

**毕 业 论 文**

题 目 深度学习加速器片上网络



自适应路由模块设计与实现



姓 名 刘芸裴



学 号 14073205



指导教师 蔡旻



日 期 2018 年 5 月



**独 创 性 声 明**

本人声明所呈交的论文是我个人在导师指导下进行的研究工作及取得的研究成果。尽我所知，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，也不包含为获得北京工业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了谢意。

签名： 日期：



**关于论文使用授权的说明**

本人完全了解北京工业大学有关保留、使用学位论文的规定，即：学校有权保留送交论文的复印件，允许论文被查阅和借阅；学校可以公布论文的全部或部分内容，可以采用影印、缩印或其他复制手段保存论文。

（保密的论文在解密后应遵守此规定）

签名： 导师签名： 日期：



[摘要 6](#_Toc515839649)

[Abstract 7](#_Toc515839650)

[1绪论 8](#_Toc515839651)

[1.1片上网络概述 8](#_Toc515839652)

[1.2片上网络研究现状 9](#_Toc515839653)

[1.3 NoC关键技术难点 10](#_Toc515839654)

[1.4 NoC的缺点 10](#_Toc515839656)

[1.5 NoC的优点 12](#_Toc515839663)

[1.6路由器结构 13](#_Toc515839671)

[1.7本文的基本结构 14](#_Toc515839672)

[2相关技术 15](#_Toc515839673)

[2.1蚁群算法 15](#_Toc515839674)

[2.2.1定义 15](#_Toc515839675)

[2.2.2背景 15](#_Toc515839676)

[2.2.3工作原理 16](#_Toc515839677)

[2.2.4 Zynq-7000系列 16](#_Toc515839678)

[2.3 Verilog 18](#_Toc515839679)

[2.4 Chisel 18](#_Toc515839680)

[2.5 本章小结 19](#_Toc515839681)

[3.XY路由算法 19](#_Toc515839682)

[3.1 XY路由算法 19](#_Toc515839683)

[3.2 XY路由的搭建 20](#_Toc515839684)

[3.3本章小结 21](#_Toc515839685)

[4.奇偶转弯路由器 21](#_Toc515839686)

[4.1 奇偶转弯模型 21](#_Toc515839687)

[4.2奇偶转弯性能评估 22](#_Toc515839688)

[4.3奇偶转弯规则 22](#_Toc515839689)

[4.4奇偶转弯模型算法 23](#_Toc515839690)

[4.5本章小结 24](#_Toc515839691)

[5 路由器与网络模块设计与实现 25](#_Toc515839692)

[5.1 路由模块 25](#_Toc515839693)

[5.1.1 Config模块 25](#_Toc515839694)

[5.1.2 Demux模块 25](#_Toc515839695)

[5.1.3 Direction模块 26](#_Toc515839696)

[5.1.4 Network模块 26](#_Toc515839697)

[5.1.5 NetworkIO模块 27](#_Toc515839698)

[5.1.6 Packet模块 27](#_Toc515839699)

[5.1.7 Router模块 28](#_Toc515839700)

[5.1.8 RouterIO模块 29](#_Toc515839701)

[5.2 测试模块 29](#_Toc515839702)

[5.2.1RouterTester模块 29](#_Toc515839703)

[5.2.2 XYRoutingExampleTester模块 30](#_Toc515839704)

[5.2.3 OddEvenRoutingExampleTestermo模块 32](#_Toc515839705)

[结论 33](#_Toc515839706)

[参考文献 34](#_Toc515839707)

[致谢 35](#_Toc515839708)

# 摘要

深度学习作为一类主流的机器学习算法，在图像识别、语音识别以及自然语言处理等多个领域有着广泛应用。然而深度学习应用程序具有计算量大、访存密集且不规则等特点，传统的CPU和GPU等多核体系结构受自身限制，无法有效满足深度学习应用的硬件需求。而FPGA作为一种灵活高效的半定制化硬件技术，被学术界广泛选作面向特定应用的加速器平台技术。

本文主要使用Scala语言和一定的C语言，在Intellij IDEA平台上，搭建了XY路由器和奇偶转弯路由器，并且实现了它们的一些基本功能。主要通过前期对NoC的结构和背景以及FPGA的背景和型号进行了一定的了解。了解到了国内外对于Noc在FPGA上开发的一些基本情况。接着学习路由器的结构，最后自己动手编写路由器。从路由器各个模块的搭建，到两种路由算法的实现，最后进行了测试模块的编写并对之前编写的算法进行了测试，验证了其正确性。通过本次的学习，知道了路由器搭建个基本过程和注意事项。同时也对其他更高等的自适应算法进行了一定的了解。知道了这些算法的实现对于FPGA开发的重要性。

关键词：深度学习，加速器，片上网络，自适应路由，FPGA

# 

# Abstract

As a kind of mainstream machine learning algorithm, deep learning is widely used in image recognition, speech recognition and natural language processing. However deep learning application has a large amount of calculation and to fetch intensive and irregular, traditional CPU and GPU multi-core architecture restricted by their own, unable to effectively meet the hardware requirements of deep learning applications. As a flexible and efficient semi-customized hardware technology, FPGA has been widely selected as accelerator platform technology for specific applications in the academic world.

In this paper, Scala language and a certain C language are mainly used to build XY router and odd-even turning router on Intellij IDEA platform, and some basic functions of them are realized. The structure and background of NoC, as well as the background and model of FPGA were studied. I have learned some basic information about the development of Noc on FPGA at home and abroad. Then learn the structure of the router and write the router yourself. From the construction of each module of the router to the implementation of two routing algorithms, the test module is finally written and the algorithm previously written is tested to verify its correctness. Through this study, I got to know the basic process of router building and matters needing attention. At the same time, we also have some knowledge of other higher adaptive algorithms. The importance of these algorithms for FPGA development is known.

Keywords: deep learning，accelerator，NoC，adaptive routing，FPGA

# 1绪论

# 1.1片上网络概述

片上网络network-on-chip（简称NoC）是[片上系统](https://baike.baidu.com/item/%25e7%2589%2587%25e4%25b8%258a%25e7%25b3%25bb%25e7%25bb%259f) system-on-chip（简SoC）的一种全新的通信方法。它是[多核技术](https://baike.baidu.com/item/%25e5%25a4%259a%25e6%25a0%25b8%25e6%258a%2580%25e6%259c%25af)的主要组成部分。NoC方法是一种全新的片上通信方法，其性能明显比传统总线式系统（bus）的性能要高出很多。基于NoC的系统能更好地适应在未来复杂多变的多核SoC设计的机制。NoC构架主要基于光学或电子技术，分别称片上光网络（optical network-on-chip简称ONoC）以及为片上电网络（electrical network-on-chip简称ENoC）。[1]

NoC的网络结构（如图1.1所示）NoC片上网络模型在一定程度上借鉴了TCP/IP的网络模型，它由5层构成，分别是：应用层，传输层，网络层，数据链路层和物理层。[2]

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| OSI模型 |  | TCP/IP模型 |  | NoC模型 |
| 应用层 | 应用层 | 应用层 |
| 表示层 |
| 会话层 |
| 传输层 | 传输层 | 传输层 |
| 网络层 | 网络层 | 网络层 |
| 数据链路层 | 主机至网络层 | 数据链路层 |
| 数据链路层 | 物理层 |

图1.1 三种网络模型图

物理层：物理层位于整个模型的最低层，因为它是整个网络模型的基础。它为网络设备组件的数据通信提供传输媒体以及互连设备,为数据传输提供可靠的环境。物理层的相关协议产生，检测并且监测物理电压，以便于发送和接收携带着数据信息的信号,它规定了整个网络使用的高地电平、信息的位宽以及时针频率等重要的物理参数。

数据链路层：数据链路层位于物理层的上一层,即处在网络模型的第二层。它同时连接着第一层物理层和第三层网络层,将原始的电路信号转变为逻辑信号,同时，它将网络层传输来的信息进行分拆传送给物理层。由于信号在物理设备之间传送时会受到各种各样的因素影响,为了弥补物理层的这一缺点,同时为了能够为网络层提供可靠的，0错误的数据服务，那么久需要数据链路层具备错误控制功能,错误检测功能以及纠错功能。与此同时,它本身还具备了模块同步，流量控制以及能量耗损控制等诸多方面的功能。是功能强大的一层。

网络层：网络层是NoC网络模型中的第三层,它分别连接着第二层数据链路层与第四层传输层。它主要实现的事路由功能, 将源节点路由到目标节点数据进行分组。它将综合考虑从整个网络到每个模块的运行状态，拥堵程度程度,服务质量和具体的路由开销等诸多因素，以此来为数据的分组规划出一条最佳的路径,最终完成数据传输的功能。与此同时,网络的服务质量，拥堵控制等问题，也是需要在网络层解决的问题。

传输层：传输层是NoC网络模型中的第四层，位于网络层之上,是NoC网络模型中最为重要的一层。它主要负责端到端的传输,并且将应用层的信息划分为不同的数据分块，并且根据服务要求的不同，完整且精准地将分块由源端传输到目的端。与此同时，它也会负责处理网络流量控制等问题。

应用层：应用层是NoC网络模块的第五层。它的功能是针对用户的需求提供用户交互接口和一些其它的网络服务。

## 1.2片上网络研究现状

在国外，NoC概念刚刚被提出不久，它的相关技术研究便马上成为了一个十分热门的前沿学术领域，并且吸引了越来越多的著名科研机构以及集成电路设计的大公司投入到这个领域进行研究。[3]其中比较具有影响力的院校有：博洛尼亚大学、斯坦福大学、卡梅隆大学、普林斯顿大学等学校。 此外，一些举世闻名的集成电路设计公司如飞利浦、阿尔法特和因特尔等公司也纷至沓来，加入到了NoC的研究行列当中。这足以说明NoC将会对未来产生多么大的影响。

完整的NoC设计方法学目前是存在着诸多方面的问题的，它们对于NoC的发展都是起到了至关重要的作用的，并且，这些问题也已经在学术界引起了的广泛研究和讨论。卡梅隆大学的学者提出了NoC设计空间的概念，并且将NoC的研究问题，归纳为三大类关键性问题：基础架构、通讯机制、映射优化。

## 1.3 NoC关键技术难点

NoC关键技术主要包括系统建模、拓扑结构、路由方法、交换方法、缓存策略、服务质量和映射优化等，近年来都不同程度地取得研究进展。

## 1.4 NoC的缺点

（1）存储结构问题。储器是NoC中十分重要的组件，在现有的片上多处理器系统中，存储器占到百分之七十的芯片面积，并且在不久的将来这个数值会上升到百分之九十。从能耗的角度，存储器的功耗也高达系统功耗的百分之九十，这对芯片的散热、封装和可靠性等诸多方面都带来了严重的问题。NoC系统需要大量的存储元件，并被组织成复杂的存储子系统（memory subsystem），这个存储子系统将支持NoC的并行数据存储、传输及交换。NoC中大量的存储资源必将占用多个路由节点，且由于处理单元与存储资源之间的数据交换非常频繁，若在数据包传输路径上路由节点数目过多，会带来很大的通信延时。如何有效缩短源节点到目的节点间的距离对提高整个NoC系统性能十分关键。

（2）软件并行化问题。未来，基于多核的高性能处理芯片可能会遇到一些困难。比如一些对于使用传统的串行程序自动并行化方法较难实施的应用。如果不能有效地利用NoC片上并行处理资源，那么并行计算的实际性能将会大打折扣。因此，如何通过有效的模型和方法，高效地利用NoC的众多处理单元，并且最大程度地降低应用的开发难度，就成为了当下迫在眉睫的问题。 [4]  
 与并行计算机发展过程中遇到的问题大同小异，NoC并行处理体系结构所面临的主要问题是如何将应用中蕴含的不同粒度、不同层次的并行性效率最大化地提取出来并且能够映射到多核的并行硬件结构上去。这一问题的解决涉及到包括程序设计语言、程序设计模型、硬件支撑和编译系统等在内的诸多方面。   
 总而言之，解决开发并行程序问题可以有三种方法：

1.串行程序自动并行化。这条路目前还处于试验阶段，所以对于现阶段来说，更为贴合实际的目标的行为，是在人机交互方面实行自动并行化。

2．设计全新的并行程序设计语言。由于现有的并行程序设计语言太过繁琐，而且普及度低，我们可以通过设计一种新的并行程序设计语言来实现开发并行程序。但是这种方法存在的弊端就是需要全部改写原有程序，工程量十分庞大，对于用户来说成本和风险也很高，并且还不能够保证效率。但是，随着多核的出现，若想把并行计算环境向大众推广，就必须有一种全新的，并且是容易被接受的并行程序设计语言。[5]

3.把串行语言加并行库或进行伪注释制导语句的扩展，即增加一个库或一些新的制导语句。这是目前相比较其他途径而言，比较容易被大众所接受，并且性能相对较高的途径。但它存在的弊端是，程序开发效率低，难度也相对比较大。

（3）功耗管理问题。虽然NoC能够显著提高芯片的能效（energy-efficiency），但它存在着一个不容忽视的问题。由于多核系统片上集成规模的骤然猛增，功耗问题尤其突出。如何在NoC设计中提高芯片的能效，对各种计算资源进行合理调度，分配从而使得功耗最大限度降低仍旧是NoC设计所面临的诸多难题之一。   
 NoC的结构主要包括：核间互连、处理器核，及片上存储这三个主要部分。由此我们分析得出，NoC的低功耗研究可以围绕处理器核功耗优化、功耗评估、片上存储功耗优化和片上网络功耗优化这四个方面，对NoC的各个部分展开。其中NoC低功耗设计的基础便是功耗评估。   
 功耗是导致包括NoC在内的多核技术出现问题的最主要的诱因，也是片上多核处理器设计的最主要的限制因素。对于NoC的不同设计层次和设计模块，都存在着卓有成效的能降低功耗的方法，而这些方法相互之间可能是互相影响并且互相牵制的。因此，我们需要深入学习和了解从NoC体系结构到电路工艺等各方面的丰富知识，才能够在设计初期做出正确的多核架构的选择。就一般情况来说，想要获得更大的降低功耗的效率，就需要从更高的设计抽象层次着手考虑低功耗设计问题。

## 1.5 NoC的优点

NoC可以有效解决SoC诸多互连问题，具有明显的优势，具体表现在以下几方面：

重用性好。NoC重用性主要体现在通信服务的重用。从底层硬件来看，路由器、 互连线、上层通信协议、网络管理等诸多方面都可以重用。因此我们在设计全新的SoC芯片时，只需要将新的IP核与原有NoC软件硬件集成就可以了，这样可以有效缩短新产品设计的周期。

（1）扩展性好。片上路由器与IP核之间通过通用网络接口来协调，同时釆用NoC架构的片上系统，这样可以使得通信模块和系统计算模块之间相互独立，通过网络接口便可以实现即插即用的方式来配置SoC。与此同时，NoC地址空间在理论上可以做到无限扩展，因此，如果采用NoC的通信架构，片上集成的IP核数目将不会受到限制。

（2）预测性好。由于NoC的通信网络结构固定，物理布局规则，整个系统性能就会变得可以预测。固定的通信硬件和规则的物理布局十分有利于提前评估和分析系

统的性能。除此之外，通过构建精确的NoC系统原型和物理模型，还可以对系统的信号完整性进行准确评估。

（3）通信机制灵活。NoC在被提出之初，就采用了GALS通信机制。采用 NoC通信架构的片上系统，使得每个IP核都能拥有一个独立的时钟域。不同时钟域之间采用异步通信的方法实现数据传输，有效避免了全局性时钟域带来的数据偏差，时钟抖动等诸多问题。

（4）系统可靠性高。采用NoC作为通信架构，可以通过在片上路由器和网络接口中增加一些纠错和检错模块，用来处理由于信号串扰所带来的数据误差。除此之外，当路由器或者芯片中的某些部分连线出现问题时，NoC还可以通过一些容错手段来规避故障，以此保证数据传输的可靠性。[6]

## 1.6路由器结构

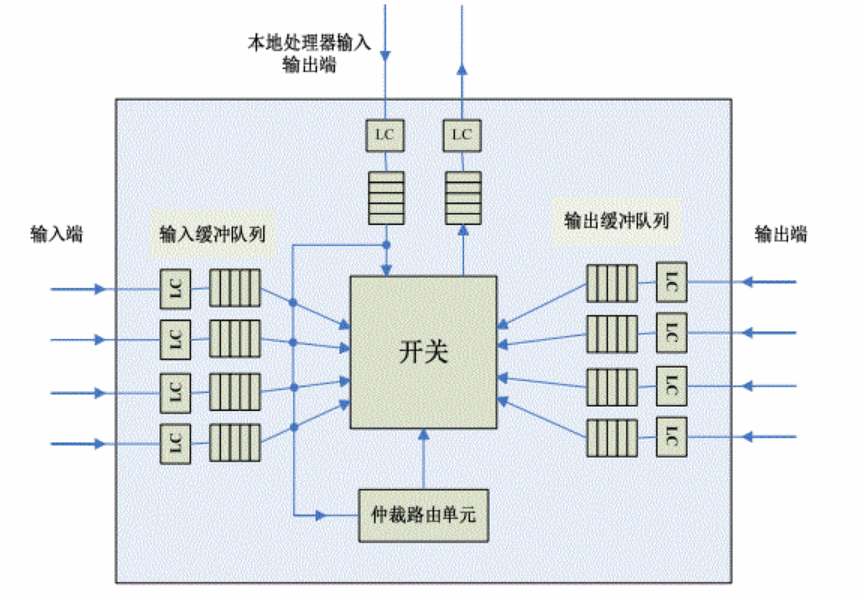
****

图1.2路由器的结构图

路由器是NoC的主要部件。它的主要功能是负责通信数据的发送和存储，并在模块内部完成仲裁，网络时序控制以及路由端口的选择等的功能。如图1.2所示，路由器由交叉开关、仲裁器（arbiter）、缓冲器（buffer）、以及输入端和输出端（IO）组成。

交叉开关的功能是连接输入输出缓冲队列。在开关内部,如果交叉点是关闭的,那么输入缓冲的数据可以传送至输出缓冲,反之将不可以传送。交叉点是打开状态或者闭合状态是由多路选择器（demux）来决定的。

仲裁器是路由器的核心单元。它的功能是对数据路的优先级进行仲裁功能。仲裁模块会根据程序中设定的仲裁机制，从仲裁表中选择一个分组进行发送,被选中的分组会被送到输出缓冲队列中等待发送,而其他没有被选中的分组要等待下次仲裁的到来。

因此，目前的NoC普遍采用输入缓冲结构，由于是队列，所以采的是先进先出（First-in First-out)结构。

如果路由器输入缓冲划分为多个队列，则每个缓冲队列可称为一个虚拟通道(Virtual Channel)。虚拟通道可以有效避免死锁（Dead lock free），但是需要较多的片上缓冲，因此对于芯片硬件的开销比较大。

## 1.7本文的基本结构：

本文共五章：

第一章讲述了Noc的一些基本概念，包括Noc目前的研究现状，NoC的优点和实现难点以及理论，实际两方面的解决方法，接着介绍了路由器的结构。最后说明了本文的基本结构。

第二章介绍了相关技术。包括相关的算法，FPGA的背景和工作原理，和实验用到的zynq-7000系列。包括Verilog语言和实验主要用到的Chisel语言的大致介绍与对比。此章只做大致上的简单说明，实际实现和设计算法的内容将会在第三章和第四章进行深层次的运用和讲解。

第三章介绍了路由算法中最基本的XY路由算法，包括它的结构和模块之间的关系以及如何用Scala语言实现它。因为XY路由算法是最基本的算法，实现了它我们才能实现更多其他算法。

第四章详细解释了奇偶转弯模型的具体实现过程。在这章中你将能知道我们如何在Scala上实现奇偶转弯算法。和上一章相结合来看，实现不同的模型的意义在于，给研究人员展现不同的模型之间，结构和算法以及实现思路的相同点与不同点。所以，多种模型的展示是非常有必要的。

第五章对两种模型进行整体性能测试，比较它们吞吐率的不同，从而对它们的性能进行评估。然后得出结论。

文章最后一部分介绍了论文的参考文献和致谢词。

# 2相关技术

## 2.1蚁群算法

蚁群系统是在研究蚂蚁觅食的过程当中，发现单个蚂蚁的行为比较简单，但是蚁群整体的行为中却可以体现出一些智能的行为。其中最为突出的是，蚁群可以在不同的条件下，寻找到达食物的最短路径。这是因为蚂蚁会在其经过的路径上释放“信息素”，告知后面的蚂蚁前面路径的信息。而蚁群内的每一只蚂蚁个体都对“信息素”具有识别能力，因此，它们会沿着“信息素”浓度较高路径行走，而每只路过的蚂蚁都会在路上留下“信息素”，这就形成一种正反馈的机制，如此循环一段时间后，整个蚁群就可以找到最短路径了。

蚁群算法根据模拟蚂蚁寻找食物的最短路径行为来设计的仿生算法，因此就一般而言，蚁群算法用来解决最短路径问题，并真的在各个领域上都取得了相对比较好的成效。目前，蚁群算法已应用到自适应的路由算法当中。[7]

### 2.2.1定义

FPGA（Field－Programmable Gate Array），即现场可编程门阵列，是一个含有可编程元件的半导体设备。它是在GAL 、PAL、CPLD等可[编程器](https://baike.baidu.com/item/%25e7%25bc%2596%25e7%25a8%258b%25e5%2599%25a8)件的基础上发展的产物。[8]它是一种半定制电路，它具备以下两个特点，一是避免了定制电路的不足，二是克服了原有可[编程器](https://baike.baidu.com/item/%25e7%25bc%2596%25e7%25a8%258b%25e5%2599%25a8)件门电路数有限的缺点。

### 2.2.2背景

以硬件描述语言所完成的[电路](https://baike.baidu.com/item/%25e7%2594%25b5%25e8%25b7%25af)设计，可以经过简单的综合与布局，快速的[烧录](https://baike.baidu.com/item/%25e7%2583%25a7%25e5%25bd%2595)至FPGA上进行测试，是现代IC[设计验证](https://baike.baidu.com/item/%25e8%25ae%25be%25e8%25ae%25a1%25e9%25aa%258c%25e8%25af%2581)的技术主流。这些可编辑元件可以被用来实现一些基本的逻辑门[电路](https://baike.baidu.com/item/%25e7%2594%25b5%25e8%25b7%25af)（比如AND、OR、XOR、NOT）或者更复杂一些的组合功能比如解码器或数学方程式。在大多数的FPGA里面，这些可编辑的元件里也包含记忆元件例如[触发器](https://baike.baidu.com/item/%25e8%25a7%25a6%25e5%258f%2591%25e5%2599%25a8)（Flip－flop）或者其他更加完整的记忆块。[9]

[系统设计师](https://baike.baidu.com/item/%25e7%25b3%25bb%25e7%25bb%259f%25e8%25ae%25be%25e8%25ae%25a1%25e5%25b8%2588)可以根据需要通过可编辑的连接把FPGA内部的逻辑块连接起来，就好像一个[电路](https://baike.baidu.com/item/%25e7%2594%25b5%25e8%25b7%25af)试验板被放在了一个[芯片](https://baike.baidu.com/item/%25e8%258a%25af%25e7%2589%2587)里。一个出厂后的成品FPGA的逻辑块和连接可以按照设计者而改变，所以FPGA可以完成所需要的逻辑功能。

FPGA一般来说比ASIC（专用集成电路）的速度要慢，实现同样的功能比ASIC电路面积要大。但是他们也有很多的优点比如可以快速成品，可以被修改来改正程序中的错误和更便宜的造价。厂商也可能会提供便宜的但是编辑能力差的FPGA。因为这些[芯片](https://baike.baidu.com/item/%25e8%258a%25af%25e7%2589%2587)有比较差的可编辑能力，所以这些设计的开发是在普通的FPGA上完成的，然后将设计转移到一个类似于ASIC的芯片上。另外一种方法是用CPLD（Complex Programmable Logic Device，复杂可编程逻辑器件)。

### 2.2.3工作原理

FPGA采用了[逻辑单元](https://baike.baidu.com/item/%25e9%2580%25bb%25e8%25be%2591%25e5%258d%2595%25e5%2585%2583)阵列LCA（Logic Cell Array）这样一个概念，内部包括可配置逻辑模块CLB（Configurable Logic Block）、输入输出模块IOB（Input Output Block）和内部连线（Interconnect）三个部分。现场可编程门阵列（FPGA）是可[编程器](https://baike.baidu.com/item/%25e7%25bc%2596%25e7%25a8%258b%25e5%2599%25a8)件，与传统逻辑[电路](https://baike.baidu.com/item/%25e7%2594%25b5%25e8%25b7%25af)和门阵列（如PAL，GAL及CPLD器件）相比，FPGA具有不同的结构。FPGA利用小型查找表（16×1RAM）来实现组合逻辑，每个查找表连接到一个D[触发器](https://baike.baidu.com/item/%25e8%25a7%25a6%25e5%258f%2591%25e5%2599%25a8)的输入端，触发器再来驱动其他逻辑电路或驱动I/O，由此构成了既可实现组合逻辑功能又可实现时序逻辑功能的基本[逻辑单元](https://baike.baidu.com/item/%25e9%2580%25bb%25e8%25be%2591%25e5%258d%2595%25e5%2585%2583)模块，这些模块间利用金属连线互相连接或连接到I/O模块。[10]FPGA的逻辑是通过向内部静态[存储单元](https://baike.baidu.com/item/%25e5%25ad%2598%25e5%2582%25a8%25e5%258d%2595%25e5%2585%2583)加载编程数据来实现的，存储在存储器单元中的值决定了[逻辑单元](https://baike.baidu.com/item/%25e9%2580%25bb%25e8%25be%2591%25e5%258d%2595%25e5%2585%2583)的逻辑功能以及各模块之间或模块与I/O间的联接方式，并最终决定了FPGA所能实现的功能，FPGA允许无限次的编程。

2.2.4 Zynq-7000系列  
如图2.1所示，Zynq-7000系列是全可编程片上系统，主要包含PS（processing system）和PL（Programmable Logic）两部分。PL采用28nm工艺；PS以2个CortexA9的ARM核为核心，还包括片上存储器、片外存储器接口（DDR）和一系列的外设接口。Zynq-7000系列将ARMCPU和外设集成在一个芯片内，使得Zynq-7000系列皆具处理器和FPGA双重特性，特别适用于软硬件协同设计。[11]

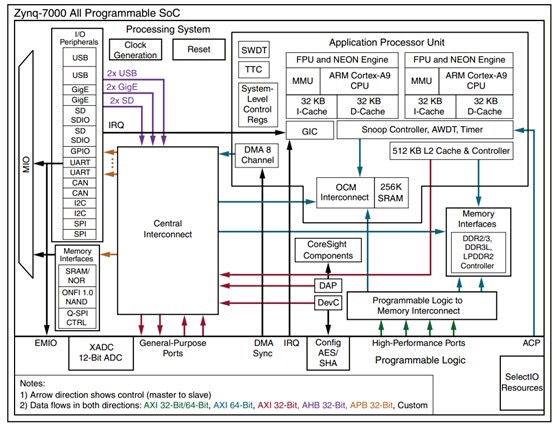


图2.1：Zynq-7000 All Programmable SoC 概览

PL和PS接口的两种类型：​

（1）功能接口。包括AXI接口、EMIO接口、中断、DMA流控制、时钟和调试接口。在FPGA开发人员设计PL模块时，就可以使用这些信号，从而和PS进行数据交互。​

（2）配置接口。包括PCAP（处理器配置访问端口）、SEU、配置状态信号和Program/Done/Init信号。​

Zynq中的AXI接口有3种，共9个：​

（1）AXI\_ACP接口：ARM多核架构下的一种接口，加速器一致性端口，用来管理DMA之类不带缓存的AXI外设，PS端是Slave接口。​

（2）AXI\_HP接口：高性能/带宽的AXI3.0标准接口，共有4个，PL模块作为主设备进行连接。​

（3）AXI\_GP接口：通用AXI接口，共有4个，包括两个32位主设备接口和两个32位从设备接口。​

## 2.3 Verilog

Verilog HDL是一种硬件描述语言，用于从算法级、门级到开关级的多种抽象设计层次的数字系统建模。被建模的数字系统对象的复杂性可以介于简单的门和完整的电子数字系统之间。数字系统能够按层次描述，并可在相同描述中显式地进行时序建模。

Verilog HDL语言从C编程语言中继承了多种操作符和结构。Verilog HDL提供了扩展的建模能力，其中许多扩展最初很难理解。但是，Verilog HDL语言的核心子集非常易于学习和使用，这对大多数建模应用来说已经足够。完整的硬件描述语言足够对从复杂的芯片到完整的电子系统进行描述。

## 2.4 Chisel

Chisel是UC Berkeley开发的一种开源硬件构造语言，它是一种建立在Scala语言之上的语言。Scala是一门多范式(multi-paradigm)的编程语言。Scala 源代码被编译成Java字节码，可以运行于Java虚拟机之上，并可以调用现有的Java类库。

Chisel是Scala语言在芯片/硬件设计领域的一个特定应用，预定义了一组专门的类和对象，利用Scala高级语言特性进行建模。

Chisel并不是Verilog的替代品，它目前还在不断地完善。它具备以下几个特点：

1.嵌入Scala编程语言。

2.代数构造与布线。

3.抽象数据类型和接口。

4.批量连接。

5.层次+面向对象+功能构建。

6.在Scala中使用元编程的高度参数化。

7.支持特定领域语言的分层。

8.相当大的标准库，包括浮点单元。

9.多时钟域。

10.生成低级别Verilog，设计用于传递到标准ASIC或FPGA工具。

11.基于修改后的BSD许可证的GITHUB开源。 

12.文档完整集。

## 

## 2.5 本章小结

本章总结了本文基于FPGA及型号Zynq-7000进行了简单的介绍。也对蚁群算法做了一个大概的介绍。由于时间有限，本次未能实现蚁群算法，但是后面主要介绍的XY路由算法和奇偶转弯路由算法和蚁群算法有很多相同之处。后面就是关于Verilog和Chisel这两种硬件语言进行了简单的介绍和对比。总的来说，Verilog这种硬件描述语言，来自于C语言，随处都是C语言的痕迹。对于C语言学习者来说，Verilog十分容易上手。而Chisel语言是建立在Scala之上的一种硬件构语言。语法类似Java，也可以被编译成java，并且Chisel还可以生成Verilog语言。Chisel语言也是本次实验中编写程序所使用的主要语言。

# 3.XY路由算法

## 3.1 XY路由算法

XY路由器是片上网络路由器中应用最为广泛的一种路由器，XY路由算法是一种确定性路由算法。它是基于2D mesh拓扑结构（二维网络结构）的 一种算法。XY路由算法可以说是最简单的一种，它的优点是通信延时低，适用于低拥堵的网络情况。所以当网络阻塞时，网络的性能将会出现不可避免的下降。XY路由算法的优点在于算法简单，易于实现，相对于其他算法来说开销较低。

2D mesh拓扑结构内的每一个通讯节点，都与相邻的节点直接相连接。输出排队的缓冲策略如图3.1所示，在该排队策略下，路由器在每个输出端口为每个输入端口提供一条虚拟通道的缓冲队列。

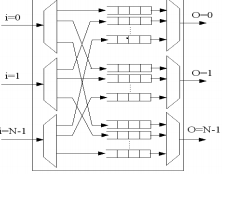


图3.1输出排队策略

这样，设计中的输入端口就和输出端口之间形成了一个全映射，在数据传输时，只需要在每个输出端口从几条输出虚拟通道中仲裁出一条输出即可。

## 3.2 XY路由的搭建

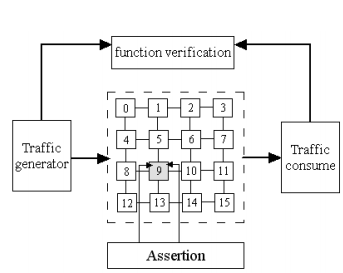
我们的拓扑结构是一个如图3.2的拓扑结构

图3.2XY路由器拓扑结构

在XYRouting模块中，我们具体实现了XY路由器的基本功能。我们首先通过当前地址取出X，Y坐标，通过目的地址取出destX，destY坐标。然后进行比较，如果当前当前X坐标大于目的destX坐标，说明应该向西边输出，反之向东。如果当前Y坐标大于目的destY坐标，那么说明应该向北输出，反之向南。通过这样运算，此我们就可以让包在路由节点之间相互输送了。具体算法如下图（图3.3 XY路由算法）。

val x = current x //定义x，使它的值为当前位置x坐标的值

val y = current y //定义y，使它的值为当前位置y坐标的值

val destX = dest x //定义destX，使它的值为终点位置x坐标的值

val destY = dest y //定义destY，使它的值为终点位置y坐标的值

when x > destX //当x 的值大于destX的值

dir = west //方向为西

elsewhen x < destX //当x 的值小于destX的值

dir = east //方向为东

when y > destY //当y的值大于destY的值

dir = north //方向为北

elsewhen y < destY //当y 的值大于destY的值

dir = south //方向为南

otherwise

dir = local //以上情况均不是，方向为本地

图3.3 XY路由算法

## 3.3本章小结

本章重点介绍实验主要使用的算法之一，XY路由算法，从它的机制到它的规则进行了详细的介绍。它的规则很简单，根据初始地址和目的地地址的XY坐标，遵循先X后Y的规则，到达下一节点后，再根据当前的地址和目的地址进行比对，再进行下一次数据的传输。最终将数据传输到达目的节点。

# 4.奇偶转弯路由器

## 4.1 奇偶转弯模型

奇偶转弯模型是通过限制某个方向的转弯而达到避免形成环的目的，但是其不是限制在某个固定位置的转弯，而是通过规则，把节点坐标分成奇节点和偶结点，通过在奇偶两种位置分别来限制不同的转弯规则，以此来避免形成环。在讲具体的定义之前，先来阐述一个基本概念，在一个环中一共有8种转弯，分别定义ES,EN,WS,WN,ME,NW,SE,SW。其中EN代表东向北的转弯，其它转弯符号的含义以此类推。

## 4.2奇偶转弯性能评估

衡量互连网络性能的两大最重要指标一个是时延另一个是吞吐率。就一般而言，我们说一种性能好的路由算法它具备低时延、高吞吐率的特点，反之，性能不好的路由算法具备高时延、低吞吐率的网络性能。下面我将重点介绍吞吐率的定义：

吞吐率：吞吐率是指对网络或者其他设施中，在单位时间内成功地传送的数据的数量。该参数指的是在网络中，单位时间内所发送的最大信息量，也代指网络可接收的最大流量。在仿真实验中，我们以网络中的每个节点，每个时钟周期所能接收的全部flit数目作为吞吐率的单位。在本次试验中，我们没有涉及到flit，只是停留在packet层面的传输。

## 4.3奇偶转弯规则

奇偶转弯模型有如下规则：

任何包不准在偶数列实施EN和ES转弯。

任何包不准在奇数列实施NW和SW转弯。

由以上规则我们可以推导出这样一个结果：

如果包在奇数列上进行了南北方向的上移动，那么接下来它将不能往东转弯。

如果包在偶数列上进行了南北方向上的移动，那么接下来它将不能往西转弯。

那么进而我们可以分析出，在奇偶转弯的规则下，不会形成环，所以也不会产生最右信道（rightmost channel)。同理，我们也可以通过改变以上的规则用来限制形成最左信道（Leftmost Channel)、最上信道（Upmost Channel)、最下信道（Downmost Channel)以此方式来达到死锁避免的目的。如图4.1所示，用0（odd）来代表奇数列，用E（even）来代表偶数列，实线代表允许的转弯的方向，虚线代表不允许的转弯的方向。按照上述规则，基于奇偶转弯模型的路由算法不可能形成最右信道rightmost， 所以由此可知，基于奇偶转弯模型的路由算法是无死锁（deadlock free）的。[4]

O E O E

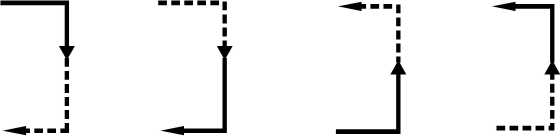


图4.1奇偶转弯规则

## 4.4奇偶转弯模型算法

我们设当前位置x坐标为c0，当前位置y坐标为c1，原地址x坐标为s0，目的地址x坐标为d0，目的地址y坐标为d1。再定义e0和e1，其中

e0 = d0 - c0，e1 = -(d1 - c1)。

根据上面提到的规则，如果包在奇数列上进行了南北的移动，那么接下来它将不能往东转弯。如果包在偶数列上进行了南北的移动，那么接下来它将不能往西转弯。我们就有了如如下图所示（图4.2 奇偶转弯算法）的算法。

if e0 == 0 //如果目的地和当前在一列

if e1 > 0 //目的地在当前位置北方

direction = north //向北输出

else

direction = south //向南输出

else if e0 > 0 //如果目的地在当前右侧

if e1 == 0 //讨论在同一行的情况

direction = east //向东输出

else if c0 % 2 == 1且c0 == s0 //当前在奇数列且当前位置和初始在同一列

if e1 > 0 //如果目的地在当前北方

direction = north //向北输出

else

direction = south //向南输出

if d0 % 2 == 1且e0 != 1 //如果目的地在奇数列，且目的地和当前地址同列或在相邻列

direction = east //向东输出

else

direction = west //向西输出

if c0 % 2 == 0 //如果当前在偶数列

if e1 > 0 //目的地在当前北方

direction = north //向北输出

if e1 < 0 //目的地在当前南方

direction = south //向南输出

图4.2奇偶转弯算法

## 4.5本章小结

本章我们对奇偶转弯路由器进行了详细的讨论，从它的定义，到它的算法。通过对奇偶模型算法的规则的了解，通过对奇偶两列分别制定不同的转弯禁止规则，限制成环，从而达到防止死锁的目的。我们也触类旁通的对其他防止死锁的算法进行了延展的思考，通过类似奇偶转弯的规则禁止某些方向的转弯。从而达到不同方向转弯算法。但并未讨论其结果。

# 5 路由器与网络模块设计与实现

## 5.1 路由模块

首先我们搭建了一下几个模块，分别是配置Config，多路分配器Demux，方向Direction，网络Network，网络输入输出NetworkIO，包裹Packet，节点Router，节点输入输出RouterIO， XY路由输入输出XYRoutingIO。各个模块之间相互关联，功能上相互配合，就搭建了路由器的框架。

### 5.1.1 Config模块

在Config模块中，我们定义了路由节点的5个端口方向数，其分别是东南西北和本地，这个在后面的Direction模块会详细的提到。由于我们的拓扑结构是一个4\*4的mesh，所以我们需要定义16个路由节点。然后宽度用路由节点数开方，用每一个节点的x坐标用它的id取余4，用y坐标用它的id整除4，就得到了它的xy坐标。然后定义其类型是UInt（无符号整型）。具体算法如下图（5.1 id地址算法）

def width = 4 //定义宽度为4

def x=id%4 //节点的x坐标为当前节点的序号对4取余

def y=id/4 // 节点的y坐标为当前节点的序号整除4

图3.7 config模块

图5.1 id地址算法

### 5.1.2 Demux模块

多路选择器Demux的功能就是要把上一个节点输出的packet从多个节点中选择一个，放到他的输入端。那么这里规定，如果输入端的状态不为ready，那么就不输出。如果状态为ready，那么我们才把packet放到输出端。我们这里还用到了zipWithIndex，这个功能实现的是把数组和索引连接在一起，那么在这里就是把out里面的元素都加上了索引。

在Demux多路选择器模块中，我们先定义了它的三个IO接口，如下面图表（5.2 Demux接口）。

|  |  |  |  |
| --- | --- | --- | --- |
| 名称 | select | in | out |
| 宽度 | 8.W | 8.W | 8.W |
| 描述 | Uint,Flipped | Decoupled | Vec,Decoupled |

图5.2 Demux接口

select，in，out。select顾名思义，就是选择端口，实现的是选择功能。宽度是8.W。它的数据类型是UInt，Flipped。in是输入端口，宽度是8.W。它的数据类型是Decoupled型。out是输出端口，宽度是8.W。它的数据类型是Vec，Decoupled型。

### 5.1.3 Direction模块

在Direction模块中我们定义了5个方向的端口，本地0，按照逆时针顺序，北为1，东为2，南为3，西为4。和他们分别的相反方向。即本地对本地，北对南，东对西，南对北，西对东。

### 5.1.4 Network模块

在Network模块中,我们对io的三个端口行了初始化，分别是in.ready，out.valid，out.bits。然后把routers的16个节点都映射了出来，然后把它的in.valid和in.bits以及out.ready端口进行了初始化。

接下来我们把这16个端口的东，南，西，北四个方向进行了连接。当然，这里的边界情况也进行了排除，比如在西北角的节点，西边和北边是没有节点的，所以只用链接东边和南边的端口。再比如在第一行除了最西边和最东边，两个角落的节点之外，北边都是没有节点的，所以不用连接北边的端口。以此类推。具体算法如下图（5.3端口连接方式）所示，这里我们设该节点的顺序为i，横坐标为x，纵坐标为y。

if y > 0 // 如果节点y坐标不在最北边

connect (i,i-4,direction north ) //把它和它序号减4（北边）的节点连接

if x < 4 // 如果节点x坐标不在最东边

connect (i,i+1,direction east ) //把它和它序号加1（东边）的节点连接

if y < 4 // 如果节点y坐标不在最南边

connect (i,i+4,direction south ) //把它和它序号加4（南边）的节点连接

if x > 0 // 如果节点x坐标不在最西边

connect (i,i-1,direction west ) //把它和它序号减1（西边）的节点连接

5.3端口连接方式

### 5.1.5 NetworkIO模块

NetworkIO这个模块的构造相对简单，这里我们只把它的in和out端口进行了初始化。定义了它的端口类型，宽度。具体如下表格（图5.4 NetworkIO端口）。

|  |  |  |
| --- | --- | --- |
| 名称 | in | out |
| 宽度 | 16.W | 16.W |
| 描述 | Flipped | Decoupled |

图5.4 NetworkIO端口

### 5.1.6 Packet模块

在Packet模块中，我们定义了几个量：id地址，src源地址，dest目的地址，size大小。具体如下表格（图5.5 Packet端口）。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 名称 | id | src | dest | size |
| 宽度 | 8.W | 4.W | 4.W | 8.W |
| 描述 | UInt | UInt | UInt | UInt |

图5.5Packet端口

### 5.1.7 Router模块

在Router模块中，我们先对每个router的io端口进行了初始化。然后定义了inputBuffer（输入缓冲），dirOut（方向输出），demuxes（多路分配器）和arbiter（仲裁器），并且进行了映射。其具体宽度和描述如下图（图5.6 Router端口）

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 名称 | inputBuffer | dirOut | demuxes | arbiter |
| 宽度 | 5.W | 5.W | 5.W | 5.W |
| 描述 | VecInit | Wire,Vec | VecInit | VecInit |

图5.6 Router端口

接下来具体介绍一下它们的实现逻辑

inputBuffer：一个输入缓冲需要接受来自五个方向的包，所以我们先定义它的宽度为5.W，类型是VecInit，然后使五个方向分别映射出5个Queue，我们使Queue的最大数为16,包进到inputBuffer里，就会放到其方向相对应的队列中。

Demuxes：多路选择器，也需要接受五个方向的包，我们定义它的宽度为5.W，类型是VecInit。然后使五个方向分别映射5 Demux，调用Demux的功能，选出inputBuffer的Queue中状态为ready的包准备进行下一步。

arbiters:仲裁器，也需要接受五个方向的包，我们定义它的宽度为5.W，类型是VecInit。然后使五个方向分别映射5个RRarbiter，调用RRarbiter的功能，选出inputBuffer的Queue中状态为ready且响应的包，准备分配到下一节点的输入端。

dirOut: 也需要接受五个方向的包，我们定义它的宽度为5.W，类型是VecInit。在完成上述步骤后，已经找到了当前需要输出的包，那么它的功能就是将这个包输出到下一节点的输入端。

有了上述的逻辑，我们再讲它们按顺序zip起来，就可以实现包在端口中的传输了。

### 5.1.8 RouterIO模块

在RouterIO模块中，我们把Router的输入输出端口进行了初始化，并且定义了它的类型为Vec，Decoupled。宽度5.W。具体如下图所示（5.7 RouterIO端口）

|  |  |  |
| --- | --- | --- |
| 名称 | in | out |
| 宽度 | 5.W | 5.W |
| 描述 | Vec,Flipped | Vec,Decoupled |

图5.7RouterIO端口。

## 5.2 测试模块

### 5.2.1RouterTester模块

RouterTester模块是对Router进行测试的模块，对于选定的节点，我们在它的输入端口输入当前地址值，目的地地址的值，源地址值，包的大小。如果当前状态端口为ready，那么就让数据输出到输出端口，看输出端是否接收到了这些数据。以此来判断Router的端口配置的算法是否正确。算法如下图所示（图5.8 RouterTester的算法）

while peek router.io.in Direction.local = ready//当本地状态为ready

poke router.io.in Direction.local id 0 // 输入id为0

poke router.io.in Direction.local src, 5)// 输入源地址为5

poke router.io.in Direction.local dest, 11// 输入目的地址为11

poke router.io.in Direction.local size, 16// 输入大小为16

val id = peek router.io.out Direction.east id// 输出东端口的id

val src = peek router.io.out Direction.east src//输出东端口的源地址

val dest = peek router.io.out Direction.east dest//输出东端口的目的地址

val size = peek router.io.out Direction.east size//输出东端口的大小

printf out: id=$id, src=$src, dest=$dest, size=$size\n//按格式输出

图5.8 RouterTester的算法

数据进入到当前节点的local节点，由于源地址为5，目的地址为11，目的地址在源地址的东侧，应当向东边的端口输出，我们看东边端口得到的数据是否正确。

这里经过试验后得到的数据为out：id=0，src=5，dest=11，size=16，结果与本地端口输入的数据一致，结果正确。

经过多次试验，结果均为正确，验证了之前端口配置算法的正确性。

### 5.2.2 XYRoutingExampleTester模块

XYRoutingExampleTester模块，这个模块是测试对于给定的当前地址值和目的地地址值，根据之前的算法，packet是否可以从正确的方向输出到下一端口，从而验证之前的算法的正确性。

这里我们调用了PeekPokeTester程序，它的功能是给定一个输入值，来看它运行之后的输出结果。Current为当前地址，dest为目的地地址，src为源地址，dir为方向。具体过程如下图（图5.9 XYRoutingExampleTester算法）

val dest = Input //定义dest为输入值

val dir = Output //定义dir为输出值

val id,io.dest,io.dir //用到的变量有id，dest，dir

XYRoutingExampleTester extends PeekPokeTester //调用PeekPokeTester

Valdir= peek //输出dir的值

printf out: current= ,dest= ,dir= //显示current，dest，和输出的dir的值

图5.9 XYRoutingExampleTester算法

这里我们测试两组数据：

第一组：current=7，src=6，dest=15

第二组：current=8，src=0，dest=3

为了方便我们这里再画一个简易的拓扑图（图5.10简易拓扑图）

0 1 2 3

4 5 6 7

8 9 10 11

12 13 14 15

图5.10 简易拓扑图

相对应的，我们得到两组数据：

第一组数据为：out：current=7， dest=15，dir=3

结合拓扑图我们来看，当前的地址为7，目的地为15，根据XY路由算法先X后Y的顺序，由于在同一列，所以不需要进行X的移动，只需要向南移动即可。dir对应的方向为南，正确。

第二组数据为：out：current=8， dest=3，dir=2

结合拓扑图我们来看，当前的地址为8，目的地为3，根据XY路由算法先X后Y的顺序，由于目的地在当前的东侧，需要向东移动， dir对应的方向为东，正确。

此外我们还进行了很多次的测试，结果均为正确。验证了之前算法的正确性

### 5.2.3 OddEvenRoutingExampleTester模块

由于奇偶转弯模块和之前的XY路由模块是共用的路由器，只是算法不同，这里我们就不做重复的介绍了，那么接下来就直接介绍奇偶转弯算法的检测模块

OddEvenRoutingExampleTester类似于XYRoutingExampleTester，都是将当前地址，源地址和目的地址输入，看它是否可以按照规则进行正确方向的输出。具体算法如下图（图5.11 奇偶转弯检测模块算法）这里我们设源地址为src，目的地址为dest，dir为方向。

val src = Input //设src为输入值

val dest = Input //设dest为输入值

val dir = Output //设dir为输入值

OddEvenRoutingExampleTester extends PeekPokeTester //调用OddEvenRoutingExampleTester

Val dir = peek //看dir的输出值

printf out: src = dest= ,dir= //显示src, dest并输出的dir的值

图5.11 奇偶转弯检测模块算法

0 1 2 3

4 5 6 7

8 9 10 11

12 13 14 15

图5.12 简易拓扑图2

这里我们让src为2，dest为4，看输出结果如何。这里为了方便观察，我们再画一个简易的拓扑图（图5.12简易拓扑图2）

按照奇偶转弯的规则，最短路径应为由2到1,1到0，0到4。第一步就应该为向西传输。

试验后得到的输出为：out：src=2，dest=4，dir=4，符合正确结果。

经过多次试验，结果均为正确，我们可以证明之前的算法正确。

在本次实验当中由于奇偶转弯路由算法的逻辑关系较为复杂，第一次完成的时候并未成功，于是在Scala中又加入了一个OddEvenRoutingVerify模块。在这个模块中，我们使用了相对熟悉的C语言算法实现奇偶转弯路由算法，验证后再与之前的算法进行对比，最终找到了逻辑上的错误，并且更正，才有了最终的结果。

## 5.3 本章小结

在本章中，我们介绍了搭建了路由器所使用的模块，每个模块中每个变量的定义，宽度和描述。搭建了这些模块，模块之间功能相互配合，再配合之前两种算法，就分别实现了XY路由器和奇偶转弯路由器的基本功能。

在完成了模块的搭建之后，我们还需要进行单元测试，于是对两种算法和路由器节点分别进行了单元测试，并且测试多组数据，以此提高测试的准确性，保证了算法的正确，也就保证了功能可以正常实现。

# 结论

本文由片上网络（NoC）引入，介绍了NoC的研究现状，优点和缺点，包括路由器的结构，分析了NoC未来的发展趋势。接着介绍了FPGA，及其中的型号Zynq-7000。因为NoC系统更可能会直接选择使用FPGA作为其最终的硬件实现方案。所以基于FPGA的NoC应用系统设计是具有实用性的。但是由于设备有限，没有能在FPGA上实现

但是我们还是设计出了基本的XY路由器和奇偶转弯路由器。

通过在Intellij IDEA上用Scala语言，搭建路由器中各种功能的模块，并让它们相互连接，相互配合，实现了路由器的基本功能。

通过实现算法并对这两种算法详细了解后，我们知道了XY路由算法是一种确定性路由算法，那么它在网络拥堵低的情况下，效率是十分高的。算法简单，容易实现。而奇偶转弯算法就相对复杂一些，通过限制在四个方向的转弯，从而避免成环，是放死锁的。相对于XY路由算法来说，有一定的自适应性。

在今后的工作中，我们希望可以实现更多的算法，满足不同路由器和不同网络状况的需求。也希望能尽可能的完善算法，使路由器实现更多的功能。

# 参考文献

[1] Tapani Ahonen et al. A brunch from the coffee table – case study in NoC platform design. In J. Nurmi, H. Tenhunen, J. Isoaho, and A. Jantsch, editors, Interconnect-Centric Design for Advanced SoC and NoC, pages 425–453. Kluwer Academic Publishers, 2004.

Systems on Chips

[2]Benini L, De Micheli G. Networks on chips: a new SOC Paradigm. IEEE Computer, pp.70-78,2002.

[3] J. P. Soininen, A. Jantsch, M. Forsell, A. Pelkonen, J. Kreku, and S. Kumar Extending platform-based design to network on chip systems. In Proceedings of the International Conference on VLSI Design, January 2003

[4] Zheng Li, Jie Wu, Li Shang, Robert Dick, and Yihe Sun. Latency criticality aware on-chip communication. In Proceedings of the IEEE Conference on Design, Automation, and Test in Europe, March 2009.

[5] Ahemani, A. Jantsch, S Kumar, A. Postula, J. Oberg, M. Millberg, D. Lindqvist Network on chip: an architecture for billion transistor era. In Proceeding of the IEEE NorChip Conference, November 2000

[6] F. Karim et al. An Interconnect Architecture for Networking

[7]康与云著．元功能链驱动的机电产品矩阵式创新设计方法：山东人民出版社，2015.11

[8] Israel Gazit, Miroslaw Malek. Fault Tolerance Capablilities in Multistage Network-Based Multicomputer Syatems. IEEE Transactions on Computers. Val 37

[9] CHEN P, JIANG XJ. System for CAN BUS communication based on FPGA J] . Computer Measurement & Control, 2009,17 (12)

[10] IKEGAMI K, ABE K, NOMURA K. Designing nonvola­tile reconfigurable switch-based FPGA through overall circuit performance evaluation J] . 2012 IEEE 26th In­ternational Parallel and Distributed Processing Symposi­um Workshops & PhD Forum (IPDPSW)

[11] Yan WEN. Hardware and Software Codesign Speech Recognition System Based on ZYNQ7000[A]. Advanced Science and Industry Research Center.Proceedings of 2014 International Conference on Computer,Network Security and Communication Engineering(CNSCE 2014)[C].Advanced Science and Industry Research Center:,2014

[12] 陈军.Torus交换结构流量均衡和容错路由算法研究：电子科技大学,2007

# 致谢

四年的大学时光转转瞬即逝，在收尾时，也不知是因为劳累还是因为年龄的增长，我长出了人生中的头两根白头发，感慨良多。在完成毕业设计及论文这段时间内，有很多需要感谢的人。没有他们的帮助和陪伴，我是没有办法仅仅凭一己之力完成的。

首先感谢学校对我这四年的培育和教导，感谢信息学部对毕业生工作上的支持。在制作毕设这段时间里，我十分感谢我的导师蔡旻老师。我从他这里得到了很多很多的帮助，对于基础知识相对不够扎实的我，他每次都能十分有耐心的教导。从配置实验环境到算法的编写，他每周都会为尽职尽责地给我和其他两位同学指点迷津，答疑解惑。在进度上循序渐进，在方式上循循善诱，是一位出色的老师，很荣幸这次可以做蔡老师的毕设。也感谢其他同学，在茶余饭后，对所做的实验有不懂之处的时候，可以相互讨论，相互交流。让我体会到了学习的乐趣。也感谢这次毕设，让我学习到了很多硬件上的知识，丰富了我的知识库,增长了见识。这些都是对我以后的人生非常有帮助的。最后感谢所有学科老师，是老师们的帮助让我走到了今天，谢谢你们！