表示尾微片。对于头微片对应的数据表示的是地址信息，对于本文的设计，高八位表示 X 维的地址，低八位表示 Y 维的地址。数据微片和尾微片的数据位包含的是具体的数据信息。

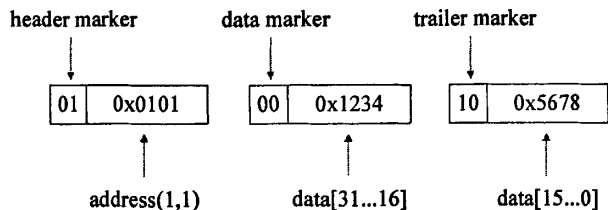


图 4-7 数据包格式

在每个头微片中，我们存储的地址信息既不是源地址，也不是目的地址，

是基于源和目的相对地址，是通过本地资源节点根据网络的拓扑结构计算得来的相对地址。在数据传输过程中，头微片每经过一个路由节点，对应维数的地址信息将自动减 1，当对应维数的地址等于 0 时，表示当前节点是对应维的目标节点。然后再将数据通过 `doutput` 传给下一维的子路由器，然后继续在下一维上传输直到到达目的节点，最后再将数据传给本地资源节点。图 4-7 表示的是一个包含有 32 位数据的数据包(0x12345678)，该数据包从源地址 (1, 1) 发送到目的地址 (0, 0) (相对地址是 (1, 1))。数据包的被重新划分成三个微片，头微片包含相对地址信息，中间的数据微片包含 16 位的数据信息，尾微片也包含 16 位的数据信息。

在传输数据报文的时候，本地资源节点首先将数据包分割成若干个的微片。微片的长度规定为 16 位。该网络支持任意长度的报文，也就是说对微片的个数没有限制。数据包的头微片用高两位来进行标识，后面的 16 位包含两个 8 位的地址信息 (X 维和 Y 维)。当接受到带有地址信息的首微片时，路由器将在输入链路和合适的输出链路 (可能是虚通道 0, 虚通道 1 或者 `doutput` 端口) 之间建立一个连接。该数据包的其他的微片将沿着相同的路由路径进行转发。当路由节点接受到尾微片时表示是最后一个微片，表示该数据包传送结束，尾微片也包含数据信息，也需要进行转发。这就是所谓的基于虫孔交换机制的路由。

4.5. FPGA 设计流程

FPGA 的设计流程与 ASIC 类似，一般来说，完整的 FPGA 设计流程包括电路设计与输入、功能仿真、综合、实现与布局布线、布线后仿真与验证、加载配置与调试等主要步骤，如图 4-8 所示。

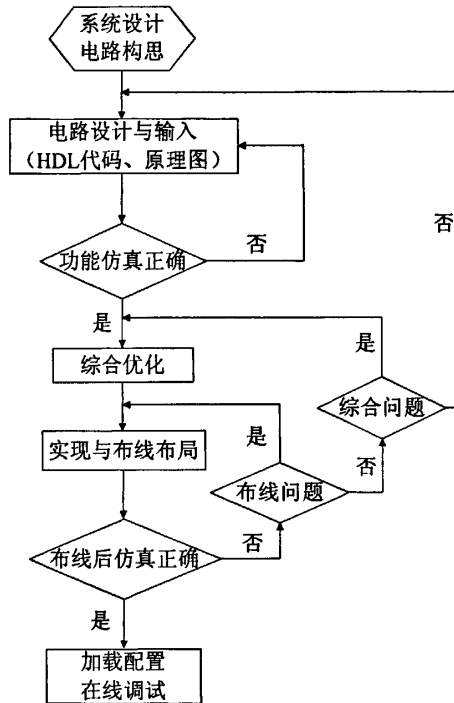


图 4-8 FPGA 设计流程图

➤ 电路设计与输入

电路设计与输入是指通过某些规范的描述方式，将电路设计构思输入给 EDA 工具。常用的设计输入方法有硬件描述语言(HDL)和原理图设计等。原理图设计输入法在早期应用比较广泛，这种方法比较直观，便于理解。但是在大型设计中，它的可维护性较差，不利于模块构造与重用。目前，最常用的设计方法是 HDL 设计输入法，其中应用最为广泛的是 VHDL 和 Verilog HDL 语言。这两种语言的优点是利于自顶向下设计，利于模块划分与复用，可移植性好，通用性好，设计不因芯片的工艺与结构的不同而变化，更利于向 ASIC 的移植。在本设计中采用的是 Verilog HDL 设计输入法。

➤ 功能仿真

电路设计完成后，要用专用的仿真工具对设计进行功能仿真，以验证电路功能是否符合设计要求。功能仿真有时也被称为前仿真，通过前仿真能及时发现问题，加快设计进度，提高设计的可靠性。常用的仿真工具有 Mentor 公司的 ModelSim、Synopsys 公司的 VCS、Cadence 公司的 NC-Verilog 和 NC-VHDL 等。本设计中使用的是 ModelSim 仿真工具。

➤ 综合优化

综合优化 (Synthesize) 是指将 HDL 语言、原理图等设计输入映射成与、或、非门，RAM，触发器等基本逻辑单元，并根据一定的约束条件进行优化，

最后产生网表 (Netlist) 文件, 供布局布线使用。常用的综合优化工具有 Mentor 公司的 Precision RTL、Synopsys 公司的 FPGA Compiler II 等。本设计中使用了 Quartus II 中的 Analysis&Synthesis 具进行分析与综合。

➤ 实现与布局布线

设计经过综合优化之后生成了网表文件, 这与芯片实际的配置情况还有较大差距。因此, 应该使用 FPGA 厂商提供的软件工具, 根据所选芯片的型号, 将综合输出的逻辑网表适配到具体的 FPGA 器件上, 这个过程叫做实现过程。在实现过程中最主要的过程是布局布线 (Place And Route, PAR)。布局 (Place) 是指将逻辑网表中的硬件原语或者底层单元合理地适配到 FPGA 内部的固有硬件结构上, 布局的优劣对设计的最终实现结果 (速度和面积) 有很大影响。布线 (Route) 是指根据布局的拓扑结构, 利用 FPGA 内部的各种连线资源, 合理正确的连接各个元件。FPGA 结构相对复杂, 为了获得较好的实现结果, 特别为了满足时序要求, 一般采用时序驱动的引擎进行布局布线, 对于不同的时序约束, 获得的布局布线结果一般有较大差异。本设计使用 Quartus II 内嵌的布局布线工具, 以时序驱动的方式对设计进行布局布线。

➤ 布线后仿真与验证

将布局布线后的时延信息、反标到设计网表中, 所进行的仿真称为布局布线后仿真, 简称后仿真。布局布线之后生成的仿真时延文件包含的时延信息、最准确, 既包含了门延时, 又包含了实际布线延时, 较好的反映了芯片的实际工作情况。通过后仿真能检查设计时序与 FPGA 实际运行情况是否一致, 确保设计的可靠性和稳定性。后仿真的仿真工具与功能仿真一样, 使用的还是 ModelSim。有时为了保证设计的可靠性, 在后仿真之后还要做一些验证。验证的手段比较丰富, 可以用 Quartus II 内嵌时序分析工具完成静态时序分析 (Static Timing Analyzer, STA); 也可以用第三方验证工具 (如 Synopsys 的 Formality 验证工具、PrimeTime 静态时序分析工具等); 还可以用 Quartus II 内嵌的 ChipEditor 分析芯片内部的连接与配置情况。

➤ 调试与加载配置

设计开发的最后步骤就是在线调试或者将生成的配置文件下载到芯片中进行测试。可以使用 Quartus II 的 Assembler 具生成编程文件, 然后使用 Quartus II 的 Programmer 工具与 USB-Blaster 下载电缆一起对器件进行编程和配置, 实现 FPGA 原型芯片。本设计中是把生成的配置文件下载到芯片中进行测试的。同时可以使用 Quartus II 内嵌的 In-System Memory Content Editor 工具, 对芯片工作情况进行观测。该工具可以通过 JTAG 口, 在线实时地对 FPGA 中集成的单端口 RAM 进行操作。

FPGA 原型芯片实现的整个流程都是使用 Altera 公司 Quartus II 软件实现。任何仿真或验证步骤出现问题, 就需要根据错误的定位返回到相应的步骤更改

或重新设计。本设计中原型芯片的具体实现过程如下。

首先，需要在 Quartus II 软件中创建一个工程，将设计输入文件添加到工程中，同时，对设计的输入输出信号进行管脚分配。在本设计中设计输入文件包括设计源程序（Verilog HDL 格式），存储数据文件（MIF 格式），第三方 EDA 工具产生的文件（VQM 格式），以及 Quartus II 软件中由 MegaWizard Plug-In Manager 工具产生的 IP 核宏模块（存储器和锁相环 PLL）。

其次，在设计输入完成之后，就需对使用 Quartus II 内嵌的各种工具，对设计进行综合、布局布线以及时序分析。在本设计中，对设计所加的约束都使用的是 Quartus II 软件中的默认值。当出现设计不满足约束条件时，根据 Quartus II 软件提示的错误，需要准确定位错误，从而解决问题。需要注意，这里错误定位和解决问题的过程是一个相当复杂，多次反复的过程。

最后，上述步骤完成之后，Quartus II 软件会生成一个 sof 文件，我们将这个文件通过 USB-Blaster 下载电缆下载到 EPS150 芯片上，通过 FPGA 开发板上的其他资源如 LED 灯、现实接口等对设计结果进行观察，如果能满足设计要求，就完成了设计的 FPGA 原型实现。在 Quartus II 的综合报告中可以得到本设计对 FPGA 的资源利用率。

第五章 路由节点的仿真验证及性能分析

路由节点将作为一个 IP 核应用于基于片上网络的 SoC 开发和系统集成,根据国际电联对 IP 核的定义——可以作为可重用设计单元的模块及其验证环境,因此对路由节点进行验证是整个设计流程中的关键环节之一。在本章中我们首先介绍片上网络的验证方案,然后再对基于 FPGA 设计的路由节点及其子模块进行仿真验证,并对其进行测试,给出的仿真结果。最后分析片上网络路由节点的性能。

5.1. 路由节点的验证方案

随着集成电路设计复杂度的不断提高,设计验证变得越来越重要。大规模电路设计必须经过系统、全面的仿真验证,以保证其所有逻辑都经过验证,并且功能正确^[24]。为了提高仿真验证工作的效率,保证验证过程的全面性,需要在验证实施前设计全面、有效、可行的仿真验证方案。对片上互连网络而言,需要根据其特点进行特殊设计,以保证仿真验证过程的正确、有效。总的来说,片上互连网络的特点主要包括:

- ▶ 片上互连网络具有分层结构,从下至上可分为物理层、网络链路层及应用层。物理层为单个路由器或网络接口,完成底层数据的传输;网络链路层建立在物理层构建的网络之上,并加入网络接口与通信协议,负责将数据从源资源节点正确地转发到目的资源节点;应用层构建在物理层与网络链路层之上,为资源节点之间的通信提供接口和同步。
- ▶ 片上互连网络使用全局异步局部同步的时钟策略,不同的资源节点之间工作在各自不同的时钟域,各个时钟域之间通过异步进行转换。
- ▶ 片上互连网络完成的功能主要是节点之间的数据通信,它主要考虑数据传输的正确、有序,保证不同节点之间正确、有效的通信。

5.2. 验证流程

根据以上对片上网络特点的分析,我们在对片上网络路由节点进行建模时,采用了自顶向下的主流方法将其分解为较为简单的设计子模块。而验证过程则以相反的方向进行,从较简单的子模块开始,逐步过渡到更高设计层次,更复杂的单元。在片上网络路由节点设计的早期我们就根据其功能定义制定了验证方案,并且在用硬件描述语言完成对电路建模的同时,设计相应的验证平台,同步进行局部逻辑验证。这样做有以下几点优势:

- ▶ 根据子模块电路建模时发现的疑点和难点,调整验证平台的设计,确定验证的重点环节。
- ▶ 在验证过程中及时发现局部设计中存在的错误或隐患,及时修改设计,大

大缩短了开发过程中的调试迭代路径，加快了开发速度。

- ▶ 子模块所需验证用例集合较小，便于设计高覆盖率的验证平台。

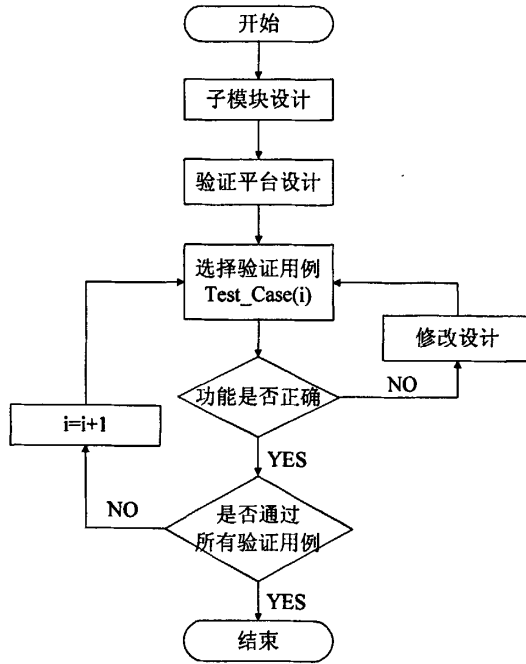


图 5-1 子模块验证流程图

子模块验证流程如图 5-1 所示。在完成子模块的电路建模后，我们根据其功能定义给出相应的验证平台。该验证平台包含多个验证用例构成的验证集。我们首先将第一个验证用例加载到子模块上。根据输出，如果子模块完成的功能和预期的不一致，我们将定位故障点，并且对电路模型进行修正，迭代过程将持续到设计通过该验证用例。然后我们将判断是否通过了所有的验证用例，如果没有则更换验证用例，重复以上工作，迭代过程持续到设计通过所有验证用例。在每个子模块都完成验证后将子模块组合为网络组件，在仿真平台上验证其功能正确性。

5.3. 路由节点的验证过程及验证结果

对片上网络路由的验证工作有两个切入点。

第一：功能仿真

功能仿真的主旨在于验证综合后的电路结构是否与设计意图相符合，是否存在有歧异的综合结果。功能仿真的输入是根据综合后得到的一般性逻辑网表抽象出的仿真模型，不考虑延时信息。对于片上网络路由节点中较为简单的模块，当我们确信硬件描述语言的代码不存在综合歧异时，就省略功能仿真。但是如果在布局布线后仿真发现有电路结构与设计意图不符合的现象，则会回溯到功能仿真对综合结果进行补救分析。功能仿真一般也称为“前仿真”。

第二：布局布线后仿真

布局布线后仿真是指电路已经映射到特定的工艺环境后，综合考虑电路的路径延迟与门延迟的影响，验证电路的行为是否能够在一定时序条件下满足设计构想的过程^[40]。布局布线后仿真的主要目的在于验证是否存在时序违规。其输入为从布局布线结果抽象出的门级网表、验证平台以及扩展名为 SDO 和 SDF 的标准延时文件是由集成电路制造厂商提供的其物理硬件时序特征的描述，这种时序特征一般包含单元逻辑的门延时最小值、典型值、最大值。SDO 和 SDF 文件包含的延时信息最全，不仅包含了门延时，还包含了布线后信号在传输线上的延时数据。布局布线后仿真最为准确，能够较好地反映芯片的实际工作情况。布局布线后仿真也被称为时序仿真，或“后仿真”。

在对片上网络路由节点的各子模块完成建模后，我们利用 ALTERA 公司的 QuartusII 集成开发环境对模型进行了综合，并且选用 STRATIX III 系列 FPGA 进行了电路的布局和布线，得到电路的门级网表 ECUBE_ROUTER.vo 以及 ECUBE_ROUTER.sdo。最后结合这两个文件和我们设计的验证平台 Testbench.v 利用 Mentor Graphics 公司提供的 ModelSim 软件进行验证。

本文的路由模块设计仿真验证是基于其他子模块仿真正确的情况下进行的组合功能仿真。

首先我们看一下一次握手协议的仿真情况，片上网络路由节点是按照四相握手协议的规定以异步方式将数据包从源节点传输到目标节点的。为了验证路由节点的握手协议的功能，我们在验证平台的设计中，首先给出 dreq_in 请求信号，置 dreq_in 为高电平，并将数据发送到数据总线，等待子路由模块的响应，当路由模块给出 dack_out 信号后（置 dack_out 为高电平），表示数据已经经过几个时钟周期的处理，并进入该子路由模块缓冲区等待向下个路由节点传输，缓冲区在收到数据后就开始向本节点的 Y 维子路由模块或者下个节点的 X 维子路由模块发出 req_out 请求信号，当等待到下个子路由模块的应答信号 ack_in 时，撤销请求信号。一次数据路由传输结束。仿真波形如图 5-2 所示。

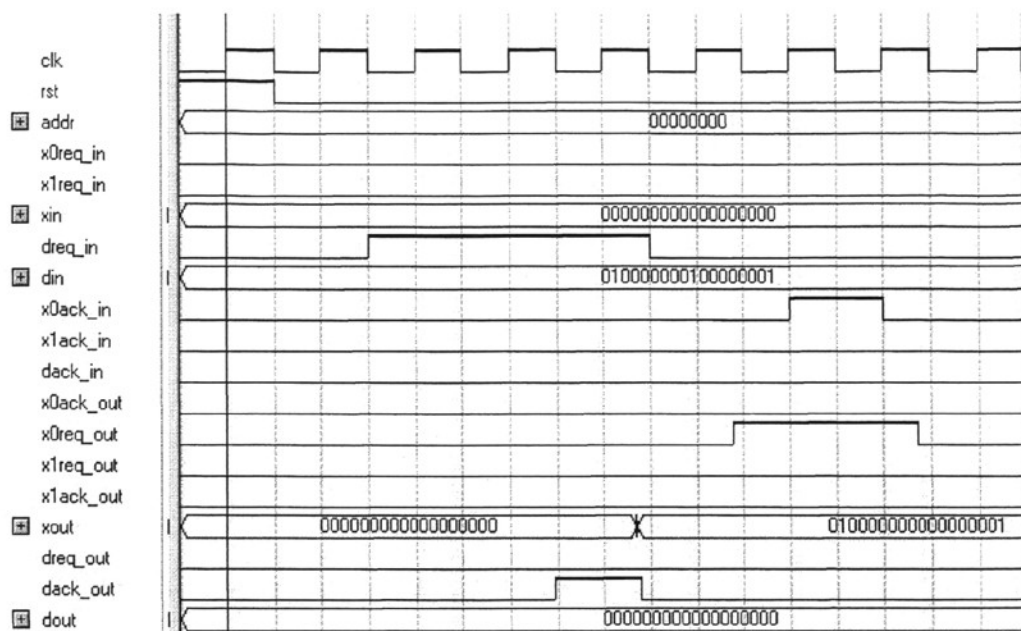


图 5-2 握手协议仿真波形

下面再以一次完整的消息包传输过程进行仿真验证，该仿真同样是对单个子路由模块进行的仿真，该仿真验证过程是用两组不同方向的数据的同时访问路由模块，一组数据是由本地节点向其他 X 维节点发送数据，一组是由 X 维的邻节点向本节点的 Y 维子节点发送数据。波形如图 5-3 所示。

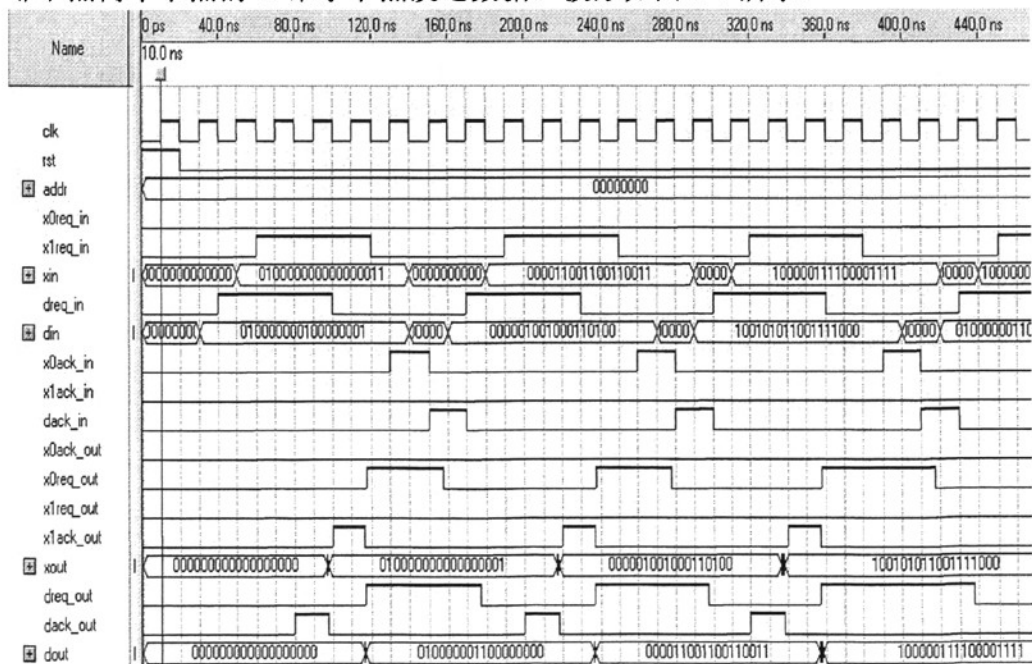


图 5-3 子路由模块数据传输仿真波形

该仿真主要考虑以下几个因素，首先是数据微片类型的处理问题，其次是数据的顺序，准确性问题，再次是路径选择问题等。由仿真结果显示，din 输

入的数据，头微片的路由选择是 0 虚拟通道，且头微片的数据信息发生了变化，X 维地址自动减 1，在后面的几个时钟周期里传输的是数据微片和尾微片，路由没有发生变化，且数据信息也没有变化，验证正确。再看 xin 输入的数据，使用虚拟通道 1 进行请求，由于 xin 输入的头微片信息的地址在 X 维地址已经是 0，所以 xin 的数据经过处理后自动进入 dout 缓冲区，再由 dreq_out 信号请求进行数据传输，其后面的数据微片和尾微片的传输也显示仿真验证过程是正确的。

通过分析以上时序仿真波形以及对路由节点对数据微片的处理过程，我们可以确认该路由模块能够正确实现我们预期的所有功能。

5.4. 路由节点的硬件参数分析

路由节点是片上网络的关键组件，它所能达到的最高时钟频率(运行速度)、芯片面积消耗量、数据吞吐率、包传输延迟等性能参数对构建片上网络具有重要意义。下文将给出我们设计的片上网络路由节点的性能分析，需要说明的是，分析的数据样本均基于 ALTERA 公司 Stratix III^[46] 系列 FPGA 上进行综合和布局布线的门级网表。

5.4.1. 运行速度

在对路由节点行为模型进行综合时，我们将 EDA 软件的综合策略设置为均衡电路的速度和面积，并且对模型中的状态机，存储器，冗余逻辑进行了优化。通过测试，路由节点可以达到的最高工作时钟频率为 263 MHz，最小时钟周期为 3.8 ns。

5.4.2. 芯片面积消耗量

实现路由节点所消耗的面积，即电路规模是路由节点性能的重要衡量指标。我们设计的路由节点在 FPGA 上完成布局布线后的结果如图 5-4 所示。

| | |
|---------------------------|--|
| Flow Status | Successful - Tue Mar 31 01:19:08 2009 |
| Quartus II Version | 8.0 Build 215 05/29/2008 SP 0.01 SJ Full Version |
| Revision Name | ECUBE_ROUTER |
| Top-level Entity Name | ECUBE_ROUTER |
| Family | Stratix III |
| Device | EP3SL150F780C4ES |
| Timing Models | Preliminary |
| Met timing requirements | N/A |
| Logic utilization | < 1 % |
| Combinational ALUTs | 595 / 113,600 (< 1 %) |
| Memory ALUTs | 0 / 56,800 (0 %) |
| Dedicated logic registers | 309 / 113,600 (< 1 %) |
| Total registers | 309 |
| Total pins | 146 / 488 (30 %) |
| Total virtual pins | 0 |
| Total block memory bits | 0 / 5,630,976 (0 %) |
| DSP block 18-bit elements | 0 / 384 (0 %) |
| Total PLLs | 0 / 4 (0 %) |
| Total DLLs | 0 / 4 (0 %) |

图 5-4 芯片面积开销

芯片上的逻辑资源是有限的，如果路由节点相对于资源节点中的其他 IP 核占用的芯片面积过大，则会影响片上网络的规模。关于这个问题，相关文献上建议，片上网络通信组件（路由节点）所占用的芯片面积一般不超过 IP 核的 20%，否则实现片上网络的代价过高，需要简化电路设计或相关功能^[25]。从片上网络路由节点的设计结果来看，其耗费的 ALUT（adaptive look up table）数量为 595 个单位。根据我们在对比试验中得出的结论，在同型号的 FPGA 上实现 32 位的 NiosII 处理器 IP 核大约需要耗费 2700 个单位的 ALUT，实现 32 位的 C68000 处理器大约需要耗费 4000 个单位的 ALUT。片上网络路由节点的芯片面积消耗量和 NiosII 处理器相比为 16.6%，和 C68000 处理器相比为 11.2%。可见我们提出的路由节点芯片面积消耗量符合要求。

第六章 总结与展望

6.1. 工作总结

随着集成电路产业的飞速发展, NoC 技术已成为下一代集成电路的主流设计技术。使用多处理器系统代替传统的单处理器系统, 在提高系统并行性方面显示出了巨大的优势。迄今为止, 国际上对于 NoC 的研究多集中于拓扑结构和路由器设计以及路由算法方面的研究等。

本设计的研究工作主要包括两个方面, 一是基于二维 Mash 网格拓扑结构上的路由算法的研究, 在基于 Turn Model 模型研究的基础上, 对 XY 路由算法进行了改进, 提出了一种 XY-YX 的路由算法, 并且经过证明该算法满足防死锁的条件, 最后在 NoC 仿真平台 NIRGAM 上进行了仿真, 仿真结果和 XY, mini OE 算法进行了比较, 结果显示具有较好的性能。

二是基于对二维 Torus 网络拓扑结构系统研究的基础上, 设计了一个支持 E-cube 维序路由算法, 虫孔交换机制的路由节点, 路由节点的设计是用 Verilog HDL 语言在 Quartus II 开发环境下进行的, 并且该设计通过了 RTL 级的仿真验证, 并满足了通信节点的功能要求, 实验结果表明该路由节点占用了较少的系统资源。

6.2. 未来展望

本文根据二维 Torus 网络拓扑结构系统的特点, 设计了一种可以在二维 Torus 拓扑结构上进行扩展的路由节点。该节点的设计采用的是虫孔交换机制, 缓冲区的大小即是微片的大小, 这样的设计不仅提高了运行效率, 而且还在很大程度上节省了系统资源的硬件开销, 其次路由节点采用 e-cube 路由算法, 路由节点的标识使用二进制格雷码进行编码, 这样便于硬件实现, 提高系统运行的效率。为后续的 NoC 硬件平台的搭建打下基础。

进一步的研究工作可从以下几方面展开:

深入研究各种路由算法, 通过权衡路由算法的优缺点和硬件实现的优缺点, 将路由算法通过硬件来实现, 并最终在 F P G A 开发板上进行仿真验证。

其次在 NoC 硬件平台搭建的问题上, 我们还需要做很多工作, 路由节点是一个可以重构的 IP 核, 但是一个完整的 NoC 平台还需要有资源节点和资源网络接口, 资源节点可以采用常用的 IP 核比如 Altera 的 Nois II 软核或者 ARM 核等等, 资源网络接口主要是负责路由节点和资源节点的通信, 这个需要根据实际需求自己定制, 最后还需要将这三组模块进行组合连接通信。而对于同构的 NoC 网络, 资源节点和网络资源接口是可以重构的, 而对于异构的 NoC 网络, 资源节点都不一样, 所以在设计的时候要考虑不同异构网络资源节点的接口的实现统一问题等等。

再次在 NoC 硬件平台搭建完成后，我们需要对该平台进行验证，所以还需对每个资源节点写软件操作系统来进行验证。从软件支持的角度，研究任务在多核之间的动态调度。实现功能划分的最佳方法是由操作系统动态地进行任务调度。因此，在整个系统中集成一个操作系统，根据各处理器不同的特性以及当前的工作状态，实时地对其进行任务调度将对系统性能的提升起到关键作用。利用可重构技术建立通用的多核系统平台，设计通用的多核通讯接口。异构多核的最大特点就是不对称性，不同的处理器在接口、时序等方面都存在不匹配，如何实现他们之间的通讯是设计多核系统的难点。

在上述研究的基础上，需要建立一个友好的用户界面，使用户可以很灵活的配置多核系统，并且通过软件可以对各系统的性能进行比较。通过这个界面，用户可以在多个处理器中，根据对处理器属性的分类合理选择系统需要的处理器，在通用的多核系统平台上搭建系统，然后输入需要完成的应用，通过软件自动的为各处理器分配执行任务，每次系统执行完任务将给出对系统的性能分析报告。用户可以反复选择不同的处理器进行实验，通过比较每次性能分析报告选择出最适合用户应用程序的多核系统。

参考文献

- [1] L.Benini, G.De Micheli. Networks on chip: a new SoC paradigm [J]. IEEE Computer, vol.35, no.1, Jan, 2002.
- [2] M.Sgroi, M.Sheets, A.Mihal et al. A.Sangiovanni-Vincentelli, Addressing the system-on-chip interconnect woes through communication-based design [A].38th Design Automation Conference (DAC'01) . June 2001.
- [3] S.Kumar, et al, A network on chip architecture and design methodology [A]. IEEE Computer Society Annual Symposium on VLSI (ISVLSI'02). April 2002.
- [4] D.Pham et al. The Design and Implementation of a First-Generation CELL Processor [A] ISSCC Dig.Tech.Paper, 2005.
- [5] ITRS1999, International Technology Roadmap for Semiconductors [C], 1999.
- [6] S. Kumar, A. Jantsch, J. Soininen, M. Forsell. A networks on chip architecture and design methodology [J], Proceedings of IEEE Computer Society Annual Symposium on VLSI. 2002: 105 - 112.
- [7] W. Dally, B. Towles. Route Packets, Not Wires: On-Chip Interconnection Networks[C], Proceedings. the 38th Design Automation Conference, 2001 (6) : 684 - 689.
- [8] Rashinkar P. System-On-a-Chip Verification Methodology and Techniques [M]. USA: Kluwer Academic Publishers, 2001: 6- 10.
- [9] Sarbazi-Azad, H.Ould-Khaoua, M.Mackenzie, L.M. Performance analysis of k-ary n-cubes with fully adaptive routing Parallel and Distributed Systems[C], 2000. Proceedings. Seventh International Conference on Publication Date: 2000 : 249-255.
- [10] Leiserson, Fat-trees: Universal Networks for Hardware-Efficient Supercomputing [J], IEEE Transactions on Computers, 34(10), 1985: 892-901.
- [11] F. Karim et al. An Interconnect Architecture for Networking Systems on Chips [J], IEEE Micro, 22(5), 2002: 36- 45.
- [12] Marcello Coppola, Riccardo Locatelli, Giuseppe Maruccia, et al, Spidergon: a novel on- chip communication network[C], System-on-Chip, 2004. Proceedings. 2004 International Symposium on, 2004: 15-18.
- [13] P. Guerrier and A. Greiner, A Generic Architecture for On- Chip Packet-Switched Interconnections, Design, Automation and Test in Europe

- Conference and Exhibition 2000[C]. Proceedings, 2000: 250- 256.
- [14] W.J. Dally and C.L. Seitz, The Torus Routing Chip [J], Technical Report 5208: TR: 86, Computer Science Dept, California Inst. Of Technology, 1986: 1- 19.
- [15] IBM (PERCS) Mootaz Elnozahy. PERCS: IBM effort in HPCS [EB/OL]. <http://www.ncsc.org/casc/meetings/vision-public.pdf>. 2003.
- [16] Cray Inc [EB/OL]. The cascade project. <http://www.cray.com/cascade/2002>.
- [17] P.P. Pande, C. Grecu, A. Ivanov and R. Saleh, Design of a Switch for Network on Chip Applications [J], Proc. Int'l Symp. Circuits and Systems (ISCAS), vol. 5, 2003: 217- 220.
- [18] BeniniL, De Micheli G, Networks on chips: a new SoC paradigm[J], IEEE Computer, Jan, 2002: 70-78.
- [19] Christopher J. Glass and Lionel M. Ni. The Turn Model for Adaptive Routing[C], the 19th Annual International Symposium on Computer Architecture, NEWYORK: ACM, 1992: 278-287.
- [20] NIRGAM Manual [M], University of Southampton UK and Malaviya National Institute of Technology India: Lavina Jain, 2007.
- [21] Ge-Ming Chiu, The Odd-Even Turn Model for Adaptive Routing [J], IEEE Transactions on Parallel and Distributed System, 11(7), 2000: 729-738.
- [22] Willian J. Dally and C. L. Seitz, Deadlock-free message routing in multiprocessor interconnection networks[J], IEEE Transactions on Computers, C-36 (5), 1987: 547-553.
- [23] Cristian Grecu, Andrè Ivanov, Partha Pande, et al. Towards Open Network-on-Chip Benchmarks[C]. Proceedings of the First International Symposium on Networks-on-Chip (NOCS'07), Princeton, New Jersey, 2007:205~208.
- [24] N. Sherwani, Algorithms for VLSI Physical Design Automation [M], Kluwer Academic Publishers, 1995: 187-189.
- [25] Pinto et al. Efficient Synthesis of Networks on Chip, Proc[C].Int'l Conf. Computer Design 2003, 2003: 146-150.
- [26] Chang Wu, Yubai Li, Song Chai, Design and Simulation of a Torus Structure and Route Algorithm for Network On Chip[C], the 7th International conference on ASIC, Guilin, China, 2007.
- [27] IBM Corporation. The Connect Bus Architecture [EB/OL]. <http://www.chips.ibm.com>, 1999.
- [28] ARM Corporation. The AMBA Specification [EB /OL]. www.arm.com, 1999.

- [29] Silicore Corporation.the WISHBONE System Architecture Specificatio [OL/EB]. www.silicore.com/wishbone.htm, 2001.
- [30] OMI, PI-bus VHDL toolkit, version3.1, open microprocessor systems initiative [S].1996:35.
- [31] Altera Corporation. The Avalon Specification Reference Manual [OL/EB].http://www.altera.com.cn/literature/manual/mnl_avalon_bus.pdf.
- [32] 葛晨阳,徐维朴,孙飞. IP 复用技术的研究[J]. 微电子学, 32 (4), 2002: 257 - 260.
- [33] 高明伦, 杜高明, NoC: 下一代集成电路主流设计技术[J]. 微电子学, 36(4), 2006.
- [34] 周干民. NoC 基础研究[D]. 合肥: 合肥工业大学博士学位论文. 2005.
- [35] Ni.L.M, McKinley.P.K.A survey of wormhole routing techniques in direct networks [J], Computer, 26(2), 1993: 62-76.
- [36] Zeferino.C.A, Kreutz.M.E, Susin.A.A, RASoC: a router soft-core for networks-on-chip[C], In Proceedings of Design, Automation and Test in Europe Conference and Exhibition Designer'Forum, 2004: 198-203.
- [37] Edgard de Faria Corrêa, Leonardo Alves de Paula e Silva, Flávio Rech Wagner, et al.Fitting the router characteristics in NoCs to meet QoS requirements[C], In Proceedings of the 20th annual conference on Integrated circuits and systems design, 2007: 105-110.
- [38] Nilanjan Banerjee, Praveen Vellanki, Karam S. Chatha, A Power and Performance Model for Network-on-Chip Architectures[C], In Proceedings of the Design, Automation and Test in Europe conference and Exhibition(DATA'04), 2004: 1250-1255.
- [39] Everton Carara, Fernando Moraes, Ney Calazans, Router architecture for high-performance NoCs[C], In Proceedings of the 20th annual conference on Integrated circuits and systems design, 2007: 111-116.
- [40] Cong, J, et al. Relaxed Simulated Tempering for VLSI Floorplan Designs[C], Proc. of ASP Design Automation Conference, 1999.
- [41] Ran Ginosar, Fourteen Ways to Fool Your Synchronizer[C], In Proceedings of the Ninth International Symposium on Asynchronous Circuits and Systems, 2003.
- [42] David.S.Bormann and Peter Y.K.Cheung, Asynchronous wrapper for heterogeneous systems[C], In Proc. International Conf. Computer Design (ICCD), 1997.

- [43] J. Duato, S. Yalamanchili, L. NI 著, 谢伦国等译, 并行计算机互连网络技术——一种工程方法[M]. 电子工业出版社, 2004.
- [44] John D. Owens. William J. Dally. Ron Ho, et al. Research Challenges for On-Chip Interconnection Networks[J], IEEE Micro, 27(5), 2007: 96-108.
- [45] Mininoc, [EB/OL], <http://www.es.ele.tue.nl/mininoc/>.
- [46] <http://www.altera.com.cn/products/devices/stratix-fpgas/stratix-iii/st3-index.jsp>.
- [47] Akash Kumar, Andreas Hansson, Jos Huisken, et al. An FPGA Design Flow for Reconfigurable Network-Based Multi-Processor Systems on Chip[C], Proceedings of the IEEE Design, Automation & Test in Europe Conference & Exhibition, DATE '07, 2007:1-6.
- [48] Li Ping Sun, El Mostapha Aboulhamid, David J.P, Network on chip using a reconfigurable platform[C], In Processing of Midwest Symposium on circuits and systems, 2003: 819-822.

附 录

➤ 攻读硕士学位期间撰写的论文

欧阳一鸣, 董少周, 梁华国. 基于 2D mesh 结构的 NoC 路由算法设计与仿真. 计算机工程. (已录用)

➤ 研究阶段参加的科研项目

自然科学基金重大研究计划 (90407008);

国家自然科学基金重点项目 (60633060);

安徽省自然科学基金 (050420103)。

➤ 实验环境及设计软件

PC 机

Linux (Redhat 8.0 or Redhat 7.2) 操作系统

GNU C++ compiler version gcc-2.95.3 or gcc-3.2.3

SystemC-2.1.v1

Nirgam-1.1

Windows XP 操作系统

Quratus II 8.0

ModelSim SE 6.2