电子科技大学

硕士学位论文

片上网络路由算法及应用研究

姓名：蒋勇男

申请学位级别：硕士

专业：通信与信息系统

指导教师：李玉柏

20080401

摘要

技术的飞速发展把芯片设计带入了十亿晶体管的领域。有预测到这个十年末，单片芯片上将允许集成多达数十亿个晶体管。随着系统芯片芯核数量的增多，传统的片上系统(SoC)的设计方法将不再满足十亿晶体管级别的设计要求。这时，片上网络(Network On Chip，NOC)做为一种新的解决途径被提出来了。片上网络的可升级性(scalable)和易扩展性很好的满足了新设计的要求。片上网络逐渐发展成为传统片上总线之外的一种新的通信结构。

文章第一部分主要是介绍我们构建的片上网络仿真平台,该平台为以后的研究工作提供了一个坚实的基础,我们可以在此平台上进行存储深度优化,功能映射研究;关于路由算法,则是主要设计实现了支持有一定自适应性的 XY 维序路由,以及 OE 路由模式的路由节点,考虑到应用研究时数据传递的需求,该平台也支持不同数据模式的传输,包模式传输,flit 传输.最后对于网络平台的设计可行性从网络通信中的延时,数据吞吐率以及整个平台的资源消耗方面进行了验证.

文章第二部队则主要是片上网络应用研究,首先对视频处理算法进行了介绍,并对其中帧间编码部分进行了功能分解,将其映射到我们设计的仿真网络平台中,通过分析数据流量图,我们采用内嵌存储以及根据映射时尽量避免热点的原则给出了优化后的功能映射以及数据流量分析,从一个侧面论证了片上网络在并行处理上的优势.

关键词：片上网络，路由技术，路由算法，并行分解

Abstract

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

The first part of this paper is to talk about the simulate network model we have made, which also is important for our research work, we can research how to optimize the depth of FIFO, how to map the special function in to our simulate network model. We also achieve the router model which support adaptive routing, such as XY with adaptable adjust, OE turn model for adaptive routing. Consider about the need of data transfer, packet transfer and flit transfer model are also designed. At last, we validate our simulate network model from three aspects: delay of data routing in the network, throughput and resource cost.

Second part of this paper is to research about the application, firstly, we introduce the video compressing, then divide intra frame code model in to several part and map in to our simulate network model, analyze the data transfer report, as a result, we put the memory into our processing models which are also special function models ,analyze the data transfer report , prove the advantage of NOC in parallel processing.

Keywords: Network on Chip, Routing Technology, Routing Algorithm, Parallel processing

图目录

表目录

缩略词表

英文缩写	英文全称	中文释义
SoC	System on Chip	片上系统
NOC	Network On Chip	片上网络
IP	Intellectual Property	知识产权
NI	Network Interface	网络接口
HOL	Head of Line	队头阻塞
ASIC	Application Specific Integrated Circuits	专用集成电路
FPGA	Field Programmable Gate Array	现场可编程门阵列
OE	Odd_even	奇偶转向路由
ISO	International Standardization Organization	国际标准化组织
ITU	International Telecommunication Union	国际电信联盟
IEC	International Electrotechnical Commission	国际电工委员会
DPCM	Differential Pulse Code Modulation	差分脉冲编码
JVT	Joint Video Team	联合视频工作组
DCT	Discrete Cosine Transform	离散余弦变换
CAVLC	Context-based Adaptive Variable Length Coding	基于上下文的自适应变长编码
UVLC	Universal vaVriable Length Coding	统一可变长编码
CABAC	Context-based Adaptive Binary Arithmetic Coding	基于上下文的自适应二进制算术编码

独 创 性 声 明

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

签名：_____ 日期：2008年 6月3日

关于论文使用授权的说明

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

(保密的学位论文在解密后应遵守此规定)

签名：_____ 导师签名：_____

日期：2008年 6月3日

第一章 绪论

1.1 课题背景

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

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

(1) 可扩展性面临的问题

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

(2) 单一时钟同步问题

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

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

1.2 片上网络研究动态

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

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

在国内，对片上网络的研究也引起人们的重视，比如：清华大学微电子学研究所的孙义和等人对特定应用的 NOC 设计方法学进行了研究；清华大学电子工程系的曾烈光等人对高性能片上互连网络及其可测性设计进行了研究；西北工业大学航空微电子中心荆元利，樊晓桠，张盛兵，高德远，周昔平等人提出了基于片上网络的模块化测试方法；湖南嵌入式计算机系统重点实验室的李仁发，李肯立，徐成等人研究了片上 IP 网络化互连的策略； 湖南农业大学的任峻，凌纯清，李仁发等人提出了片上网络化运动控制器的设计与实现方法；哈尔滨工业大学的周文彪、张岩、毛志刚等人研究了片上网络的低功耗自适应数据保护方法；中国科学院计算机研究所的韩银、张磊等人对片上系统的超多核测试和可靠性设计进行了研究；国防科技大学计算机学院的卢锡城对嵌入式高性能网络 IC 设计进行了研究。

本文则主要构建一套成熟的片上网络仿真平台，并对之前进行的一些工作做了总结，设计了具有比较完善功能的路由节点模型。同时对几种典型的路由算法进行了仿真验证，选择合适的作为我们在日后进行各种拓展研究的基础。最后对片上网络如何走向应用，以及如果进行功能的划分进行了初步的探讨，同时针对片上网络给出了比较合理的划分结果。

1.3 论文结构及内容安排

本论文共分为六大部分。第一章介绍了课题背景、片上网络研究动态，说明论文结构及内容安排；第二章总结 NOC 中几种常见的拓扑结构、常用的路由技术及路由算法；第三章介绍了了我们构建的 Mesh 结构网络仿真平台，重点介绍了路由节点的设计；第四章介绍了即将应用于片上网络应用研究的视频处理标准 H.264 的相关知识；第五章则探讨了如何在基于片上网络的基础上进行功能划分，并给出了相关仿真结果；第六章对本设计作了总结，分析了设计中存在不足，展望了未来应该开展的工作。

第二章　片上网络概念和路由技术

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

(1) 片上路由器结构简单，不宜采用较复杂的路由算法。

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

(2) 片上网络的网络协议。

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

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

2.1　片上网络拓扑结构

片上网络的拓扑结构主要是指NOC中各个节点的连接方式。不同拓扑结构的网络其信道的访问技术，利用率以及信息的延迟，吞吐量，设备开销都各不相同，所以，在片上网络中，对拓扑结构的研究至关重要。

通常NOC拓扑结构分为两类，一类是直接型拓扑，另一类是间接型拓扑[4]。

在直接型的NOC体系结构中，处理单元(PEs)和交换单元(SEs)结合作为一个节点。

2.1.1 直接网络拓扑

在直接网络中，每个PE像瓦片一样被放置和连接。瓦片表示处理单元或者存储单元。虽然瓦片可能有不同的功能，但是它们有着同样物理尺寸而且有规则的平面布置图。包交换技术被应用在瓦片之间的通信。每个瓦片能独立地执行数据包的路由和仲裁。

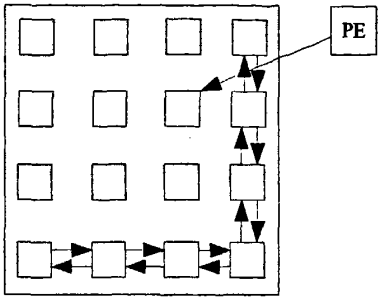

图2-1 瓦片结构

最简单的一种结构，就是标准的2D网格，如图2-2所示。这种网络拓扑呈现了每一个非外围的路由节点都与它四个最临近的节点双向连结，同时也与它相关的PE双向连接。外围节点则至少有一个边没有连接。2D花托是2D网格的不同形式。这种情况下，每个节点与其他四个节点相连，包括外围节点，这意味着一边的节点能够直接把数据包传到另一边。继续开发额外的连通性，2D花托可以演变成一种类似于特殊立方体的拓扑结构，我们称这种网络结构为旁路花托。它描述了一个更高维数的网络，因为每个节点不仅和它最邻近的节点相连，也和越过相邻节点的下一个节点相连。

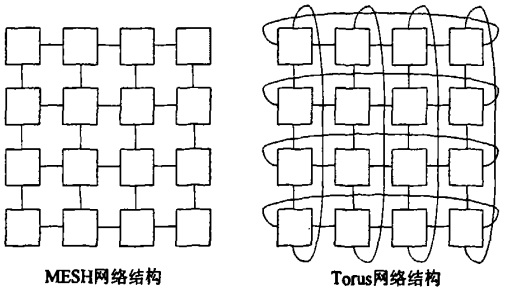

图2-2 Mesh网络与Torus网络

2.1.2 间接网络拓扑

在间接网络拓扑中，处理器节点(CPU, DSP, MCU)与交换节点分开，网络交换由专门的 Crossbar 执行，每个 Crossbar 通过网络接口(NI)和各个交换节点连接进行通信。间接网络可以使得节点设计和网络设计分开进行，提高设计效率。如蝶形网络，十字开关交换网络，Spin 网络等等。

间接网络也采用了瓦片平面布置结构。而且利用了一个二维空间的网孔拓扑结构。与直接网络的瓦片结构相比较，每个节点的路由和网络接口都是由开关网络来执行的。处理节点和路由节点也是分开执行各自的功能的。

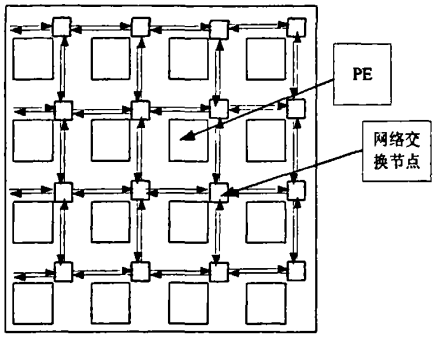

图2-3 二维间接网络

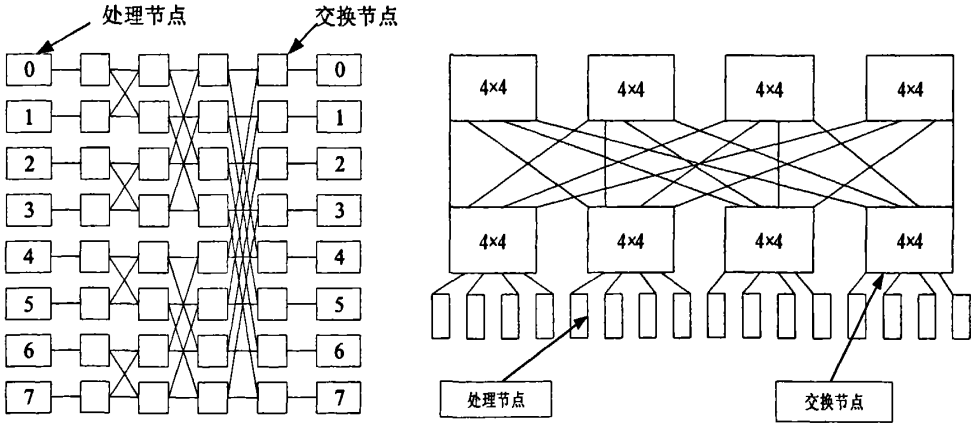

图2-4 典型间接网络拓扑

在图 2-4 中我们给出了几种典型的间接网络拓扑结构,其中左图为蝶形拓扑结构, 每条源－目的路由使用一个专门的数据路径; 在任何的两个处理器节点之间的

延迟是一样的，而且延迟由开关网(switch fabrics)上的中间阶段数目所决定的；右图为 Spin 网络[5]利用一个 fat-tree 拓扑结构，在该结构中每个处理器位于 fat-tree 的一个树叶节点上。

2.2 NOC 的关键技术——路由技术

本文所介绍的常用路由技术主要包括以下内容，包交换技术、虚拟通道技术[6]和死锁[7]避免技术。包交换技术关注的是数据包是怎样从输入通道交换到输出通道的。采用的包交换技术不同，所产生的延迟就不同，网络的延迟与包交换技术直接相关。虚拟通道技术主要是结合虫洞路由一起使用，可以大大的降低阻塞发生的概率。死锁避免技术，主要介绍了死锁的原理，在路由算法设计中也有所涉及。

2.2.1 包交换技术

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

2.2.1.1 存储转发(Store-and-Foward)

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

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

$T = (L/BW + R) \times H$。

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

2.2.1.2 虚拟直通

与存储转发方式缓存整个数据包不同，在虚拟直通中，数据包被分成了许多的片段(flits)。每个包的头片段控制路由，其余的片段跟在头片段之后，以“流水”的方式在网络中传输，如图 2-5。当阻塞发生时，整个数据包都存储在阻塞发生的路由器缓存里。这样，每个路由器只需要有限的缓存空间，即阻塞发生时所需缓存的数据包的长度大小就够了。

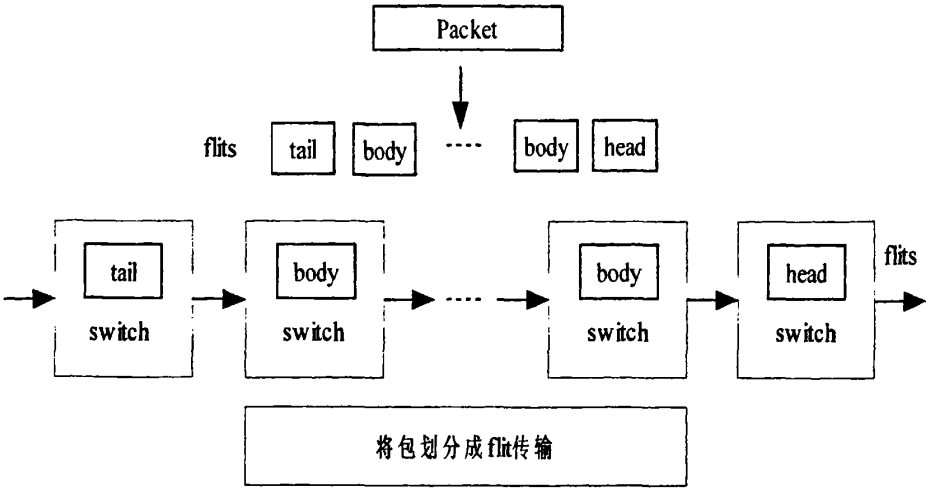

图2-5 数据包分片示意图

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

$T = L/BW + R \times H$。

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

2.2.1.3 虫洞路由

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

在延迟方面，也与虚拟直通一样，都为：

$T = L/BW + R \times H$ 。其中，L 是数据包的长度，BW 是网络的带宽，R 是每个节点的寻径延迟，H 是所经过的节点数。存储转发和虫洞路由延迟比较如图 2-6。

从上面的描述中可以看出，虚拟直通方式和虫洞路由方式是很相似的。事实上，在没有阻塞的情况下，虚拟直通和虫洞路由是完全一样的。它们两者的差别在于发生阻塞时对被阻塞的数据包的处理：虚拟直通将整个的数据包储存在发生阻塞的路由器中；而虫洞路由是将各个片段分散到各个节点中。如图 2-7 所示。

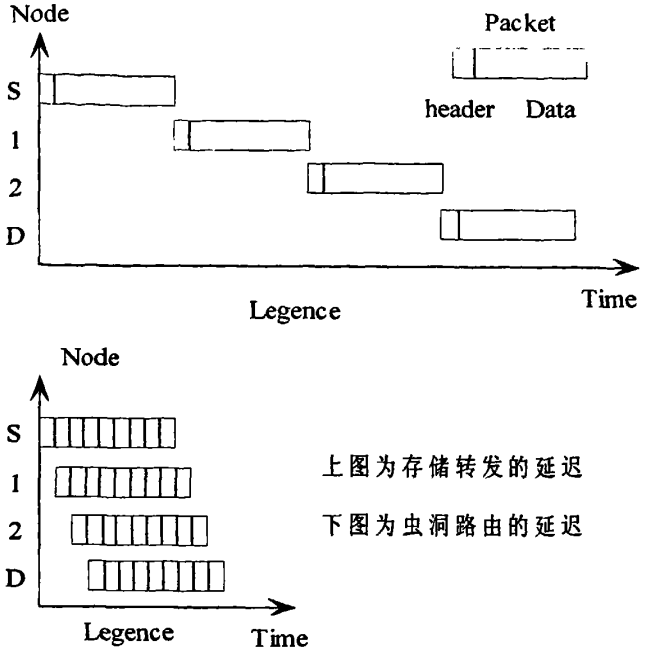

图2-6 虫洞路由与虚拟直通的比较

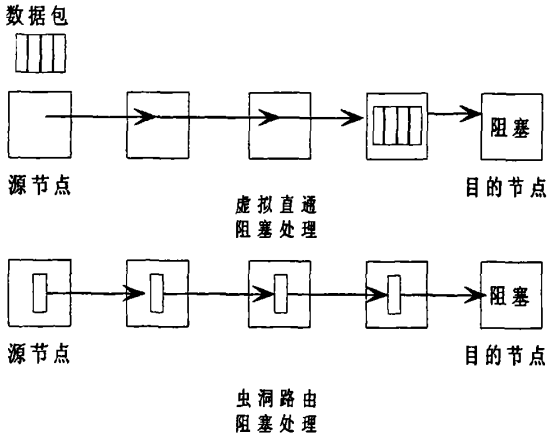

图2-7 阻塞示意图

虫洞路由的优点：

(1) 相对的使网络延迟对路径长度不那么敏感。从公式 T = L/BW + R×H 可以看出来，延迟由两部分组成，前一项是发送数据包的时间，后一项是在各个节点寻径时间的总和。在实际应用中，一般数据包的长度比较大，远远大于各个节点

寻径时间的总和，所以延迟主要由前一项决定。这个优点很好的适应了 NOC 可升级性(scalable)的要求。

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

虫洞路由带来的缺点：

(1) 只要一条通道开始传送包的第一个片段，它就被这条数据包所占用了，必须等到传完所有片段才能被释放，而被其它消息使用。

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

2.2.1.4 偏转路由(Deflection Routing)

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

2.2.2 虚拟通道

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。

2.2.2.1 死锁问题研究

当多个数据包循环等待的时候，就会发生死锁，如图 2-8 所示：

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

对于死锁的避免而言，在设计路由算法时也很重要，这里我们可以看到通过虚通道技术来避免死锁，同样在路由算法设计中通过打断某个方向的转弯也可以，

在后面自路由算法中介绍的几种算法均可以避免死锁的发生。

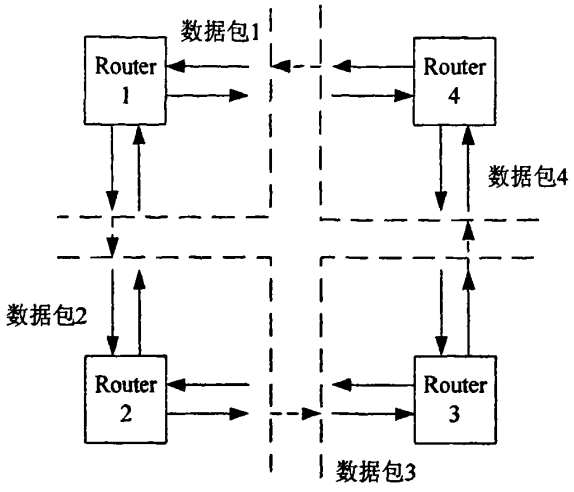

图2-8 死锁问题的发生

2.2.2.2 虚拟通道

通常，我们在采用了虫洞路由中的网络引入虚拟通道，即把一个缓存划分为多个虚拟通道，这些虚拟通道通过时分复用一条物理信道。如图 2-9 所示，一个 12 个 flit 的缓存被划分为 4 条虚拟通道，每条虚拟通道有 3 个 flit 的缓存。

在虫洞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。

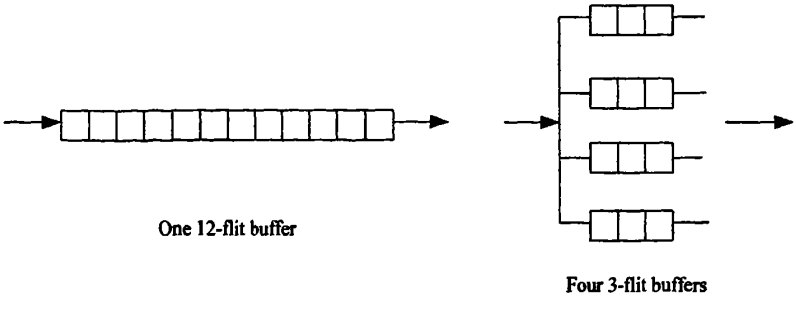

图2-9 虚通道技术示意图

为了解决虫洞路由交换机制的阻塞，Dally 提出了虚通道技术。该技术的思想就是在每个物理通道上设置多个缓冲区，每个缓冲区度应一条虚通道。所有虚通

道以时分复用的方式共享该物理通道。这样可以使得活动(active)的数据包绕过阻塞(blocked)的数吞吐据包，从而达到提高物理通道利用率和网络吞吐率的目的。同时虚通道的引入可以比较容易实现自适应无“死锁”算法，因此许多采用虫洞路由交换技术的路由器都采用虚通道技术。

虚通道的应用也带来一些问题[8]：路由器对数据包头的处理需要较大的开销；缓冲区数据增加导致路由器交叉开关，仲裁单元的增加，从而使路由器的复杂性增加；交叉规模的扩大，增加了仲裁时间，导致路由时间增大，由此使得路由器延迟增加；不同的虚通道上的负载不平衡性对网络的性能也有较大的影响。

但是考虑到片上网络的特性，首先是需要保证高度的准确性，因为与网络接口(NI)相连接的每个 IP 都有固定的执行功能，到每个 IP 的数据要确定高的精度。其次，在 MPSoC 中，各个功能块的功能也是固定的，我们可以根据通过一定的路由算法采取静态分析策略找出路由的最小路径以及最理想的避让策略。

2.3 片上网络中典型路由算法

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

路由算法可以按照不同的标准分为不同的几类。比如说源路由(Source Routing)和分布式路由(Distributed Routing)，确定路由(Deterministic Routing)和自适应路由(Adaptive Routing)。

2.3.1 确定性路由

确定路由是一种常见的路由，它的路由路径只于起点地址和终点地址有关，给定起点和终点地址，路由路径就被确定了，与当前网络的状态无关。而在确定路由中，使用的最多的则是维序路由[9][10](Dimension-Ordered Routing)，因为它有着非常简单易实现的路由逻辑。在维序路由中，每个数据包一次只在一个维上路由，当在这个维上到达了恰当的坐标之后，才按由低维到高维的顺序在另外的维上路由。因为数据包是按照着严格的单调的维数变化的顺序在通道内路由，所以维序路由也是没有死锁的。在这里我们将重点介绍 2D Mesh 网络结构中的 XY 维序路由。

XY 维序路由是最简单的一种路由算法，它的路由过程是：先 X 方向将数据包

包送至目的节点所在的列；再沿 Y 方向将信数据包送至目的地处理器所在的行，如图 2-10 所示：

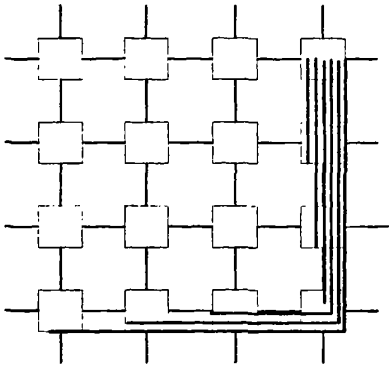

图2-10 XY 路由寻路

当路由器开始处理一个数据包的时候，因为采用的是 XY 路由，它先会判断 X 方向上是否到达了目的节点所在的列，如果没到达，再比较目的 X 和当前 X，如果目的 X 比当前 X 大，则还应该向东路由；如果目的 X 比当前 X 小，则说明还应该向西路由。如果判断出来 X 方向上已经到达了目的节点所在的列，则再比较目的 Y 和当前 Y，如果相等，则说明已经该数据包已经到达了目的地，则收包，向 IP 路由。如果目的 Y 大于当前 Y，则还应该向南方路由；如果目的 Y 小于当前 Y，则说明还应该向北方路由。

举例来说，一个从(2，2)发出的数据包，其目的地是(3，4)，则它采用 XY 路由算法的路由路径是：(2，2)—>(3，2)—>(3，3)—>(3，4)。具体如图 2-11 所示。

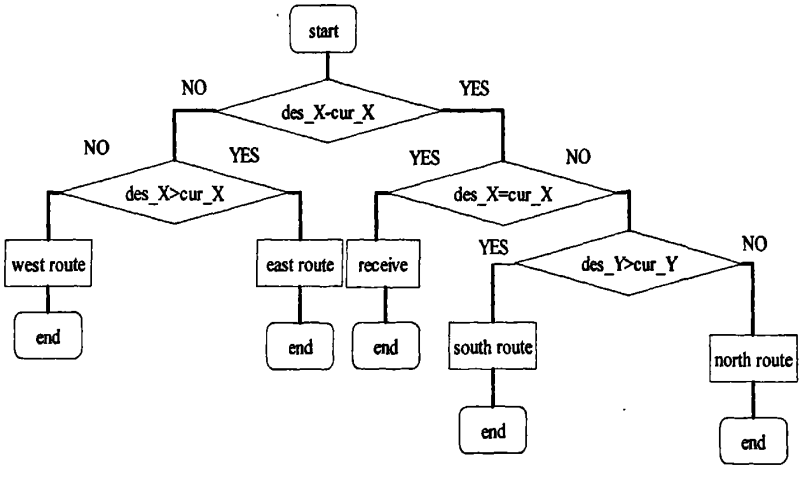

图2-11 XY 路由寻路规则

即一次只在一个方向(维数)上路由，直到在该方向(维数)上当到达了与目的地址相同的节点，再按照单调的顺序改变方向在其它维上路由，XY 路由是由 X 方向改变到 Y 方向的顺序，而这正是维序路由算法的典型特征。

2.3.2 自适应路由

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

所谓的自适应路由，就是指数据包的路由路径不只与起点地址和终点地址有关，还要考虑网络的状态。也就是说，有同一对起/终点的地址的数据包，在不同的网络状态下，它们的路由路径也可能不同。

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

在片上网络中，由于路由器结构所限，一般都采用的是比较简单的自适应算法，如带有一定自适应性的维序路由。

2.3.2.1 带有一定自适应性的维序路由

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

下面以 4×4 的 2D Mesh 网络中的带有一定自适应性的 XY 路由为例，具体解释一下这种算法的路由过程。如图 2-12 所示，假设一个数据包路由的起点为(1，4)，终点为(4,1)。如果按照一般的 XY 路由的话，它的路由路径应该是(1,4)->(2,4)->(3,4)->(4,4)->(4,3)->(4,2)->(4,1)。这时如果假设数据包在节点(3,4)发生了阻塞，不能继续向(4,4)发送。如果按照原来的 XY 路由，数据包就不能在往下发送，被阻塞在了(3,4)中。这时如果采用的是带有一定适应性的 XY 路由，在向节点(4,4)发送传送请求没有被允许之后，则它就会向 Y 维方向上的(3,3)节点发送传送请求，被允许之后，数据包就被发往节点(3,3)了。到达(3,3)后，又会先向 X 维方向上的(4,3)

发送请求。就这样最终的路由路径为(1,4)->(2,4)->(3,4)->(3,3)->(4,3)->(4,2)->(4,1)。

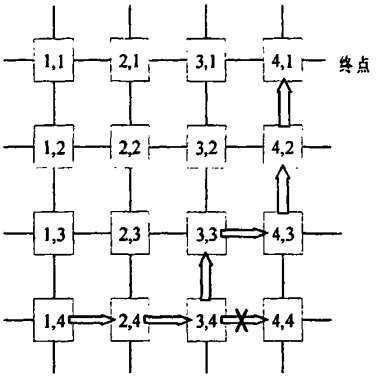

图2-12 带有一定自适应性的维序路由

从图 2-12 显示的路由路径可以看出，带有一定自适应性的 XY 路由和一般的 XY 路由的差别就是在某个节点发生阻塞之后，它可以不按照先 X 维再 Y 维的顺序路由，而可以是向另一个维发送数据包。这样就可以从一定程度上缓解网络的拥塞。

2.3.2.2 Odd_Even 路由机制

odd_even 路由[11]机制主要是利用禁止转弯模型来避免死锁，因为死锁的产生是由于在传输中形成了环,互相请求,互相等待造成,如上面死锁图示，现在应用比较广泛的几种路由方式 OE(odd_even routing),XY 维序路由，先西路由(West-first routing)，后北路由(North last routing)，negative first 路由等均是打断某些方向的转弯,来避免死锁的发生.

在 OE 路由模型中,根据在网络中的位置我们可以用(X1,X0)来表示某个具体的路由节点,我们规定以下几点:

1) X1 为网络中节点所在列,X0 为网络中节点所在行.

2) X1 为偶,则该节点所在列为偶列,X1 为奇,则该节点所在列为奇列.

3) 网络中偶列所在路由节点禁止 West_to_North,West_to_South 转向,奇列所在路由节点则禁止 North_to_West,South_to_West 转向.

OE 路由模型中,前提是禁止 180 度转弯的发生,同时由于 OE 路由算法是自适应性的,因此每个节点在处理相关方向数据时,可自适应的选路,同时在保证最小路径的前提下,我们可以通过比对数据的源地址和目的地址,结合该列所禁止的转向来

选择响应的输出方向,以便 Arbiter 进行仲裁.在 OE 路由模型中生成的测试数据如下所示,总共 19 位,其中第 18 位为有效数据标志为,16 位为包头数据标志位,0-3 位为目的地址,4-7 为源地址.

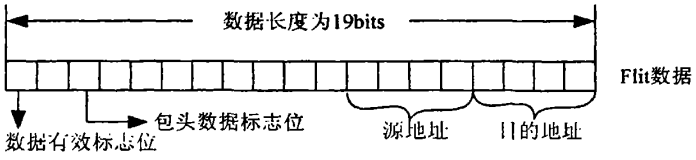

图2-13 OE 路由数据格式

在实际设计中,我们比较源地址 Sx,Sy 和目的地址 Dx,Dy 的关系,首先有 Sy 和 Dy 之间的大小关系,确定 Westbound 或者 Eastbound,对于 Westbound 传输的消息而言,我们认为总是可以向西传输,直到 Sy=Dy 为止,但是目的节点所在列会有某些禁止的转向,所以我们在寻路时将考虑避免传输到目的节点在这条禁止转向的路径上的上一个节点.这就需要我们在路由算法设计时对于某个具体的节点,要从某些当前可行的传输方向中剔除上述可能出现的传输方向,从而选择一条不会出现因违背 OE 路由规定而导致不能到达目的节点的路径。

基于上述最基本设计思想,我们可得到遵循 OE 路由方法中的禁止转向规定的路由表。源节点采用了黑色图示，目的节点采用了红色。图 2-14 则显示了若干条节点之间的通信情况。

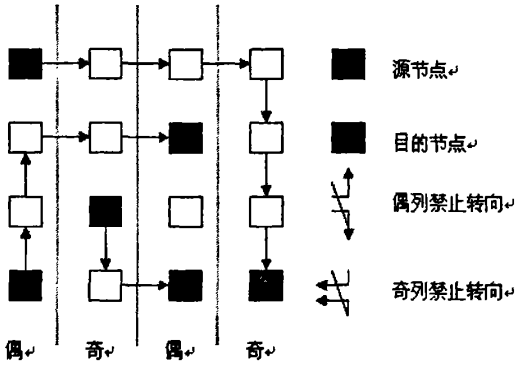

图2-14 OE 路由寻址模式

具体的路由算法我们可以参照图 2-15。

我们规定如下：目的地址为(d0,d1),源地址为(s0,s1)，当前地址为(c0,c1)。最初的可选路由方向为空。

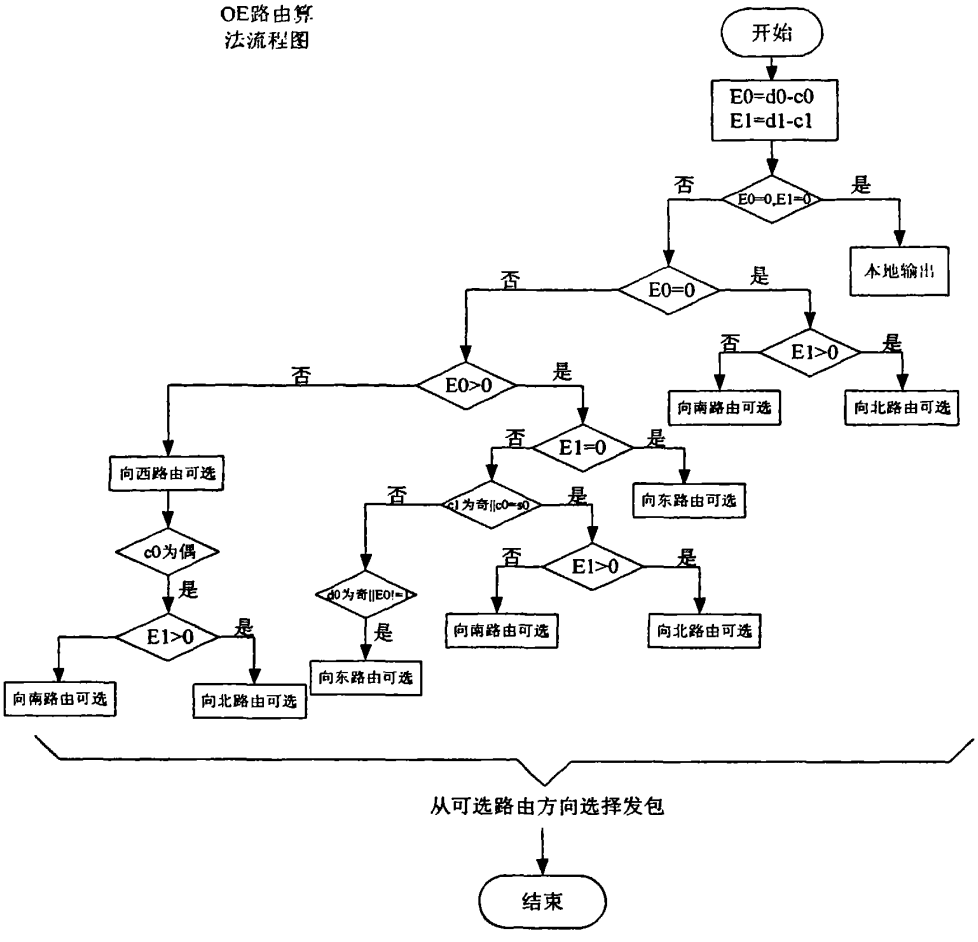

图2-15 OE 算法流程

2.4 本章小结

本章主要做了对片上网络的基本概念进行了回顾与总结，着重介绍了片上网络的相关知识，介绍了两种片上网络的拓扑结构，给出了各自类型的具体网络拓扑结构介绍和示意图；对片上网络的路由技术方面，重点关注了包交换技术，虚拟通道技术，其中提到了死锁这个在网络设计和路由算法设计中很重要的概念；在路由算法方面，介绍了确定性路由和自适应路由算法，并给出了 XY 维序路由，OE 转弯路由的具体实现方法。

第三章　NOC 结构和典型仿真平台的设计

上一章已经介绍了片上网络的拓扑结构，以及几种典型的路由算法，考虑到片上网络最终都要走向应用研究，因此建立一套成熟且实用的仿真平台对于以后进行功能映射显得尤为重要，同时鉴于每个网络节点对应的处理模块都需要与其他对应节点进行通信，因此我们也需要对于不同的传输模式进行相关的仿真。

在这一章中我们将重点介绍我们构建的仿真平台，其中对于节点的设计给予了足够的关注，因为一个路由节点设计的好坏将直接影响到网络在处理中的性能，同时对两种路由算法进行了仿真(XY 维序路由，OE 路由)，在延时以及资源消耗分析后采用比较适合的路由算法。搭建一套成熟的仿真平台，为后面的片上网络研究提供坚实的基础。

3.1　Mesh 网络拓扑结构

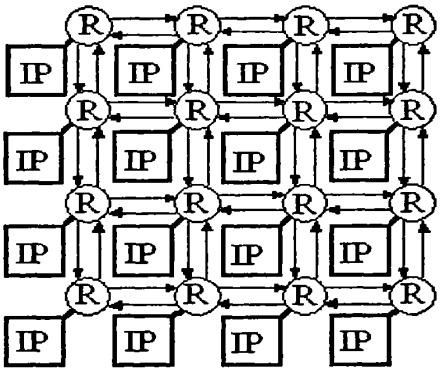

图3-1 Mesh 拓扑结构

图 3-1 显示了用于 NOC 的 Mesh 拓扑结构。其中，每一个路由节点 R(Router)都与一个 IP(或者是处理器核)通过一个本地的接口(Network Interface)相连接，IP 产生的数据由此进入网络，并按照网络的规则进行传送或者通信，而不再采用传统的总线方式，这样，极大的提高了并行处理的速度和性能。

Mesh 结构也是目前在 NOC 中研究得最早的一种拓扑结构。结构规则，简单易于实现，但是由于 Mesh 结构中每个节点并不完全相同，会极大地影响网络性能。

从图中可以清楚的看到，在一般的 Mesh 结构中，边缘节点与内部节点在物理链路的连接上就具有差别，例如，顶点与仅与另外两个节点连接，边上的节点与三个节点连接，而内部节点则直接连接了另外四个节点。在我们进行应用研究时，不可避免地会出现处理的热点，也就是某个处理模块需要很大的通信量，某个模块可能需要比较少的通信量，因此在功能映射时我们可以进行相关处理。

3.2 拓扑网络的路由算法

路由算法是决定网络性能的另外一个重要方面，在 NOC 的片上环境中，路由节点应该尽量设计得简单，不宜消耗过多的资源，因此，路由算法也要尽量简单化，具有尽可能小的路由表。另外，避免死锁和活锁也是路由算法必须考虑的问题。在上一章中已经介绍了路由算法，在这里进行仿真平台的验证中我们采用了 XY 维序路由和确定型的 OE 路由，这类确定性的路由算法(为了下续研究需要，OE 中采用了固定路由表，而只是遵循 OE 的基本转弯规则)实现方式简单，针对 NOC 的具体情况，能方便的进行软件和硬件仿真，因此也成为了目前 NOC 结构研究中一种被广泛采用的方法。

3.3 仿真平台的建立

我们构建的片上网络仿真平台采用了 Mesh 网络拓扑结构，每个路由节点下映射 IP 处理模块，之间通过网络接口 NI(network interface)进行通信交互，每一个路由节点则是包括数据有效性检测模块，FIFO 缓存模块，Arbiter 仲裁模块，以及 Crossbar 寻路输出模块。

数据传输模式实验了包交换以及帧(flit)交换模式，路由算法采用 XY 维序路由以及确定性 OE 路由。图 3-2 为 4×4Mesh 整体网络图。

3.3.1 路由节点设计

路由节点的设计决定了网络的处理性能，我们根据在实际中需要应对的情况来进行针对性的设计，在下面 Mesh 网络结构中，以最中间的处理节点为例来进行说明(比较典型，其他节点和它相同，或者简化)，首先节点需要涉及到东，南，西，北，中五个方向的数据处理，由于节点不可能实时处理完所有的数据，因此我们必须给每个方向都设计相对应的缓存；对于网络中传输的数据而言，有可能因为

软件，硬件错误出现无效的数据(由我们约定的数据格式来判断)，因此每个方向在进行处理之前都需要数据有效性校验；网络中传输的数据都是有传输的目的地，不可避免地会出现寻路的竞争以及寻址输出，我们需要设计相应的仲裁模块来进行寻路和竞争的处理；最后对于处理后的数据需要由交换模块输出。所以我们设计的路由节点将包括数据有效性校验模块 Mux,缓存模块 FIFO,仲裁模块 Arbiter,选通输出模块 Crossbar。我们首先给出路由节点顶层模块图(如图 3-3 所示)，在后续各节将详细介绍每个功能模块的设计与验证。图 3-4 则是我们构建的平台的信号连接图。

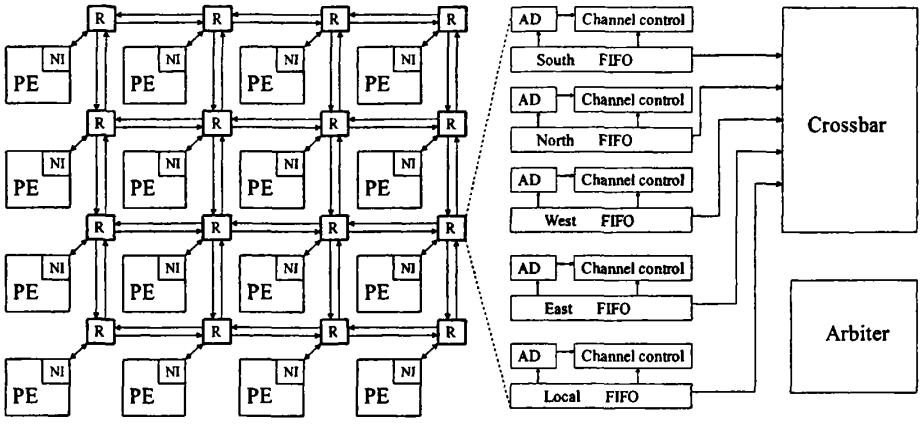

图3-2 Mesh 仿真结构

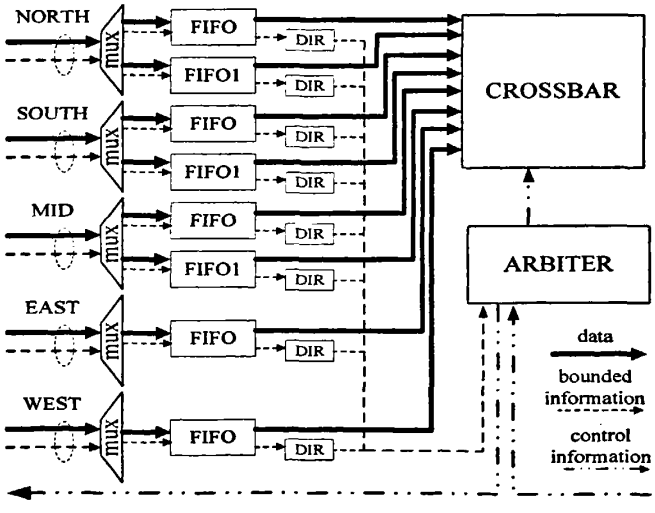

图3-3 路由节点模块

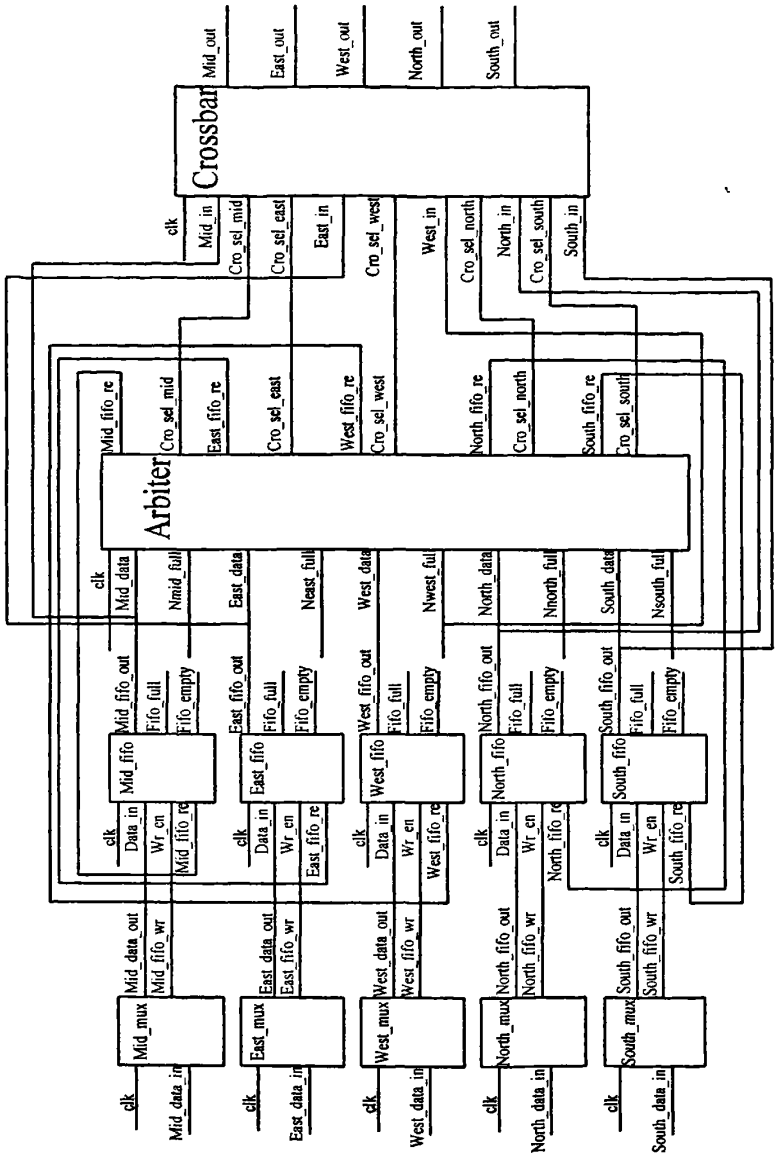

图3-4 整体设计信号框图

在上面我们已经给出了路由节点的顶层模块设计，每一个路由节点需要包括五个 Mux，五个 FIFO 来处理相应方向的数据请求，Arbiter 仲裁模块将产生 Crossbar 的控制信号，来进行数据的选通输出。各模块具体介绍如下。

1. 数据有效性校验模块 Mux 仿真

在片上网络各节点通信中，可能会出于种种原因会出现无效甚至错误数据，所以我们为每个节点都设计了 Mux 模块，用来对到来的数据进行有效性验证,当标志位为 1 则输入到 FIFO 模块,否则输出为无效数据。Mux 模块虽然功能单一，但是却是后续模块得以正常运作的前提。图 3-5 为 Mux 模块的仿真示意图。

2. 数据缓存模块 FIFO

在仿真网络中，单一路由节点对于接受到的数据进行处理需要时间，经验证表明仲裁约需要 7 个时钟的响应时间，如果节点带有处理模块，则需要更多的时间，因此我们为每个节点的每个方向都设计了缓存模块，采用标准的 FIFO 设计。图 3-6 为缓存模块仿真。

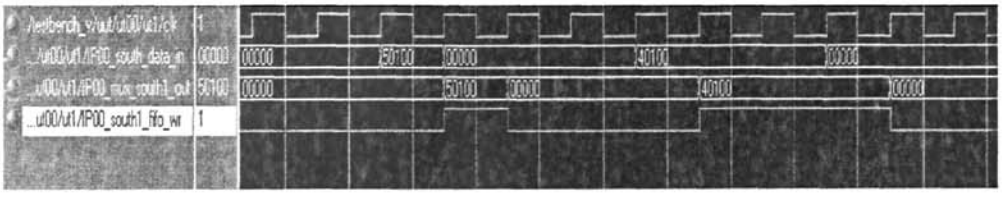

图3-5 数据有效性校验模块 Mux

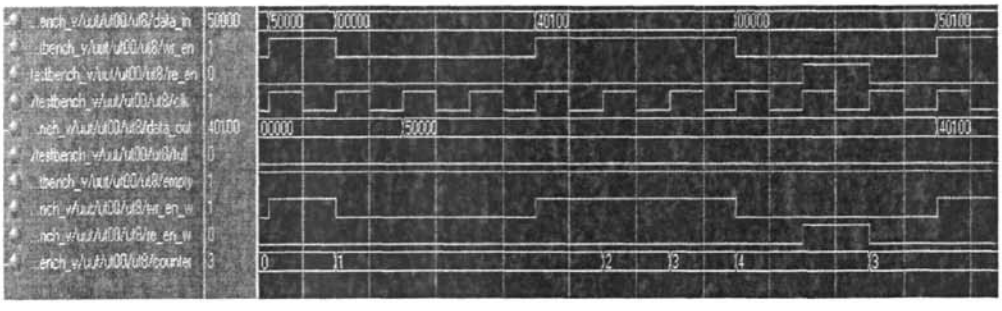

图3-6 数据缓存模块 FIFO

3. 仲裁模块 Arbiter

仲裁模块在整个路由节点中占据最大的比重，无论从资源消耗，时间消耗上来讲，这一模块的设计质量将极大程度地影响整个路由节点甚至于网络的表现性能，在 ISE 9.1 中进行资源评估时可以很直观地反映出来。Arbiter 模块用来对该节点各个方向来的数据进行仲裁处理,在网络中以中间四个节点最为明显，有 east，west，north，south，mid 五个方向的数据需要处理，两个或者三个输入端口竞争同一个输出端口的情况时有发生，当生成的数据的插入率大，即网络比较繁忙的时候尤其如此。因此好的竞争机制显得尤为重要，我们知道在计算机网络中，有很

多类似的应对机制，例如时间片轮转，优先级机制等等。

这里我们采用了优先级处理机制,为每个方向设置了相应的 priority，采用轮询处理，每一个输出方向均有自己的响应顺序，例如都是从 router01 的 East 方向输出,其响应优先级为 West>Mid>South，(因为每个 router 节点实际向一个方向输出的输入数据来源有所不同)当三个方向有竞争同一个方向时，我们首先响应 West,并将 priority 加一，以备下一次轮询处理， 将产生的 cro_sel 信号输入到 Crossbar 模块，控制数据选通模块的输入输出。图 3-7 为竞争处理情况。

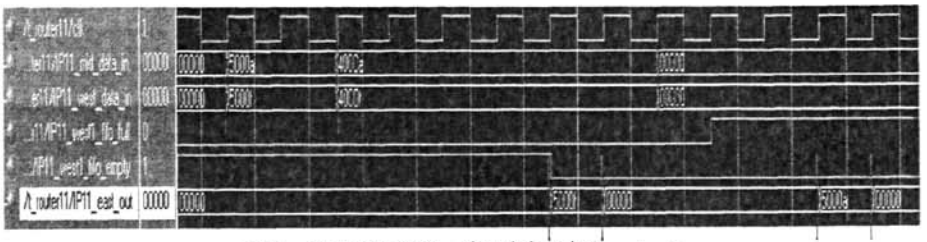

图3-7 输出竞争处理

整个仲裁模块的处理过程采用了时序状态机模式，共有 4 个状态，req 请求状态，req_get 请求获得状态,arbiter 方向仲裁状态,read 读出数据状态。

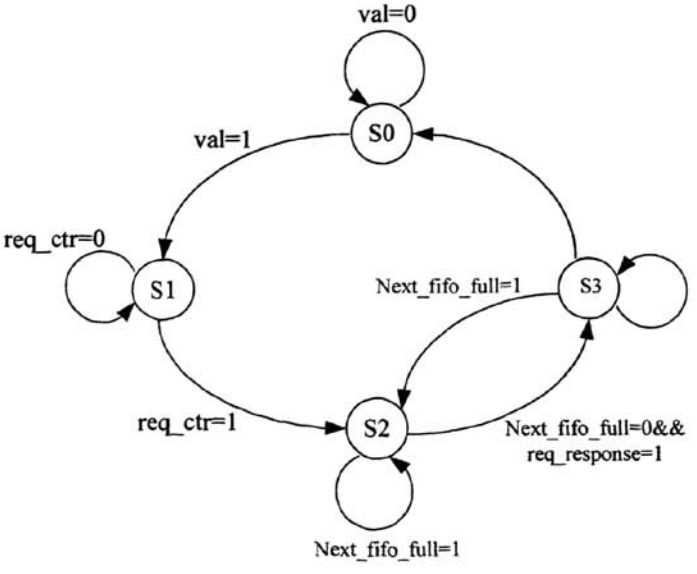

图3-8 状态机转移图

Req 状态(S0)：在 XY 维序路由中(OE 路由中)，我们将根据所接受到数据中数据有效(val)为进行检验，如果为有效数据则进入到下一状态。

Req_get 状态(S1)：在这个状态根据目标地址与当前地址的关系比较来对每个方向的请求赋值，同时需要考虑请求输出方向是否已经占用(req_ctr)，如果占用则将请求置为 7(表示不予处理)，没有占用则根据路由方式置值，具体路由方式我们已经在第二章路由算法中介绍，如果没有请求则相应方向的请求置为 7。此时对于 south1_req_1，north_req_1,mid_req_1,east_req_q,west_req_1 均已有相应方向的请求值。这里我们规定对于每一个节点来说，对于任意方向请求的置值如下，请求 east 方向输出置为 0，south 方向输出置为 1，west 方向输出置为 2，north 方向输出置为 3，mid 方向输出置为 4。

Arbiter 状态(S2)：在这个状态，我们将得到进入仲裁所需要的状态，对于每个请求的输出方向都自己的响应顺序，例如输入端口都请求 east 方向输出时，我们首先响应中间输入的请求，如果下一级对应端口的 FIFO 已满，则不予处理保持该状态等待；如果预请求的下一级节点的相应方向 FIFO 未满，(例如对于网络中的节点而言，IP00 节点 east 方向输出的下一级就是 IP01 节点 west 输入方向的 FIFO)且在该输入上数据有效时，则将该输入方向选通(req_response)，同时将其他几个竞争方向的 req_1 信号仍置为 7，表示不予处理，等待下一次竞争处理；如果不满足以上条件，则依次响应 south,north,east 方向的输入请求，处理如上所述。

对于每个输出端口来说，按优先级进行仲裁，例如对于该节点的 mid 输出端口而言如果 priority_mid=0,意味着我们将优先相应 east 方向的输出，同时将 east_fifo_re_en，cro_sel_east 拉高，将其他方向的 FIFO 读使能信号置低，cro_sel 信号也置为 7，如果该优先方向上没有请求输出信号，则按照默认顺序进行轮询，例如依次处理 south,west,north 方向。对于其他的输出端口而言，也一样有自己的优先级对应顺序，以及默认轮询处理顺序。

Read 状态(S3)：将各个方向输入 FIFO 读使能信号置为 1，同时也将相应方向的 Crossbar 输入信号拉高。

图 3-9 为仲裁模块的仿真情况。

4. Crossbar 数据选通模块

Crossbar 模块同样有五个输入端，输出亦然，根据仲裁模块中的结果产生相应输入方向的 cro_sel 信号,控制该方向的 FIFO 读出以及数据输出。(图以 IP00 节点

为例示)

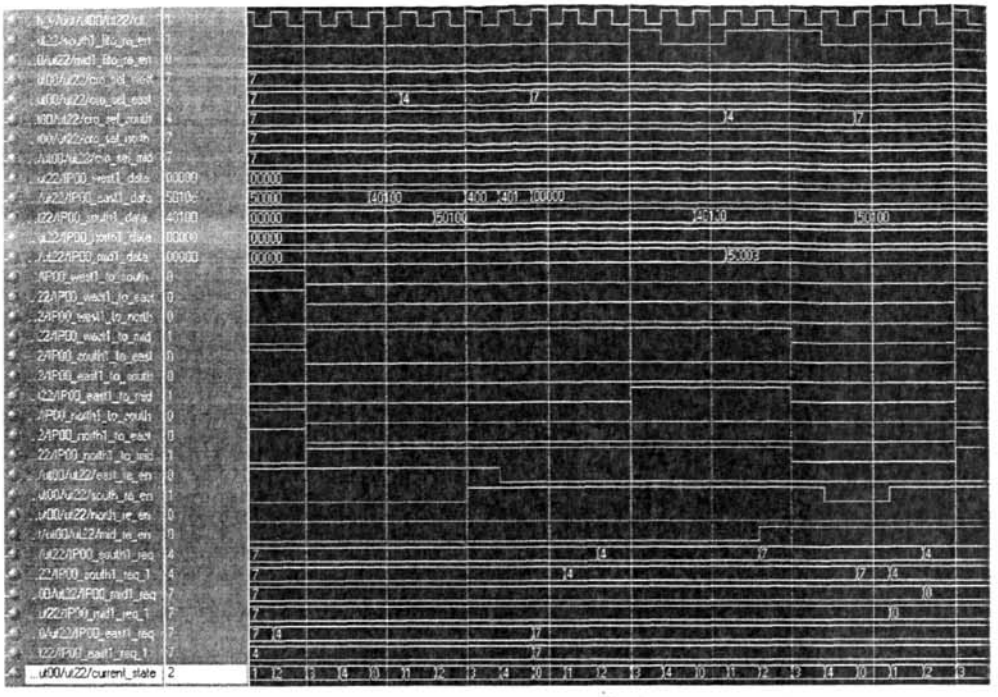

图3-9 仲裁模块 Arbiter

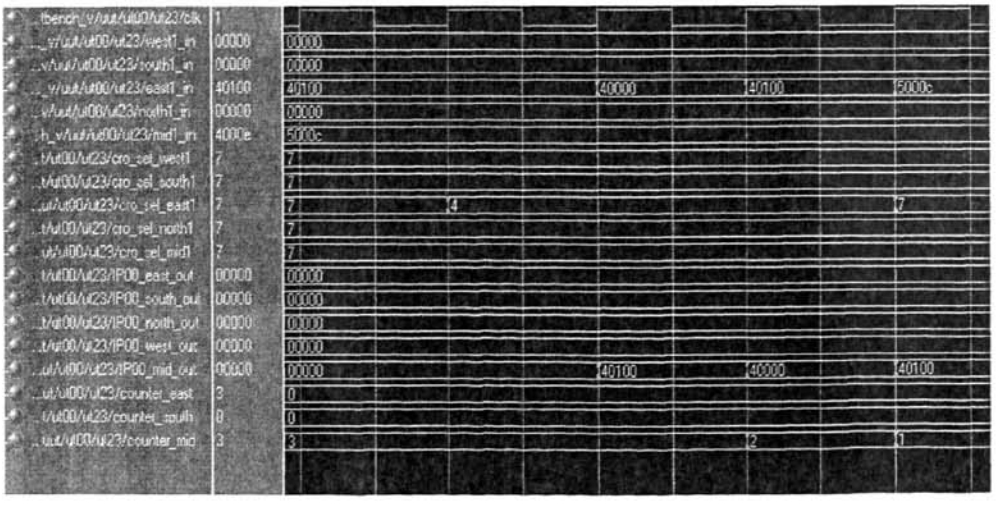

图3-10 数据选通模块 Crossbar

3.3.2 传输模式研究及仿真实现

在构建的仿真平台上，涉及到具体的应用需要不同的传输模式，比如当探讨

hot_potato 的路由方式时，我们的数据在网络中是以 flit 的形式传输的，而当我们把具体的应用划分成小的处理模块，就像我们在后面划分的运动补偿各模块时，每个节点所映射的 IP 处理模块之间处理的数据往往是很多连续的数据流，在网络中传输的时候，将以包的形式传输，二者的数据格式如下图所示：

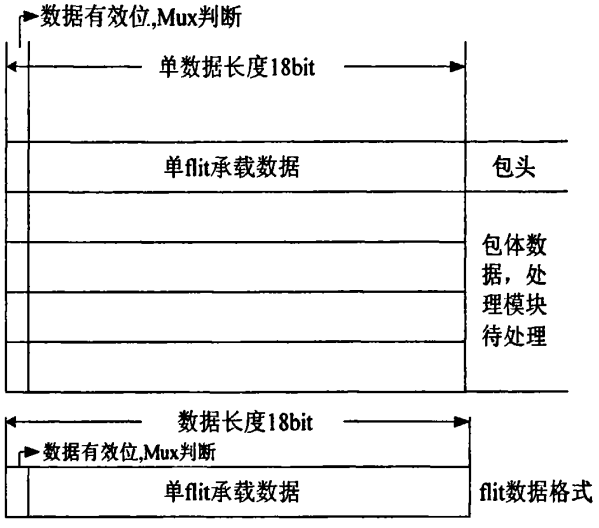

图3-11 包数据格式，flit 数据格式

相应地针对帧数据传输和包数据传输模式传输 Arbiter 需要采用不同的设计方式，对于帧数据传输，每个帧都进入状态机处理,状态机的状态分别为 req，req_get，Arbiter，read。其中 req 状态为路由节点的五个方向均有数据请求，则处理，并产生 req_1 信号，进入下个状态 req_get 后，将进行最初步的仲裁,将各自方向的 req_1 信号赋值给 req 信号，进入 Arbiter 状态进入优先级响应判决，选通传输方向，同时未选通方向将被置为最初始状态，read 状态则数据输出，而对于包传输模式,因为是虚拟直通模式，我们在仲裁时还需要需要考虑下一级传输方向的 FIFO 情况，当下一级 FIFO 可以容纳整个包时，我们才能对包头帧进行状态机仲裁，产生响应的 req 信号。后续帧则不用进行状态机裁决,只需按照包头帧选通方向传输即可。

图 3-12，图 3-13 给出了帧传输模式与包传输模式的仿真格式:

从下面两图我们可以看出，在包传输模式中(图 3-12),我们仅对包头帧进行状态机仲裁，而后续帧则直接按预先选通的路径传输，对于帧数据传输(图 3-13)，我们则是每个数据都要进行状态机仲裁，再进行传输。

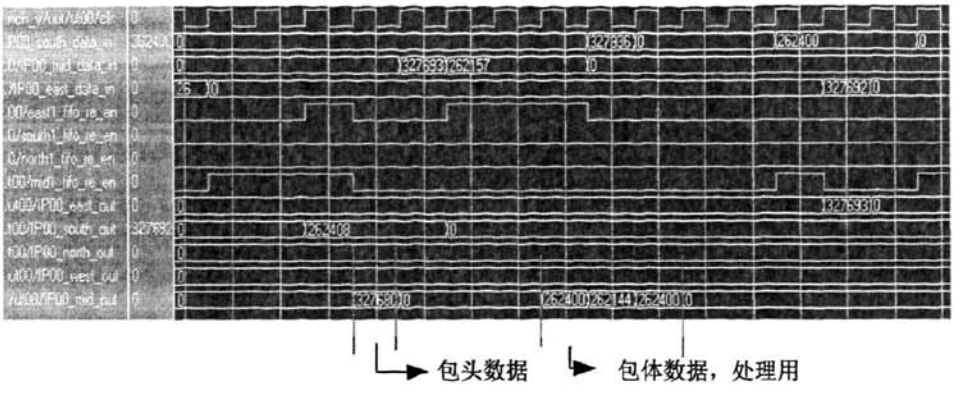

图3-12 包模式数据传输

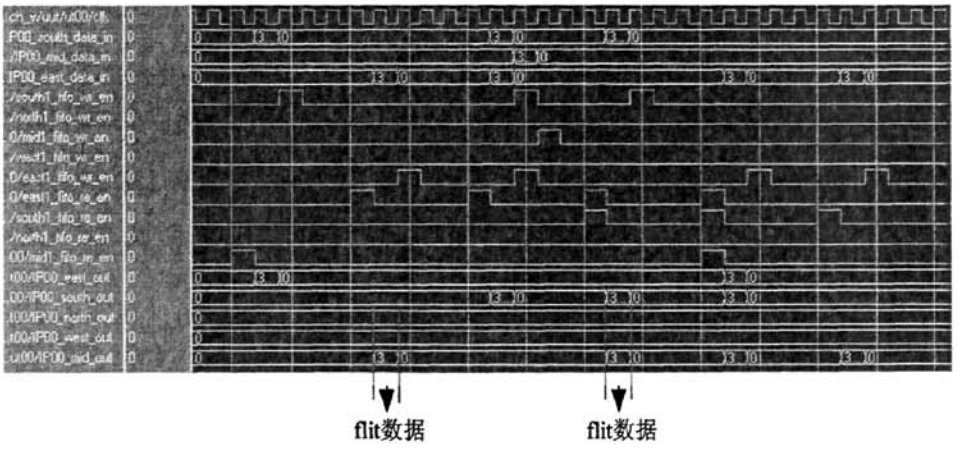

图3-13 帧模式数据传输

3.4 测试结果分析

整个测试过程采用 ModelSim + Matlab 来完成。通过往每个节点中 Mid 方向注入数据，直到目的节点收到数据，测出网络的延迟和吞吐量。测试示意图如图 3-14 所示。

Source data 为产生数据源，它是我们采用 Matlab 根据我们设计的数据包格式以及数据注入网络的时刻产生的数据。Initial Time 表示数据包注入网络的时刻；Receiver Time 表示每个数据切片离开网络的时刻；Finish Time 表示最后一个数据切片离开网络的时刻。对这些参数的产生以及测试按照如下方法进行：

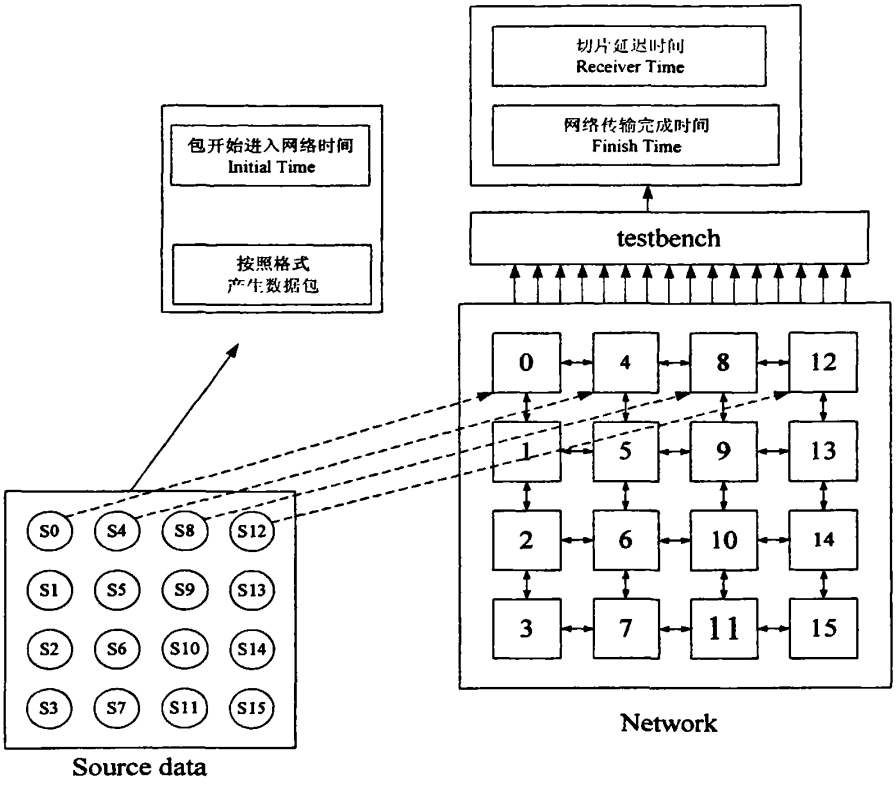

图3-14 网络测试示意图

首先，我们假定数据到达网络的过程是一个 Poisson 过程(以参数 Lambda 调节数据插入率)，并且每个节点首个到达数据的时间是随机的。由于 Poisson 过程是一个计数过程，不利于我们对数据进行统计测试，因此，在数据产生的过程中，我们用时间间隔为指数分布的序列(均值为 1/Lambda)来表示数据到达的时间，方法如下：

通过 T = exprnd(1/Lemda,1,Length)得到数据包插入网络的时间间隔。通过产生的 T 值，设置数据进入网络的时刻 Time_insert，Time_insert 是数据产生时间的叠加。Time_insert (i+1)= Time_insert (i)+T(i+1)。如果 T=[5 12 8 6 14 15 9 8 12 10]，则可以得到 Time_insert =[5 17 25 31 45 60 69 77 89 99]。在每个 Time_insert 时刻，产生的数据包并通过语句 dlmwrite('mem00.txt',mem，'')将得到的数据保存为与 ModelSim 兼容的.txt 文件以便将数据在 Testbech 中读入网络。通过这样的方法，我们依次产生 mem00～mem33 的数据。

在用 Matlab 产生好数据后，在 ModelSim 的 Testbench 中，我们通过使用 $readmemb("memXX.txt",mem)将每个节点产生的数据读入网络中(XX 的值从 00～33)。同时，我们在 Testbench 中设置一个全局变量 timer，timer 用于全局计时，每经过一个时钟周期 timer 的值加 1。当目的节点接收到有效数据的时候，通过 $fdisplay(FILE_ID,"%d",timer) 记录下当时的 timer 的值，并通过 FILE_ID=$fopen("Receive.dat")写入 Receive.dat 文件，这样就得到了数据离开网络时的时刻，最后一个 Receive 的值就是我们测试的数据中最后一个离开的网络时间。

我们用数据离开网络的时刻减去数据进入网络的时刻就得到了数据在网络中的延迟，然后求平均，就能得到数据在网络中的平均延迟。并且，我们可以根据第一个数据片进入网络的时刻、最后一个数据片离开网络的时刻以及发送包的总数量得到网络的平均吞吐量。

由图 3-15，图 3-16 可以看出在均匀模式下当插入率比较大时 XY 路由算法的延迟要小于 OE 的路由延时情况，在数据插入率比较小的时候则不是很明显。在热点模式下同样是插入率大时 XY 延时要小于 OE，插入率小的时候也不是很明显，考虑到我们在后面的应用研究中很有可能会出现热点情况且模块之间有很大的通信量，所以我们可以在平台中采用 XY 维序路由算法，对于其他比较简单的应用研究则二者均可。

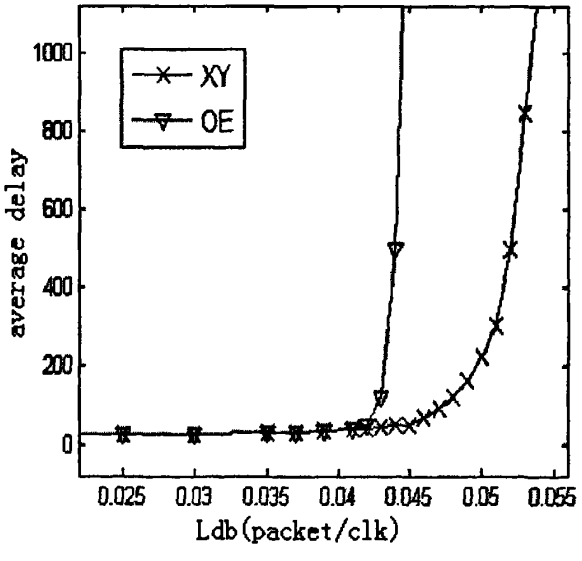

图3-15 均匀模式下延迟比对

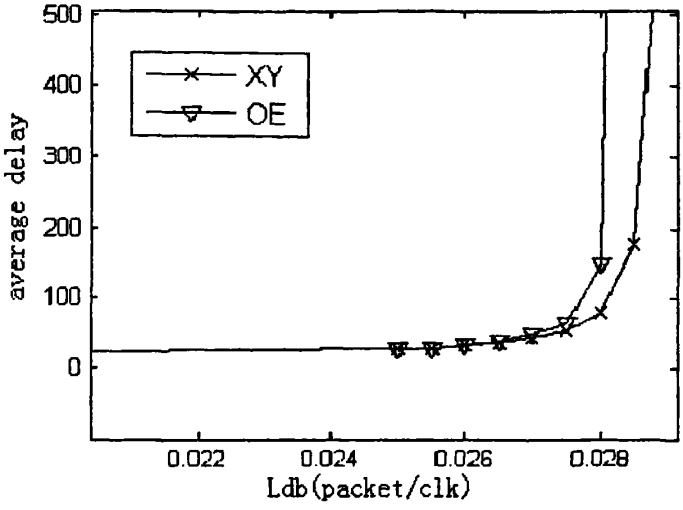

图3-16 热点模式下延迟比对

表3-1 OE 路由模式

OE module	Occupied slices	4 input LUTs	Total equivalent gate	Numbers
Mux	0	0	160	5
FIFO	93	118	1687	5
Arbiter	173	361	3846	1
Crossbar	154	307	2602	1

表3-2 XY 维序路由模式

XY module	Occupied slices	4 input LUTs	Total equivalent gate	Number
Mux	0	0	160	5
FIFO	93	118	1687	5
Arbiter	146	193	2231	1
Crossbar	154	307	2602	1

在资源消耗上 XY 则略小于 OE 路由(应用 ISE 9.1 进行资源评估)，二者主要体现在 Arbiter 资源消耗上，因为在设计网络路由节点时，二者的主要区别就是在仲裁器的设计上有所不同。

综合上述两种考虑我们最终选择 XY 维序路由作为我们仿真平台的路由算法，OE 路由则针对某些特定的需求来使用。

3.5 本章小结

这一章中主要介绍了我们设计的片上网络仿真平台，包括以下几个方面：

- 设计了比较成熟的路由节点，对其中每个小的模块进行了功能验证。
- 将独立的路由节点连成了 4×4 的 Mesh 网络，为我们在网络中进行路由算法验证，应用研究提供了一个平台。
- 验证了两种路由算法：XY 维序路由,OE 路由，同时对其资源消耗，网络延时情况作了比较，为我们在后续工作中进行应用研究奠定了基础。
- 对不同需求下的传输模式进行了设计，实现了包模式传输(在后面设计功能划分时应用了包模式传输)，流片 flit 模式传输。

第四章　片上网络应用研究

对于片上网络而言，最终都将走向应用研究，由于笔者以前有过在 VC 环境下进行视频处理的基础，因此我们尝试将视频处理 H.264 应用到片上网络的研究中，这一章我们将着重介绍一下 H.264 的相关知识，为下一章进行处理模块的映射做好预先研究准备。

4.1　H.264 相关背景

视频压缩编码标准[12][13][14]的制定工作主要是由国际标准化组织(International Standardization Organization, ISO)和国际电信联盟(InternationalTelecommunication Union, ITU)完成的。由 ITU 组织制定的标准主要是针对实时视频通讯的应用，如视频会议和可视电话等，它们以 H.26x 命名：而由 ISO 和 IEC(International Electrotechnical Commission，国际电工委员会)的共同委员会中的 MPEG[15]组织(Moving Picture Expert Group)制定的标准主要针对视频数据的存储(如 VCD 和 DVD)、广播电视和视频流的网络传输等应用，它们以 MEPG-x 命名。H.264 标准为 ITU 和 ISO/IEC 联合制定的，在 ISO/IEC 中，该标准的正是名称为 MPEG-4 AVC(Advanced Video Coding)标准，作为 MPEG-4 标准的第十部分；在 ITU-T 中的正式名称为 H.264 标准[15][16]。

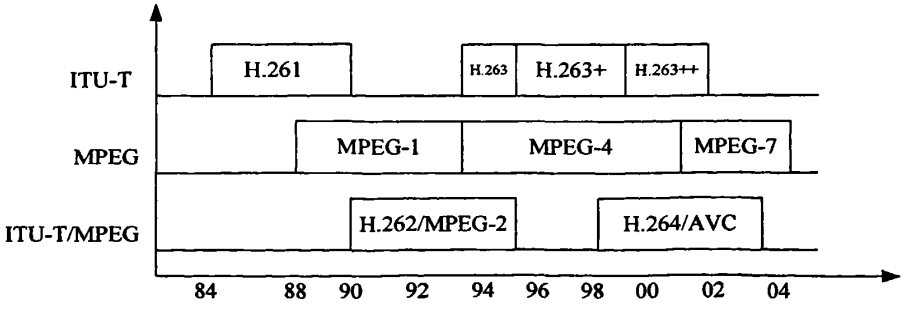

图4-1 视频编码标准发展

较以往标准，H.264 主要在以下几方面做了较大改进：

(1)帧内预测

以前的标准只利用了一个宏块内部的相关性，而忽视了宏块之间的相　性，所以一般编码后的数据量较大。为了能进一步利用空间相关性，H.264/AVC 引入了帧内预测以提高压缩效率。简单地说，帧内预测编码就是用周围邻近的像素值来预测当前的像素值，然后对预测误差进行编码。这种预测是基于块的，对于亮度分量，块的大小可以在 16×16 和 4×4 之间选择，16×16 块有 4 种预测模式，4×4 块有 9 种预测模式；对于色度分量，预测是对整个 8×8 块进行的，有 4 种预测模式。除了 DC 预测外，其他每种预测模式对应不同方向上的预测

(2)帧间预测

与以往的标准一样，H.264/AVC 使用运动估计和运动补偿来消除时间冗余，但是它具有以下五个不同的特点:

① 预测时所用块的大小可变 H.264 一共采用了 7 种方式对一个宏块进行分割，每种方式下块的大小和形状都不相同，这就使编码器可以根据图像的内容选择最好的预测模式。与仅使用 16×16 块进行预测相比，使用不同大小和形状的块可以使码率节省 15%以上。

② 更精细的预测精度

在 H.264/AVC 中，亮度分量的运动矢量(MV)使用 1/4 像素精度。色度分量的 MV 由亮度 MV 导出。如此精细的预测精度较之整数精度可以使码率节省超过 20%。

③ 多参考帧(multiple reference frames)

H.264/AVC 支持多参考帧预测，即可以有多于一个(最多 5 个)的在当前帧之前解码的帧可以作为参考帧产生对当前帧的预测(motion-compensated prediction)。较之只使用一个参考帧，使用 5 个参考帧可以节省码率 5-10%。

④ 循环滤波器(Loop Filter)

它的作用是消除经反量化和反变换后重建图像中由于预测误差产生的块效应，即块边缘处的像素值跳变，从而一来改善图像的主观质量，二来减少预测误差。与以往的 Deblocking Filter 不同的是，经过滤波后的图像将根据需要放在缓存中用于帧间预测，而不是仅仅在输出重建图像时用来改善主观质量，也就是说该滤波器位于解码环中而非解码环的输出外。

(3)整数变换

H.264/AVC 对帧内或帧间预测的残差(residual)进行 DCT 变换编码。为了克服浮点运算带来的硬件设计复杂，更重要的是舍入误差造成的编码器和解码器之间不匹配的问题，新标准对 DCT 的定义做了修改，使得变换仅用整数加减法和移位操作即可实现，这样在不考虑量化影响的情况下，解码端的输出可以准确地恢复编码端的输入。当然这样做的代价是压缩性能的略微下降。此外，该变换是针对 4×4 块进行的，这也有助于减少块效应。为了进一步利用图像的空间相关性，在对 Chroma 的预测残差和 16×16 帧内预测的预测残差进行上述整数 DCT 变换之后，标准还将每个 4x4 变换系数块中的 DC 系数组成 2×2 或 4×4 大小的块，进一步做 Hadamard 变换。

(4)熵编码

对于 Slice 层以上的数据，H.264/AVC 采用 Exp-Golomb 码，这是一种没有自适应能力的 VLC。而对于 Slice 层(含)以下的数据，如果是残差，H.264/AVC 有两种熵编码的方式：基于上下文的自适应变长码(Context-based Adaptive Variable Length Coding,CAVLC)和基于上下文的自适应二进制算术编码(Context-based Adaptive BinaryArithmetic Coding, CABAC)；如果不是残差，H.264/AVC 采用 Exp-Golomb 码或 CABAC 编码，视编码器的设置而定。

4.2 H.264 编码技术探讨

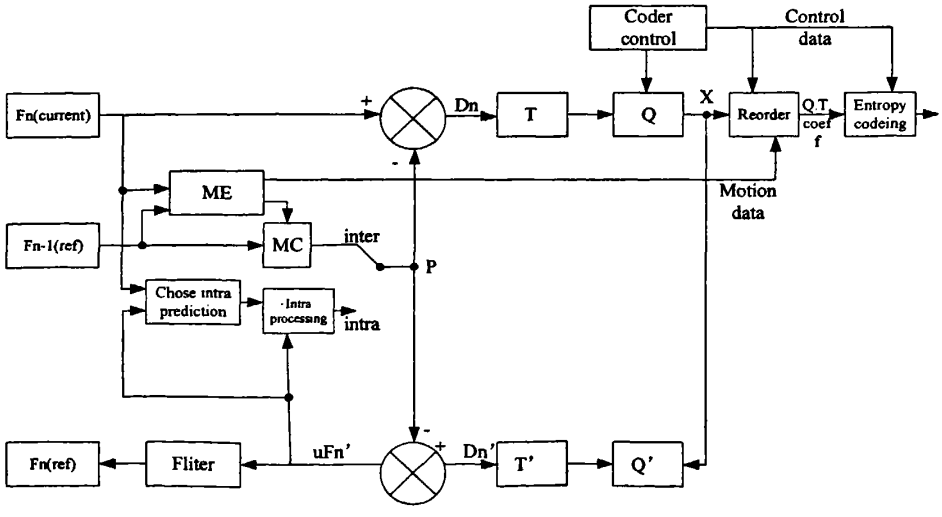

图4-2 编码框图

由上图可知，编码过程主要涉及帧内预测、变换量化相关、熵编码、 运动搜索、块滤波几个主要过程，下面我们将详细介绍每一个过程。

4.2.1 帧内预测

帧内预测的目的是生成对当前宏块的预测值。一个宏块由一个 16×16 的 亮度分量和两个 8×8 的 色度分量构成，亮度块有两类帧内预测方式，按标准中的记号表示为：Intra_16x16 和 Intra_4x4；而两个 色度 分量则采用相同的预测方式。Intra_16x16 是对整个 16×16 大小的 亮度 进行预测，一般用于图像比较平坦的区域，共有 4 种预测方式。而 Intra_4x4 方式是将 16×16 大小的 亮度 划分为 16 个 4×4 大小的亮度块，然后对每个 4×4 大小的块进行预测，共有 9 种预测方式。

Intra_16x16 帧内预测模式根据与当前宏块邻近的 33 个像素来生成 luma 分量的预测数据，共有 4 种预测方式：垂直(vertical)、水平(horizontal), DC 和平面(plane)，如图 4-3 所示。图中白色部分代表当前宏块参考的邻近像素，它们分别来自当前宏块的上方(以 H 表示)、左方(以 V 表示)和左上方，因为这些宏块在编码顺序上位于当前宏块之前，所以用它们来预测是合理的。

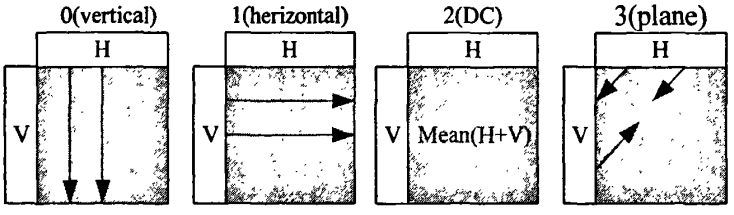

图4-3 帧内预测模式

在进行预测之前，首先要判断这些邻近像素是否可用，如果这些像素不可用，例如邻近像素所在的宏块位于其它 slice 之中或当前宏块位于图像边缘时，某些预测模式就用不起来。具体判断的细节请参考标准。对于垂直模式，如果 H 可用的话，预测值即为 H，否则不能使用此模式。对于水平模式，如果 V 可用的话，预测值即为 V，否则不能使用此模式。对于 DC 模式，如果 H 和 V 都可用，就用这 32 个像素的均值作为预测值，如果只有 H 或 V 可用，就用这 16 个像素的均值作为预测值，如果 H 和 V 都不可用，例如当前宏块位于一个 Slice 的开头，则预测值为 128。

对于 plane 模式，要求必须所有的 33 个邻近像素都可用。为便于叙述，引入一个坐标系，其中横向为 x 轴，纵向为 Y 轴，定义当前宏块左上角像素的坐标为(0,0)。用 P(x,y)表示位于坐标(x,y)处的 33 个邻近像素值，其中 H 对应 p(x,-1)，x=0..15，V 对应 p(-1,y)，y=0..15，而左上角处的邻近像素值为 p(-1,-1)。预测值用 predL(x,y)，x,y=0..15 表示。则 plane 预测模式的步骤及公式如下：

1)计算中间变量 H 和 V:

$$V=\sum_{Y'=0}^{7}(Y'+1)\times(P[-1,8+Y']-P[-1,6-Y'] \tag{4-1}$$

$$X_{QD}\,H=\sum_{X'=0}^{7}(X'+1)\times(P[8+X',-1]-P[6-X',-1] \tag{4-2}$$

2)计算中间变量 a,b,c

$$a=16\times(p[-1,15]+p[15,-1]) \tag{4-3}$$

$$b=(5\times H+32)>>6 \tag{4-4}$$

$$c=(5\times V+32)>>6 \tag{4-5}$$

其中 Clip1(x)将代表 x 将位于 0 到 255 之间，即

$$Clip(x)=\begin{cases}255 & x>255\\ 0 & x<0\quad 0\\ x & 0\le x\le 255\end{cases} \tag{4-6}$$

在此模式下，编码器将当前宏块 16x16 的 luma 分量划分为 16 个 4×4 的块，然后根据每个 4×4 块周围的邻近像素对该块做预测。按理说，对一块像素做预测，其上下左右的像素都应当作为参考，但由于编码顺序的原因，H.264 只选择了 13 个像素作为参考，这 13 个邻近像素与当前块的位置关系如图 4-4 中 A-M 所示。

同 Intra_16x16 模式一样，Intra_4×4 在开始预测之前，首先需判断 A-M 这些参考像素是否可用，如果有些参考像素不可用，那么有些预测模式也就不能用了。前面已经提及，编码器端用于预测的参考数据是经过反变换与反量化后的重建图像，所以判断的主要依据就是看这些像素是否在当前的 4×4 块之前已经完成编码，即已经是经过重建了的图像。在决定了哪些参考像素可用后，就可以通

过它们产生预测值了。H.264 一共定义了 9 种 Intra 4x4 预测方式，除了 DC 方式(模式 2)之外，其它 8 种都是向某一个方向上进行预测，也就是做外插。

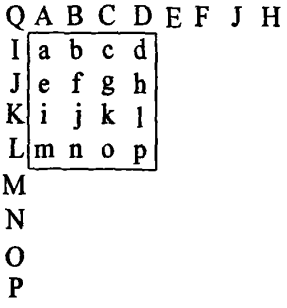

图4-4 帧内预测象素位置

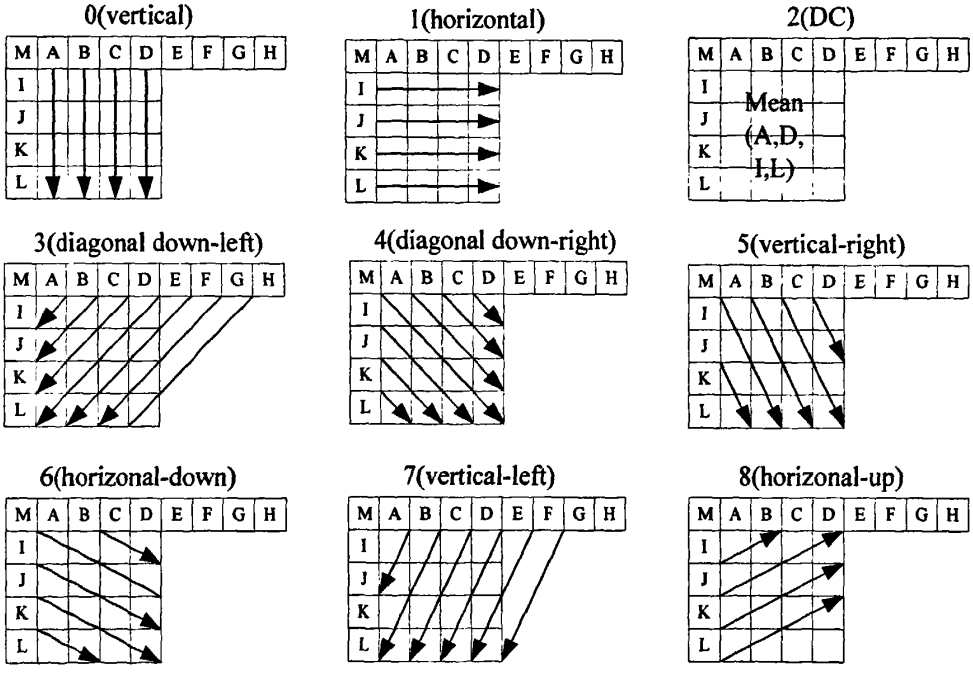

图4-5 Intra4×4 帧内预测对应模式

4.2.2 变换量化相关

H.264 把运动估值和 Intra 预测的残差结果从时域变换到频域，使用了类似于 4×4 离散余弦变换 DCT(Discrete Cosine Transform)的整数变换[17] [18]，而不是像 MPEG-2 和 MPEG-4 那样采用 8×8 DCT 的浮点数变换。由于是整数变换，因

而不存在反变换的误匹配问题，核心变换不需要乘法，只用简单的加法和位移来完成，在变换中不进行归一化运算，而是结合到量化过程中完成。

4.2.2.1 整数变换

不同于 JPEG，MPEG 以前的 DCT 变换，H.264 采用整数变换主要是为了解决在帧内预测时失配问题，同时由于变换只需要移位加减来实现，可以极大提高硬件效率。

$$
\begin{aligned}
&Y=(C_f X C_f^{T})\otimes E_f \\
&=\left(\begin{bmatrix}1&1&1&1\\2&1&-1&-2\\1&-1&-1&1\\1&-2&2&-1\end{bmatrix}X\begin{bmatrix}1&2&1&1\\1&1&-1&-2\\1&-1&-1&2\\1&-2&1&-1\end{bmatrix}\right)\otimes\begin{bmatrix}a^2&ab/2&a^2&ab/2\\ab/2&b^2/4&ab/2&b^2/4\\a^2&ab/2&a^2&ab/2\\ab/2&b^2/4&ab/2&b^2/4\end{bmatrix}
\end{aligned}
\tag{4-7}
$$

其中 $a=1/2, b=\sqrt{2/5}, d=1/2$ 其反响变换公式为：

$$
\begin{aligned}
&X'=C_i^{T}(Y\otimes E_i)C_i \\
&=\begin{bmatrix}1&1&1&1/2\\1&1/2&-1&-1\\1&-1/2&-1&1\\1&-1&1&-1/2\end{bmatrix}\left(\begin{bmatrix}Y\end{bmatrix}\otimes\begin{bmatrix}a^2&ab&a^2&ab\\ab&b^2&ab&b^2\\a^2&ab&a^2&ab\\ab&b^2&ab&b^2\end{bmatrix}\right)\begin{bmatrix}1&1&1&1\\1&1/2&-1/2&-1\\1&-1&-1&1\\1/2&-1&1&-1/2\end{bmatrix}
\end{aligned}
\tag{4-8}
$$

其中 a=1/2, $b=\sqrt{2/5}$。

4.2.2.2 哈达变换

$$
Y_D=\left(\begin{bmatrix}1&1&1&1\\1&1&-1&-1\\1&-1&-1&1\\1&-1&1&-1\end{bmatrix}[W_D]\begin{bmatrix}1&1&1&1\\1&1&-1&-1\\1&-1&-1&1\\1&-1&1&-1\end{bmatrix}\right)/2 \tag{4-9}
$$

$$
X_{QD}=\left(\begin{bmatrix}1&1&1&1\\1&1&-1&-1\\1&-1&-1&1\\1&-1&1&-1\end{bmatrix}[Z_D]\begin{bmatrix}1&1&1&1\\1&1&-1&-1\\1&-1&-1&1\\1&-1&1&-1\end{bmatrix}\right) \tag{4-10}
$$

其中 W_D 为正向变换的待处理矩阵，Y_D YD 为处理后的矩阵。Z_D ZD 为反向变换的待处理矩阵，X_{QD} 为处理后的矩阵。

在 16×16 点 Intra 模式下，对于亮度 DC 系数，相邻的 4×4 点像素块先通过整数变换，由于变换后得到 DC 系数仍有很大的相关性，再将每个 4×4 点的 DC 系数抽出，采用 4×4 点哈达变换[19]进行处理，所得的输出再经过量化得到最终输出的 DC 系数块；对于色度 DC 系数块，将每个 8×8 宏块分割成 4 个 4×4 点块，每个 4×4 的点块先进行整数变换进行处理，再将每一个 4×4 点块的 DC 系数取出，采用 2×2 点哈达马变换进行处理，输出系数再经过量化得到最终输出的 DC 系数块。

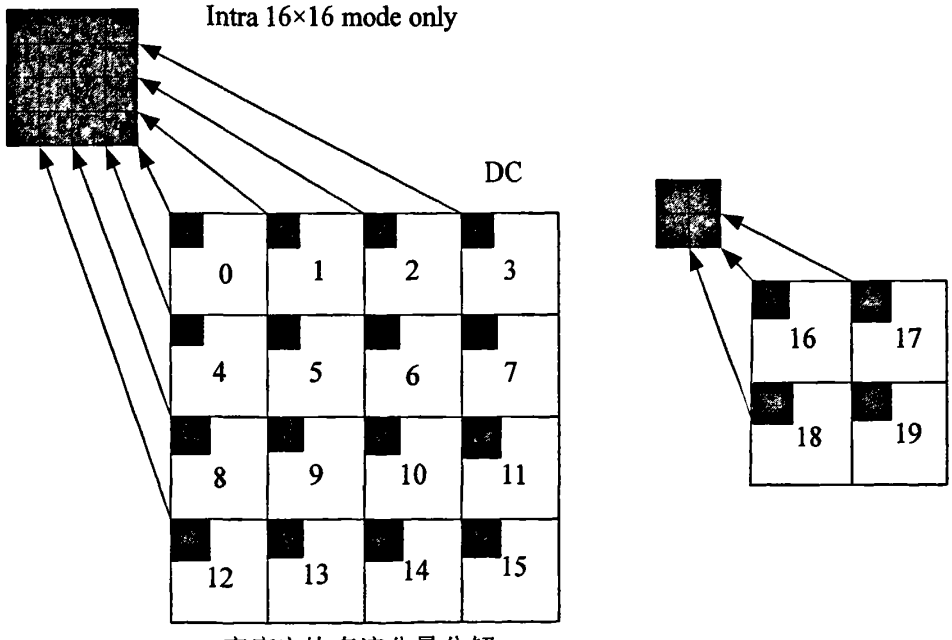

图4-6 哈达变换元素示意图

在进行了二维核心变换后，缩放过程是和量化过程相结合的。H.264 中采用分级量化的方法，一共支持 52 个等级(由 QP 表示)。量化步长选择的范围很大，使编码器可以准确灵活地控制视频流的码率和质量。量化步长基本符合指数关系，QP 每增加 6，量化步长就加倍，。为了避免视觉熵明显的文理效应，色度分量的

QP 值大致限定为亮度 QP 值的 80%。

表4-1 量化表

QP	0	1	2	3	4	5	6	7	8	9	10	•••
Q_{step}	0.625	0.6875	0.8125	0.875	1	1.125	1.25	1.375	1.625	1.75	2.25	•••
QP	•••	24	•••	30	•••	36	•••	42	•••	48	•••	51
Q_{step}	•••	10	•••	20	•••	40	•••	80	•••	160	•••	224

4.2.3 熵编码

在完成变化与量化之后，这些数据就送往熵编码[20]器，完成整个变换编码的最后一步。H.264 一共有 3 种熵编码:(1)Exp-Golomb 码(Exponential Golomb codes) ； (2)CAVLC (Context-based Variable Length Coding) ； (3)CABAC(Context-basedAdaptive Binary Arithmetic Coding)。H.264 对这三种码的使用范围做了规定:(1)不出现在残差数据中;(2)仅出现在残差数据中:(3)仅出现在 Slice 层以下(从 Slicedata 开始)的数据中。(1)和(2)都是采用查表方式，但是(1)的表是固定的，而(2)在编码过程中会根据周围宏块以及在之前编码的数据信息，选择不同的表，从而具有上下文自适应功能。(3)属于自适应算术编码，能够获得比(2)更好的压缩性能和自适应能力，但由于复杂度过高，故不常采用，本文主要介绍前两种熵编码方法。

4.2.3.1 Exp-Golomb 编码

Exp-Golomb 码的编码过程分为两步：第一步将待编码的数据转换为一个中间变量 codeNum。根据转换方式，可将 Exp-Golomb 码分为 4 种，按 H.264 的记号分别表示为 ue,me,se 和 te。其中 ue 和 se 分别用于无符号整数和有符号整数的编码，me 用于对 coded_block_pattern 编码，这是表示当前宏块中哪些子块含有非零系数的一个参数，te 用于编码 ref_idx_10 和 ref_idx_l1，该参数表示使用缓存中的哪个参考帧做帧间预测。

$$M = floor(log2\ [code\ \_\ num\ +1] \tag{4-11}$$

$$INFO = code\ \_\ num\ +1-2 \tag{4-12}$$

式中 code_num 为欲编码码字

表4-2 指数哥伦布码表

Exp_Golomb 码	码字
1	0
01^{x_0}	1-2
$001^{x_1x_0}$	3-6
$0001^{x_2x_1x_0}$	7-14
$00001^{x_3x_2x_1x_0}$	15-30
$000001^{x_4x_3x_2x_1x_0}$	31-62
•••••	•••••

4.2.3.2 CAVLC 编码

CAVLC 的设计考虑了如下几个事实：1)经过变换与量化后的预测残差中含有较多的 0，这样在 Zig-Zag 扫描之后，用 Run 和 Level 表示预测残差可以取得较好压缩效果。这一点在以前的标准中也用到了。这里 Level 表示非零系数值，Run 表示非零系数之前的 0 的数目。2)残差末尾的几个非零系数一般为±1(trailing 1s,T1s),C AVLC 对它们单独进行了编码。3)作为空间相关性的一种表现，当前块中的非零系数数和周围块的中的非零系数数应该差不多，CAVLC 利用这一点自适应地选择编码当前块中非零系数数的码表。4)位于低频处的系数值一般较大，而位于高频处的则相反，CAVLC 利用这一点自动地选择编码 Level 的码表。5)位于低频处的非零系数一般是连着的，中间没有零，此时用(Run,Level)形式来表示它们就显得效率不高，因此 CAVLC 将 Run 和 Level 分开单独进行编码。编码流程如图 4-7 所示。

第一个被编码的语法元素(syntax element)是 coeff_token，它包含两个信息：所有的非零元素个数(total_coeff，包括 T1s)和 T1s 的个数。Total_coeff 可取到 16，T1s 可取 0 到 3。一共有 5 个码表供编码器进行选择：除了专用于 chroma DC 系数的码表之外，其余 4 个码表分别对不同的 total_coeff 取值范围做了优化，表 0 适用于 total_coeff 较小的情况，对比较小的 total_coeff 赋予较短的码字，表 1 则适用于 total_coeff 取值中等的情况，对中等取值的 total_coeff 赋予较短的码字，

依此类推。如前所述，由于空间相关性，当前块中的 total_coeff 与周围块的十分相近,因此可以依据周围块的 total_coeff 从 4 个码表中选择一个最好的表进行编码，这就体现了 CAVLC 的上下文自适应的能力。

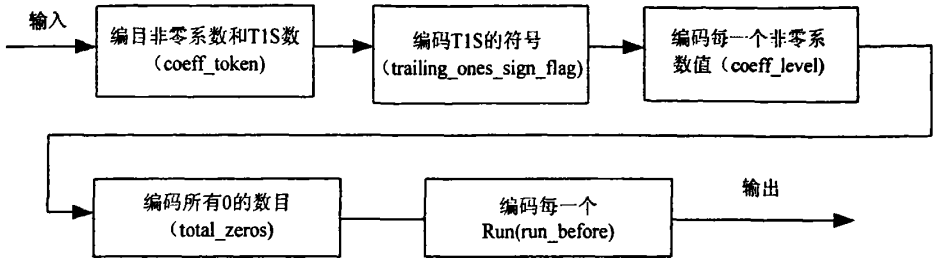

图4-7 CAVLC 编码流程

随后被编码的是 T1s 的符号，0 表示正，1 表示负。CAVLC 对残差数据的编码顺序与 Zig-Zag 扫描的顺序相反，即从高频向低频走。

第三个被编码的是 Tls 之后的 Level 数据。对它的编码也体现了 CAVLC 的上下文自适应的能力。同 coeff_token 类似，CAVLC 根据 Level 的不同取值范围建了 7 个码表 VLC0-VLC6,VLC0 倾向于给较小的 Level 赋予较短的码字，而 VLC1 会给稍大一点的 Level 赋予较短的码字，依此类推。随着 Level 值由小到大地变化，CAVLC 会根据先前被编码的 Level 决定是否地转至下一个码表，判断依据就是给每个码表设一个阈值，如果大于则跳转。

表4-3 Level 编码阙值

VLC	阙值
VLC0	0
VLC1	3
VLC2	6
VLC3	12
VLC4	24
VLC5	48
VLC6	N/A

初始码表是 VLC0，但如果 total_coeff>10 且 T1s<3 的话则选 VLC1。每编码一个 Level，编码器便将其与表 4.2 中的阈值进行比较，如果 Level>阈值，则转至下一个码表，从表中可以看出，VLC0 最多被使用 1 次，因为它的阈值为 0。这里

需要说明一个特殊情况，当 T1s<3 时，第一个 Level 肯定不会取±1,所以对这个 Level 编码时将其绝对值减 1 以减小码长。

CAVLC 的第四步是编码最后一个非零系数之前所有 0 的个数 total_zeros。以往的标准中，每一个非零系数都与一个二元组(Run, Level)相联系，编码是针对这个二元组进行的。前面讲过，残差开头的几个非零系数一般是连着的，它们之间没有 0，因此它们的 Run 就没必要编码。因此 CAVLC 就将二元组(Run, Level)拆开，Run 与 Level 分别编码，这样就能够提高压缩效率。显 total_zeros<totale_coeff，所以 totale_coeff 可以决定 totale_zeros 的取值范围，并且通过观察可以发现，不同的 totale_coeff 下 totale_zeros 的分布也有不同，所以 CAVLC 根据不同的 totale_coeff，建立了不同的码表用于对 total_zeros 的编码，这也体现了 context adaptive。

最后一步就是对 Run 进行编码。CAVLC 引入了一个变量 zerosLeft，表示还没有编码的 0 的个数，初值为 totale_zeros。显然 Run 的取值范围必须在 0 到 zerosLeft 之间，这样 zerosLeft 就决定了可以用多少 bit 来表示 Run。例如，如果 zerosLeft=1，那么 Run 只能取 0 或 1，所以只需 1 bit 就可以对其进行编码，而如果 zerosLeft=6，那么就需要 3bit 因为 Run 的取值范围在 0 到 7 之间。当然，这说的是定长码的情况，CAVLC 根据不同 zerosLeft 下 Run 的分布建立了不同的 VLC 码表，供编码器选择。这也体现了 contextadaptive 的特点。

当 Run= zerosLeft 时，编码器就可以停止编码，哪怕这个 Run 对应的 Level 不是第 1 个系数。因为剩下的非零系数(如果有的话)前面都没有 0 了，不需要再进行编码了。这就体现出 Run 和 Level 分开编码的优越性。

4.2.4 运动估计

运动估计可以把 inter 帧的压缩性能进一步的提高，常见的运动补偿是使用相邻帧适当位置的图像块来预测和匹配当前的图像块。通常的标准是将图像划分为许多子块，并认为子块内所有像素的位移量是相同的，这意味着将每个子块视为一个“运动的物体”。对于某一时间 t1，图像帧中的某一子块如果在另一时间 t2 的帧中可以找到若干与其十分相似的子块，则称其中最为相似的子块为匹配块，并认为该匹配块是时间 t1 的帧中相应子块位移的结果。位移矢量由两帧中相应子块的坐标决定。下图说明了当前帧和参考帧的匹配过程：

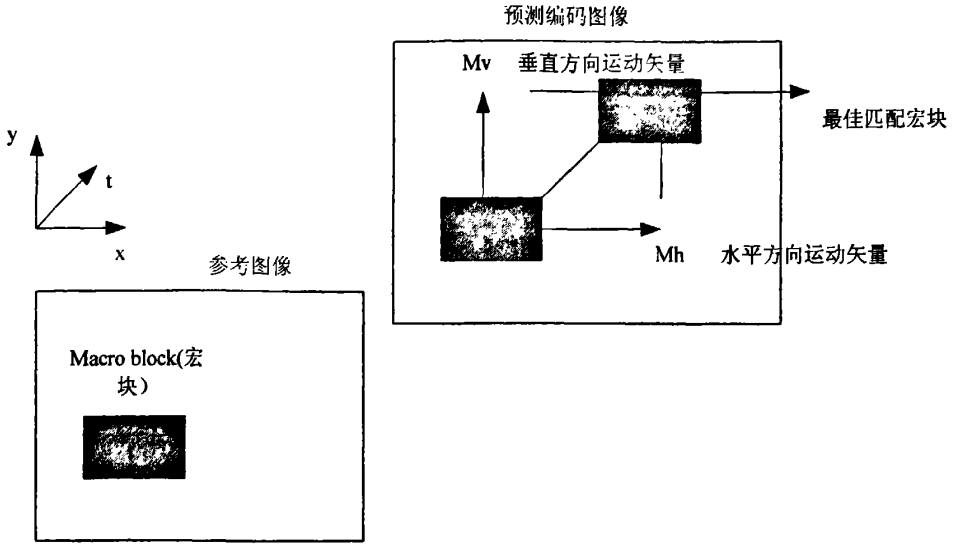

图4-8 运动估计示意图

H.264 中添加了不同块大小的搜索方式，主要是为了更好的表现图像细节，但是如果分块越细，相应的运动矢量也多，因此将在算法中引入率失真来衡量。

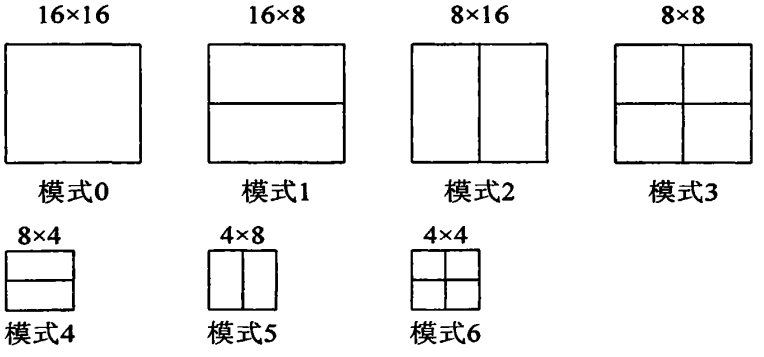

图4-9 搜索分块模式

4.2.4.2 搜索准则

很多策略殊途同归，都是以预测像素差值最小为目的，在运动估计中匹配准则对精度的影响不是很大，由于 MAD 准则不需要作乘法，实现简单方便，所以常被采用。而 SAD,MAD 只是一个因子的差别，通常用 SAD 代替 MAD，这样易于硬件实现，实际应用较多。

最小绝对差 MAD

$$MAD(i,j)=\frac{1}{MN}\sum_{m=1}^{M}\sum_{n=1}^{N}\left|f_k(m,n)-f_{k-1}(m+i,n+j)\right| \tag{4-13}$$

绝对误差和 SAD

$$SAD(i,j)=\sum_{m=1}^{M}\sum_{n=1}^{N}\left|f_k(m,n)-f_{k-1}(m+i,n+j)\right| \tag{4-14}$$

均方误差 MSE

$$MSE(i,j)=\frac{1}{MN}\sum_{m=1}^{M}\sum_{n=1}^{N}\left|f_k(m,n)-f_{k-1}(m+i,n+j)\right|^2 \tag{4-15}$$

归一化互相关函数 NCCF

$$NCCF(i,j)=\frac{\sum_{m=1}^{M}\sum_{n=1}^{N}f_k(m,n)f_{k-1}(m+i,n+j)}{\left[\sum_{m=1}^{M}\sum_{n=1}^{N}f_k^2(m,n)\right]^{1/2}\left[\sum_{m=1}^{M}\sum_{n=1}^{N}f_{k-1}{}^2(m+i,n+j)\right]^{1/2}} \tag{4-16}$$

4.2.4.3 搜索算法

运动搜索的算法：

全搜索(精准，不实用，消耗太多资源)

从原点出发顺时针由近到远搜索，最小 SAD 块对应即为运动矢量位置。

搜索示意如图 4-10 所示。

三步法：

T.Koga 提出[21]，若其最大搜索长度为 7，搜索精度为一个像素，步长为 4，2，1，所以得名三步法。

当搜索范围大于 7 的时候，仅用三步法是不够的，但它为以后很多改进算法提出了好的基础，可以进行相关探讨。

搜索示意如图 4-11 所示。

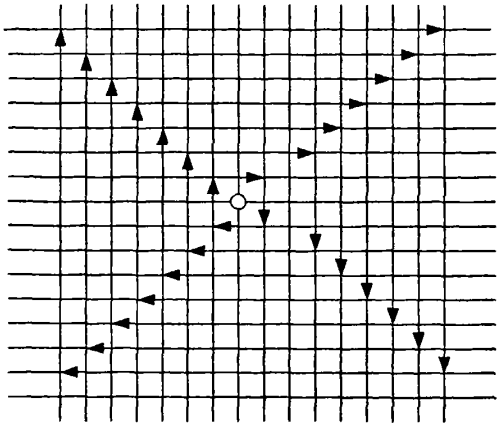

图4-10 全搜索示意

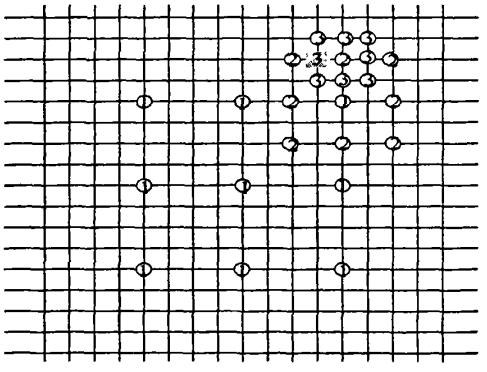

图4-11 三步法搜索示意图

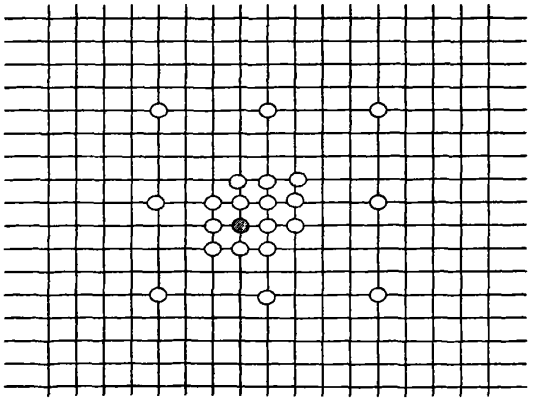

图4-12 新三步法搜索

新三步法：

引入提前中止机制。

1．首先搜索中心和八个点，最小 SAD 为中心点跳至步骤 3，否则步骤 2。

2．以上次搜索最小点为中心，继续第一步类似搜索，直到最小点为中心点。

3．三步法小搜索。

搜索示意如图 4-12 所示。

菱形搜索法

经过研究表明运动矢量基本分布在零矢量的周围。如图

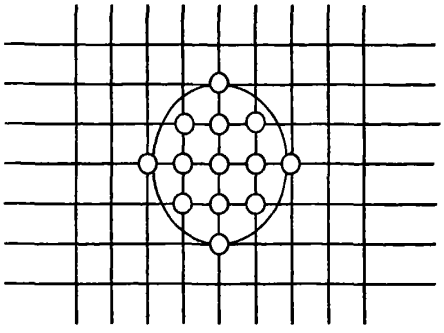

图4-13 运动矢量分布图

故菱形搜索策略显得比较高效，下图为菱形搜索策略的模板：

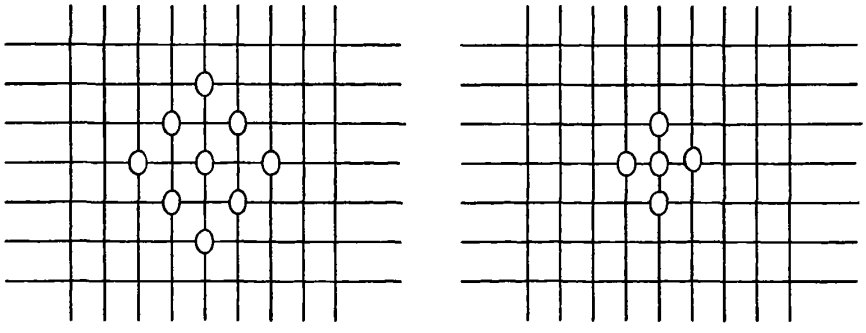

图4-14 左图为大菱形搜索，右图为小菱形搜索

搜索策略采用类似新三步法的方式，同时引入提前中止机制，下图为一次搜索示意。

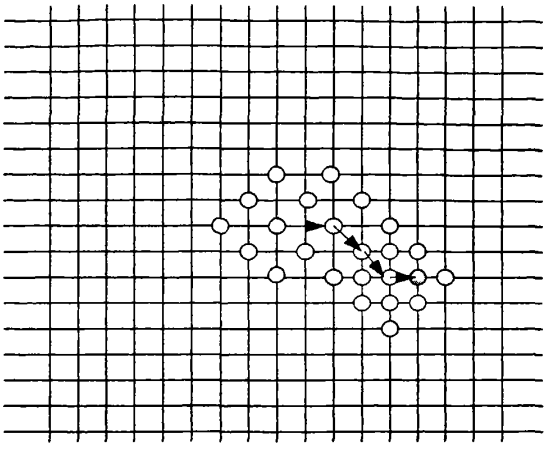

图4-15 菱形搜索示意

4.2.4.4 象素内插

H.264 中引入 1/4 像素精度内插，实验表明 1/8 精度以下对于图像效果提升已无明显效果。

A,B 为整精度位置，bb,cc 为 1/2 像素位置，a,b 为 1/4 像素位置。

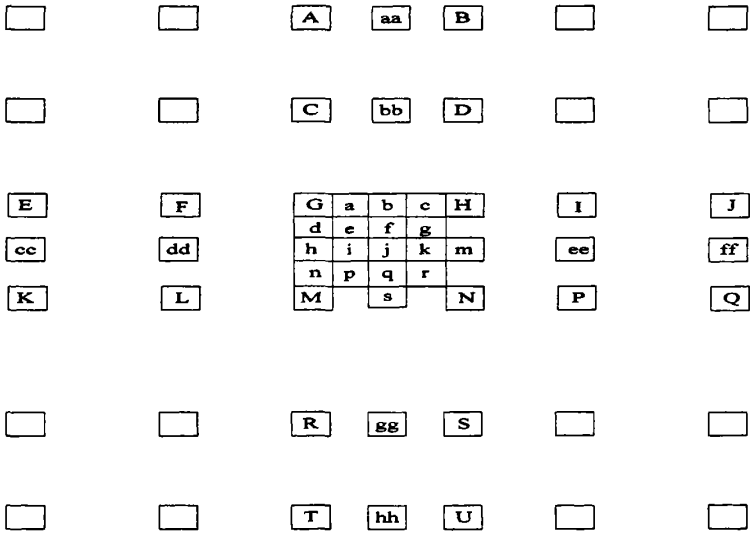

图4-16 象素内插

具体如下：

$$b_1 = E - 5 \times F + 20 \times G + 20 \times H - 5 \times I + J \tag{4-17}$$

$$h_1 = A - 5\times C + 20\times G + 20\times M - 5\times R + T \tag{4-18}$$

最终 b,h 点值由下两式得出:

$$b = \max(0, \min(255, (b_1 + 16)/32)) \tag{4-19}$$

$$h = \max(0, \min(255, (h_1 + 16)/32)) \tag{4-20}$$

$$j_1 = cc - 5\times dd + 20\times h_1 + 20\times m_1 - 5\times ee + ff \tag{4-21}$$

或 $$j_1 = aa - 5\times bb + 20\times b_1 + 20\times s_1 - 5\times gg + hh \tag{4-22}$$

最终 j 点由下式求得

$$j = \max(0, \min(255, (j_1 + 512)/1024)) \tag{4-23}$$

四分之一像素精度 a=(G+b+1)/2;

而四分之一像素精度 e=(b+h+1)/2.

色度系数象素内插

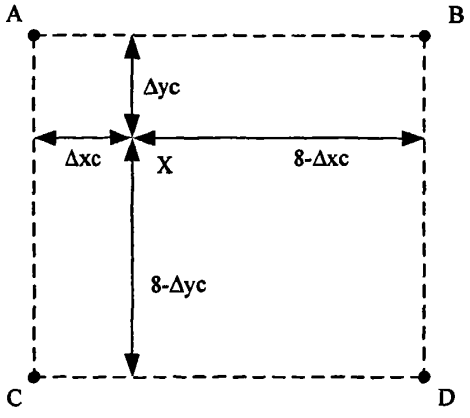

图4-17 色度象素内插

$$X = ((8-\Delta x_c)\times(8-\Delta y_c)\times A + \Delta x_c\times(8-\Delta y_c)\times B + (8-\Delta x_c)\times\Delta x_y\times C + \Delta x_c\times\Delta y_c\times D + 32)/64 \tag{4-23}$$

4.2.5 块效应滤波

H.264 去块滤波处理是应用于已解码完成的宏块，是基于 4×4 块边界的，也就是说对一个宏块的 16×16 亮度分量，需要对其 4 条水平边界和 4 条垂直界进行滤波，而 8×8 的色度分量则只需要对其 2 条水平边界和 2 条垂直边界进滤波，如图 2.6 所示，虚线为需要滤波的边界。在块边界，滤波的强度是和的编码模式、运动矢量和残差数值相关的，而对于单个像素，基于量化系数门限值可以取消对任意单个像素的滤波在编码环和解码环中使用去块滤波系统主要有两个优点：图像中由于运补偿、变换及量化产生的虚假边界可以被平滑，降低图像块效应，提高了主视觉效果；滤波后的帧用于后续帧的运动补偿与预测，从而避免了虚假边界累误差导致的图像质量进一步降低。

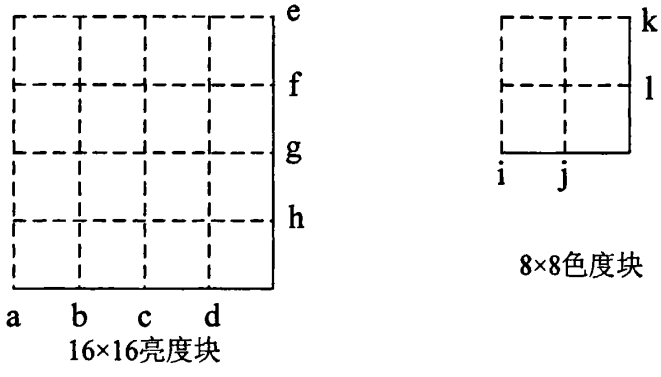

图4-18 块效应滤波边界示意图

4.3 本章小结

这一章主要介绍了视频处理标准 H.264 的整个处理流程，在实际的编程实现中，我们发现运动补偿，运动估计是整个视频处理的重中之重，相对于它而言，变换量化，熵编码(主要是由索引查表)，帧内预测，块效应滤波(对显示效果有提高)则要简单一些。当然在运动搜索的效果也要好于帧内预测的效果，因此我们在下一章将对运动估计进行分解，来试图进行初步的并行优化。

第五章 功能映射探讨

在这一章我们将初步研究片上网络应用的任务划分，在最大程度上做到并行划分，这样可以使网络负载相对均衡，我们将视频处理标准里面的运动估计映射到我们已经构建的 Mesh 仿真平台上，进行了功能划分以及数据流量研究，在功能映射上做了一定的优化。

5.1 并行处理相关

并行处理[22]的本意是将一个问题分解成若干部分，由各个处理器对各个部分分别进行计算。一个理想的并行计算是能被立即分解成许多完全独立部分且他们被同时执行的计算。这被形象地成为易并行。对这种问题进行并行化应是一目了然的，且无需特殊技术或算法就可得到一个成功解。一个真正意义上的并行计算意味着在各个进程间没有通信，即它是一个完全分离的计算图。如下图所示：

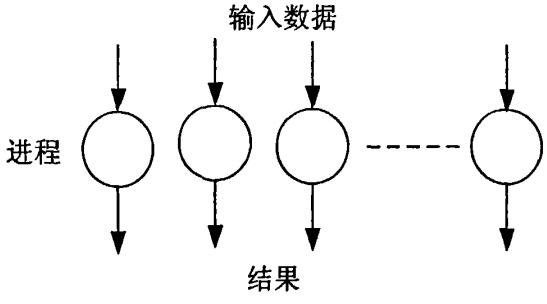

图5-1 分离的计算图

每个进程需要不同数据，并由其输入产生最终结果，而不需要使用其他进程生成的结果。如果在整个计算期间，所有可用处理器均被分配有进程。事实上有大量数据是属易并行的，或至少是接近易并行的。如上图所示，通常用于各个独立部分为相同运算，因此被称为 SPMD(单程序多数据)更为合适。由于数据不是共享的，因此使用分布式存储器多处理机或者消息传递多计算机更为合适。如果它们需要同一数据，则该将数据拷贝到每个进程中去。

近似易并行计算是那种需将计算结果分布和收集，以及用某种方式加以组合

的计算。这就意味着在开始和最后只有一个进程处于运行状态。如果要创建动态进程，通常的方法是采用主从结构。首先创建一个进程，由它再启动相同的从进程。图 5-2 示出了这种最终结构。

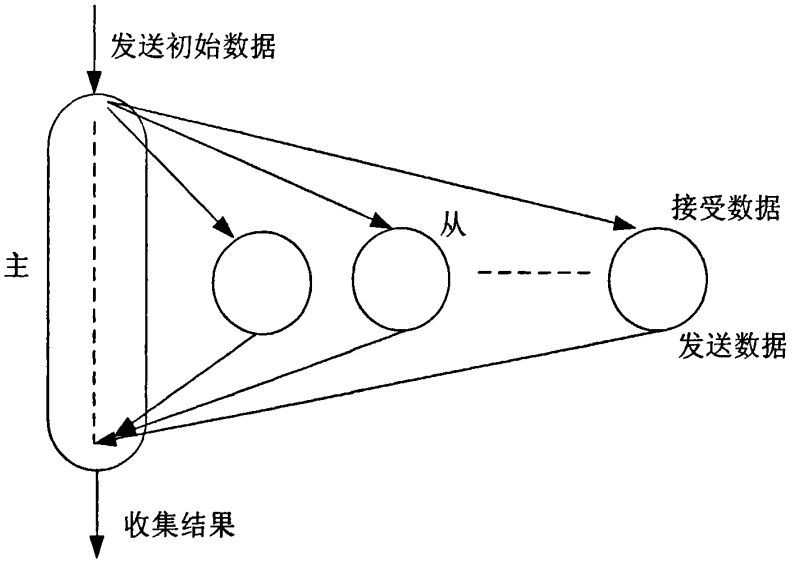

图5-2 主从进程实现易并行计算图

5.2 易并行计算举例(图像的几何变换)

将图像存于计算机中以便可以用某种方法对图像加以改变，被显示的图像常源于两种方法，一种是从摄像机那样的外部源得到，而且这种图像可能需要以某种方法被改变；显示的图像也可以人工创建。不管哪一种方法都可在已存储的图像上进行许多图形操作。例如可以移动图像，对图像缩放，旋转。还需要对图像进行处理操作，如平滑和边缘检测，特别对于源自外部的图像，因为它很可能是“有噪声的”。而这些通常是易并行的(因为每个象素是独立于其他象素的，因而这的确是一个易并行计算)。

通常输入数据是象素图，存于文件中或者复制到数组中。并行编程主要关心将象素图分解为一些象素组提供每个处理器加工，这是因为一般来讲象素数要远远大于处理器数/进程数，(H.264 的宏块基于以上概念)有两种一般的编组方法：以正方形/矩形区域编组和以列/行编组。我们可以简单地将一个进程(处理器)分配到

一个显示区域。如果对于相邻区域间不存在通信的情况，出了易于编程外，采用哪种方式都无关紧要。

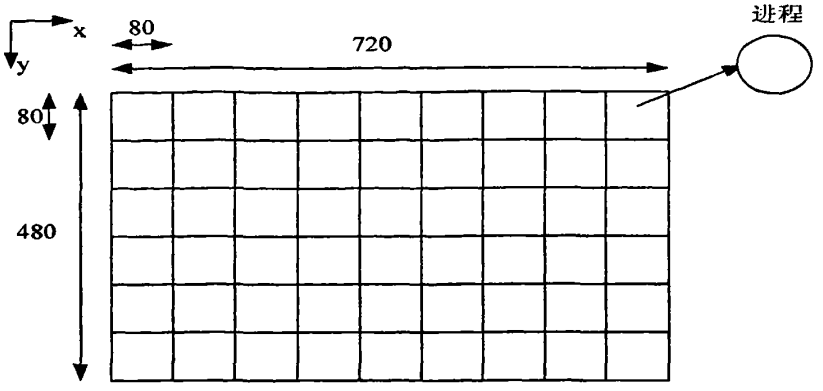

图5-3 每个进程的正方形区域

假设我们使用一个主进程和 54 个从进程(当然可以适当减少进程数)，由主进程返回新的坐标值以供显示。从进程进行相关地处理。这一形式为我们后面进行功能模块的映射提供了一个很好的基础。

5.3 针对片上网络的路由节点处理模块功能映射

5.3.1 模块处理设计分析

根据我们前面并行处理的一些原则(或者举例)，我们可以看到对于图像处理而言的确存在易并行处理的可能性。而片上网络取代总线结构的一个优点就是各处理模块(IP 核)的可并行性，接下来我们进行尝试功能映射。

考虑到视频处理本身需要大量的数据量通信，以及处理时间，因此对于我们片上网络的应用研究来说，我们的重点是如何进行功能模块的并行处理映射，当然在有了好的划分机制后可以适当地进行功能的扩展，因此在这里我们对视频处理中很有代表性的运动搜索，也就是帧间编码的重点进行功能映射的划分。

当然对功能模块进行并行划分可以最大限度地提高处理速度，以及利用网络资源，由前面一章中的介绍可以看到在运动估计时，运动补偿对一帧进行处理时对每一个宏块有几种模式的块划分估计，从这里来讲我们对于每一个宏块都可以进行同时进行处理，对于不同模式处理而言，它们之间也不需要太多的通信，因此我们根据易并行处理的原则可以认为它们是基本并行的，进行功能划分时可以

把它们作为不同的处理模块进行映射，在最初的划分时，鉴于菱形搜索是各个模式都需要的，我曾经把它单独作为一个模块来进行划分，后来经过小组讨论，如果将菱形模块作为单独的模块而言，当然可以从资源消耗上来取得一定的效果，但是势必引入通信中的热点，因此在最后的划分时，将菱形搜索模块包含在各个模式搜索模块之中。

对于各种模块进行处理时需要访问存储区来进行得到所需的数据，包括参考帧和处理帧，因此我们需要开辟存储区，作为共享存储设计。象素内插在这里我们分为一个独立的模块，7 种模式进行访问 Memory 时，自然进行内插，因此对于通信而言我们可以认为只是内插模块与存储模块之间的数据流动；对于模式选择模块而言，7 种模式都将与其有通信流量，最终的结果将影响到运动补偿模块，当然运动补偿是基于数据存储与访问的。基于上面的划分思想我们可以得到如下的功能映射框图。

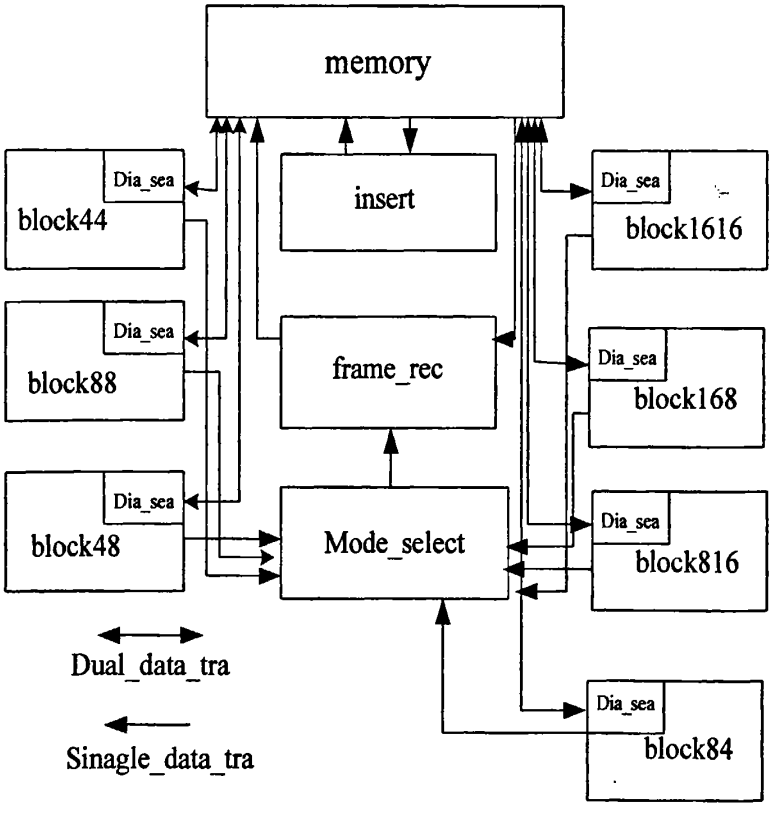

图5-4 针对片上网络的功能映射框图

5.3.2 模块实现

对于程序设计中，我们将涉及到的模块以及传输量说明如下：

block44：计算 4×4 块的运动向量，将运动向量连同相应的 SAD 值发往 mv84 和 mv48 模块。

block84(block48)：计算 8×4(4×8)块的运动向量，连同相应的 SAD 值发往 mv84(mv48)模块。

mv84(48)：根据 SAD 值确定 8×4、4×8 块最终的运动向量，并连同其 SAD 值发往 mv88_1 和 mv88_2 模块。

block88：计算 8×8 块的运动向量，将运动向量连同相应的 SAD 值发往 mv88_1 和 mv88_2 模块。

mv_88_1(_2)：完全等同的两个模块，根据 SAD 值确定 88 块最终的运动向量，并连同其 SAD 值发往 mv168(mv816)。

block168(816)：计算 16×8(8×16)块的运动向量，将运动向量连同相应的 SAD 值发往 mv168 和 mv816 模块。

mv168(mv816)：根据 SAD 值确定 16×8(8×16)块最终的运动向量，并连同其 SAD 值发往 mv1616。

block1616：计算 16×16 块的运动向量，将运动向量连同相应的 SAD 值发往 mv1616。

mv1616：根据 SAD 值确定最终的运动向量 mvx[44][36]和 mvy[44][36]，将最终结果发往 frame_rec。

mode_select：根据各搜索模块的 SAD 值进行宏块模式选择。

frame_rec：帧重构模块，根据最终的运动向量和当前处理帧重构出重构帧。

Dia_sea：进行菱形搜索。

对于菱形搜索我们采用了如下的处理流程，在实现中我们发现菱形搜索对于运动估计而言在占用的时间和代码量上都是重中之重，因此我们给予了足够的重视。具体运算如图 5-5 所示。

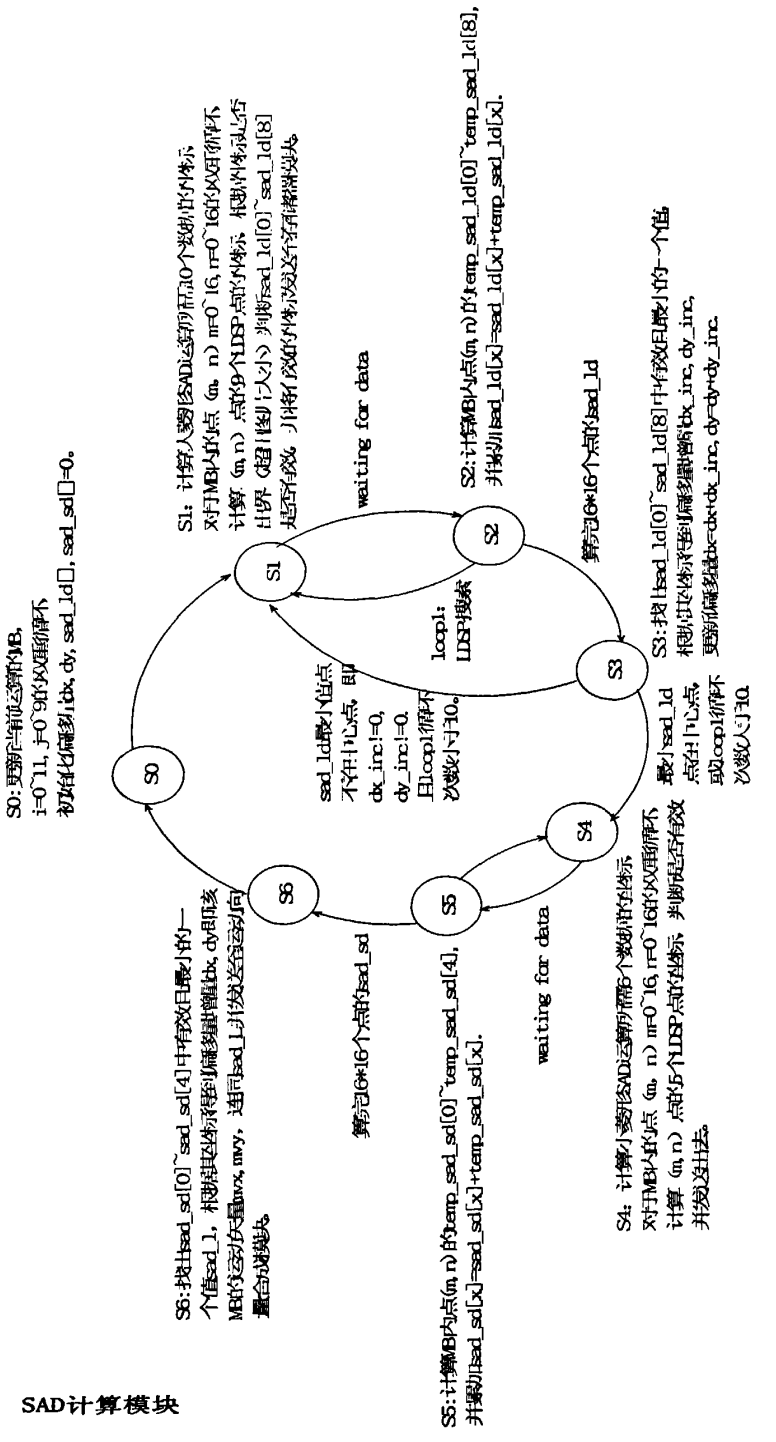

图5-5 菱形搜索处理流程

具体过程如下：

假设对 frame_prc 中的(i,j)宏块进行匹配。设初始的偏移量为(dx,dy)=(0,0)。

第一次 LDSP,计算 sad_ld[0]~sad_ld[8]:

$$sad\_ld[0]=\sum_{m=0}^{15}\sum_{n=0}^{15}|frame\_prc(i\times16+m,j\times16+n)-frame\_ref(i\times16+m+0+dx,j\times16+n+0+dy)|$$

$$sad\_ld[1]=\sum_{m=0}^{15}\sum_{n=0}^{15}|frame\_prc(i\times16+m,j\times16+n)-frame\_ref(i\times16+m+2+dx,j\times16+n+0+dy)|$$

$$sad\_ld[2]=\sum_{m=0}^{15}\sum_{n=0}^{15}|frame\_prc(i\times16+m,j\times16+n)-frame\_ref(i\times16+m+1+dx,j\times16+n+1+dy)|$$

$$sad\_ld[3]=\sum_{m=0}^{15}\sum_{n=0}^{15}|frame\_prc(i\times16+m,j\times16+n)-frame\_ref(i\times16+m+0+dx,j\times16+n+2+dy)|$$

$$sad\_ld[4]=\sum_{m=0}^{15}\sum_{n=0}^{15}|frame\_prc(i\times16+m,j\times16+n)-frame\_ref(i\times16+m-1+dx,j\times16+n+1+dy)|$$

$$sad\_ld[5]=\sum_{m=0}^{15}\sum_{n=0}^{15}|frame\_prc(i\times16+m,j\times16+n)-frame\_ref(i\times16+m-2+dx,j\times16+n+0+dy)|$$

$$sad\_ld[6]=\sum_{m=0}^{15}\sum_{n=0}^{15}|frame\_prc(i\times16+m,j\times16+n)-frame\_ref(i\times16+m-1+dx,j\times16+n-1+dy)|$$

$$sad\_ld[7]=\sum_{m=0}^{15}\sum_{n=0}^{15}|frame\_prc(i\times16+m,j\times16+n)-frame\_ref(i\times16+m+0+dx,j\times16+n-2+dy)|$$

$$sad\_ld[8]=\sum_{m=0}^{15}\sum_{n=0}^{15}|frame\_prc(i\times16+m,j\times16+n)-frame\_ref(i\times16+m+1+dx,j\times16+n-1+dy)|$$

假设最小的 sad 为 sad_ld[4],则偏移量更新为(dx,dy)=(-1,1)+(0,0),再重复上面过程，直到最小的 sad 为 sad_ld[0]。在这个过程中，偏移量(dx,dy)会不断的累加。

接着进行 SDSP，SDSP 只进行一次，即计算 sad_sd[0]~sad_sd[4]:

$$sad\_sd[0]=\sum_{m=0}^{15}\sum_{n=0}^{15}|frame\_prc(i\times16+m,j\times16+n)-frame\_ref(i\times16+m+0+dx,j\times16+n+0+dy)|$$

$$sad\_sd[1]=\sum_{m=0}^{15}\sum_{n=0}^{15}|frame\_prc(i\times16+m,j\times16+n)-frame\_ref(i\times16+m+1+dx,j\times16+n+0+dy)|$$

$$sad\_sd[2]=\sum_{m=0}^{15}\sum_{n=0}^{15}|frame\_prc(i\times16+m,j\times16+n)-frame\_ref(i\times16+m+0+dx,j\times16+n+1+dy)|$$

$$sad\_sd[3]=\sum_{m=0}^{15}\sum_{n=0}^{15}|frame\_prc(i\times16+m,j\times16+n)-frame\_ref(i\times16+m-1+dx,j\times16+n+0+dy)|$$

$$sad\_sd[4]=\sum_{m=0}^{15}\sum_{n=0}^{15}|frame\_prc(i\times16+m,j\times16+n)-frame\_ref(i\times16+m+0+dx,j\times16+n-1+dy)|$$

其中的 dx,dy 为进行完 LDSP 后得到的偏移量。

假设最小值为 sad_sd[3],则最终该宏块的运动矢量为(-1,0)+(dx,dy)。而此 sad 值则被保存用作最终运动向量确定。

对于其它大小的块如 16*8、8*16、8*8、8*4、4*8、4*4，也是用同样的方式计算此块的运动矢量，只是在Σ求和的时候的范围要改变一下。

对于运动矢量的最终确定考虑到 QCIF 图像，我们最小可以划分为 44*36 个 4*4 像素的块，因此我们可以得到 mvx[44][36]和 mvy[44][36]个运动向量。最终根据这两组 44*36 的运动矢量重构出我们的重构帧 frame_rec。

对于 16*16 的宏块菱形搜索，我们可以得到 mvx16_16[11][9]和 mvy16_16[11][9]个运动向量；

对于 16*8 的块菱形搜索，我们可以得到 mvx16_8[11][18]和 mvy16_8[11][18]个运动向量；

对于 8*16 的块菱形搜索，我们可以得到 mvx8_16[22][9]和 mvy8_16[22][9]个运动向量；

对于 8*8 的块菱形搜索，我们可以得到 mvx8_8[22][18]和 mvy8_8[22][18]个运动向量；

对于 8*4 的块菱形搜索，我们可以得到 mvx8_4[22][36]和 mvy8_4[22][36]个运动向量；

对于 4*8 的块菱形搜索，我们可以得到 mvx4_8[44][18]和 mvy4_8[44][18]个运

动向量；

对于 4*4 的块菱形搜索，我们可以得到 mvx4_4[44][36]和 mvy4_4[44][36]个运动向量。

对于 8*4 的块，可以分为两个 4*4 的块，若 8*4 块的小菱形搜索所得的最小 SAD 值大于两个 4*4 块的小菱形搜索所得的最小 SAD 值之和，即：

SAD8_4>SAD4_4[0]+SAD4_4[1]

则我们采用 4*4 块的运动向量。4*8 的块类似。对于 8*8 的块，可以分为两个 8*4 块或者两个 4*8 的块，同样的，我们采用 SAD 值最小的一种分法的运动向量作为最终的运动向量。

以此类推，得到 mvx[44][36]和 mvy[44][36]。

frame_rec 模块核心算法：

QCIF 图片被分为了 11*9 个 16*16 的宏块，每一个宏块最终又被细分为了 4*4 个 4*4 的像素块。

重构的过程如下：

```
for(i=0;i<11;i++)
 {for(j=0;j<9;j++) //11*9 个宏块的循环
   {for(m=0;m<4;m++)
     {for(n=0;n<4;n++) //宏块内部 4*4 个 4*4 块的循环
       {
          dx=mv_x[i*4+m][j*4+n];
          dy=mvy_[i*4+m][j*4+n];  //取出对应位置的运动向量
          for(x=0;x<4;x++)
          {for(y=0;y<4;y++)//4*4 个像素的循环
{frame_rec[i*16+m*4+x][j*16+n*4+y]=frame_prc[(i*16+m*4+x+dx)][(j*16+n*4+y+dy)];
           }
       }
     }
   }
}
```

图5-6 重构模块核心算法实现

5.4 片上网络映射优化

首先我们对划分的模块进行了数据流量的测算，这里我们采用 bit 作为测量的基准，当然我们可以知道存储模块部分将是负载最重的模块，因为几乎所有的模块都将访问存储模块，而运动补偿模块有最少的流量，因为我们可以视为其只在最后进行一次整体的补偿，而不象其他搜索模块那样进行多次的搜索。下表为整个网络的各处理模块的通信流量：

表5-1 网络处理模块的通信流量

处理模块	流量值(bit)
Count_16_16	4208529743
Count_16_8	4119956279
Count_8_16	4119956279
Count_8_8	2904696671
Count_8_4	3660119151
Count_4_8	3660119151
Count_4_4	909218703
Count_insert	1512945094
Count_mode_select	2546214575
Count_frame_rec	104546

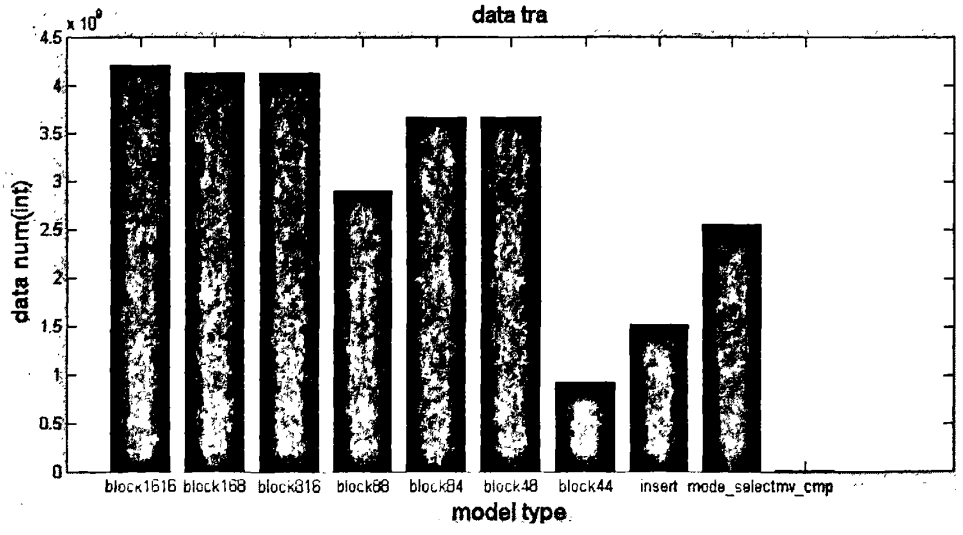

图5-7 网络处理模块承重图

由承重图可以看到对于 mode_select 和 frame_rec 模块而言，我们发现其具有相对比较少的数据流量，因此在进行片上网络优化时进行了相应处理，本身

mode_select 模块可以看作是 frame_rec 模块的先决模块，所以我们在映射时进行了合并，合并后的模块为 mv_cmp 模块，而 frame_rec 模块的主要功能是进行帧重构。

对于存储模块 MEM 而言，在进行流量计算时我们发现它具有最大的流量，因为几个搜索模块 block44,block84 等都需要与它进行通信，显然在处理时会成为通信的瓶颈，一个解决方式是我们可以采用分布存储的设计方式，开辟几个存储区，这样可以在一定程度上分担通信流量，但是考虑到网络节点所带的 IP 处理核，我们希望是具有实际处理功能的模块，而不仅仅是数据访问这种简单的存储区设计，因此这里我们出于用资源换时间的考虑，在每一个 block 中都加入了小的内存，我们同样也可以把 insert 模块嵌入。进行网络映射后如下图所示：

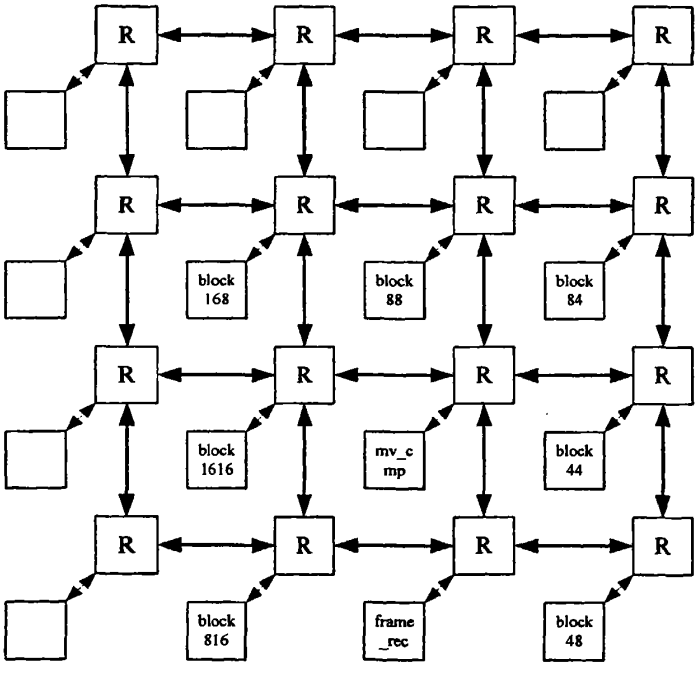

图5-8 片上网络功能映射

在进行网络映射的时候因为已经把模块缩减到了 9 个因此在映射的位置上是比较灵活的，对于现在我们做的这种映射由于节点之间有大量数据交换的话往往是比较近的节点之间，这也是我们进行映射的初衷，即尽量让大量的数据交流发生在相邻节点之间，对于路由算法而言反而是用黑总线路由可能具有比较高的效率，但是考虑到功能与网络都进行扩展的话，因此我们仍然沿用我们的 XY 路由模式。在接下来的网络数据分析时，我们采用简化的数据流量分析图 5-9。

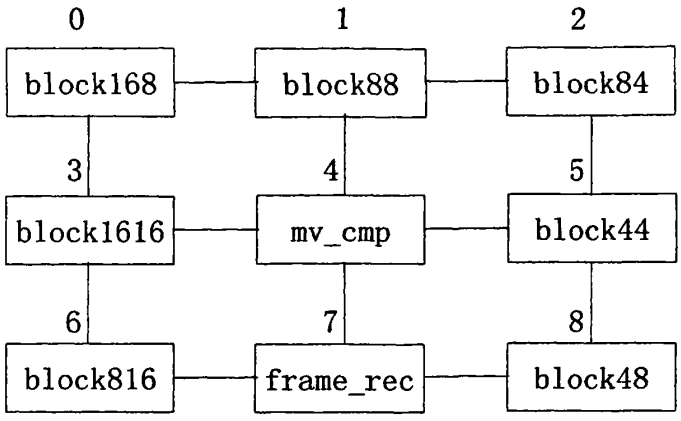

图5-9 数据包流量分析简图

由于在进行数据流量分析时，我们对于每个模块都采用了内嵌存储区的设计，因此我们传输的将是各模块的运动矢量估计，具体流量分析如下：

节点 3(1616 模块)发送 11*9 个包(每个包封装一个 1616 分块的 mvx，mvy 和 sad)到节点 1(运动向量比较模块)；包格式如图 5-10 所示。

节点 0(168 模块)发送 11*18 个包到节点 1；

节点 6 发送 22*9 个包到节点 1；

节点 4 发送 22*18 个包到节点 1；

节点 2 发送 22*36 个包到节点 1；

节点 8 发送 44*18 个包到节点 1；

节点 5 发送 44*36 个包到节点 1；

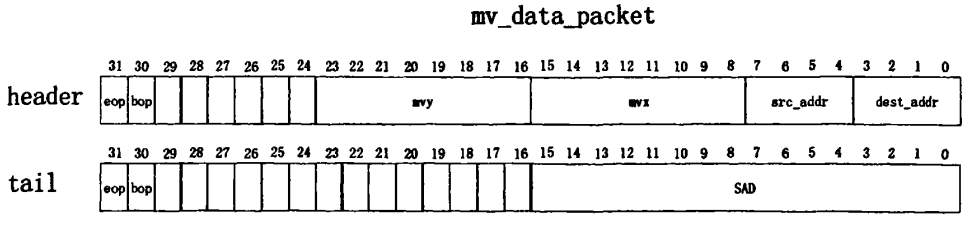

图5-10 mv_data_packet 格式

节点 1 在收完所有分块(1616，168，816，88，84，48，44)的运动矢量之后，算出最终的运动矢量(44*36)之后，发送到节点 7(帧重构模块)，包格式：

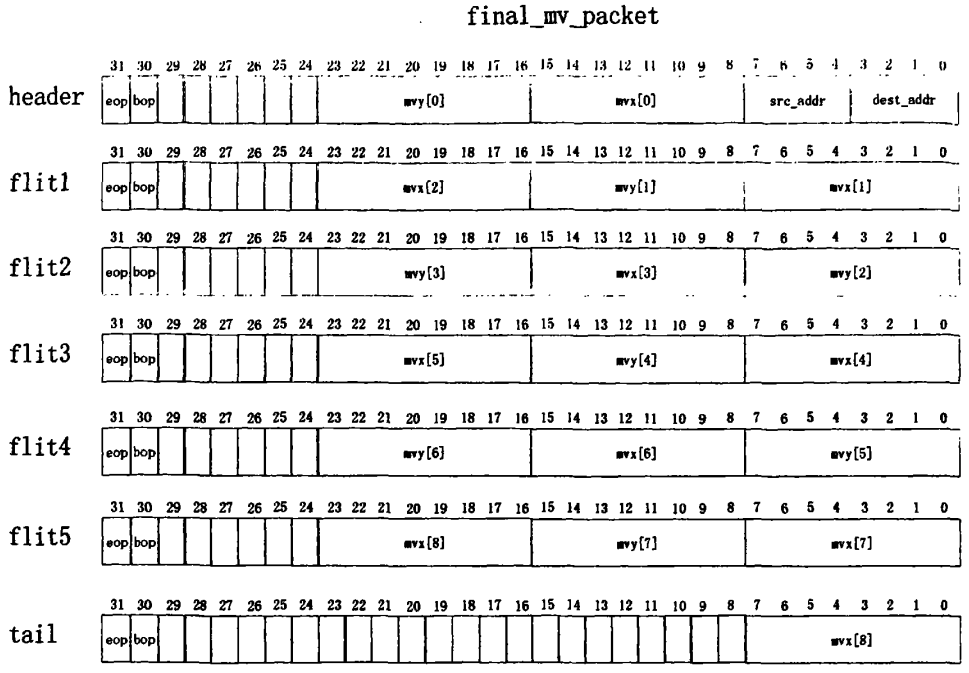

图5-11 final_mv_packet 格式

经计算我们得到数据流量处理如下：

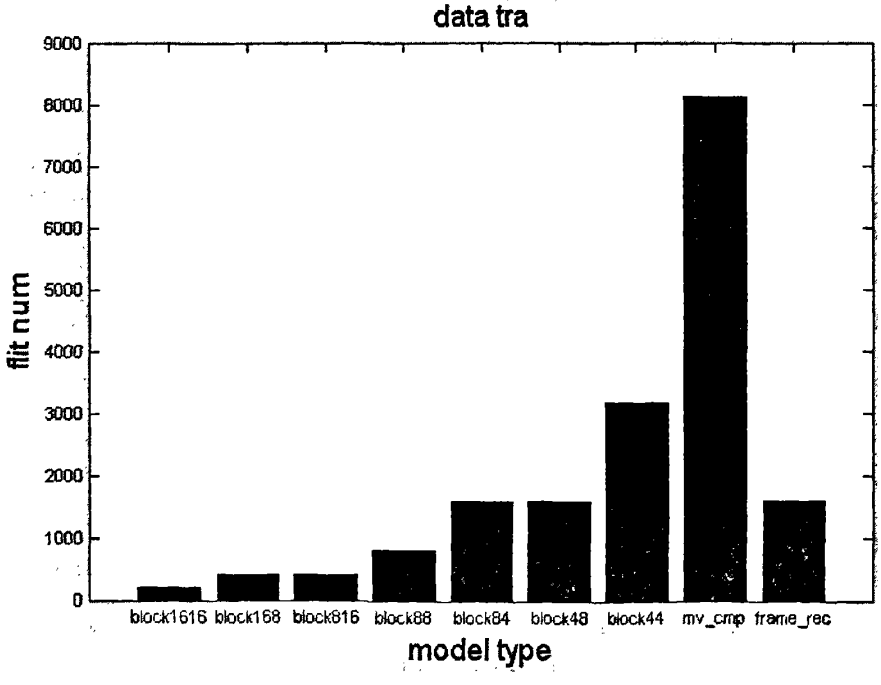

图5-12 片上网络映射后的数据流图

可以看出在进行重新划分，以及内存嵌入后，数据流量有了明显改善，同时在合并模块后，各个模块之间的通信流量更加均衡。(当然存在热点 mv_cmp，我们将其映射到中间位置，来减少路由寻址时的时间消耗)经过我们进行功能模块的划分后处理时间上比单纯在硬件上速度会有提高(因为单纯的硬件处理时要顺序进行，而我们映射后会存在近似并行处理，自然有一定的提高，具体的时间优势由于我们正在进行 verilog 的移植，将在后续的研究中深入)；对于热点的通信情况，出于优化的考虑，针对各个方向的 FIFO 深度应该有动态的配置，这些在我们小组另一同学的课题研究中有所涉及。

5.5 本章小结

本章我们对处理模块的并行映射进行了初步的探讨，并将视频处理中的帧间编码关键技术运动估计与补偿嵌入到我们设计的片上网络仿真平台中，采用存储区内置的设计方法，减小了节点与节点之间的通信量。接下来我们希望对整个处理流程进行功能映射，这里存在存储的问题，本身图像处理尤其是视频处理需要很大的数据流量，因此需要我们在接下来的研究中进一步优化我们的设计，处理模块的实现效率也需要优化，力求用更少的资源，更快的速度，更好地完成处理功能。相信我们在接下来的研究中将有所突破。

第六章 结论与展望

6.1 结论

本论文对片上网络的路由方面的技术以及片上网络的应用方面进行了较深入的讨论，主要内容如下：

介绍了片上网络技术的研究背景和当前发展状况；

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

深入分析了 Mesh 结构，构建了 4×4 的 Mesh 网络结构，进行网络仿真，重点研究了 XY 维序路由以及 OE 路由模式，这样不仅可以防止死锁、活锁，而且简单易实现。同时，论文中还给出了该算法下，Mesh 网络延时和资源消耗方面的仿真结果；

对节点的仿真验证、测试和得到的结论进行了分析以及网络延时进行了分析。

分析了视频处理标准 H.264，同时将运动搜索模块进行了并行分解以及应用到我们设计的片上网络仿真平台中，进行处理模块的映射处理，初步讨论了片上网络的并行应用。

6.2 展望

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

最后，片上网络的最终目的是走向应用，IP 核的设计、动态配置网络节点的功能以及片上网络功耗的评估等等，在未来我们将重点关注。随着对其研究的不断深入，NOC 技术一定会得到越来越快速的发展。

致谢

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

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

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

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

最后，衷心感谢为评阅本论文而付出辛勤劳动的各位专家和学者。

蒋勇男

2008 年 4 月于电子科技大

参考文献

[1] Axel Jantsch, Hannu Tenhunen. "Network on Chip". Kluwer [M] Academic Publishers .2003, pp6.

[2] Axel Jantsch, Johnny Qberg, Hannu Tenhune. "Special issue on networks on chip". Journal of Systems Architecture, February 2004, pp.61-63.

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

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

[5] P.Guerrier, A.Greiner. "A Generic Architecture for On-Chip Packet-Switched Interconnections". Proceedings of Design Automation and Test in Europe, March 2000, pp. 250-255.

[6] Duato, J.; Robles, A.; Silla, F.; Beivide, R, "A comparison of router architectures for virtual cut-through and wormhole switching in a NOW environment," Parallel and Distributed Processing, 1999. 13th International and 10th Symposium on Parallel and Distributed Processing, 1999. 1999 IPPS/SPDP. Proceedings, 12-16 April 1999 Page(s):240-247

[7] Franck Binard. "Deadlock Preventive Adaptive Wormhole Routing on k-ary n-cube ".Interconnection Networks. University of Ottawa, December 7, 2003.

[8] Alexberson george Anderson. "Sybase and Client/Server Computing". Osborne Mcgraw-hill, 1995.

[9] J. Wu. "A deterministic fault-tolerant and deadlock-free routing protocol in 2D Meshes based on odd-even turn model". Proceedings of the 16th international conference on Supercomputing, 2002, pp. 67-76.

[10] J. Duato, S. Yalamanchili, L. Ni. "Interconnection Networks, an Engineering Approach", IEEE Computer Society Press, 1997.6-45.

[11] Ge-ming chiu, "The odd_even turn model for adaptive routing",IEEE Computer Society.2000.2-6.

[12] 刘榴娣，刘明奇。实用数字图像处理。北京：北京理工大学出版社。2006，70-89

[13] 沈兰荪，卓力。视频编码与低速率视频传输。北京：电子工业出版社。2001，20-35

[14] Gray S.Greenbaum. "Remarks on the H.26L Porject:Streaming Video Requirements for Next Generation Video Compression Standards"[R], ITU-T video coding experts group meeting ,Monterey,Feb.1999, doc Q15-G-11

[15] 李宾，高平。H.264 编码系统的特点及应用前景[J]。数字电视与数字视频，2003，No.6

[16] 王嵩，薛全，张颖，陈建乐。H.264 视频编码新标准及性能分析[J]。数字电视与数字视频，2003，No.6

[17] G. J. Sullivan. "Efficient Scalar Quantization of Exponential and Laplacian Random Variables". IEEE Transaction on Information Theory. 1996, 42(5):11

[18] H. S. Malvar, A. Hallapuro, M. Karczewicz, and L. Kerofsky. "Low- Complexity Transform and Quantization in H.264/AVC". IEEE Transaction on Circuit and System for Video Technology. 2003, 13(7):255~257

[19] Iain E.G. Richardson. "H.264/MPEG-4 Part10 White Paper".Transform and quantization.March 2003

[20] C.E. SHANNON. "A Mathematical Theory of Communication" [M].The Bell System Technical Journal. July,October,1948. Vol.27,pp.379-423,623-656

[21] T.Koga,"Motion-compensated inter-frame coding for video conferencing"[A],in Proc.NTC81,New Orleans,LA,Nov. 1981,pp.C9.6.1-9.6.5

[22] 陆鑫达,并行程序设计(Parallel Programming)。机械工业出版社，2002，67-78

个人简历

出生年月：1982 年 10 月；

2001 年 9 月—2005 年 7 月： 电子科技大学机电学院，获得学士学位；

2002 年 9 月—2005 年 7 月： 电子科技大学通信学院，获二学位；

2005 年 9 月—2008 年 6 月： 电子科技大学通信学院，攻读硕士；

攻读硕士学位期间的研究成果

参加的科研项目：

(1) 国家自然科学基金项目——复杂 SOCs 片上通信关键技术研究。

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

发表、录用的论文与申请专利：

蒋勇男，李玉柏，MESH 结构 OE 路由节点设计,2007 年中国通信年会