2010

A study of on-chip FPGA system with 2D mesh network

Ka-ming Keung
Iowa State University

A study of on-chip FPGA system with 2D mesh network

by

Ka-Ming Keung

A dissertation submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Major: Computer Engineering

Program of Study Committee:
Akhilesh Tyagi, Major Professor
Soma Chaudhuri
Chris (Chong-Nuen) Chu
Arun K Somani
Zhao Zhang

Iowa State University

Ames, Iowa

2010

DEDICATION

For those who give me academical, mental and financial supports for my Phd study.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGEMENTS

I would like to take this opportunity to thank my adviser Dr. Akhilesh Tyagi with deep gratitude. This thesis could not be completed without his patience, encouragement. His insights have always inspired me and renewed my hopes for completing my graduate education. I would also like to thank my committee members for their efforts and contributions to this work. I would additionally like to thank Professor Mahadevan Gomathisankaran, Veerendra Allada and Swamy Ponpandi. The valuable discussion with them make my thesis get into better shape.

ABSTRACT

The advance in fabrication technology hugely increases the number of available transistors on a single chip. It allows the industry to build the entire system on a single chip which was only realizable on a board in the past. On-chip System not only reduces the computer physical size, but also increases the computation performance because modules/cores/intellectual properties (IPs) are packed closely together. When simply increasing the clock frequency to increase the computer performance becomes harder because of the wire delay, putting more computation units on a single chip becomes a good alternative for improving computer performance. Building more cores on a chip in the future is expected.

With many IPs on a chip, traditional bus is no longer able to provide enough bandwidth to support the communication between IPs. Providing a high performance on-chip network infrastructure for the IP communication becomes a key to high performance on-chip computation. This thesis focuses on an on-chip network supporting on-chip system.

This thesis is composed of two main parts. In the first part, a high performance deadlock free dual-coded on-chip router using adaptive multicast routing is built. Compared with the traditional deterministic XY unicast router, this router can reduce both packet latency and energy consumption.

In the second part, a co-processor placement algorithm for an on-chip system built from FPGAs with an on-chip network is proposed. The algorithm aims to place the communicating modules as close as possible. In addition, an algorithm for sharing a FPGA by multiple co-processors and an algorithm for supporting polymorphic co-processor are proposed to increase on-chip FPGA system throughput.

CHAPTER 1. INTRODUCTION

Fabrication technology has been consistently improving. The transistor feature size has reduced from 180nm in 2000 to 32 nm in 2009. International Technology Roadmap for Semiconductors (ITRS) [16] predicts that the feature size will be further reduced to 16 nm by 2018, INTEL Corp. predicts that 16 nm will arrive as early as 2013 and NVIDIA Corp. predicts that 11 nm will arrive in 2015. The reduction in transistor size has both pros and cons. The main pro is the increased number of transistors that can be put on a single chip. The main con is the increased wire delay. Increased wire delay prohibits the chip designer from gaining computer performance by simply increasing the clock frequency. How to effectively use these ample number of on-chip transistors in the upcoming era becomes an interesting question. The simplest answer is to put the entire system on a chip to reduce the communication latency between computer's modules(or IP). In the past, the industry had always added new components onto a chip whenever the feature size decreased. At the beginning, cache was added. From then, floating point computation units, memory controllers, network controllers, peripheral controllers and graphic computation units were added. To further increase computer performance, the industry packed as many central processing units (cores) as possible onto a single chip recently. With the proper uses of parallel computing algorithms, these cores can collaborate efficiently. Hence, users would be able to enjoy a massive performance gain. Some existing on-chip parallel computing system prototypes include INTEL 80-core teraflop research chip [1], Wavescalar [38] and TRIPS processor [17].

Communication between these on-chip IPs is provided by a bus. As the number of on-chip IPs increases, a traditional bus would no longer be able to provide sufficient bandwidth for the IP communication. Communication delay due to bus congestion increases IP idle time and

limits system performance. Providing a high performance on-chip network infrastructure for the IP communication becomes a key to high performance on-chip computation.

1.1 Motivation for an advanced router

Router is the main component of an on-chip network. It routes a packet from a source to a destination. The simplest router supports XY-routing and unicast routing. An advanced router can reduce packet latency and energy consumption. There are many advantages in utilizing a deadlock free and adaptive multicast on-chip router which support multicast binary code, multicast unary and on-the-fly code transformations. These advantages will be discussed in detail in this section.

1.1.1 A need for an adaptive router

An adaptive router can increase system performance.

In a 2D Mesh on-chip network, the most basic on-chip router uses deterministic XY-routing. On the one hand, XY-routing requires low hardware resources, but on the other hand, the router suffers from congestion problem. In XY-routing, a packet is first routed to the X direction until it reaches the column of the destination and then route to the Y direction until it reaches the destination. When two packets are sent to a same link, network congestion occurs. It is because packet routes are fixed. A packet could not take an alternative route to traverse to its destination even if there are some less congested links leading to the destination.

Fig. 1.1(a) illustrates an XY-routing congestion problem. In this example, *IP3* sends a packet to *IP2* and *IP4* sends a packet to *IP5*. Using XY-routing, the link between *IP4* and *IP5* is congested. As a result, the packet latencies of both packets increase. In fact, there exists an alternative route for the packet from *IP3* to *IP2* to traverse to its destination without any network congestion. If adaptive routing is in use as shown in Fig. 1.1(b), the congestion problem could be avoided and packet latency would be reduced.

In this thesis, a *Path-Based* adaptive routing method is proposed. It has a better performance than both of the existing *DyXY* adaptive routing method [29] and *Regional Congestion*

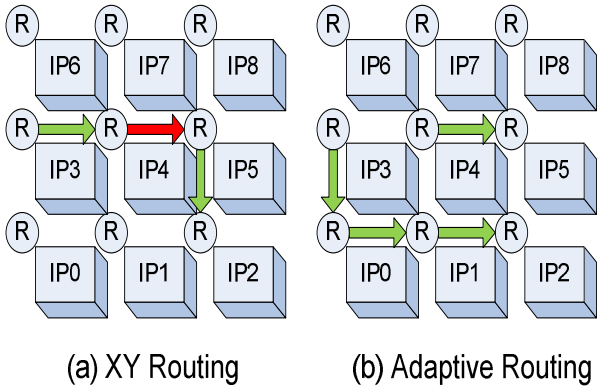


Figure 1.1 XY-Routing Congestion Problem

Aware adaptive routing method [20].

1.1.2 A need for a multicast router

Lack of multicast support enormously increases packet latency and energy consumption.

In on-chip parallel computing system, there are a lot of one-to-many packets. For example, Chip-Level-Multiprocessor (CMP) requires high speed multicast memory coherence packet transmission for memory synchronization [23, 11, 37] and multicast operand packets transmission for computation (RAW [40], TRIPS [36], WaveScalar [39]). In a co-processing computer system, microprocessor forks out data from its cache to multiple IPs for parallel acceleration. In an on-chip system with many FPGA tiles, a single configuration bitstream has to be multicast from an I/O tile to many FPGAs when multiple FPGA request a same configuration bitstream file for runtime reconfiguration [2]. Native multicast support decreases packet latency. In CMP, it reduces microprocessor stalling time due to memory synchronization. In co-processing system, it reduces acceleration startup time. In FPGA-built on-chip system, it reduces reconfiguration time.

Fig. 1.2 compares the number of buffer writes between networks with and without multicast support. In this example, IP at location (0,0) multicasts a packet to five IPs at location (4,0), (4,1), (4,2), (4,3) and (4,4) respectively. In a unicast system, the network interface has to

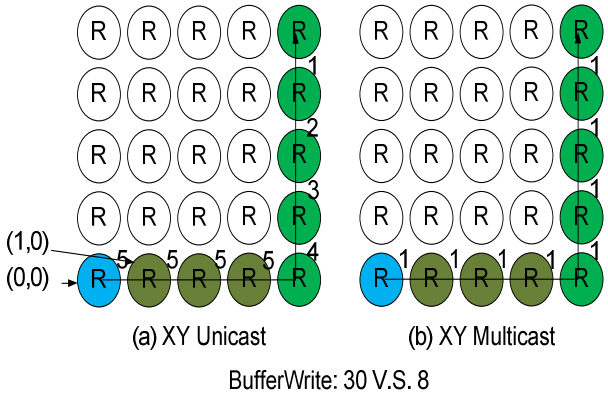


Figure 1.2 Unicast and Multicast Buffer Write Comparison

duplicate the multicast packet five times and send out each unicast packet one by one. The number of buffer writes in a network with native multicast support is 8 while the number of buffer writes in a network without native multicast support is 30. With native multicast support, the network itself would be less congested, packet latency would be lower and energy consumed by the network would be lesser.

1.1.3 A need for an adaptive multicast router

When the multicast function and the adaptive routing are combined, multicast packet divergence points can be chosen dynamically based on destinations and congestion status to further reduce the number of buffer writes and network load. Fig. 1.3 compares the number of buffer writes between networks with and without an adaptive multicast function. In this example, IP at location (0,0) multicasts a packet to four IPs at location (1,4), (2,4), (3,4) and (4,4). In the XY multicast tree (Fig. 1.2 (a)), the packet can only diverge at locations (1,0), (2,0) and (3,0). In the adaptive multicast tree (Fig. 1.2 (b)), the packet can diverge at locations (1,4), (2,4) and (3,4). The number of buffer writes in a network with adaptive multicast support is 8 while the number of buffer writes in a network without adaptive multicast support is 20.

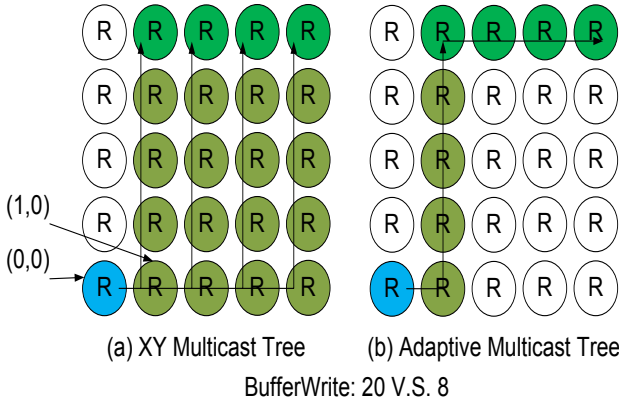


Figure 1.3 XY and Adaptive Multicast Buffer Write Comparison

1.1.4 A need for a multicast router supporting unary code, binary code and code transformation

Multicast destination addresses are commonly coded in two ways, one-hot unary code or binary code. Both coding methods offer advantages at a certain point in the design space. Unary code has fewer number of address flits when the number of destinations is high. Binary code has fewer number of address flits when the number of destinations is low. Given a unary coded multicast packet with many destinations, the number of addresses decreases every time the packet diverges. When the number of addresses decreases to a certain level, binary code becomes a better coding method instead of unary code. A router accepting two codes with transformation from unary code to binary code can minimize the number of address flits of multicast packet. Hence, the packet latency could be reduced.

Beside the transformation from unary multicast code to binary multicast code, a transformation from multicast code to unicast code is another function which can reduce the packet latency. Multicast virtual channel requires a lot of chip space. In order to minimize the physical size of a router, each router port has only one multicast virtual channel and several unicast virtual channels. Therefore, multicast virtual channel is considered to be a rare property in a router. To lessen the competition for the multicast virtual channel, a router can change the packet coding from multicast to unicast when the number of addresses of a multicast packet

drops to one.

1.1.5 A need for a multicast deadlock free router

Multicast deadlock halts the packet flow and makes the system unfunctionable. It is crucial to keep the network deadlock free. There were several attempts to avoid multicast deadlock problem. However, each attempt has its own drawback. Lin *et al.* [30] propose sending separate multicast packet copies to four separate quadrants, but the number of packet copies would be hugely increased. They also propose utilizing route pre-computation by Hamiltonian path partitioning, but then the routing would not be adaptive. Chien *et al.* propose Planar network [9] which uses two sub-networks ($X+$ and $X-$) to route the packet. Then again, some extra routers and links would be needed for the extra sub-networks. In this thesis, we try a new approach by splitting the FIFO into a data FIFO and an address FIFO. When a deadlock is going to occur, the router breaks the packet into two pieces to prevent the deadlock from occurring. Compared with Lins methods, the number of duplicate flits is fewer. Compared with Planar network, the physical size of the router is smaller.

1.2 Motivation for an on-chip system built by many FPGAs with the on-chip network

A system-on-chip realized with an FPGA is common today. Modules in an on-chip system can be reconfigured to meet the dynamically changing system demands. For example, security policy could be renewed in the FPGA modules even after the initial system implementation. Digital signal processing algorithm assigned to an FPGA can also be renewed to fit the changing input data characteristics.

Currently, soft bus is the default choice for an on-chip system modules communication. As the number of on-chip system modules grows, the bandwidth of the bus could not meet the modules' high communication demands. An on-chip system with many modules needs a native 2D mesh network to avoid communication infrastructure becoming the system's bottleneck.

Fig. 1.4 shows a layout of FPGAs with a native 2D mesh network. The network on the

bottom left corner controls the FPGA reconfiguration. Each FPGA tile could be configured as a module to support the system.

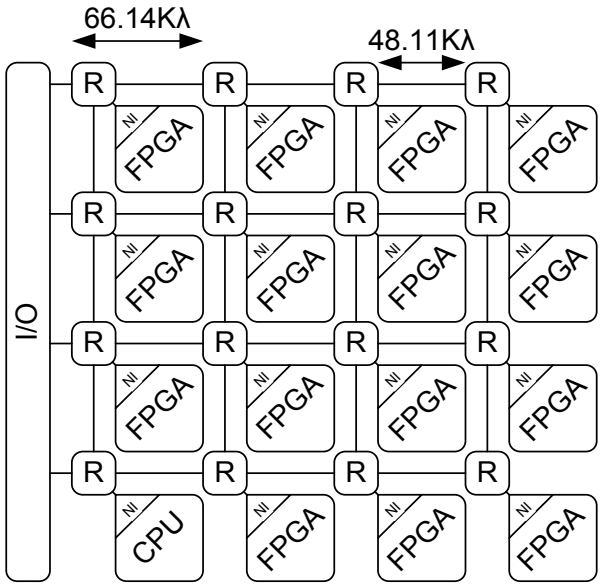


Figure 1.4 On-chip system built by FPGA with 2D Mesh network

1.2.1 A need to place system modules as close as possible

When system modules are closely placed, network's energy consumption could be reduced and system throughput could be increased. In a native 2D mesh network supporting FPGA on-chip system, modules location affects system performance. Fig. 1.5 compares two modules placements. Suppose a co-processor in this system requires two modules, *M1* and *M2*. After leaving *I/O*, the data goes through *M1* to *M2*, and then goes back to *I/O*. In Fig. 1.5 (a), *M1* is placed on location (1,1) and *M2* is placed on location (1,2). The number of hops hops traversed by each datum is three. In Fig. 1.5 (b), *M1* is placed on location (1,1) and *M2* is placed on location (2,2). The number of hops traversed by each datum is five. Placing modules closer to each other can reduce communication's energy consumption. It can also improve system performance because each packet suffers less congestion.

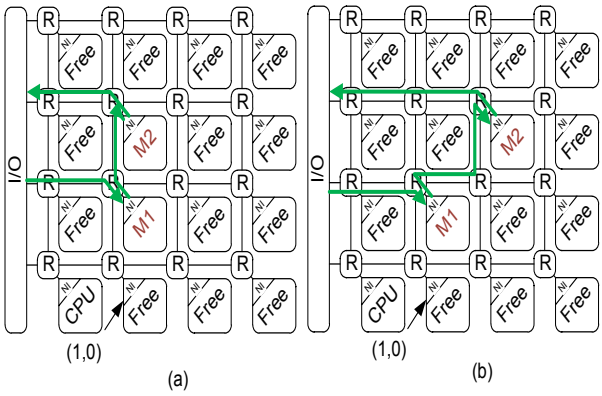


Figure 1.5 Placements Comparison

1.2.2 A need for sharing system modules among co-processors

Module sharing among co-processors can increase system throughput. In an on-chip system, one co-processor is supported by multiple modules. Within each co-processor, the module with the highest processing time is the bottleneck module of the co-processor. It is a key factor in determining the co-processor's throughput. This bottleneck module creates idle time on other modules within the same co-processor. The bottleneck module's ancestors have to wait for the bottleneck module to accept the newly generated data while the descendants have to wait for the data generated by the bottleneck module.

Fig. 1.6 shows an example of module sharing. In this example, two co-processors were mapped on the chip. Module *lib1*'s processing time (PT) is ten cycles. Module *lib2*'s processing time is two cycles. Module *lib3*'s processing time is twelve cycles. Without modules sharing, six FPGA tiles are needed to support two co-processors. Suppose module *lib2* is shared by the two co-processors. Only five FPGA tiles are needed instead of six. Note that the processing time of module *lib2* now becomes four cycles. However, the bottlenecks of both co-processor stay at twelve cycles. Therefore, their throughput will not be significantly affected. Using lower number of FPGA tiles to support the two co-processors allows more free FPGA tiles to support others co-processors. As a result, the whole system throughput increases. In this thesis, we are going to develop a bottleneck aware module sharing algorithm to increase the

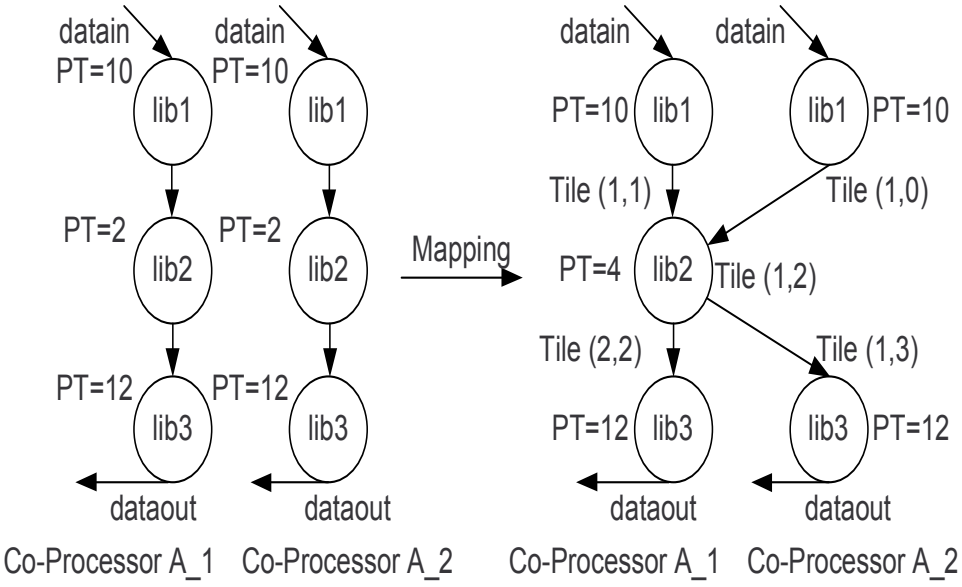


Figure 1.6 Module Sharing (a) Two Task Graphs, (b) Two Mapped Task Graphs

on-chip system throughput.

1.2.3 A need for an on-chip system with polymorphic modules

FPGA can be configured as a specific module, such as Discrete Concise Transform (DCT) module or Advanced Encryption Standard (AES) module, to speed up a specific function. It can also be configured as a soft microprocessor, *e.g.* Xilinx Microblaze, to handle a broad range of functions. The maturity of C-to-HDL technology allows code written in C to be translated into binary image for the soft microprocessor and FPGA bitstream for FPGA fabric. Therefore, the future on-chip system can choose a module running in either software mode or hardware mode to support a co-processor. Modules running in software mode usually provide lower throughput and consume more energy compared with the modules running in hardware mode. However, when the co-processor's expected runtime is short and there exists a configured microprocessor on the chip, running the modules in software mode allows the co-processor to start sooner without waiting for the tile reconfiguration. The module is ready to support the co-processor once the module instructions arrive at the soft processor L1 cache (configured by

the block RAM). This characteristic allows the co-processor to complete in a short time. In the last part of this thesis, we will extend our placement algorithm to support the polymorphic modules selection.

1.3 Thesis Organization

This thesis is organized as follows:

In Chapter 2, existing related works are reviewed.

In Chapter 3, an adaptive router using a new path-based adaptive routing algorithm is introduced.

In Chapter 4, a multicast adaptive router and a novel way to break the multicast deadlock are presented.

In Chapter 5, a multicast router supporting both unary multicast code and binary multicast code is introduced. This router also supports transformation from unary code to binary code and from multicast code to unicast code.

In Chapter 6, experimental results about multicast router and information about hardware implementation are presented.

In Chapter 7, a protocol for an on-chip FPGA system with a 2D Mesh network is presented.

In Chapter 8, a run-time placer which favors placing the co-processor modules as close as possible is introduced.

In Chapter 9, an algorithm supporting CLB modules sharing is presented.

In Chapter 10, an algorithm supporting polymorphic modules placement is discussed.

CHAPTER 2. REVIEW OF LITERATURE

This chapter reviews the architecture of an on-chip network router, the use of on-chip network and some research and design issues in an on-chip network which are closely related to this thesis.

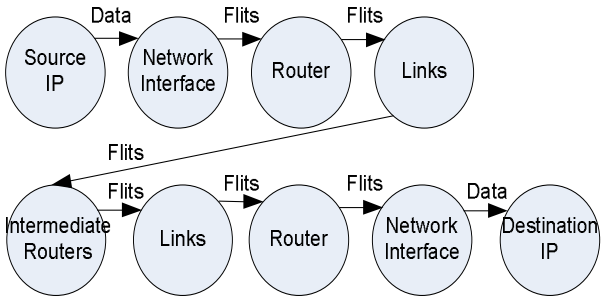


Figure 2.1 Data Flow From the Source to the Destination

2.1 On-chip Network

On-chip network contains four components. They are intellectual property (IP), network interface (NI), router and link.

Intellectual property is the component that computes data and provides data storage. Examples of Intellectual property include microprocessor, field programmable gate array (FPGA), application-specific integrated circuit (ASIC), graphics processing unit and memory controller.

Network interface (NI) is an interface between network router and IP. It packetizes data sent by a directly connected IP and transmits the packet into the network through the router. Besides, it receives other packets sent by other IPs through the router, de-packetizes the received packets and sends the data to the directly connected IP.

Router controls packet flow. It receives a packet either from the network interface or from its neighbor routers. Then, it analyzes the packet address and delivers the packet to the proper direction.

On-chip router ports are connected by links. Link width is shorter than packet length. A packet has to be divided into flits to traverse through the network. Each flit is usually 128 bits which equals the link width. Each router port can send and receive one flit per cycle through the links.

Topology defines how on-chip routers are connected. On-chip network topology includes 2D Mesh [15], Torus [27], Ring [24], FatTree [21] and Butterfly [33]. Among all of these, 2D Mesh is the most commonly used topology [19, 32, 12, 4, 5, 27] because of its regularity and simplicity. An on-chip system with a 2D-Mesh network is shown in Fig. 2.2.

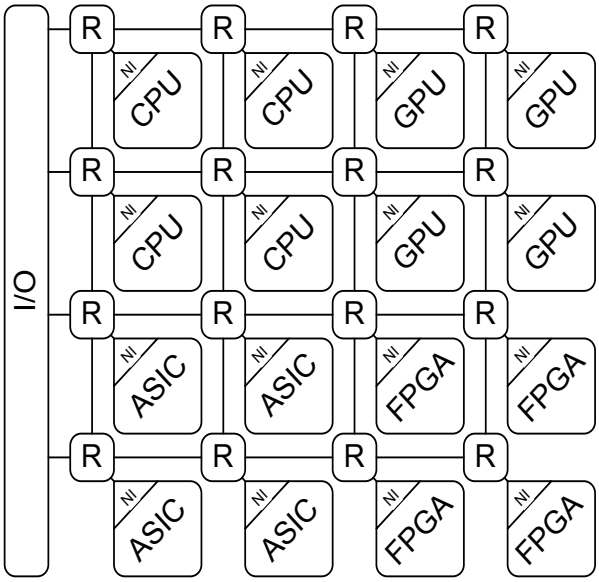


Figure 2.2 System On Chip with 2D Mesh On chip Network

2.2 Router architecture

A simple router in a 2D mesh network has three components. They are input buffer, route computation unit and switch arbiter. An input buffer temporarily stores the packet flits. An

Input buffer temporarily stores the packets. Its main component is FIFO storage. Each FIFO's slot stores one flit. The first flit going into the FIFO storage will leave first. The switch arbiter allocates the output port to the packet. Each router has five input ports and five output ports in the North, the East, the West, the South and Local. The North output port is connected to the South input port of the router on the North. The East output port is connected to the West input port of the router on the East. The West output port is connected to the East input port of the router on the West. The South output port is connected to the North input port of the router on the South. The Local port is connected to the Local IP. When the flits arrive at the router, they are stored in FIFO storage. When all address flits of a packet have arrived, the route computation unit digests the address flits and decides which direction the packet is traversing to. The packet then requests the output port in the designated direction. Once the packet has obtained the output port, the flits traverse to the next router.

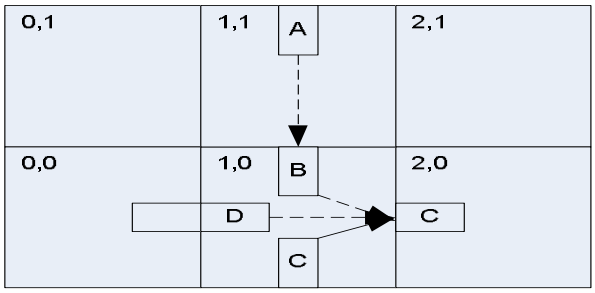


Figure 2.3 Physical Channel Blocking Problem

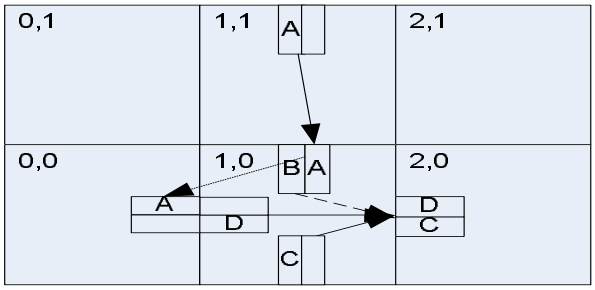


Figure 2.4 Solving Blocking problem by Virtual Channel

If there is only one set of FIFO storage in each port, the router is called a physical channel router. Physical channel router is small but suffers from packet blocking problem. Fig. 2.3 shows an example of packet blocking problem. In router (0,1), Packet B, C and D are competing for the storage set on router (0,2)'s West port. Packet C is the winner which is currently occupying the storage set that Packet B and D are still waiting for. Packet B is temporarily stored in the storage set on router (1,0)'s North port. Packet D is temporarily stored in the storage set on router (1,0)'s West port. Packet A's destination is (0,0), but it is now temporarily stored in the storage set on router (1,1)'s North port. It is supposed to traverse to the South. As there is only one set of FIFO storage on router (1,0)'s North port and it is being occupied by Packet B, Packet A has to wait until Packet B has released it. However, Packet B will not release the storage set on router (1,0)'s North port until it has been traversed to the storage set on router (2,0)'s West port which is currently occupied by Packet C. To relieve the blocking problem, virtual channel is introduced. Virtual channel router increases the number of FIFO storage sets on each router port. These FIFO storage sets are called virtual channels. Fig. 2.4 shows a network with virtual channel routers. Each router's port contain two FIFO storage sets. With the extra storage set, Packet A can now be routed to its destination by going through another FIFO storage set on router (1,0)'s North port while Packet B is still occupying one storage set on the same router's port. The packet latency of A is reduced and hence, network performance is improved. The link from router (0,0) to router (1,0) in physical channel router was idle, because Packet A was blocked at router (1,1). On the contrary, the link from router (0,0) to router (1,0) in virtual channel router was used by Packet A. It shows that virtual channel router also increases wire utilization. Even though virtual channel router seems to be superior than physical channel router, it has some drawbacks. The extra FIFO storage sets make the design more complex and requires more space and energy. Also, a more complex virtual channel arbiter is needed to decide which packet owns which storage set on the downstream router's port. As a result, a virtual channel router has a larger physical size and a higher energy consumption.

2.2.1 Pipelined On-chip Router

The router throughput can be increased by pipelining. Peh *et al.* [34] have introduced a 6-stage pipeline on-chip virtual channel router. The six stages are Buffer Write (BW), Route Computation (RC), Virtual Channel Allocation (VA), Switch Allocation (SA), Switch Traversal (ST) and Link Traversal (LT). At Buffer Write (BW) stage, an incoming flit is written into an input buffer's FIFO storage. At Route Computation (RC) stage, a route computation unit decides which direction a packet goes. At Virtual Channel Allocation (VA) stage, a virtual channel arbiter allocates a virtual channel on the downstream router to a packet. At Switch Allocation (SA) stage, a switch arbiter allocates an output port for a packet which owns a virtual channel on the downstream router. At Switch Traversal (ST) stage, the packet traverses from the input buffer's FIFO storage to the output port. At Link Traversal (LT) stage, the packet traverses from the current router's output port to the downstream router's input port.

2.2.2 Store-and-Forward, Virtual cut-through and Wormhole Routing

As mentioned before, packet length is longer than link width. The network interface divides a packet into many flits (flow control units) and sends them out one by one. (Fig. 2.5)

Figure 2.5 Packet Format

Each packet contains at least one *Head* flit, at least one *Body* flit and one *Tail* flit. *Head* flits store the destination addresses of a packet. A route computation unit digests *Head* flits to make route decision. *Body* flits store the data sent from the source IP. *Tail* flit is the last flit of a packet. It works as a *Body* flit but also indicates the end of the packet. A virtual channel is released after the *Tail* flit leaves it.

There are three methods to route a packet. They are Store-and-Forward Routing, Virtual

Cut-Through Routing and Wormhole Routing. They are different in FIFO storage length and route decision time.

In Store-and-Forward routing, each router's FIFO storage size is equal to the packet size. Store-and-Forward router waits until the whole packet has arrived before making route decision. It has high packet latency and requires a big FIFO storage.

Virtual cut-through router makes route decision once all the *Head* flits have arrived. Compared with Store-and-Forward routing, Virtual cut-through routing has lower packet latency. However, its FIFO storage size is still the same.

Due to limited space in a chip, router's FIFO storage size is usually small. This characteristic makes Store-and-Forward routing and Virtual cut-through routing unfavorable in the on-chip network because packet size is limited by small FIFO storage size. It decreases the packet efficiency because a large portion of the packet is *Head* flits.

Wormhole routing is the most favorable routing method in the on-chip networking community. FIFO storage size is smaller than the packet size which makes the router smaller. More chip space can be reserved for other resources. In case of packet blocking, the packet spans across multiple routers. Similar to the Virtual cut-through router, route decision is made once all address flits have arrived.

2.3 Deterministic and Adaptive Routing

Flit flow control can be categorized into deterministic routing and adaptive routing. Packet route is fixed in deterministic routing. Router cannot change packet routes based on dynamic network congestion status. XY-routing is the most famous deterministic routing algorithm. A packet is first routed in the X direction until it reaches the column of the destination. The packet is then routed in the Y direction until it reaches the destination. XY-routing is simple and easy to implement in hardware. It is also free from unicast deadlock. The only drawback is that it suffers from the congestion problem stated in Chapter 1.

Another deterministic routing algorithm is pre-computed routing. In pre-computed routing, a packet route from a source to its destination is predetermined by the network interface. This

pre-computed route is stored in the packet's *Head* flits. Network routers route the packet according to the pre-computed route in the *Head* flits. When all network interfaces collaborate and a proper algorithm is used to compute the route, both unicast and multicast deadlocks could be avoided.

Adaptive routing allows routers to make route decisions based on the network congestion status. In adaptive routing, each router has congestion information of its surroundings. The adaptive router routes the packet to a less congested channel. Gratz *et al.* [20] have summarized four ways to determine the congestion condition.

1. The number of free virtual channels:

   A link will be free sooner if less virtual channels are in use on the downstream routers [13].

2. The number of free FIFO storage slots:

   Each virtual channel contains many FIFO storage slots. Counting the number of free FIFO storage slots provides a more comprehensive measurement compared with counting the number of free virtual channels. A higher number of free FIFO storage slots implies that the link will be free sooner [3].

3. Crossbar demand:

   This method uses the number of packets competing for a same link to estimate the congestion condition [20]. If the number is low, that link will be free sooner.

4. Composite metric:

   Composite metric combines the three methods described above in different proportions.

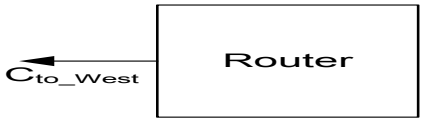


Figure 2.6 DyXY adaptive routing

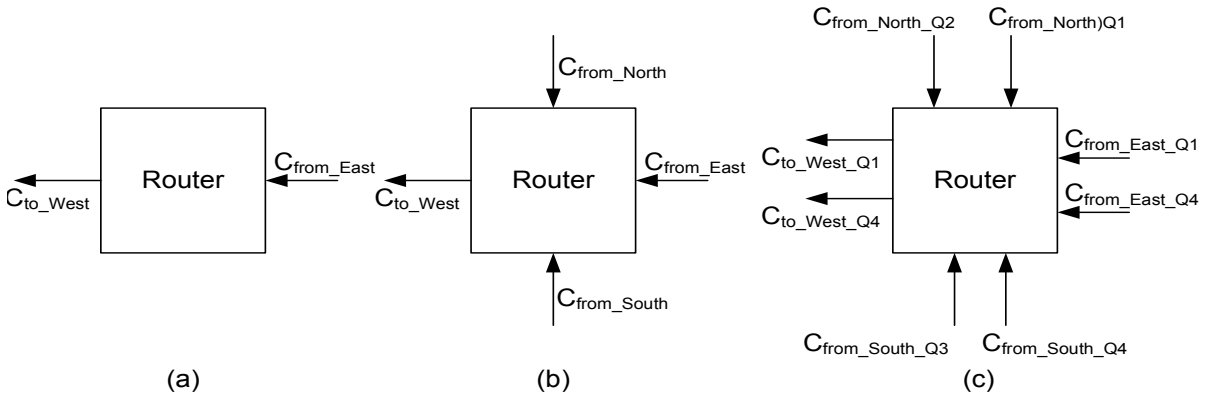


Figure 2.7 Regional Congestion Aware adaptive routing

DyXY [29] is one of the well-cited adaptive routing methods. In *DyXY*, each router receives the FIFO storage usage information from the downstream routers through the dedicated congestion status wires. In the Route Computation (RC) stage, the router routes a packet to the downstream routers with the lowest FIFO storage usage. Fig. 2.6 shows how *DyXY* router sends the congestion information (C_{to_West}) to the router on the West. The router sends the FIFO storage usage information on its West port to the router on the West. $C_{to\_West} = f(C_{West\_port})$

Regional Congestion Aware Router (RCA) [20] is another well-cited adaptive routing method. *RCA* aims to include more downstream links congestion information to increase the accuracy in making efficient routing decisions. The congestion information is aggregated and propagated starting from some distanced downstream links. The congestion information of closer links is weighted heavier than more distanced links.

There are three versions of RCA which are:

1. RCA 1D:

   The routers aggregate and propagate the congestion information in two directions. Information moves either vertically or horizontally in the same direction as the incoming information. Fig. 2.7(a) shows how *RCA 1D* transfers the congestion information (C_{to_West}) to the router on the West. The router aggregates its own congestion information and

information received from the router on the East. Then, it propagates the aggregated information to the router on the West. These aggregated information is a function of the router's FIFO storage usage information on its West port and the congestion information received from the router on the East. $C_{to\_West} = f(C_{West\_port}, C_{from\_East})$

2. RCA Fanin:

   The routers aggregate and propagate congestion information from all Fanin directions except the one that the information is going to be propagated to. Fig. 2.7(b) shows how *RCA Fanin* passes the congestion information (C_{to_West}) to the router on the West. The router aggregates its own congestion information and the information received from the routers on the East, the North and the South. Then, it propagates the aggregated information to the router on the West. These aggregated information is a function of the router's FIFO storage usage information on its West port and the congestion information received from the routers on the East, the North and the South. $C_{to\_West} = f(C_{West\_port}, C_{from\_North}, C_{from\_South}, C_{from\_East})$

3. RCA Quadrant

   The routers aggregate and propagate congestion information by quadrants. It propagates two selected congestion information to the downstream router based on the direction of the propagation. Fig. 2.7(c) shows how *RCA Quadrant* passes the congestion information (C_{to_East}) to the router on the West. The router aggregates its own information and those received from the routers on the East, the North and the South. Then, it propagates the aggregated information to the router on the West. There are two aggregated information $C_{to\_West\_Q1}$ and $C_{to\_West\_Q4}$. $C_{to\_West\_Q1}$ is a function of the router's FIFO storage usage information on its West port and the *Q1* congestion information received from the routers on the East and the North. $C_{to\_West\_Q1}$ is a function of the router's FIFO storage usage information on its West port and the *Q4* congestion information received from the routers on the East and the South. $C_{to\_West\_Q1}$ reflects the congestion information to the *Quadrant I*. $C_{to\_West\_Q4}$ reflects the congestion information to the *Quadrant IV*. $C_{to\_West\_Q1} = f(C_{West\_port}, C_{from\_North\_Q1}, C_{from\_East\_Q1})$

$$C_{to\_West\_Q4} = f(C_{West\_port}, C_{from\_South\_Q4}, C_{from\_East\_Q4})$$

2.4 Unicast and Multicast

There are mainly two types of packet communications in on-chip network. They are unicast and multicast. There is only one sender and one receiver in unicast. There is one sender and more than one receiver in multicast. The needs for multicast communication are described in Section 1.1.2. Multicast packet routing can either be path-based or tree-based. In path-based multicast packet routing, a packet follows a pre-determined path to reach all the destinations one by one. The packet diverges when it has arrived at one of the destinations. It ends when the packet has arrived at the last destination. In tree-based multicast packet routing, a packet diverges at selected routers based on the routing algorithm to reach its destinations independently.

2.5 Deadlock

There are two types of deadlocks. They unicast deadlock and multicast deadlock. Deadlock halts packet flow and makes the whole network unfunctionable.

2.5.1 Unicast Deadlock

Unicast deadlock has been well studied in previous research. It occurs when multiple packets in the virtual channels form a dependent cycle. Fig. 2.8 shows a unicast deadlock example.

Packet 55 stored in router (1,1)'s East channel needs to traverse to router (1,2)'s South channel.
Packet 66 stored in router (1,2)'s South channel needs to traverse to router (2,2)'s West channel.
Packet 77 stored in router (2,2)'s West channel needs to traverse to router (2,1)'s North channel.
Packet 88 stored in router (2,1)'s North channel needs to traverse to router (1,1)'s East channel.
No packet can traverse to the desired channel until at least one of the packets has released its own channel. No packet can release its own channel until at least one of the packets has traversed to its desired channel. As a result, a unicast deadlock is formed.

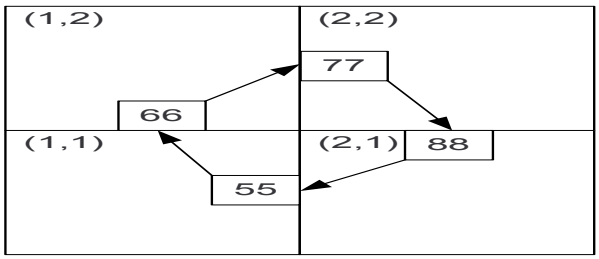


Figure 2.8 Unicast Deadlock

Unicast deadlock can be avoided by turn prohibition [18].

Deterministic XY-routing is free from unicast deadlock because some packet turns are not allowed. Packet turns from the North/South to the East/West are restricted. Under this circumstance, no dependent cycles can be formed.

Adaptive routing without restricting packet turns suffers from unicast deadlock. Glass [18] introduces *West-First, North-Last* and *Negative-First* methods to restrict some turns in the adaptive routing. However, Glass's methods create an unbalanced traffic distribution in the network. Taking this disadvantage into account, Chiu [10] proposes a *Odd-Even Turn Model*. The model prohibits the East to the North/South packet turns in even columns and the North/South to the West turns in odd columns. Instead of using turn restrictions, Dally *et al.* [13] introduce *Ordered Virtual Channels*. In their method, routers and channels are ordered. A router can only use a channel with equal or less order to send packets to the downstream routers. Kim *et al.* [26] propose assigning different channels to store packets with different turning directions to avoid deadlock.

2.5.2 Multicast Deadlock

Like unicast deadlock, multicast deadlock is formed by channel dependency. Unlike unicast deadlock, multicast deadlock cannot be avoided by turn restrictions or ordered virtual channels. Therefore, even XY-routing suffers from multicast deadlock. Lin *et al.* [30] show an example of multicast deadlock (Fig. 2.9). This example applies to routers with physical channel and routers with any number of virtual channels. The number of virtual channels is assumed to be

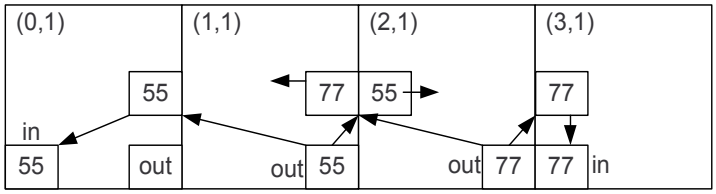


Figure 2.9 Multicast Deadlock Example

one.

IP (1,1) sends multicast Packet 55 to IP (0,1) and IP (3,1).

IP (2,1) sends multicast Packet 77 to IP (0,1) and IP (3,1).

Packet 55 at router (1,1) has successfully obtained router (0,1)'s East channel and router (2,1)'s West channel.

Packet 77 at router (2,1) has successfully obtained router (3,1)'s West channel and router (1,1)'s East channel.

Flits are sent to their obtained channels accordingly.

However, Packet 55 at router (2,1) is waiting for router (3,1)'s West channel which is occupied by Packet 77. At the same time, Packet 77 at router (1,1) is waiting for router (0,1)'s East channel which is occupied by Packet 55. As a result, a multicast deadlock is formed.

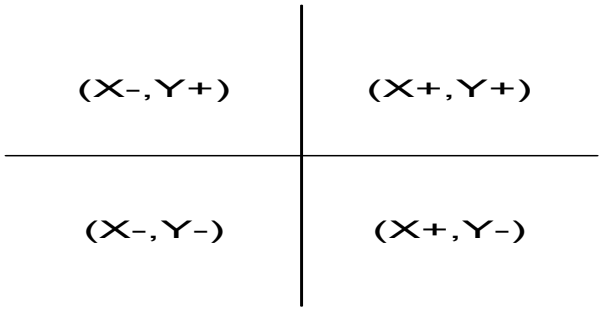


Figure 2.10 X+, X-, Y+, Y- Zones

Lin *et al.* [30] propose deterministic routing strategies to avoid multicast deadlock. It is to partition the destinations into $(X+, Y+)$, $(X+, Y-)$, $(X-, Y+)$ and $(X-, Y-)$ zones (Fig.

2.10). A sender sends one packet copy to its destinations in each zone using XY-routing.

To increase the network throughput, they also propose *Hamiltonian Path Partitioning*. In this method, the network interface of the packet source pre-computes the packet route using *Hamiltonian Partitioning* algorithm and stores the route into the packet's *Head* flits. Network router routes the packets following the designated route in the packet's *Head* flits. *Hamiltonian Partitioning* algorithm ensures the network is free from multicast deadlock.

Samman *et al.* [35] introduce a multicast physical channel router which can avoid the multicast deadlock using *Planar Network* [9]. *Planar network* (Fig. 2.11) has two duplicate sub-network $(X+)$ and $(X-)$. If the X-coordinate of the packet destination is higher or equal to the X-coordinate of the packet source, the packet would be routed through $(X+)$. Otherwise, the packet would be routed through the $(X-)$.

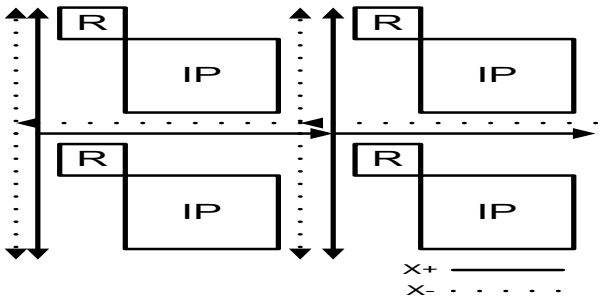


Figure 2.11 Planar Network

Bjerregaard *et al.* [6] have published a survey paper covering many areas in previous research about on-chip network. Interested readers are referred to Bjerregaard *et al.* for more information.

CHAPTER 3. Path-Based Adaptive Routing

This chapter discusses *Path-based adaptive routing*. The goal of this adaptive routing is to route a packet to its destination using less congested channels in order to reduce packet latency. The *Path-based adaptive routing* is a deadlock free routing tailored to the *Odd-Even Model* for increased accuracy in estimating congestion condition. Utilizing the slack time in Route Computation (RC) stage in a pipelined router can improve network performance without increasing router clock cycle. Motivation for *Path-based adaptive routing* is first discussed in the next section. *Path-based adaptive routing* operation and its micro-architecture will be discussed later in this chapter.

3.1 Motivation

Adaptive router is able to route the packets to their destinations faster than deterministic router by utilizing less congested links. There are a few proposed on-chip adaptive routers including *DyXY* router and *RCA* router. In this chapter, we propose *Path-based adaptive router*. It is believed to have higher throughput and lower packet latency than the two adaptive routers mentioned above.

DyXY [29] adaptive router uses congestion information of the links from its downstream routers to make routing decision. Because its congestion observation window is limited, the network performance is not satisfying. In addition, when *DyXY* routing is paired with *Odd-Even Routing* restrictions to ensure the network is deadlock free, the adaptive routing choices are very limited. As a result, *DyXY* adaptive routing performs only slightly better than XY-routing.

Compared with *DyXY*, *Regional Congestion Aware Router (RCA)* [20] uses a much larger

congestion observation window (*1D*,*Fanin* and *Quadrant*). They aggregate and propagate congestion information starting from some far away routers. The congestion information of closer links is weighted heavier than more distant links. All congestion information is reduced to a final 9-bit congestion value to aid in making routing decision. The drawback of *RCA* is the two types of noises produced in the congestion information. The first type is the unwanted congestion information of links beyond the packet destination. The second type is the unwanted congestion information of the links which are unusable when a deadlock free algorithm is applied. The unwanted noises in the final congestion value degrade the packet routing quality.

Path-Based Adaptive Router (Path) is designed to have a large congestion observation window to improve routing quality. Compared with *RCA*, *Path* is more accurate in estimating congestion condition because *Path* only takes the valid path between the packet source and destinations into account to make routing decision.

Adaptive routing lengthens the route computation stage. Router clock cycle can be conserved in *Path* by utilizing the slack time in Route Computation (RC) stage.

3.2 Basic Unicast Router

Path is based on the 6-stage pipeline router proposed by Peh *et al.* [34] It has six stages:

1. Buffer Write (BW),
2. Route Computation (RC),
3. Virtual Channel Allocation (VA),
4. Switch Allocation (SA),
5. Switch Traversal (ST).
6. Link Traversal (LT).

A packet is first written into the input buffer's FIFO storage (virtual channel) at BW stage. At RC stage, the route computation unit decides the packet outgoing direction. The

route decisions are written into a virtual channel status table. Then, the packet requests a virtual channel in the downstream router. Once the packet has obtained a virtual channel, the packet location is written into the SA unit's FIFO in charge of output port allocation in the designated direction. Packet location is represented by port identification number ($PORTID$) and virtual channel identification number ($VCID$). With ($PORTID, VCID$), the SA unit is informed that a packet located at virtual channel $VCID$ on port $PORTID$ needs an output port to go to the downstream router. At SA stage, some packets which have obtained virtual channels compete for the output port. SA unit uses FIFO to decide which packet will get the output port. This FIFO keeps the output port requests in order. At ST stage, the packet traverses from the virtual channel in the input buffer to the output port. Finally, the packet traverses to the downstream router at LT stage.

The router contains four main components: input buffer, virtual channel allocator, switch allocator and crossbar switch. Three changes have been made from the original unicast router. First, an FIFO is added into the input buffer to keep the virtual channel requests from all packets in the input buffer in order. Second, we add an FIFO into the switch allocator to keep the output port requests from all packets in order. Last, the virtual channel allocator was modified to allocate the downstream router's virtual channel request to all packets in round robin style. The operations of a basic router will be discussed in detail in this section.

3.2.1 Input Buffer

Input Buffer (fig. 3.1) is a temporary storage for the incoming packet. In a router, there is an input buffer on each port. Each input buffer contains several virtual channels (FIFO Storages) which store the incoming flits. The number of virtual channels ($NumVC$) is four in the example. A virtual channel status table stores the packet and virtual channel information. The information includes whether the packet is routed, the designated direction, the next virtual channel will be used in the downstream router ($NVCID$) and whether the virtual channel is full or empty. The table is used by the input buffer, the router's virtual channel allocator and the router's switch allocator. When a flit has arrived, the input buffer controller

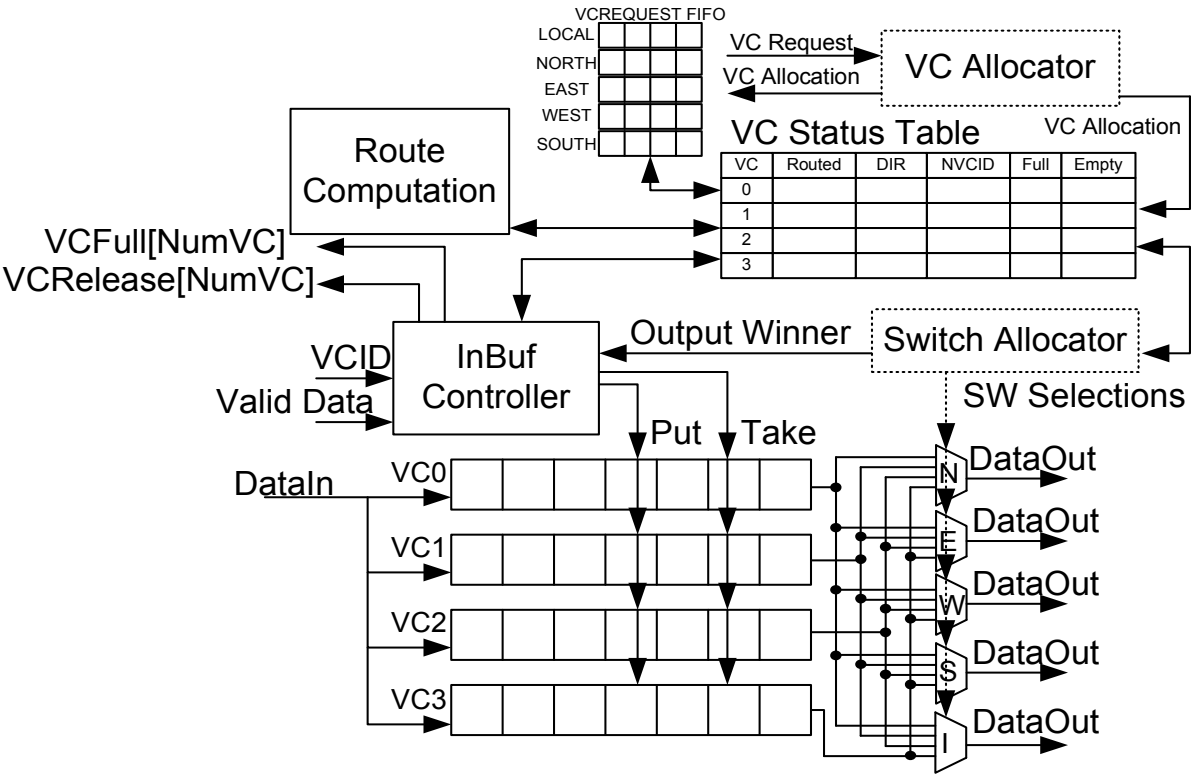


Figure 3.1 Unicast Input Buffer

Table 3.1 Input Buffer Signals

Signal	Description
VCFull[NumVC]	The virtual channel is full
VCRelease[NumVC]	The virtual channel is released by a packet
VCID	The VC used by the incoming Data
Valid Data	the incoming data is valid
DataIn	Incoming Data
Put	Put the Data into the virtual channel
Take	Remove the Data from the virtual channel
New VC Request	Adding a VC Request into the VC REQUEST FIFO
VC Request	Requesta a VC in the downstream routers
VC Allocation	Allocate a downstream VC to a packet
Output Winner	The output selected by the switch allocator
SW Selections	The winning packet selected by the switch allocator
DataOut	Data going out of the router

writes the flit into the virtual channel according to the incoming $VCID$ signal. The Route Computation then decides a routing direction (DIR) for the packet and marks the packet as routed in the table. An entry is written into *VCREQUEST FIFO*. *VCREQUEST FIFO* keeps the virtual channel requests in order. A *VC Request* signal is sent to the virtual channel allocator. When the virtual channel ($NVCID$) in the downstream router is allocated, that request entry is removed from the *VCREQUEST FIFO* and the identification of the virtual channel ($NVCID$) is written into the virtual channel status table. Switch allocator decides the router's output port winner. When a flit traverses to the downstream router, the input buffer controller removes the flit from the virtual channel.

3.2.2 Virtual Channel Allocator

Fig. 3.2 shows the virtual channel allocator micro-architecture. Table 3.2 explains the signals in the virtual channel allocator. Each router contains one virtual channel allocator. It is used to allocate the virtual channels in the downstream (remote) routers to the packet at the local router's input buffer. It contains five micro allocators to handle the IP, the North, the East, the West and the South virtual channel allocation. The allocation order is maintained by round robin (RR) algorithm. Each micro virtual channel allocator has some

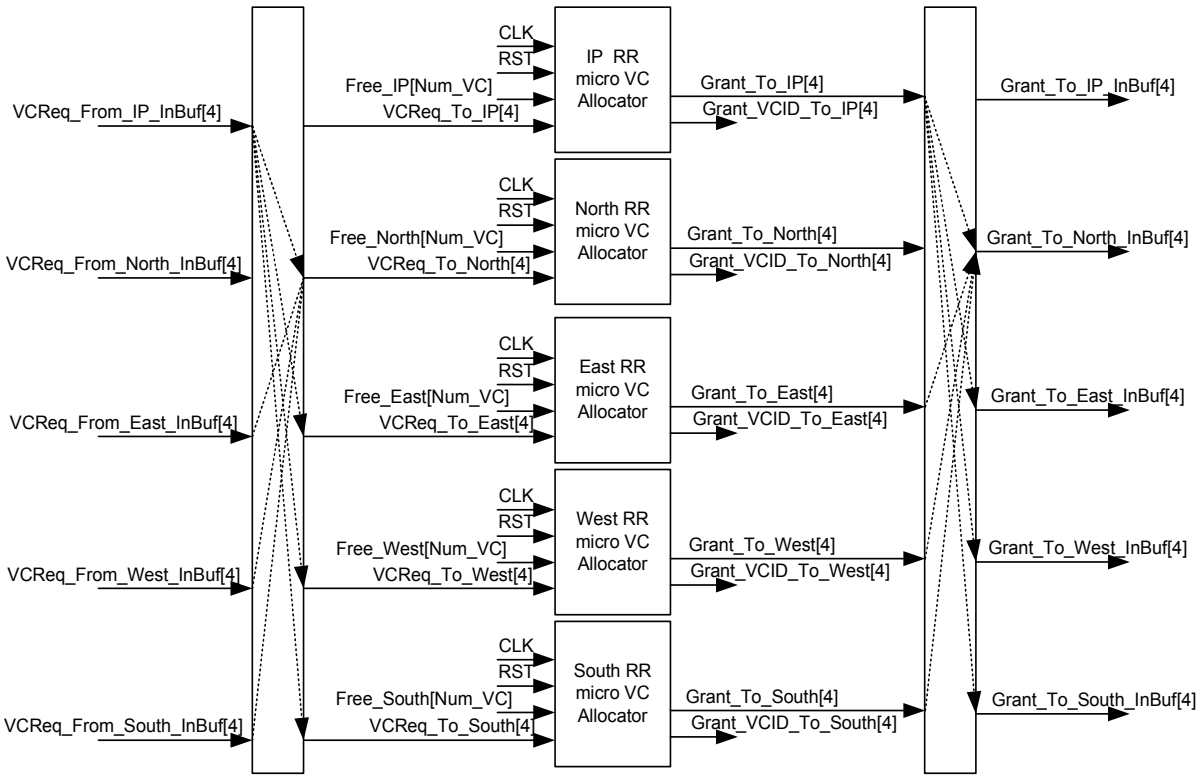


Figure 3.2 Virtual Channel Allocator

Table 3.2 Virtual Channel Allocator Signals

Signal	Description
VCReq_From_InBuf_LD	Virtual Channel Request from the Local Input Buffer at direction LD
Free_RD	Free the Virtual Channel at the Remote Router Input Buffer at the direction RD
VCReq_To_RD	Virtual Channel Request To Remote Router at direction RD
Grant_To_RD	Grant a Virtual Channel at Remote Router at direction RD
Grant_VCID_To_RD	The ID of the Virtual Channel being granted at the Remote Router at direction RD
Grant_To_InBuf_LD	Grant a Virtual Channel to the Local Input Buffer at direction LD

memory components to record the availability of the virtual channels in the downstream routers and the local IP. When the micro virtual channel allocator allocates a free virtual channel $NVCID$ of the downstream router's input buffer in the direction RD to the packet at the local router's input buffer on the port LD, it flags the corresponding $Grant_{T}o_{R}D$'s LD signal. The local router's input buffer at the direction LD records the $NVCID$ for the requesting packet once it has seen the flagged signal.

When the downstream router's input buffer releases the virtual channel, it flags the $Free\_RD$ signal to notify the local router's virtual channel allocator that the virtual channel is released and can be allocated to other packets in the future.

3.2.3 Switch Allocator

Switch allocator (fig. 3.3) allocates the output port to packets in the virtual channels in the input buffers. It contains five micro switch allocators. Each micro switch allocator has a switch FIFO to record all output port competitors. When the micro virtual channel allocator allocates a downstream virtual channel to the packet in the local virtual channel ($LVCID$) at port ($PortID$), the $(LVCID, PortID)$ entry is sent to the micro switch allocator. The micro switch allocator adds the entry into its switch FIFO for switch allocation. The entry at the head of the switch FIFO in the allocator represents the owner of the output port. The output

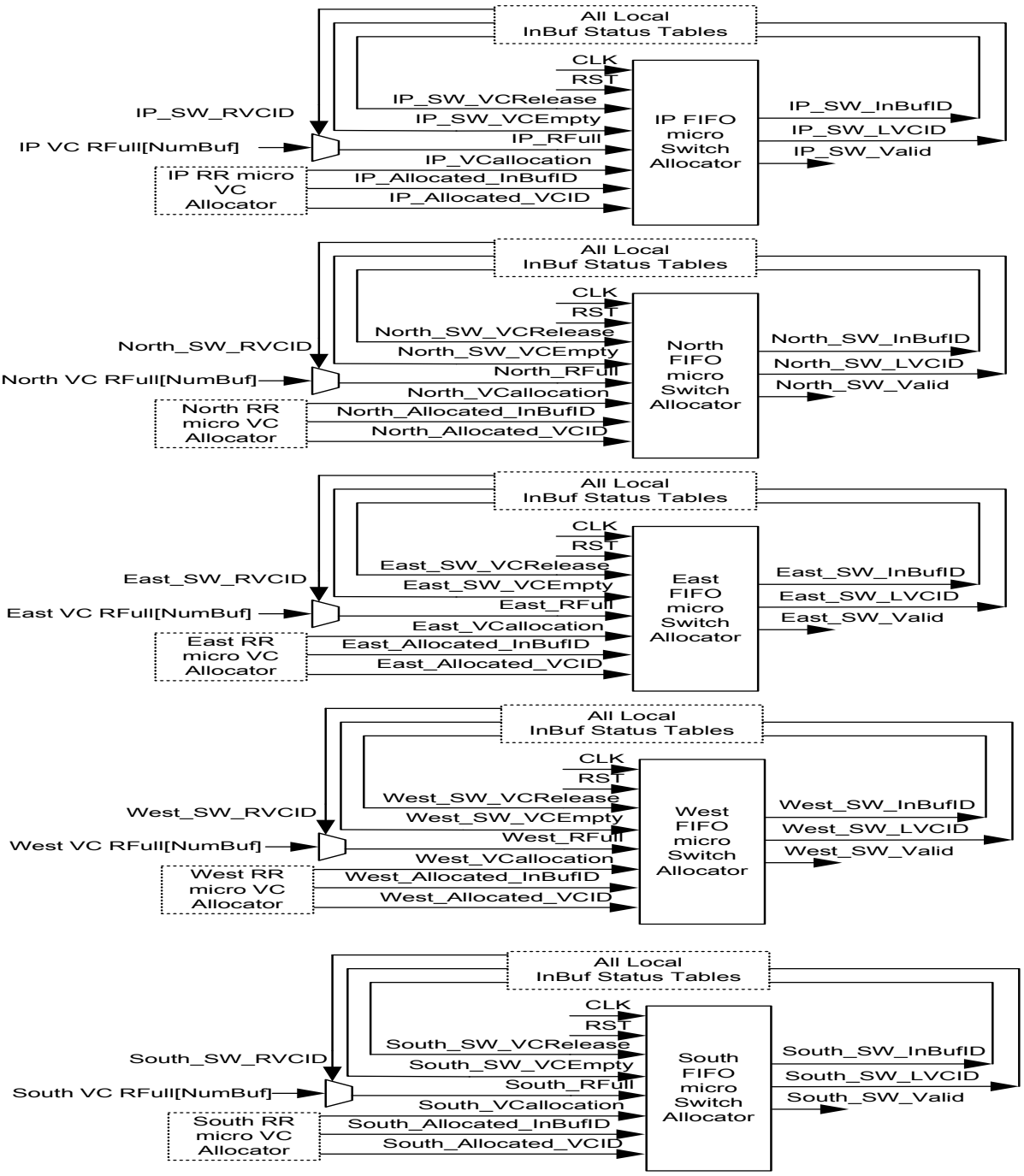


Figure 3.3 Switch Allocator

Table 3.3 Switch Allocator Signals

Signal	Description
*LD*_SW_VCRelease	VCRelease signal from the selected output owner. It's on when the packet's tail flit is leaving the selected VC
*LD*_SW_VCEmpty	VCEmpty signal from the selected output (switch) owner.
*LD*_RFull	Full signal from the remote router's target VC.
*LD*_VCallocation	VCallocation signal from *LD* micro VC allocator.
*LD*_Allocated_Inbuf	It indicates which inbuf obtains the remote VC
*LD*_Allocated_VC	It indicates which local VC obtains the remote VC
*LD*_SW_InBufID	The current *LD* output owner (InBuf)
*LD*_SW_LVCID	The current *LD* output owner (Local VC)
*LD*_SW_Valid	Valid Output Owner
*LD*_SW_RVCID	VCID of the remote VC
*LD*_VC_RFULL[NumBuf]	All remote VC full signals

port owner releases the output port when one of the three following situations occurs.

1. The tail flit is coming out from the virtual channel to the output port.

   In this case, packet transmission is complete. The local virtual channel is released. The corresponding entry in the switch FIFO is removed from the switch FIFO.

2. The packet owning the output port has no flits in its virtual channel.

   In this case, no flit will come from the virtual channel in the next cycle. The corresponding entry in the switch FIFO will be moved from the FIFO's head to the FIFO's tail.

3. The virtual channel in the downstream router is full.

   In this case, the virtual channel in the downstream router has no space to store the new flit sent by the packet. The corresponding entry in the switch FIFO will be moved from the FIFO's head to the FIFO's tail.

3.3 Path-Based Adaptive Routing

3.3.1 Congestion Estimation

Path-Based Adaptive Router (fig. 3.4) contains a congestion table to aid in routing. The congestion table receives congestion information from the local virtual channel status table and downstream routers. It also sends its own congestion information to upstream routers.

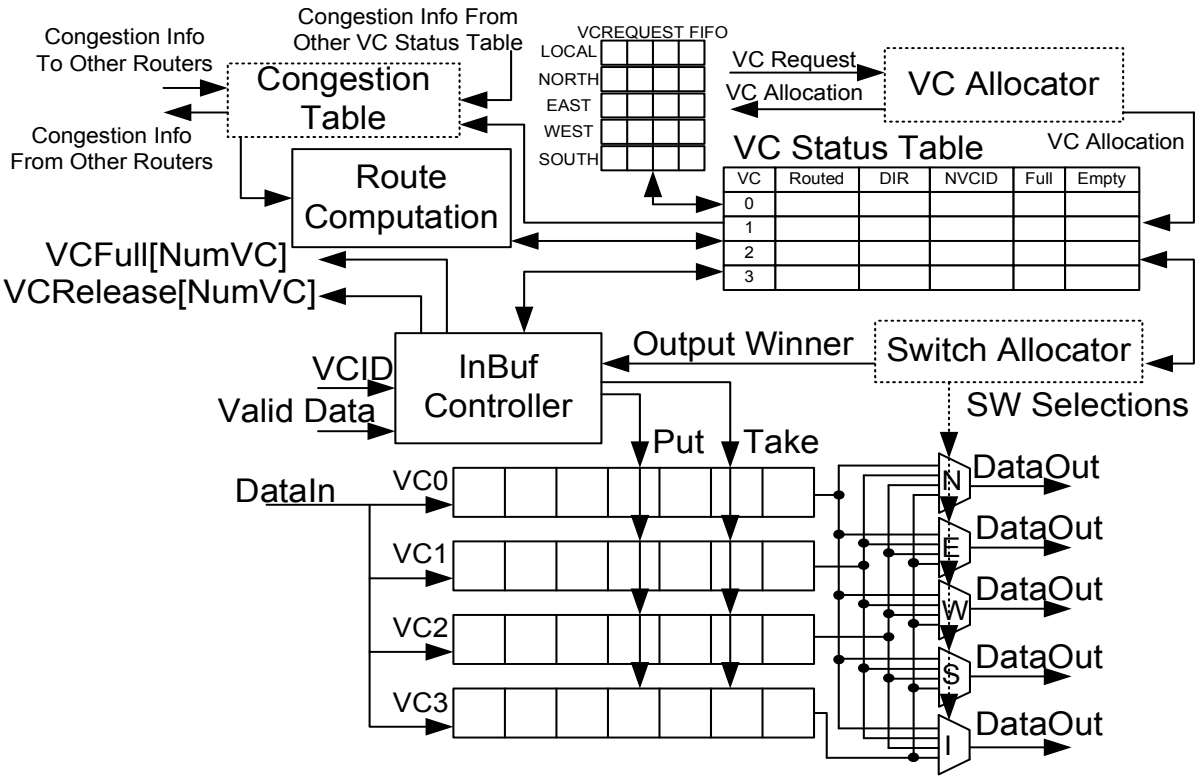


Figure 3.4 Adaptive Router's Input Buffer

Path makes use of the *crossbar demand* to estimate the channel congestion condition. In *crossbar demand* metric, a number system is used to represent the congestion condition. The congestion number of link d at the router (x,y) $C_{(x,y),d}$ equals the number of packets in the router requesting link d.

Suppose two packets are routed to the North from the South and three packets are routed to the North from the East at router (1,2), $C_{(1,2),East} = 2 + 3 = 5$

Table 3.4 Six routes from location (2,2) to location (0,4)

Choice	Path	Stop 0	Stop 1	Stop 2	Stop 3	OE Viol.	Even	Odd
1	WWNN	(2,2)	(1,2)	(0,2)	(0,3)	Free	Free	N,S
2	WNWN	(2,2)	(1,2)	(1,3)	(0,3)	Even	Free	N,S
3	WNNW	(2,2)	(1,2)	(1,3)	(1,4)	Even	Free	N,S
4	NWWN	(2,2)	(2,3)	(1,3)	(0,3)	Odd	W	Free
5	NWNW	(2,2)	(2,3)	(1,3)	(1,4)	Both	W	Free
6	NNWW	(2,2)	(2,3)	(2,4)	(1,4)	Odd	W	Free

3.3.2 Adaptive Unicast Route Decision

In *Path-based adaptive routing*, each path (p) from the router to the packet destination has a cost ($Cost_p$).

$Cost_p = \sum C_i$ when p is a valid path where i is the link along p.

A valid path is a path obeying the *Odd-Even Turn Model*'s turn restrictions.

$Cost_p = \infty$ when p is an invalid path.

Path-based adaptive routing compares the costs ($Cost_p$) of each path (p) from the current router to the packet destination and routes the packet along the first link of the path with a lowest cost.

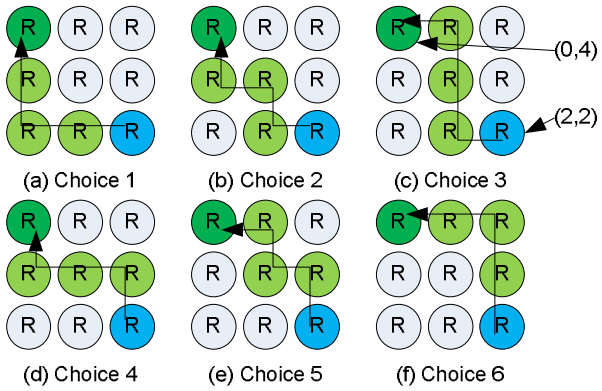


Figure 3.5 Routes from (2,2) to (0,4)

For example, to route a packet from router (2,2) to router (0,4), there are six minimal

paths (Fig. 3.5 and Table 3.4). In path 1, the packet traverses to the West at router (2,2). Then, it traverses to the West at router (1,2), to the North at router (0,2) and to the North at router (0,3). Finally, it arrives at the the destination router (0,4). In these six paths (Fig. 3.5), only path 1, 4 and 6 are valid. Path 2, 3 and 5 violate the *Odd-Even Routing Model*'s turn restrictions [10]. $Cost_{p1} = C_{(2,2),West} + C_{(1,2),West} + C_{(0,2),North} + C_{(0,3),North}$,

$Cost_{p2} = \infty$,

$Cost_{p3} = \infty$,

$Cost_{p4} = C_{(2,2),North} + C_{(2,3),West} + C_{(1,3),West} + C_{(0,3),North}$,

$Cost_{p5} = \infty$,

$Cost_{p6} = C_{(2,2),North} + C_{(2,3),North} + C_{(2,4),West} + C_{(1,4),West}$.

Note that the packet is only required to follow the first link of the chosen path. The downstream routers will make the following routing decisions according to their own congestion number.

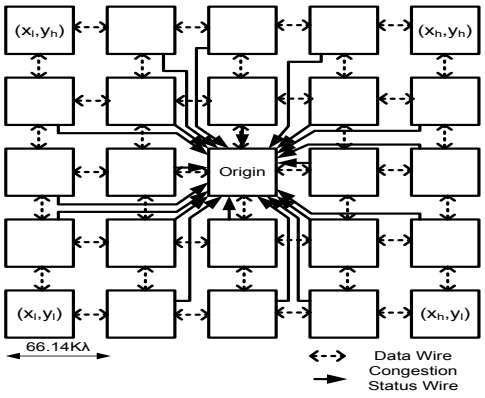


Figure 3.6 Congestion Observation Window 5×5

Each router obtains the downstream routers' congestion information through the dedicated congestion information wires (Fig. 3.6). In *Odd-Even Routing Model*, the number of path from the source (x_{src}, y_{src}) to the destination (x_{dest}, y_{dest}) is approximately $\frac{(d_y+h)!}{d_y!h!}$ where $h = \lceil \frac{d_x}{2} \rceil$, $d_x = |x_{dest} - x_{src}|$, $d_y = |y_{dest} - y_{src}|$. Fig. 3.7 and Fig. 3.8 show the number of valid path from

15	5	5	1	1	1	5	5	15
10	4	4	1	1	1	4	4	10
6	3	3	1	1	1	3	3	6
3	2	2	1	1	1	2	2	3
1	1	1	1	Origin	1	1	1	1
3	2	2	1	1	1	2	2	3
6	3	3	1	1	1	3	3	6
10	4	4	1	1	1	4	4	10
15	5	5	1	1	1	5	5	15

Figure 3.7 Num. of Valid Path (Odd)

15	15	5	5	1	5	5	15	15
10	10	4	4	1	4	4	10	10
6	6	3	3	1	3	3	6	6
3	3	2	2	1	2	2	3	3
1	1	1	1	Origin	1	1	1	1
3	3	2	2	1	2	2	3	3
6	6	3	3	1	3	3	6	6
10	10	4	4	1	4	4	10	10
15	15	5	5	1	5	5	15	15

Figure 3.8 Num. of Valid Path (Even)

an origin to a destination within a 9×9 window where the origin is in an odd column and an even column respectively. The number of valid paths grows as the distance between the origin and the destination increases. In an extreme case, the number of valid paths from router $(0, 0)$ to router $(29, 29)$ is $\frac{(44)!}{29!15!} = 2.3 \times 10^{11}$. To compare the costs of all 2.3×10^{11} paths, the route computation unit needs to be humongous and Route Computation stage needs to be extremely long. In fact, it is even infeasible for a router to have all of the link congestion information in the grid due to delay and energy consumption of the abnormally long wires. Therefore, the congestion observation window in each router is somehow limited.

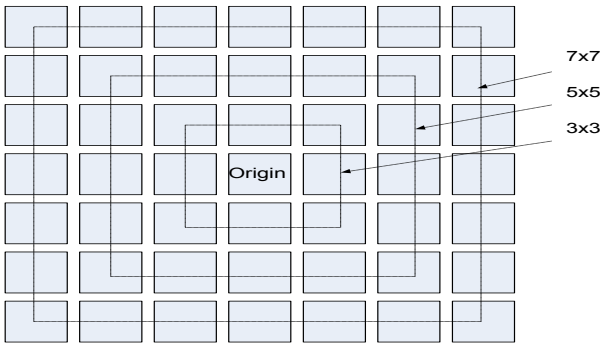


Figure 3.9 Observation Window

Even though a bigger observation window (Fig. 3.9) always leads to better routing result, its marginal gain diminishes quickly. The reason is that the actual congestion condition encountered by the packet at a distanced router could be different from the stale congestion in-

formation from an earlier time. The accuracy in estimating the congestion condition decreases as the distance between the link and the router increases.

In *Path*, a 5×5 observation window is used. Empirical studies show that a 5×5 observation window is one of the feasible observation window sizes which leads to significant network performance gain compared with non-adaptive routing. The details of the studies are shown in section 3.4.4.

In synthetic traffic:

1. A network with a 5×5 observation window has an average of 50% higher maximum throughput compared with a network with a 3×3 observation window.

2. A network with a 7×7 observation window has an average of 7% higher maximum throughput compared with a network with a 5×5 observation window.

3. A network with a 9×9 observation window has an average of 2.9% higher maximum throughput compared with a network with a 7×7 observation window.

In an on-chip video system:

1. A network with a 5×5 observation window is able to complete an average of 15.5% more co-processings than a network with a 3×3 observation window.

2. A network with a 7×7 observation window is able to complete an average of 2.26% more co-processings than a network with a 5×5 observation window.

3. A network with a 9×9 observation window is able to complete an average of 1.11% more co-processings than a network with a 7×7 observation window.

Another reason a 5×5 observation window is used is to prevent Route Computation stage from slowing down the router clock cycle. The Route Computation path increases as the observation window enlarges. The Route Computation path in a 5×5 observation window is just a little bit shorter than the Switch Allocation path which is the router's critical path.It allows *Path-based Adaptive Router* to run as fast as a basic unicast router.

When the observation window increases to a certain size, the wire delay could prohibit the downstream routers from sending congestion information to the current router in the same clock cycle. A 5×5 observation window does not have this problem.

As each router has a limited size of the congestion observation window, some destinations might be located outside. To handle these outside destinations, a virtual destination is introduced. Suppose a packet has a destination (x, y), the router assumes it has a a virtual destination (V_x, V_y). Each router has an observation window (x_l, y_l), (x_l, y_h), (x_h, y_l), (x_h, y_h) (Fig. 3.6),

$V_x(V_y) = x_l(y_l)$ if $x < x_l(y < y_l)$

$V_x(V_y) = x(y)$ if $x_l \leq x \leq x_h(y_l \leq y \leq y_h)$

$V_x(V_y) = x_h(y_h)$ if $x > x_h(y > y_h)$

Using this method, the packet keeps traversing until the actual destination is no longer outside of the window where a virtual destination is not needed.

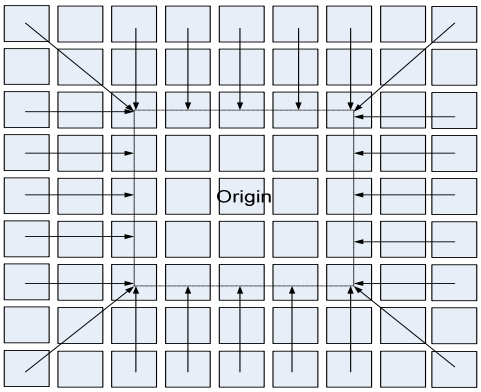


Figure 3.10 Virtual Destinations

The congestion information wires required by *Path* with 5×5 observation window is shown in Table 3.5. This table compares the number of congestion information wires needed by *Path*, XY-routing and *RCA (Quad)* using 9-bit precision with 50/50 local/remote congestion weighting.

Table 3.5 Congestion Status Wire Comparison

Routing Algorithm	# Congestion Wires (In/Out)	Overhead (64-bit/128-bit)
XY	0/0	0%/0%
RCA Quad	72/72	11.25%/5.625%
Path 5×5	(Even:155/Odd:160)/20	27.73%/13.86%

3.4 Experiments

Path-based Adaptive Routing algorithm is tested by both synthetic traffic [14] and video on-chip system built from many FPGAs with a 2D Mesh network. Details about the video on-chip system are given in Chapter 7.

3.4.1 Synthetic Traffic

The synthetic traffic patterns used are uniform traffic pattern, transpose traffic pattern and transpose2 traffic pattern. In uniform traffic pattern, each IP randomly selects a destination at the beginning of the simulation. In transpose traffic pattern, each IP (x,y) has a destination at IP ($Gsize-1-y$,$Gsize-1-x$) where $Gsize$ is the grid size of the 2D mesh network. In transpose2 traffic pattern, each IP (x,y) has a destination at IP (y,x).

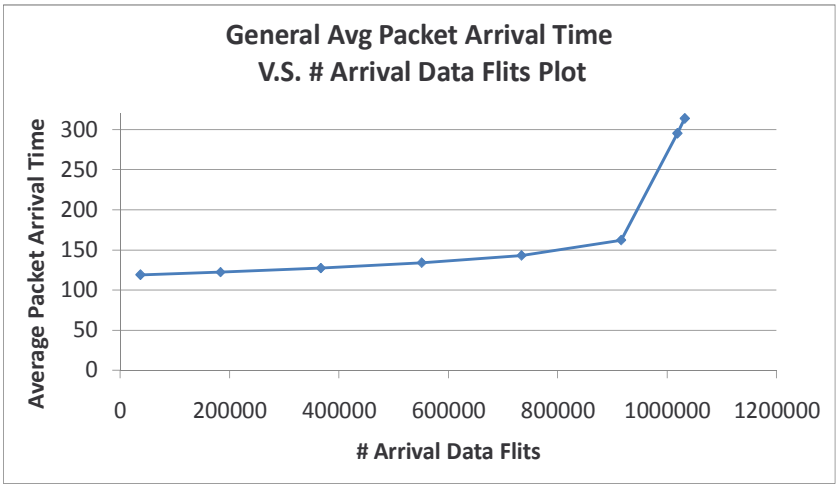


Figure 3.11 General Avg. Packet Arrival Time V.S. # flits arrival Plot

During simulation, each IP sends some packets to its destination at a random time per a certain time interval. Parameter S is used to represent the length of the time interval. Network congestion condition can be controlled by changing S. The average packet arrival time (packet latency) and the number of arrived packets during simulation were recorded. Fig. 3.11 shows a typical graph of average packet arrival time versus number of arrived packets. This graph is created from multiple simulations with different S values. When S decreases, each IP sends its packets more frequently. As a result, the number of arrived packets increases. The network also becomes more congested as S decreases. Thus, the average packet arrival time increases. There is a limit in the number of arrived packets. When it reaches the limit, the average packet arrival time is very high. At that point, the network saturates. The limit on the number of arrived packets would be the maximum throughput of the network.

3.4.2 Simulation Parameters

Simulation parameters are shown in Table 3.6.

Table 3.6 Baseline simulation parameters

Grid Size	20×20
Synthetic Traffic Simulation Cycles	$30,000$
Video System Simulation Cycles	$20,000,000$
Num. Virtual Channels on each port	3
Num. Samples	5
Flit Width	$128 - bit$
Packet Length	10 flits
Path-Based Adaptive Routing observation window	5×5
RCA Adaptive Routing Type	$9 - bit\ Quadrant$

3.4.3 Path-Based Adaptive Unicast Routing Experiments

In this section, the unicast router performances among *XY-routing*, *Path-Based Adaptive Routing*, *DyXY Adaptive Routing* and *Regional Congestion Aware Adaptive Routing* are compared. To compare the maximum throughput and average packet arrival time, network

simulations were performed with uniform, transpose and transpose2 traffic patterns. For each synthetic traffic patterns, four network simulations were performed using different routing algorithms. To compare the on-chip video system performance, four network simulations using different routing algorithms were performed.

3.4.3.1 Synthetic Traffic

The uniform traffic result is shown in Table 3.7. The transpose traffic result is shown in Table 3.8. The transpose2 traffic result is shown in Table 3.9.

Table 3.7 Unicast Maximum Throughput and Packet Arrival Time Comparison (Uniform Traffic)

Route Type	Packet Emission Rate (Flits/1000 Cycles)	#Data Flits Arrived	Avg. Packet Arrival Time
DYXY	4	38830	115.6102418
DYXY	20	192834	117.9263686
DYXY	40	384976	122.8249662
DYXY	60	579192	151.8655148
DYXY	80	583326	438.0845356
PATH	4	38830	115.6230998
PATH	20	192828	117.9965218
PATH	40	384942	122.8696196
PATH	60	578368	129.0098318
PATH	80	773556	138.0447192
PATH	100	963052	151.7619542
PATH	120	1005048	378.9351336
RCA	4	38832	115.8770748
RCA	20	192814	119.5187108
RCA	40	384956	126.5921642
RCA	60	578344	135.315809
RCA	80	704878	186.3254718
XY	4	38830	115.7923838
XY	20	192832	118.2180048
XY	40	384954	123.0518178
XY	60	578370	129.1828942
XY	80	771948	145.2329162
XY	100	911358	220.6122842
XY	120	918114	337.1739802

Fig. 3.12 compares the maximum throughput among the four routing algorithms. It shows

Table 3.8 Unicast Maximum Throughput and Packet Arrival Time Comparison (Transpose Traffic)

Route Type	Packet Emission Rate (Flits/1000 Cycles)	#Data Flits Arrived	Avg. Packet Arrival Time
DYXY	4	37040	119.1213526
DYXY	20	183352	122.7288478
DYXY	40	366686	132.1529418
DYXY	60	492770	356.809355
PATH	4	37036	119.0768918
PATH	20	183340	122.0896192
PATH	40	366706	127.2761332
PATH	60	551144	134.500288
PATH	80	733658	145.479022
PATH	100	914120	199.8641612
PATH	120	957992	287.6338648
RCA	4	37038	119.3035784
RCA	20	183356	124.3778648
RCA	40	366734	133.1998434
RCA	60	551084	147.5922776
RCA	80	706430	247.6608324
XY	4	37038	119.0432638
XY	20	183334	122.4009914
XY	40	366682	129.0420652
XY	60	551182	148.6774014
XY	80	667292	273.3474678

Table 3.9 Unicast Maximum Throughput and Packet Arrival Time Comparison (Transpose2 Traffic)

Route Type	Packet Emission Rate (Flits/1000 Cycles)	#Data Flits Arrived	Avg. Packet Arrival Time
DYXY	4	37076	118.5996482
DYXY	20	183404	121.7919792
DYXY	40	366720	130.0294158
DYXY	60	550556	158.408553
DYXY	80	698788	230.02403
DYXY	100	762268	321.4106814
PATH	4	37074	118.6837524
PATH	20	183406	121.8029222
PATH	40	366710	127.2714288
PATH	60	551210	134.776862
PATH	80	733740	145.7384672
PATH	100	915420	204.3813364
PATH	120	958946	289.7001492
RCA	4	37076	118.860271
RCA	20	183418	123.2147042
RCA	40	366694	130.4797152
RCA	60	551264	140.6497392
RCA	80	719738	201.2966292
XY	4	37076	118.7819766
XY	20	183406	122.2703378
XY	40	366702	129.1163658
XY	60	551350	147.2139512
XY	80	669072	271.1154458

Path-based Adaptive Routing has a better performance than other adaptive routing algorithms and XY-Routing algorithm. On average, *Path-based Adaptive Routing*'s maximum throughput is 58.9% higher than *DyXY-routing*, 37.1% higher than *RCA-routing* and 29.6% higher than *XY-routing*.

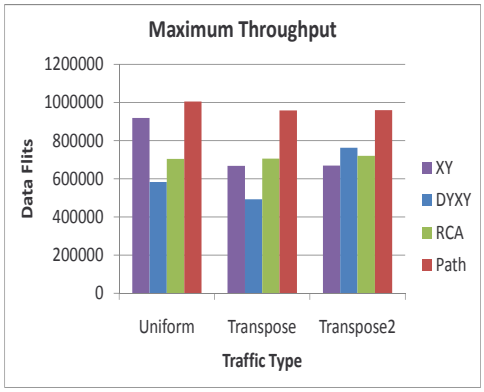


Figure 3.12 Unicast Maximum Throughput Comparison

Fig. 3.12 compares the average packet arrival time among the four routing algorithms. The data is recorded when the number of arrival flits is 579192 in uniform traffic, 366686 in transpose traffic and 550556 in transpose2 traffic. It shows *Path-based Adaptive Routing* has a better performance than other adaptive routing algorithms and XY-Routing algorithm. *Path-based Adaptive Routing*'s packet latency is 11.6% lower than *DyXY-routing*, 4.43% lower than *RCA-routing*, 3.55% lower than *XY-routing*.

3.4.3.2 Video System on FPGAs with 2D Mesh Network

The video system used in this section is explained in Chapter 7. A co-processing request queue for the video system is randomly generated. The configuration bitstream size of each CLB group is 2.2 Mbits which equals 18,000 128-bit packets. The runtime (RT) and configuration time (CT) are measured in cycles and the energy is measured in nJ. Configuration time is the elapsed time interval between the system issuing reconfiguration signals until all tile reconfiguration is complete. Runtime is the time interval from when all tile reconfiguration

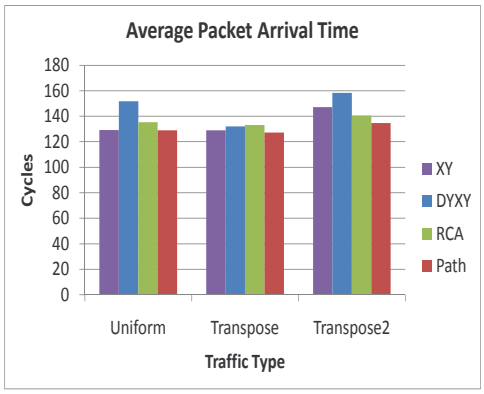


Figure 3.13 Unicast Average Packet Arrival Time Comparison

is complete to when all data is processed. Note that runtime excludes configuration time.

Table 3.10 and Fig. 3.14 compare the number of co-processor executions during simulation. System throughput of *Path-Based Adaptive Routing* is 13% higher than *XY-routing*, 16% higher than *DYXY-routing* and 16% higher than *RCA-routing*.

Table 3.10 Unicast Average Co-processing Runtime Comparison (Video System)

Route Type	Num. Co-processor Executions						
	M4EB	M4EV	M2E	M4DB	M4DV	M2D	Total
DYXY	309	304	350	374	362	292	1991
PATH	373	347	387	419	414	361	2301
RCA	305	309	353	366	355	289	1977
XY	322	315	354	370	355	316	2032

Table 3.11 and Fig. 3.15 compare the average co-processor runtime among the four routing algorithms. The average co-processor runtime of *Path-Based Adaptive Routing* is 10.4% lower than *XY-routing*, 10.7% lower than *DYXY-routing* and 9.7% lower than *RCA-routing*.

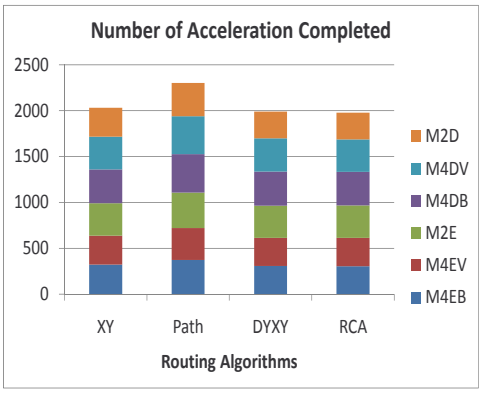


Figure 3.14 Unicast Number of Co-processor Executions Comparison

Table 3.11 Unicast Average Co-processor Runtime Comparison (Video System)

Route Type	Average Co-processor Runtime (Cycles)						
	M4EB	M4EV	M2E	M4DB	M4DV	M2D	Total
DYXY	1002624	1043602	821314	420215	415744	597043	4300544
PATH	869778	884098	715738	402521	407252	561336	3840726
RCA	982214	1005695	796662	452887	448847	568407	4254714
XY	950209	956262	795352	478222	497546	605913	4283507

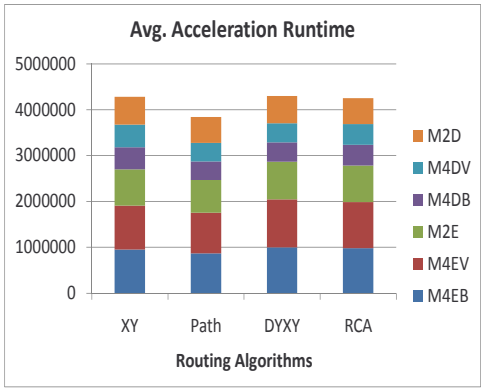


Figure 3.15 Unicast Average Co-processor Runtime Comparison

3.4.4 Path-based Adaptive Routing Observation Window Size

Congestion observation window of *Path-based Adaptive Routing* affects network performance. This section presents the effect of observation window size on network maximum throughput and packet arrival time.

3.4.4.1 Synthetic Traffic

The uniform traffic result is shown in Table 3.12. The transpose traffic result is shown in Table 3.13. The transpose2 traffic result is shown in Table 3.14.

Fig. 3.16 compares maximum throughputs of networks with different observation windows. Fig. 3.17 compares the average packet arrival times of networks with different observation windows. In Fig. 3.17, the packet arrival time is recorded when the number of arrived flits is 578338 in *uniform* traffic, 531654 in *transpose* traffic and 533178 in *transpose*2 traffic. Results show observation window extension can increase network maximum throughput. A network with a 5×5 observation window has an average of 50% higher maximum throughput compared with a network with a 3×3 observation window. A network with a 7×7 observation window has an average of 7% higher maximum throughput compared with a network with a 5×5 observation window. A network with a 9×9 observation window has an average of 2.9% higher maximum throughput compared with a network with a 7×7 observation window.

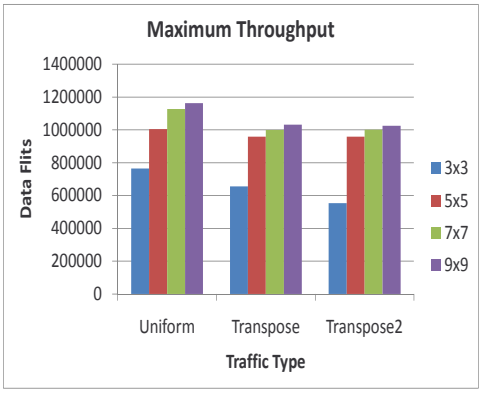


Figure 3.16 Range Test: Maximum Throughput Comparison

Table 3.12 Range Test: Maximum Throughput and Packet Arrival Time Comparison (Uniform Traffic)

Route Type	Packet Emission Rate (Flits/1000 Cycles)	#Data Flits Arrived	Avg. Packet Arrival Time
3x3	4	38830	115.8515914
3x3	20	192852	118.7588418
3x3	40	384916	124.877437
3x3	60	578338	136.0297974
3x3	80	765136	190.5885232
5x5	4	38830	115.6230998
5x5	20	192828	117.9965218
5x5	40	384942	122.8696196
5x5	60	578368	129.0098318
5x5	80	773556	138.0447192
5x5	100	963052	151.7619542
5x5	120	1005048	378.9351336
7x7	4	38830	115.6076324
7x7	20	192832	118.0267134
7x7	40	384952	122.8356854
7x7	60	578362	128.890311
7x7	80	773550	137.3567828
7x7	100	963032	148.1870066
7x7	120	1127302	261.9004008
9x9	4	38830	115.5833118
9x9	20	192838	117.8825922
9x9	40	384962	122.694256
9x9	60	578332	128.7235692
9x9	80	773604	136.917193
9x9	100	963064	146.8375398
9x9	120	1162988	188.8769778

Table 3.13 Range Test: Maximum Throughput and Packet Arrival Time Comparison (Transpose Traffic)

Route Type	Packet Emission Rate (Flits/1000 Cycles)	#Data Flits Arrived	Avg. Packet Arrival Time
3x3	4	37034	119.2555122
3x3	20	183368	123.41445
3x3	40	366720	137.0805984
3x3	60	531654	189.3443522
3x3	80	639554	267.1301784
3x3	90	655810	309.9049378
5x5	4	37036	119.0768918
5x5	20	183340	122.0896192
5x5	40	366706	127.2761332
5x5	60	551144	134.500288
5x5	80	733658	145.479022
5x5	100	914120	199.8641612
5x5	120	957992	287.6338648
7x7	4	37036	119.1206504
7x7	20	183332	122.2491402
7x7	40	366742	127.2877954
7x7	60	551092	134.052625
7x7	80	733668	143.687269
7x7	100	915332	179.0042892
7x7	120	999742	284.0542892
9x9	4	37038	119.0864214
9x9	20	183346	122.2038326
9x9	40	366726	127.2808882
9x9	60	551150	133.9389228
9x9	80	733666	142.9822276
9x9	100	915916	162.1787254
9x9	120	1018724	295.3489546
9x9	130	1032072	313.8605354

Table 3.14 Range Test: Maximum Throughput and Packet Arrival Time Comparison (Transpose2 Traffic)

Route Type	Packet Emission Rate (Flits/1000 Cycles)	#Data Flits Arrived	Avg. Packet Arrival Time
3x3	4	37076	118.8126392
3x3	20	183412	122.8528608
3x3	40	366728	136.7041812
3x3	60	533178	188.025601
3x3	80	554028	381.648575
5x5	4	37074	118.6837524
5x5	20	183406	121.8029222
5x5	40	366710	127.2714288
5x5	60	551210	134.776862
5x5	80	733740	145.7384672
5x5	100	915420	204.3813364
5x5	120	958946	289.7001492
7x7	4	37074	118.6933874
7x7	20	183430	121.9664144
7x7	40	366742	127.4098182
7x7	60	551230	134.417394
7x7	80	733710	144.0128382
7x7	100	915508	180.514023
7x7	120	1000578	283.7320486
9x9	4	37074	118.6504004
9x9	20	183426	121.93559
9x9	40	366728	127.418946
9x9	60	551254	134.289491
9x9	80	733678	143.3529208
9x9	100	915802	163.4646664
9x9	120	1018724	294.4350372
9x9	125	1025410	305.1481938

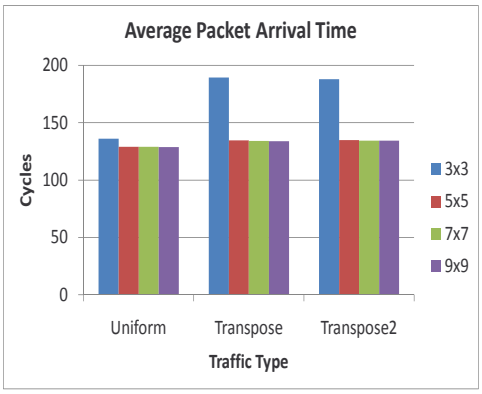


Figure 3.17 Range Test: Average Packet Arrival Time Comparison

3.4.4.2 Video System on FPGAs with 2D Mesh Network

The video system used in this section is explained in Chapter 7. A co-processing request queue for the video system is randomly generated. The configuration bitstream size of each CLB group is 2.2 Mbits which equals 18,000 128-bit packets. The runtime (RT) and configuration time (CT) are measured in cycles and the energy is measured in nJ. Configuration time is the elapsed time interval between the system issuing reconfiguration signals until all tile reconfiguration is complete. Runtime is the time interval from when all tile reconfiguration is complete to when all data is processed. Note that runtime excludes configuration time.

Table 3.15 and Fig. 3.18 compare the number of co-processor executions during the simulation. Results show that observation window extension can increase the network performance. A network with a 5×5 observation window is able to complete an average of 15.5% more co-processings than a network with a 3×3 observation window. A network with a 7×7 observation window is able to complete an average of 2.26% more co-processings than a network with a 5×5 observation window. A network with a 9×9 observation window is able to complete an average of 1.11% more co-processings than a network with a 7×7 observation window.

Table 3.15 Range Test: Average Co-processor Runtime Comparison (Video System)

Route Type	Num. Co-processor Executions						
	M4EB	M4EV	M2E	M4DB	M4DV	M2D	Total
3x3	311	306	348	365	357	304	1991
5x5	373	347	387	419	414	361	2301
7x7	380	355	406	428	426	358	2353
9x9	386	365	421	420	425	362	2379

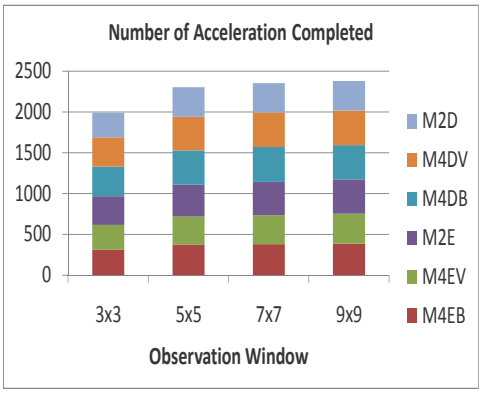


Figure 3.18 Range Test: Num. of Co-processor Executions Comparison

CHAPTER 4. Adaptive Multicast Router

4.1 Motivation

Beside unicast packet (one-to-one communication), multicast packet (one-to-many) is another common type of packets used in on-chip network. In chip-level microprocessor, memory coherence packet and data packet are multicast from one microprocessor's cache to some other microprocessors' caches sharing the same cache line. In an on-chip system, data is sent from one data producing IP to multiple data consuming IPs. In an on-chip system built from many FPGAs connected using 2D mesh network, a single configuration bitstream are multicast from the I/O to multiple FPGA tiles for high speed reconfiguration. Without native multicast support, the network interface has to copy the multicast packets several times and send each copy to its destination one by one. It is time consuming and inefficient. The goal of a native multicast support is to reduce network load and decrease multicast packet latency. In addition, a combination of multicast and adaptive routing allows routers to further reduce network load and packet latency by dynamic packet divergence point selection. One challenge in implementing a native multicast router is to keep the network free from deadlock. In this chapter, the adaptive unicast router from the last chapter is modified to support adaptive multicast routing. A novel address-data FIFO decoupling approach is proposed to keep the network free from multicast deadlock.

4.2 Adaptive Multicast Decoder

This section discusses the operation of multicast decoder in detail. To support multicast routing, a multicast decoder is added to the router. This multicast decoder can be shared by all virtual channels on all ports in the router. In a system with high multicast needs, each

virtual channel could have its own multicast decoder. Each multicast decoder has a multicast destination map to store packet destinations. Each slot in the map represents one tile on the chip.

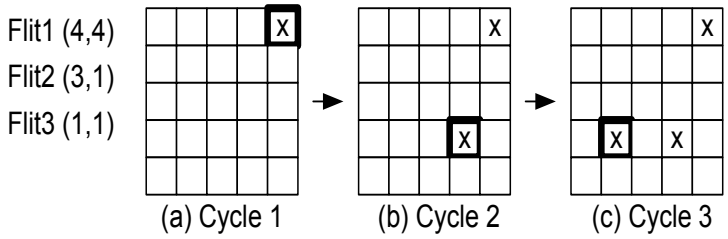


Figure 4.1 Multicast (Binary) Decoding

Fig. 4.1 shows an example of packet decoding with three addresses in a 5×5 2D mesh grid. The packet addresses are encoded in binary multicast code. Each *Head* flit has one address. The decoder reads the *Head* flits one by one and marks their destinations on the multicast map. Once the first *Body* flit has arrived, the multicast decoder starts making route decision.

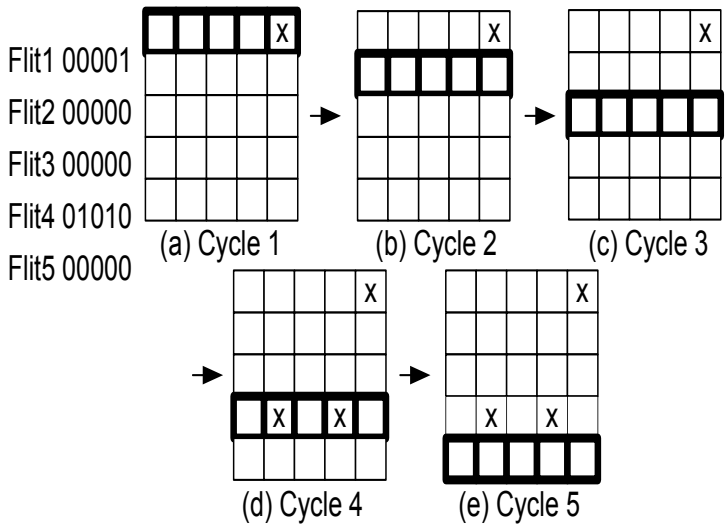


Figure 4.2 Multicast (Unary) Decoding

Fig. 4.2 shows an example of packet decoding with three addresses in a 5×5 2D mesh grid. The packet addresses are encoded in the unary multicast code. Each *Head* flit contains

one row of destinations. The decoder reads the *Head* flit and marks the destinations on the multicast map. It takes five cycles to process all the destinations. Once the first *Body* flit has arrived, the multicast decoder starts making route decision.

4.3 Dynamic Multicast Packet Divergence Point Selection

The collaboration of adaptive routing and multicast support allows dynamic divergence point selection. Compared with *XY multicast router* which has fixed packet divergence point, a router with dynamic divergence point selection has better network performance. Fig. 4.3 compares the packet multicast paths chosen by XY-Routing and adaptive routing. Tile (0,0) sends a multicast packet to tile (1,4), tile (2,4), tile (3,4), and tile (4,4). Packet route is predetermined using XY-routing. The packet diverges at routers (1,0), (2,0), (3,0) and (4,0) respectively. Adaptive multicast router can select the divergence points at routers (1,4), (2,4), (3,4) and (4,4). The network load created by the adaptive routing is 2.5 times lower than XY-routing. Adaptive multicast router reduces network load and energy consumed by buffer writes.

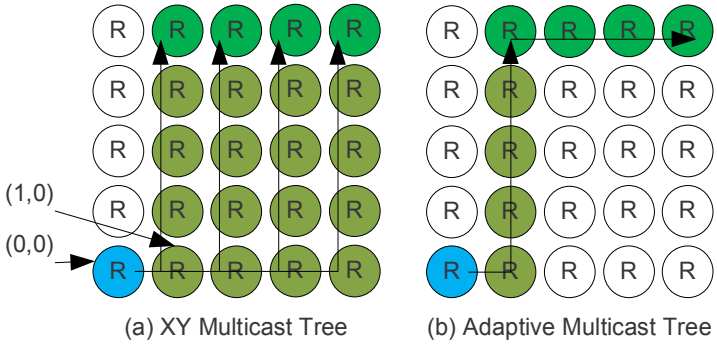


Figure 4.3 Multicast Comparison: (a) XY-Routing, (b) Adaptive Routing

4.4 Adaptive Multicast Route Decision

In adaptive multicast routing, the route computation unit decides where to diverge a packet based on the destinations. A simple and efficient method is needed to implement multicast

routing without increasing router's clock cycle. The main objective is to diverge a multicast packet as late as possible to reduce network load. First, the decoder identifies the regions holding the destinations. Then, it diverges the packet according to a lookup table. It takes less than a single cycle to complete these two steps.

4.4.1 Region Identification

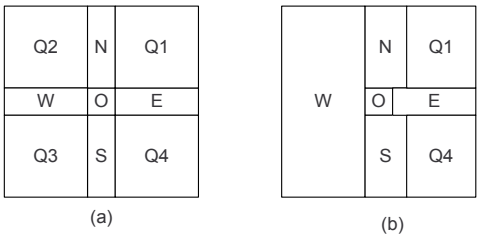


Figure 4.4 Region Identifying: (a) even column, (b) odd column

After marking the addresses on the multicast map, the router identifies which regions cover the multicast destinations. All destinations are mapped to either one of the five cardinal regions (the North, the East, the West, the South and Local) or one of the four quadrant regions (Q1, Q2, Q3 and Q4). If a destination is in the North, the East, the West, the South or Local regions, it is a cardinal destination. If a destination is in Q1, Q2, Q3 or Q4 regions, it is a quadrant destination. The region identifier creates a 9-bit vector to indicate which region is not empty. As *Odd-Even Turn Model* is applied to avoid deadlock in adaptive routing, the region identifying method needs to be modified accordingly. Fig. 4.4 (a) shows the region identifying method for the routers in odd column. Fig. 4.4 (b) shows the region identifying method for the routers in even column. Two observations should be noticed from the figures. The first one is that there are no Q2 and Q3 in the routers in odd column. A router in odd column with coordinate (x, y) considers all destinations with coordinate $(x - a, y \pm b)$ as destinations in the West region, where a, b are any positive integers. Because a packet going to the North or the South in a router in odd column cannot go to the West, all packets in routers in odd column cannot go to Q2 and Q3. The second observation is that the North and the South regions are

extended by one column to the East in the routers in odd column. A router in odd column with coordinate (x, y) considers all destinations with coordinates $(x, y + a)$,$(x + 1, y + a)$ as destinations in the North region and all destinations with coordinates $(x, y - a)$,$(x + 1, y - a)$ as destinations in the South region where a is any positive integer. Because a packet going to the East in a router in odd column cannot go to the North and the South in the next router, all packet which have destinations $(x+1, y+a)$ needs to go to the North and all packets which have destinations $(x + 1, y - a)$ needs to go to the South.

4.4.2 Route Lookup Table

After the region identifying process, the router decides the packet routing directions. The router makes route decision according to a lookup table. The pro is a quick decision can be made. The con is network load cannot be minimized because the distances between destinations to destinations and destinations to router are not being considered. As mentioned before, the lookup table is designed to diverge packets as late as possible. The table reads the 9-bit vector created in the region identifying process and creates a 5-bit (N, E, W, S, L) route decision output.

To achieve minimal routing, a packet with cardinal destinations can only be routed in one direction while a packet with quadrant destinations can be routed either in x-direction or y-direction (D_x, D_y).

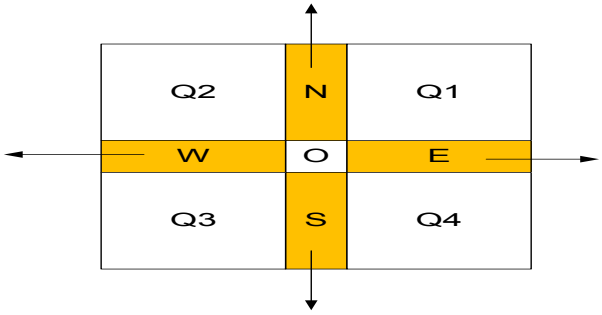


Figure 4.5 Rule 1 example

The lookup table is created using the following rules:

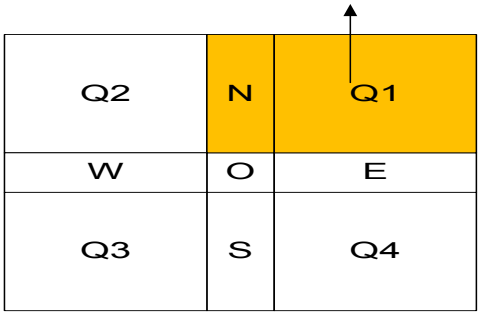


Figure 4.6 Rule 2 example

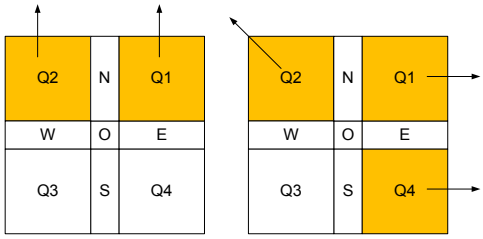


Figure 4.7 Rule 3 example

1. If a packet has cardinal destinations, it will be routed to the direction same as the region. (Fig. 4.5)

2. If a packet has quadrant destinations and another cardinal destination in the direction D_x or D_y, it will be routed to D_x or D_y where D_x has a higher priority than D_y. (Fig. 4.6)

3. If a packet has quadrant destinations but does not have any cardinal destinations in the direction D_x nor D_y, it will be routed according to a Quadrant Routing Table (Table 4.1) (Fig. 4.7).

The Quadrant Routing Table (Table 4.1) combines all unrouted quadrant destinations in two connected quadrants using their common direction D_x or D_y to route the destinations to D_x or D_y. For example, there is a packet with destinations in $Q1$ and $Q4$ which could not be routed using Rule 2 because the packet has no destinations in the North, the East and the South. The East is their common direction D_x. Therefore, destinations in $Q1$ and $Q4$ will be routed to the East.

Table 4.1 Quadrant Route Target Table

Unrouted Quad. Dest.				Route Targets				
Q1	Q2	Q3	Q4	N	E	W	S	(V_x, V_y)
0	0	0	0	0	0	0	0	NIL
0	0	0	1	0	0	0	0	$Q4 : (x_h, y_l)$
0	0	1	0	0	0	0	0	$Q3 : (x_l, y_l)$
0	0	1	1	0	0	0	1	NIL
0	1	0	0	0	0	0	0	$Q2 : (x_l, y_h)$
0	1	0	1	0	0	0	0	$Q2 : (x_l, y_h), Q4 : (x_h, y_l)$
0	1	1	0	0	0	1	0	NIL
0	1	1	1	0	0	1	0	$Q4 : (x_h, y_l)$
1	0	0	0	0	0	0	0	$Q1 : (x_h, y_h)$
1	0	0	1	0	1	0	0	NIL
1	0	1	0	0	0	0	0	$Q1 : (x_h, y_h), Q3 : (x_l, y_l)$
1	0	1	1	0	1	0	0	$Q3 : (x_l, y_l)$
1	1	0	0	1	0	0	0	NIL
1	1	0	1	0	1	0	0	$Q2 : (x_l, y_h)$
1	1	1	0	0	0	1	0	$Q1 : (x_h, y_h)$
1	1	1	1	0	1	1	0	NIL

Some quadrant destinations could not be combined with other quadrant destinations in a connected quadrant. The first reason is the absence of quadrant destinations in a connected quadrant For example, the packet only has destinations in $Q1$. The second reason is the quadrant destinations are isolated from others. For example, the packet has destinations at $Q1$, $Q2$ and $Q3$. Destinations in $Q2$ and $Q3$ are routed to the common D_x (the West). It leaves destinations in $Q1$ being isolated.

The packet with these isolated quadrant destinations will be routed adaptively as a unicast packet using a virtual address (V_x, V_y). (V_x, V_y) is the corner of the observation window in that quadrant. For example, a packet having all destinations in $Q1$ will be routed using virtual address (x_h, y_h). (x_h, y_h) is the upper-right corner in the congestion window shown in Fig. 3.6. Note that the routing decision is made in one cycle once the first *Body* flit has arrived.

When a packet diverges, two or more packets traverse to different directions. The *Head*

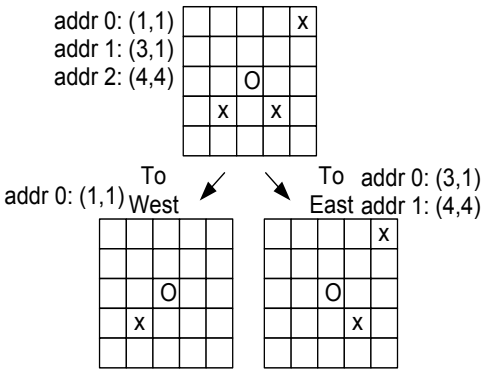


Figure 4.8 Packet Address Modification Example

flits in all diverged packets have to be modified to remove the destinations which belong to the other packets. This modification occurs when the *Head* flits leave their virtual channels. In binary-coded multicast routing, a flit modifier in the router invalidates the *Head* flits with addresses which do not belong to the traversing direction. Fig. 4.8 shows a flit invalidation example. In this example, a router has decided to route a packet with destinations in $Q1$ and $Q4$ to the East and destinations in $Q3$ to the West. When a *Head* flit leaves its virtual channel to traverse to the East, the flit modifier invalidates the flit if it contains a destination in $Q3$. *Head* flits with destinations in $Q1$ and $Q4$ will pass the flit modifier and continue traversing to the East. In unary-coded multicast routing, a flit modifier in the router modifies the *Head* flits using masks according to the routing decision which was stored in the virtual channel status table at Route Computation stage. The masks applied on packet address are the same as the region identifiers (Fig. 4.4).

4.5 Avoiding Multicast Deadlock By Address Data FIFO Decoupling

Multicast deadlock halts the packet flow and make the entire on-chip system unfunctionable. In section 2.5.2, a multicast deadlock example is given. Several methods to avoid multicast deadlocks have been proposed. Packet copying method [30] greatly increases the network load. Hamiltonian path partitioning method [30] does not apply to adaptive routing. Planar network

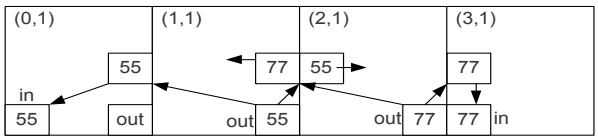


Figure 4.9 Multicast Deadlock Example

method [9] increases physical size of the router because it requires an extra sub-network.

4.5.0.1 Virtual cut-through routing

In fact, there is a simple method to avoid multicast deadlock without adding an extra sub-network or increasing the number of packet copies. It is to utilize *Virtual cut-through routing* instead of wormhole routing which is used by all of the methods mentioned above. Virtual cut-through routing is free from multicast deadlock because each virtual channel can store an entire packet. In the earlier multicast deadlock example (Fig. 4.9), the virtual channel on router (3,1)'s West port can now receive the entire Packet 77 from the virtual channel on the router (2,1)'s IP Out port. And the virtual channel on router (3,1)'s IP In port can now receive the entire Packet 77 from the virtual channel in router (3,1)'s West port. The virtual channel on router (3,1)'s West port will be released and hence, no multicast deadlock will be formed.

4.5.0.2 Address/Data FIFO Decoupling

Although *Virtual cut-through* can simply avoid multicast deadlock, it requires larger virtual channel buffer size. Due to limited on-chip space, virtual channel buffer size is usually small. Compared with *Virtual cut-through routing*, *Wormhole routing* is more preferable because it supports long packet using small virtual channel buffer. To keep both advantages of *Virtual cut-through* and *Wormhole* routing which are the deadlock free network and the small buffer size, we propose an address/data FIFO decoupling technique in *Wormhole routing*.

One observation from the multicast deadlock example above is that the virtual channel on router (3,1)'s West port is empty and it is waiting for a new flit indefinitely.

The packet in the virtual channel on router (2,1)'s IP *Out* port diverges to the virtual channels on router (1,1)'s East port and router (3,1)'s West port. The flits traversed to the virtual channel on router (3,1)'s West port successfully traverse to the virtual channel on its own router's IP *In* port. However, the flits traversed to the virtual channel on router (1,1)'s East port cannot further traverse due to the target virtual channel is occupied by Packet 55. All flits stored in the virtual channel on router (2,1)'s IP *Out* port have already traversed to the virtual channel on router (3,1)'s West port but a copy of the some of these flits stored in the virtual channel on router (2,1)'s IP *Out*'s port are still waiting to traverse to the virtual channel on router (1,1) East port. These flits staying in the virtual channel on router (2,1)'s IP Out port stop new flits from coming in and sending out. This left the virtual channel on router (3,1)'s West port empty and allocated to a packet that does not have new flits.

In fact, the deadlock is broken if Packet 77 releases the virtual channel on router (3,1)'s West port. However, it is not feasible to release the virtual channel in this situation. The *Head* flits of Packet 77 in the virtual channel on router (3,1)'s West port have already traversed to the virtual channel on router (3,1)'s IP *In* port. When a virtual channel is released, the routing information stored in the virtual channel status table is removed. If Packet 77 releases the virtual channel on router(3,1)'s West port, the *Body* flits of Packet 77 traversing to the virtual channel on router (3,1)'s West port after the deadlock is broken will not have any address information.

To avoid losing the address information, address/data FIFO decoupling in the virtual channel is applied (Fig. 4.10). Compared with the traditional single FIFO virtual channel which deletes the flit once the flit has traversed to the downstream router, address/data FIFO virtual channel stores all packet addresses until the virtual channel is released. It allows the router to break any packet into two or more packets when a potential deadlock is found.

The FIFO controller (Fig. 4.10) checks every incoming flit to see if it is a *Head* flit, a *Body* flit or a *Tail* flit. It will be stored in the address FIFO if it is a *Head* flit and in the data FIFO if it is not. When a packet traverses to the downstream router, the flits in address FIFO will leave before the flits in data FIFO.

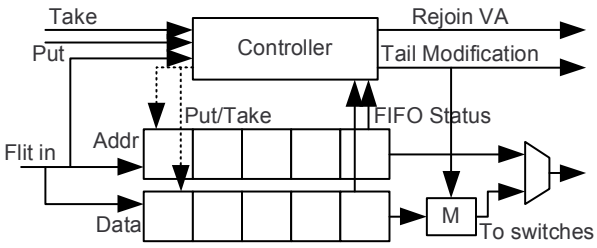


Figure 4.10 Decoupled Addr/Data FIFO

Multicast virtual channel's FIFO has 5 outputs and 5 heads. Each head selects the data going to the North, the East, the West, the South or Local. The virtual channel breaks a packet when the last flit coming out from the data FIFO to the direction D is not a *Tail* flit. It breaks the packet by changing the last flit from *Body* flit to *Tail* flit. At the same time, D head of the address FIFO is reset. The packet will release the router's D output port. On the other hand, a signal (*Tail Modification*) is sent to notify the virtual channel allocator that the coming *Tail* flit is modified from a *Body* flit to prevent the virtual channel from being released.

When a new flit arrives after the packet is broken, the packet rejoins the Virtual Channel Allocation (*VA*) stage to request a new virtual channel in the down-stream router. Because the routing decision is still available in the current router, Route Computation (*RC*) stage is skipped. When a packet has successfully obtained a virtual channel in the downstream router and an output port, the virtual channel sends the *Head* flits stored in the address FIFO followed by the newly arrived data flits to the downstream router. Note that the address FIFO is large enough to store all *Head* flits of a packet. Therefore, the virtual channel never changes any *Head* flits to *Tail* flit and it never sends any packet without data flits to the downstream router.

Fig. 4.11(step 1)-(step 9) show a packet breaking scenario based on the deadlock example given above (Fig. 4.9). In this example, Packet 77 (Fig. 4.12) is broken at the virtual channel on router (2,1)'s IP *Out* port. To avoid confusion, Packet 55 is not discussed in this scenario. The size of the address FIFO is 2 and the size of the data FIFO is 2. In Fig. 4.11(step 1)-(step 2), Packet 77 at virtual channel on router (2,1)'s IP *Out* port has already obtained the East

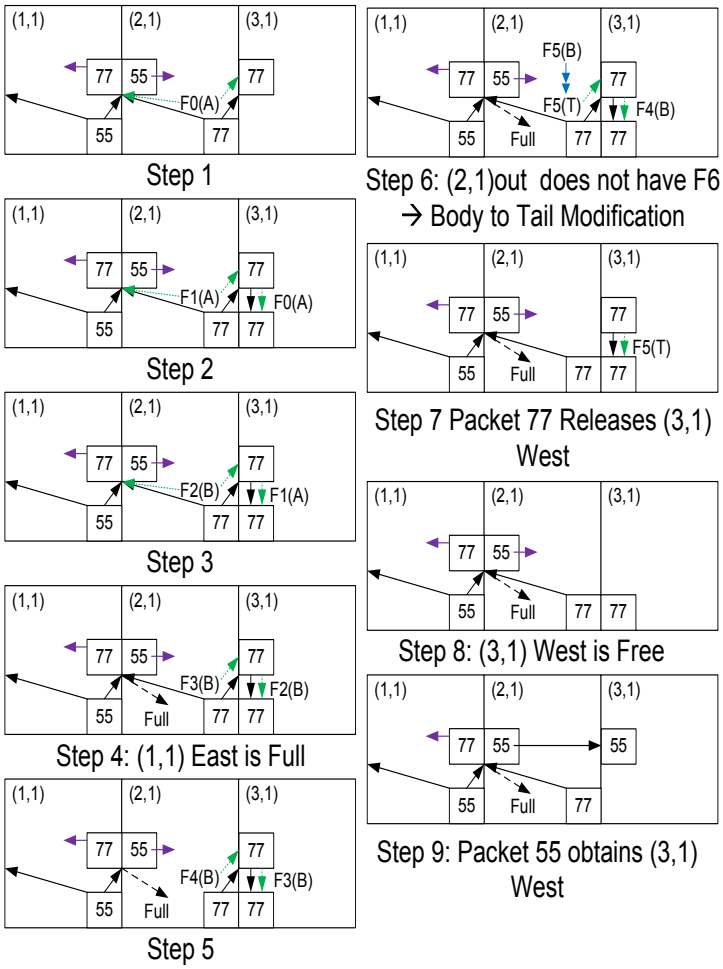


Figure 4.11 Packet 77 Break

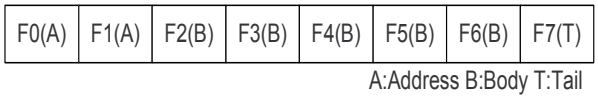


Figure 4.12 Packet 77 Sent by the IP at (2,1)

output port, the West output port, the virtual channel on router (1,1)'s East port and the virtual channel on router (3,1)'s West port. The virtual channel on router (2,1)'s IP *Out* port sends the flits *F0*, *F1*, *F2* and *F3* to the virtual channels on router (1,1)'s East port and on router (3,1)'s West port. The virtual channel on router (3,1)'s West port sends the flits to the IP at the location (3,1) accordingly. As Packet 77 cannot obtain the virtual channel on router (0,1)'s East port, the virtual channel on router (1,1)'s East port stores the flits traversed from the virtual channel on router (2,1)'s IP *Out* port. In Fig. 4.11(step 4), the virtual channel on router (1,1)'s East port sends a FULL signal to the virtual channel on router (2,1)'s IP *Out* port. Starting from Fig. 4.11(step 5), the virtual channel on router (2,1)'s IP *Out* port sends the flits *F4* and *F5* to the virtual channel on router (3,1)'s West port only and keeps the copies of flits *F4* and *F5* in its virtual channel. These two flits will traverse to the virtual channel on router (1,1)'s East port once that virtual channel is free. In Fig. 4.11(step 6), the virtual channel on router (2,1)'s IP *Out* port data FIFO sends its last flit *F5* to the virtual channel on router (3,1)'s West port. In this step, the virtual channel on router (2,1)'s IP *Out* port breaks the packet by modifying the flit *F5* from *Body* flit to *Tail* flit and send it to the virtual channel on router (3,1)'s West port. Packet 77 releases the East output port. In Fig. 4.11(step 7), the virtual channel on router (3,1)'s West port sends the flit *F5(T)* to the virtual channel on router (3,1)'s IP *In* port. As the flit *F5(T)* is a *Tail* flit, Packet 77 releases the virtual channel on router (3,1)'s West port when flit *F5(T)* leaves the virtual channel (Fig. 4.11(step 8). In (Fig. 4.11(step 9)), Packet 55 can finally obtain the virtual channel on router (3,1)'s West port and the deadlock is broken.

As the deadlock is broken, Packet 77 at the virtual channel on router (1,1)'s East port will be able to obtain the virtual channel on router (0,1)'s East port soon. The virtual channel on router (2,1)'s IP *Out* port then sends the flits *F4* and *F5* to the virtual channel on router (1,1)'s East port and accepts the flits *F6* and *F7* from the IP at location (2,1). Once the flit *F6* has arrived at the virtual channel on router (2,1)'s IP *Out* port, Packet 77 requests a virtual channel on router (3,1)'s West port at Virtual Channel Allocation (*VA*) stage. When Packet 77 has obtained the virtual channel on router (3,1)'s West port and the East output

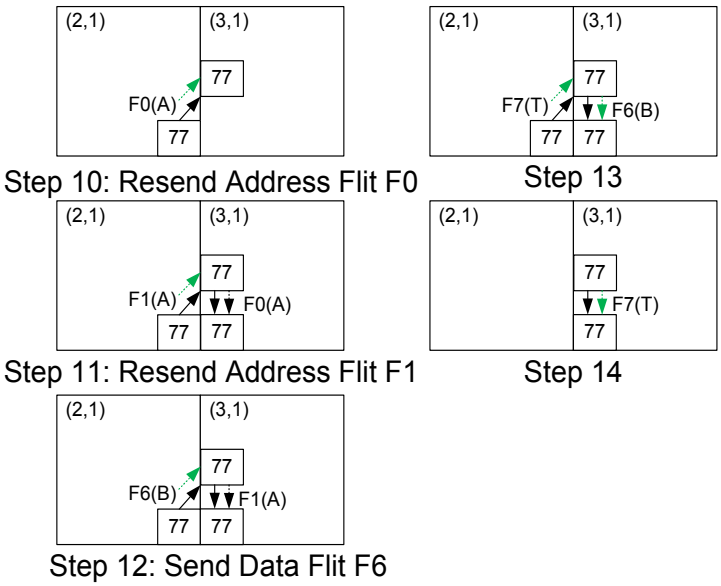


Figure 4.13 Packet 77 Second Part

F0(A)	F1(A)	F2(B)	F3(B)	F4(B)	F5(T)
F0(A)	F1(A)	F6(B)	F7(T)	A:Address B:Body T:Tail	

Figure 4.14 Packet 77 Received by the IP at (3,1)

port, the virtual channel on router (2,1)'s IP *Out* port sends Packet 77 to the virtual channel on router (3,1)'s West port by sending the *Head* flits *F0* and *F1*, followed by the data flits *F6* and *F7* (Fig. 4.13). Finally, IP at location (3,1) receives Packet 77 in two parts (Fig. 4.14).

4.6 The Micro-architecture Changes from the unicast router

There are two main changes in the adaptive multicast router from the adaptive unicast router.

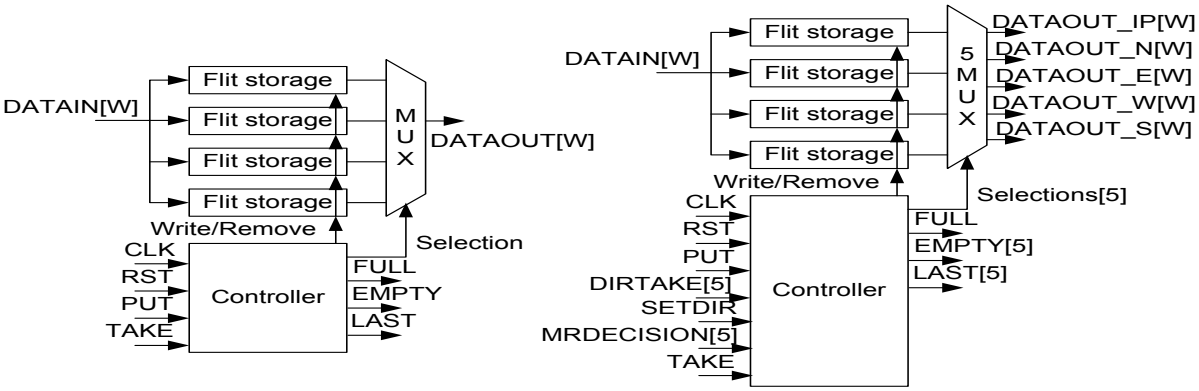


Figure 4.15 (a) 1-output FIFO (b) 5-output FIFO

4.6.1 Five output FIFO

Compared with unicast packet which makes only one output port request, multicast packet can make up to five output port requests. A packet may get all the requested output ports at different cycles. Therefore, a flit may go to one direction first and then go to other directions in later cycles. To reduce router physical size, a 5-head, 5-output FIFO instead of five 1-head, 1-output FIFOs is used in the virtual channel. Each head selects a flit to going to one routing direction. Fig. 4.15 shows the micro-architectures of a 1-output FIFO and a 5-output FIFO. Both FIFOs have FIFO depth of 4.

When a new packet has arrived and route computation is not complete yet, the FIFO assumes the data will go to all five directions. After Route Computation *(RC)* stage, the route

computation unit flags the *SETDIR* signal and uses the *MRDECISION* signal to reset the FIFO heads of the directions that the packet will not traverse to. The switch allocator uses the *DIRTAKE* signal to select a flit from the virtual channel to traverse to the downstream router in a specific direction.

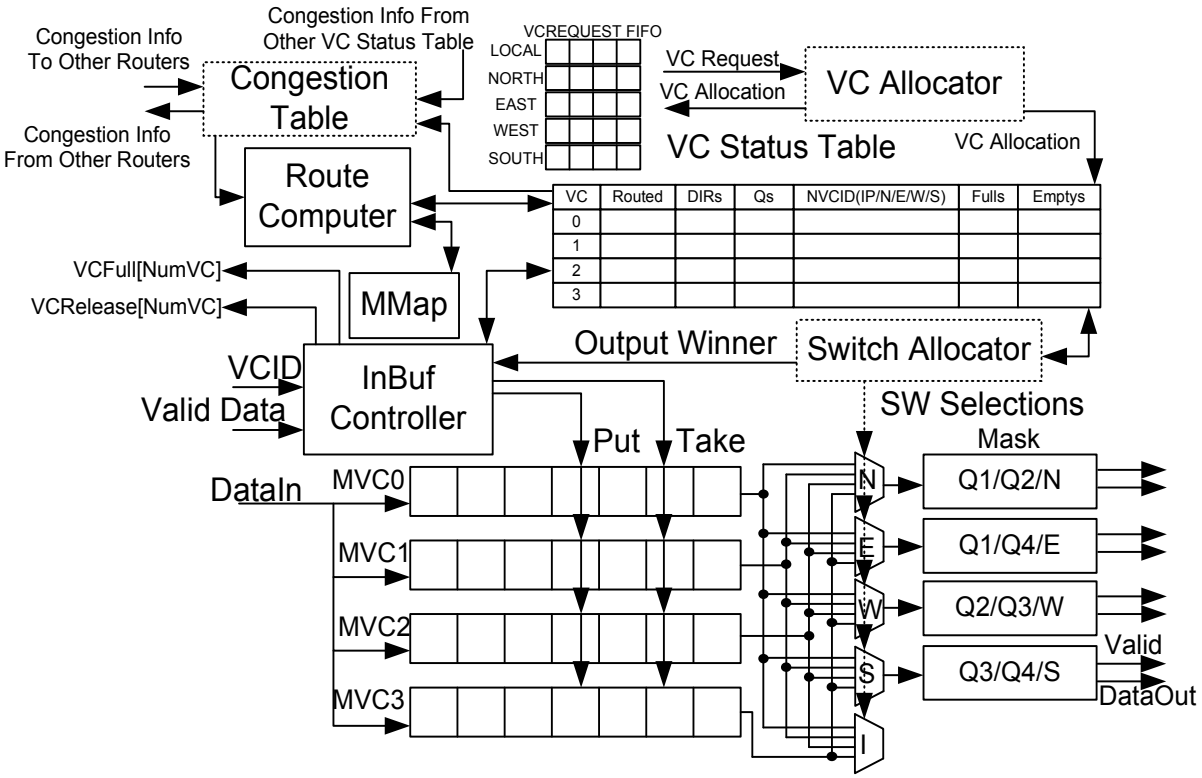


Figure 4.16 Adaptive Multicast Input Buffer

4.6.2 Multicast Adaptive Input Buffer

There are several changes in the input buffer of the adaptive multicast router. The route computation unit is connected to a multicast map (MMAP). The unicast virtual channels are changed to multicast virtual channels. The virtual channel status table is extended to contain all *NVCIDs* in the downstream routers and all full and empty signals of the 5-head FIFOs. The newly added 9-bit region identifying vectors and route decision are now recorded in the virtual channel status table. The route decisions are used to control the masks for modifying

Head flits when the flits leave the virtual channel.

4.6.3 Experiments

The adaptive multicast router experiment's details will be given in Chapter 6.

CHAPTER 5. Dual-coded Multicast Router with Dynamic Code Translation

Multicast packet address can be coded in different ways [8]. Unary code has fewer number of address flits when the number of destinations is high. Binary code has fewer number of address flits when the number of destinations is low. On-chip router prefers a low number of address flits because it reduces the packet latency. In order to integrate the advantages of both of these coding methods, a dual-coded multicast unit which accepts packets coded in either way is proposed. In addition, a packet using unary code could be trans-coded into binary code on-the-fly when the number of destinations drops below a threshold level. Multicast code could be translated into unicast code on-the-fly when the number of destinations drops to 1. These techniques could reduce packet latency and network load.

5.1 Motivation

Native multicast support can reduce the number of packet copies in an on-chip network. In an on-chip packet-switched network, each packet contains some address (*Head*) flits and data flits (Fig. 5.2). The last data flit is *Tail* flit. The address flits establish the route for all of the following flits. They contain the addresses of the packet destinations. On-chip destination address coding methods affect the number of flits for each packet. It, in turn, directly affects packet latency and energy consumption. Fig. 5.1 shows the number of start-up flits for a 9-data-flit packet with different number of destinations using different address encodings. The 2D mesh grid is 20×20. In a system which only supports unicast, the number of start-up flits is directly proportional to the number of destinations. When a packet is sent to D destinations, D copies of the packet are needed because each packet can only have one address. Each packet

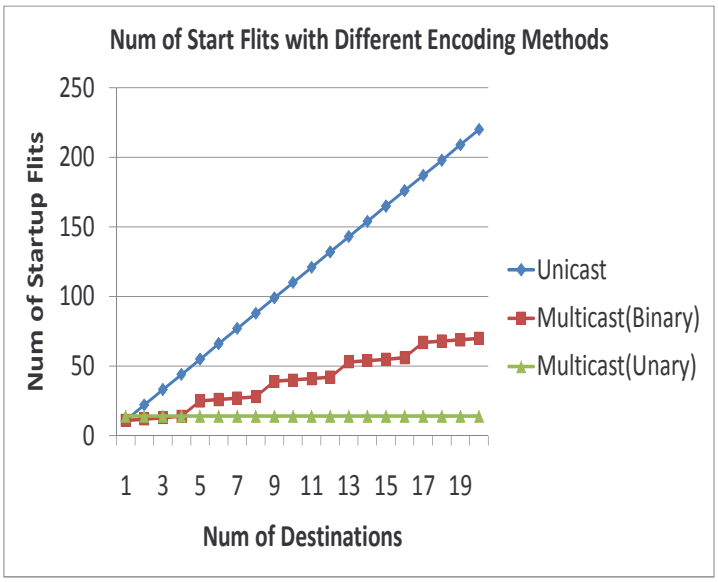


Figure 5.1 Number of Startup Flits

is $1 + L$ flits long where L is the number of data flits. Therefore, the total number of start-up flits for this packet is $D \times (1 + L)$. As each packet only has one address, the router can make routing decision once the address flit has arrived.

To reduce the number of packet copies, multicast routing can be used. Multicast destination addresses are commonly coded in two ways, one-hot unary code or binary code. A system supporting binary multicast code allows a packet to have a certain number of destination addresses M in each packet. Each address flit contains one address. In this system, when an IP needs to send a packet to D destinations, the number of packet copies is reduced to $\lceil D/M \rceil$ and each packet is $M + L$ flits long. Although the route computation starts once the first address flit has arrived at the router, the route computation unit needs to wait until the first data flit has arrived to make the routing decision. Fig. 5.1's red line shows the number of start-up flits for a multicast packet using binary code for address encoding when M=4. Note that the number of start-up flits is reduced significantly compared with the system without multicast support.

In a system supporting unary multicast code, a packet can have multiple destination ad-

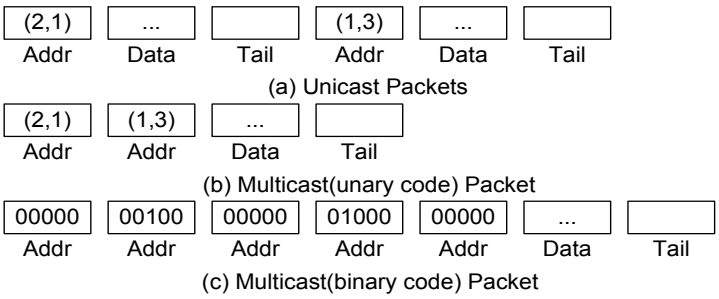


Figure 5.2 Packet Structure

Table 5.1 Packet Address Coding Scheme Comparison

Type	# MPacket	# Dest	Packet Length	Route Decision
Unicast	Low	Low	$D \times (1 + L)$	Addr. Flit Arrival
Multicast(Binary)	High	Low	$\lceil D/M \rceil (M + L)$	First Data Flit Arrival
Multicast(Unary)	High	High	$\lceil N/B \rceil + L$	Last Addr. Flit Arrival

dresses in one flit. If the number of IPs in the 2D mesh grid is N and each address flit uses a B-bit field for destination addressing, the number of start-up flits is $\lceil N/B \rceil + L$. Fig. 5.1's green line shows the number of start-up flits for a multicast packet using unary code for address encoding. The number of start-up flits is a constant. There are no extra packet copies at all. The number of start-up flits is the lowest when the number of destinations is higher than five.

Each addressing method has its own benefits which are summarized in Table 5.1. Unicast is simple and requires minimum hardware support. It is good for a system with low number of multicast packets with a few packet destinations. Binary multicast code is suitable for a system with high number of multicast packets with a few packet destinations. Unary multicast code ensures that only a single packet copy is needed. It is good for a system with high number of multicast packets with a lot of packet destinations. When a system has packets with mixed characteristics, any coding method described above alone cannot provide an optimal solution. To ensure packets can be routed efficiently, it is essential to have a router supporting all three coding methods unicast, unary multicast and binary multicast.

5.2 Multicast Code Transformation

The number of destination addresses of a packet decreases at packet divergence as the packet travels through an on-chip network. It means that the advantages of unary multicast code diminish hop by hop. When the number of addresses decreases to a level, binary multicast code becomes a better coding method instead. For example, a router decides to diverge a 10-address packet with 5 data flits into the North and the South in a 30×30 2D mesh network. Each packet can have 100 addresses using unary multicast coding. The packet to the North has 8 addresses and the packet to the South has 2 addresses. If the packet to the South keeps using unary multicast code, the packet network load remains $\lceil N/B \rceil + L$ which is 14. It requires 9 address flits. However, if the packet address coding is changed from unary multicast code to binary multicast code, the network load decreases to $2 + L$ which is 7 and the number of address flits decreases to 2.

The address coding method of the packet going to the South is changed to binary multicast code and the packet continues to traverse to its destinations. Later, the packet is diverged to the East and the South. Each diverged packet contains a single address. In this case, regardless of whether the packet changes from binary multicast coding to unicast coding or not, network load stays at $1 + L$ which is 6. Although there is no difference in network load in this coding change, it still benefits the network by reducing the multicast virtual channel competition.

As multicast virtual channel has high hardware cost, the network which has low to moderate level of multicast needs may consider having just one multicast virtual channel and several unicast virtual channels in each router input port. When the input port has only one multicast virtual channel, there is a competition among the multicast packets. Multicast packets have to wait if the multicast virtual channel is in use. It leads to an increase in packet latency. By transforming a binary multicast coded packet to unicast coded packet, the number of multicast virtual channel competitors is reduced and the network performance is improved.

The unary multicast code to binary multicast code transformation unit (Fig. 5.3) contains a T-map, a transformer and a 5-head FIFO. The FIFO depth is equal to the unary multicast to binary multicast transformation threshold (TT) which is equal to $\lceil N/B \rceil$. The T-map stores

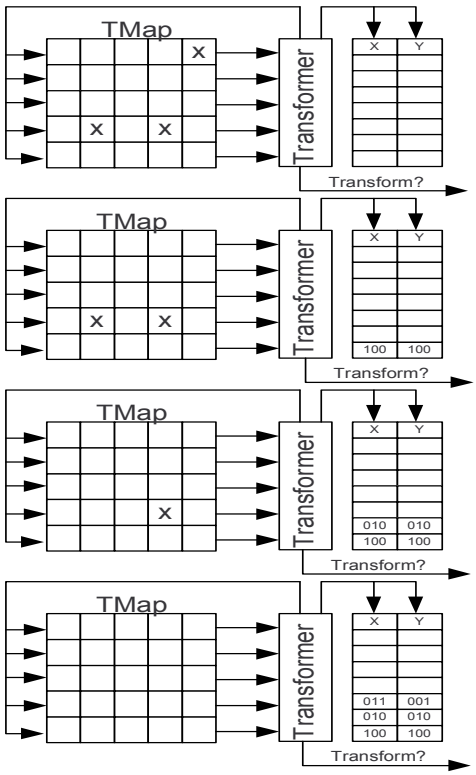


Figure 5.3 Unary Multicast Code to Binary Multicast Code Transformation

all destinations of the packet. Each valid entry in the T-map requests an address transformation. The transformer translates one address in the T-map into binary code per cycle. The transformer puts the translated binary code into the FIFO. The translated destination address in the T-Map will be removed.

If the number of destination addresses exceeds the transformation threshold (TT), the FIFO will overflow and the transformation process will be terminated. In that case, the packet continues using unary multicast code.

If the number of destination addresses is lower or equal to the transformation threshold (TT), the entries in the FIFO will replace the address flits leaving the virtual channel. If the FIFO is empty when the address flit leaves the virtual channel (It occurs when the number of destination addresses is lower than TT), that address flit will be invalidated.

The transformation starts once the first address flit has arrived at the virtual channel. In the best case, transformation adds no extra routing cycle to the system. In the worst case, the number of extra routing cycles is TT. The worst case occurs when the last unary multicast address flit has more than TT addresses and other unary multicast address flits have no addresses at all.

CHAPTER 6. Multicast Experiments and Hardware Implementations

6.1 Experimental Setup

The multicast function was tested by both synthetic traffic [14] and video on-chip system built from many FPGAs with a 2D Mesh network in Chapter 7.

6.1.1 Synthetic Traffic

The synthetic traffic patterns used are uniform traffic pattern, transpose traffic pattern and transpose2 traffic pattern. In uniform traffic pattern, each IP randomly selects a destination at the beginning of the simulation. In transpose traffic pattern, each IP (x,y) has a destination at IP ($Gsize-1-y$,$Gsize-1-x$) where $Gsize$ is the grid size of the 2D mesh network. In transpose2 traffic pattern, each IP (x,y) has a destination at IP (y,x). 5% of the unicast packets are converted to multicast packets. Every IP is randomly assigned to a multicast group of 10. Each IP sends 5 multicast packets to all members within its multicast group and 95 unicast packets to its destination node in every 100 packet.

6.1.2 Simulation Parameters

Simulation parameters are shown in Table 6.1.

6.2 Multicast Routers Experiment

6.2.1 Synthetic Traffics

In this section, the proposed routers are tested with different multicast supports using different synthetic traffic patterns. The focus is on the network maximum throughput, the

Table 6.1 Baseline simulation parameters

Grid Size	20×20
Synthetic Traffic Simulation Cycles	$30,000$
Video System Simulation Cycles	$20,000,000$
Num. Unicast Virtual Channels on each port	1
Num. Multicast Virtual Channels on each port	2
Num. Samples	5
Flit Width	$128 - bit$
Num Data Flits per Packet	10 flits
Path-Based Adaptive Routing observation window	5×5

average packet latency and the network energy consumption. The five routers tested are:

1. Unicast Router (*UNI*).
2. Binary Multicast Router without code transformation (*BIN(NT)*).
3. Binary Multicast Router with code transformation (*BIN(T)*).
4. Unary Multicast Router without code transformation (*UNA(NT)*).
5. Unary and Binary Multicast Router with code transformation (*BOTH(T)*).

Unicast router (*UNI*) has three unicast virtual channels on each input port. Multicast routers have two unicast virtual channels and one multicast virtual channel on each input port. A multicast virtual channel can handle both unicast and multicast packets. Binary Multicast Router with code transformation (*BIN(T)*) can transform binary multicast packet to unicast packet. Unary and Binary Multicast Router with code transformation (*BOTH(T)*) can transform unary multicast packet to binary multicast packet and transform binary multicast packet to unicast packet.

6.2.2 Comparison between Unicast Router and Multicast Routers

In this section, the five unicast and multicast routers mentioned above are tested using *Virtual Cut-through* routing. The length of each virtual channel is 14.

6.2.2.1 Throughput

The throughput comparison of the five routers is shown in Table 6.2, Fig. 6.1, Fig. 6.2 and Fig. 6.3.

Table 6.2 Multicast Router Maximum Throughput

Traffic	Uniform		Transpose		Transpose2	
Routing	XY	Path	XY	Path	XY	Path
NIL	1176782	1409556	833244	1267076	831616	1266188
BIN(NT)	1115466	1051240	841404	960674	840416	978882
BIN(T)	1179860	1437002	840844	1200954	839960	1126668
UNA(NT)	1079354	1009786	841116	936434	851066	952108
BOTH(T)	1182006	1438484	848132	1206106	853240	1158750

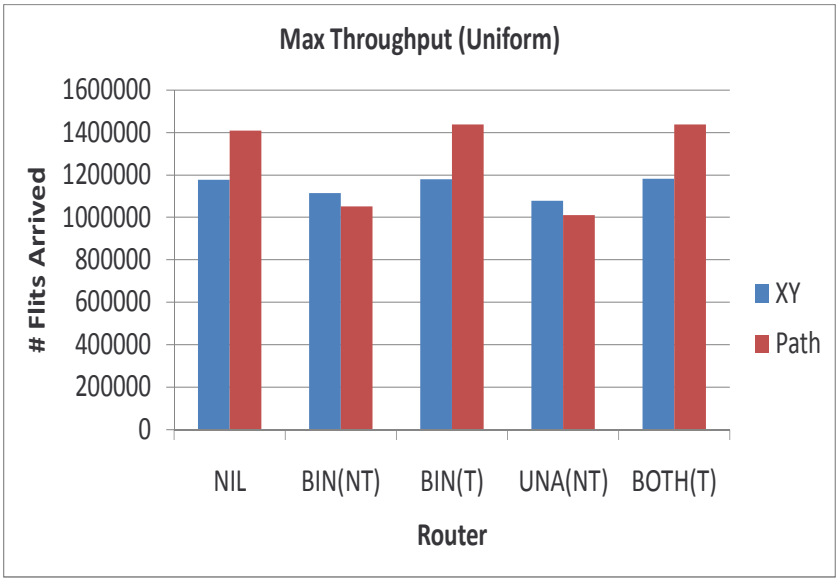


Figure 6.1 Multicast Router Maximum Throughput (Uniform Traffic)

The figures show that the unicast router has higher maximum throughput than the multicast routers. It is because a multicast router has only one multicast virtual channel in each input port. When the network is highly congested, many multicast packets block the output of the IPs. If the code transformation function is used, this blocking problem can be lessened.

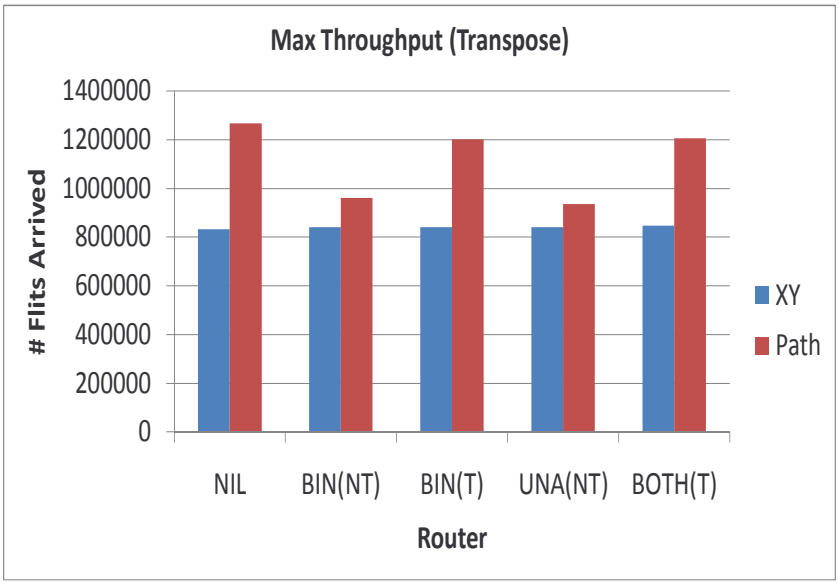


Figure 6.2 Multicast Router Maximum Throughput (Transpose Traffic)

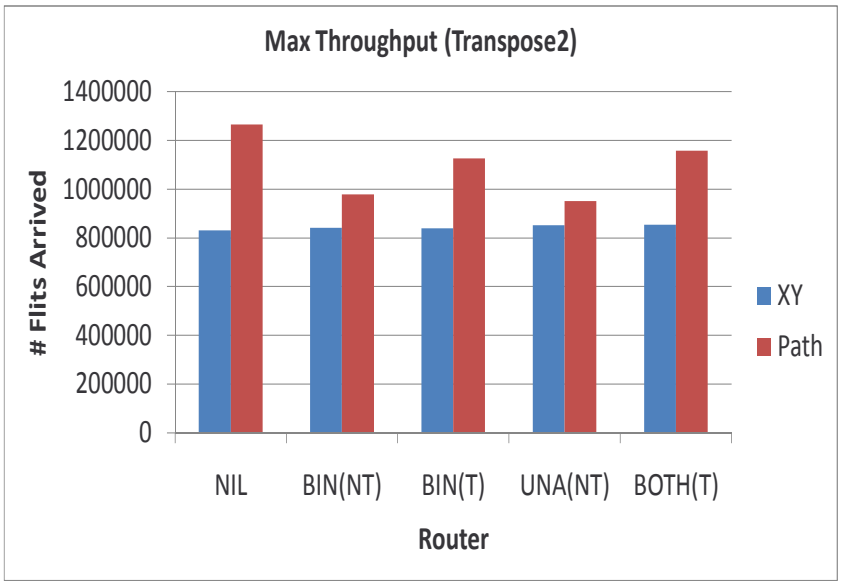


Figure 6.3 Multicast Router Maximum Throughput (Transpose2 Traffic)

Routers with code transformation has higher maximum throughput than those without.

Adaptive routers have higher maximum throughputs than XY routers in most of the simulations. The adaptive multicast routers with code transforming and the adaptive unicast router have higher maximum throughputs than their counterparts using *XY-Routing* in all synthetic traffics. The adaptive multicast routers without code transforming have higher maximum throughputs than their counterparts using *XY-Routing* in *transpose* and *transpose2* traffic patterns. However, the adaptive multicast routers without code transforming have lower maximum throughputs than its counterpart using XY multicast router in *uniform* traffic pattern. It is because the maximum throughput is affected by the packet traffic pattern created by the IPs. When the network is highly congested, *uniform* traffic pattern prefers a more regular packet flow which can be created by the multicast routers without code transformation using *XY-Routing*. Less on-chip packets would be blocked in this packet flow.

6.2.2.2 Packet Latency

The packet latency comparison of the five routers is shown in Table 6.3, Fig. 6.4, Fig. 6.5 and Fig. 6.6. The data is recorded when the number of arrival flits is 100,000.

Table 6.3 Multicast Router Packet Latency

Traffic	Uniform		Transpose		Transpose2	
Routing	XY	Path	XY	Path	XY	Path
NIL	111.75	111.73	114.35	114.35	114.80	114.80
BIN(NT)	103.65	104.16	107.00	107.61	107.27	107.85
BIN(T)	103.38	104.04	106.78	107.51	107.06	107.76
UNA(NT)	103.94	104.01	107.13	107.20	107.57	107.59
BOTH(T)	100.07	100.87	103.54	104.38	103.93	104.69

The figures show that all multicast routers have lower packet latencies than the unicast router. The adaptive multicast router supporting both binary and unary codes with the code transformation function have the lowest packet latency. Adaptive multicast routers have higher packet latencies than XY-routing multicast routers because of the extra cycles for address

decoding.

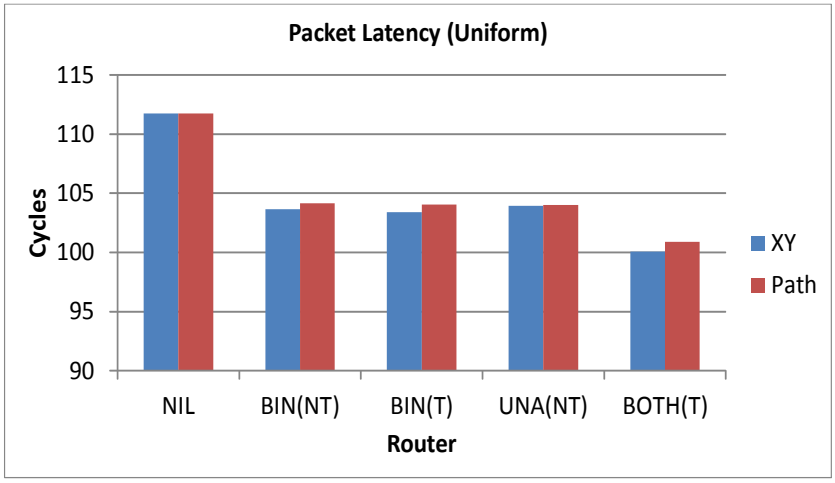


Figure 6.4 Multicast Router Packet Latency (Uniform Traffic)

6.2.2.3 Energy Consumption

The energy consumption comparison of the five routers are shown in Table 6.4, Fig. 6.7, Fig. 6.8 and Fig. 6.9. The data is recorded when the number of arrival flits is 200,000.

Table 6.4 Multicast Router Flits Energy Consumption (pJ)

Traffic	Uniform		Transpose		Transpose2	
Routing	XY	Path	XY	Path	XY	Path
NIL	405.743	410.227	422.234	426.898	422.129	426.812
BIN(NT)	396.926	392.705	413.289	409.323	413.311	409.335
BIN(T)	386.770	383.675	403.184	400.286	403.188	400.270
UNA(NT)	392.761	384.230	408.736	400.707	408.743	400.879
BOTH(T)	369.946	366.982	386.174	383.567	386.150	383.667

In the unicast router, adaptive routing consumes more energy per flit than XY-routing because of the extra logic component. The unicast router consumes more energy per flit than the multicast routers because of the extra packet copies. Code transformation reduces energy consumption by reducing the number of address flits. Adaptive multicast router consumes less

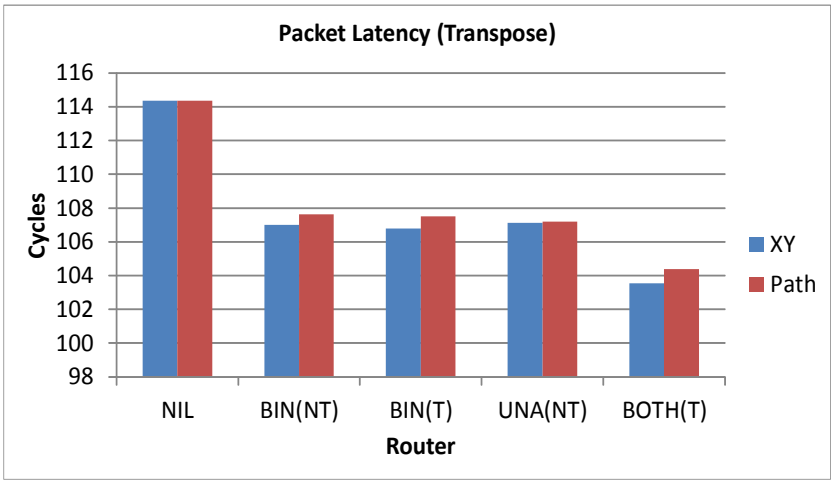


Figure 6.5 Multicast Router Packet Latency (Transpose Traffic)

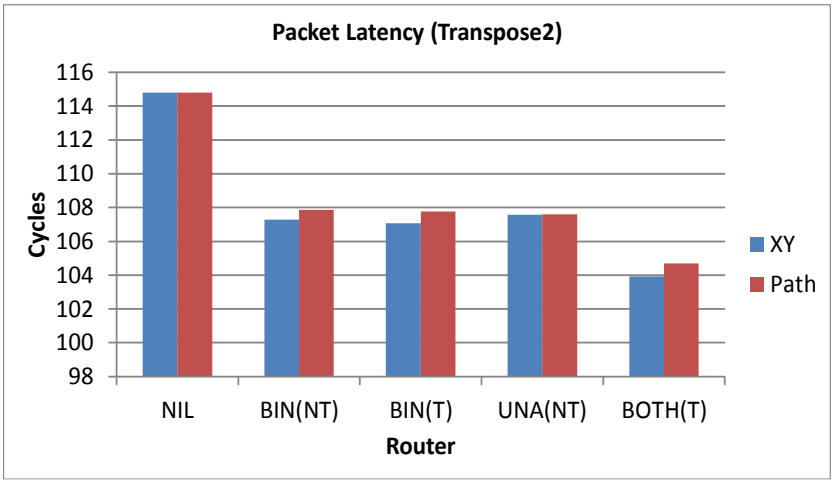


Figure 6.6 Multicast Router Packet Latency (Transpose2 Traffic)

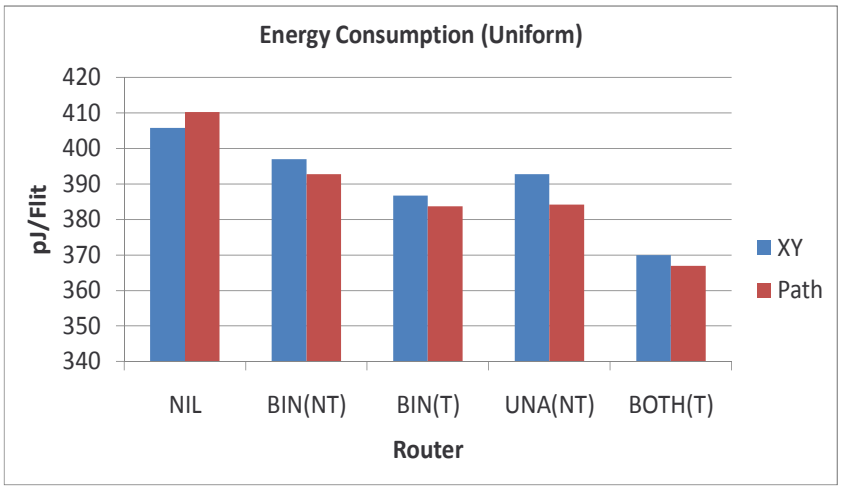


Figure 6.7 Multicast Router Flits Energy Consumption (Uniform Traffic)

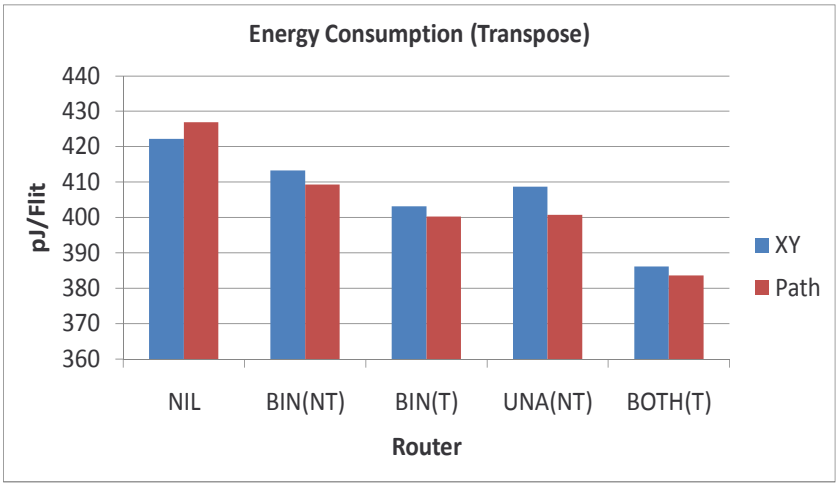


Figure 6.8 Multicast Router Flits Energy Consumption (Transpose Traffic)

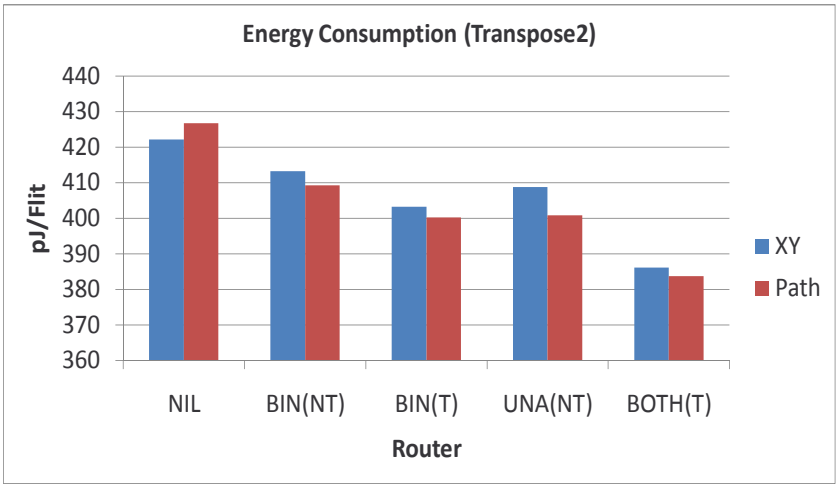


Figure 6.9 Multicast Router Flits Energy Consumption (Transpose2 Traffic)

energy than XY-routing multicast router because the dynamic packet divergent point selection reduces flit copies. The adaptive multicast router supporting both binary and unary codes with the code transformation function has the lowest energy consumption.

6.2.3 Video System on FPGAs with 2D Mesh Network

In this section, the video on-chip system performances using the five routers are compared. The video system used in this experiment is discussed in Chapter 7. The main uses of multicast function in the video system are:

1. sending a configuration bitstream from one FPGA tile or I/O to many other FPGA tiles for high speed parallel FPGA reconfigurations and

2. routing data from one producer module to many consumer modules.

A co-processing request queue for the video system is randomly generated. The configuration bitstream size of each CLB group is 2.2 Mbits which equals 18,000 128-bit packets. The runtime (RT) and configuration time (CT) are measured in cycles and the energy is measured in nJ. Configuration time is the elapsed time interval between the system issuing reconfigura-

tion signals until all tile reconfiguration is complete. Runtime is the time interval from when all tile reconfiguration is complete to when all data is processed. Note that runtime excludes configuration time.

The average co-processor runtime, the average FPGA configuration time and the average FPGA configuration energy are shown in Table 6.5, Fig. 6.10, Fig. 6.11 and Fig. 6.12.

Table 6.5 Multicast Routers Comparison (Video on-chip system using FPGA)

Router	Avg. Run Cycles	Avg Conf. Cycles	Avg. Conf. Energy (pJ)
xy_NIL	522810	78624	3180393
xy_BIN(NT)	511710	64604	2858594
xy_BIN(T)	501242	49574	2714574
xy_UNA(NT)	544389	74168	3073291
xy_BOTH(T)	502952	47715	2404750
p_adap_NIL	496585	71792	3110929
p_adap_BIN(NT)	498072	64417	2838091
p_adap_BIN(T)	494126	48724	2618276
p_adap_UNA(NT)	515521	71161	3005689
p_adap_BOTH(T)	486276	43991	2388566

6.2.3.1 Co-process Runtime

The adaptive routers can reduce average co-processor runtime. The multicast routers (except the unary multicast without transformation function) perform better than the unicast router in terms of the co-processor runtime. Unary multicast router without transformation function has the highest co-processor runtime because the unary multicast packets have long address flits and they block the multicast virtual channel and network interface output. Co-processors in the adaptive multicast router supporting both binary and unary codes with code transformation function have the lowest runtime.

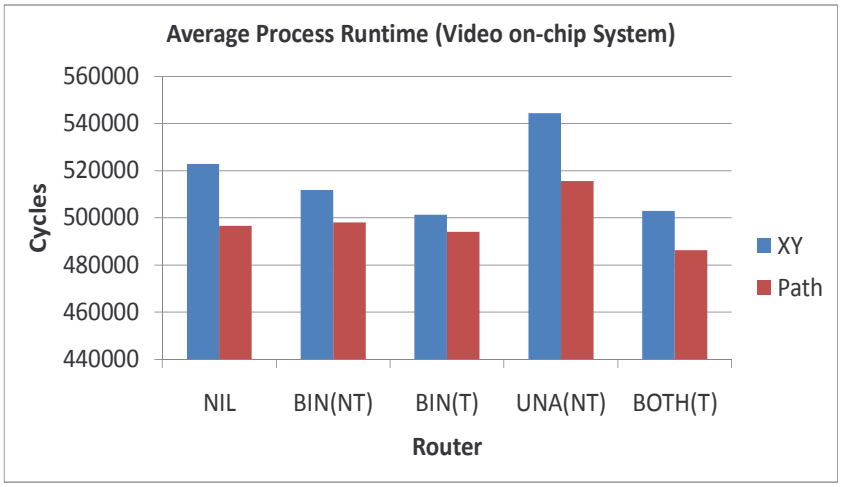


Figure 6.10 Average Video System Co-processor Runtime Comparison

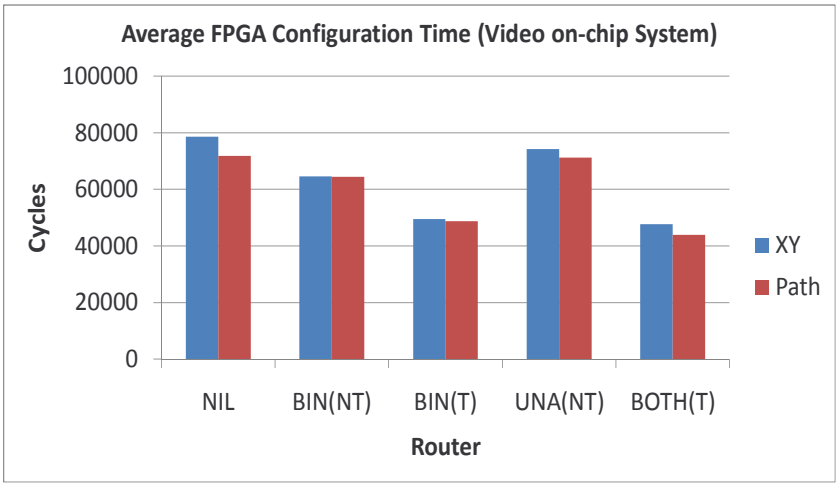


Figure 6.11 Video System FPGA Configuration Time Comparison

6.2.3.2 FPGA Reconfiguration Time

The adaptive routers reduce the average FPGA configuration time. The multicast routers perform better than the unicast router in terms of reconfiguration time. FPGA IP in the adaptive multicast router supporting both binary and unary codes with code transformation function has the lowest reconfiguration time.

6.2.3.3 FPGA Reconfiguration Router Energy

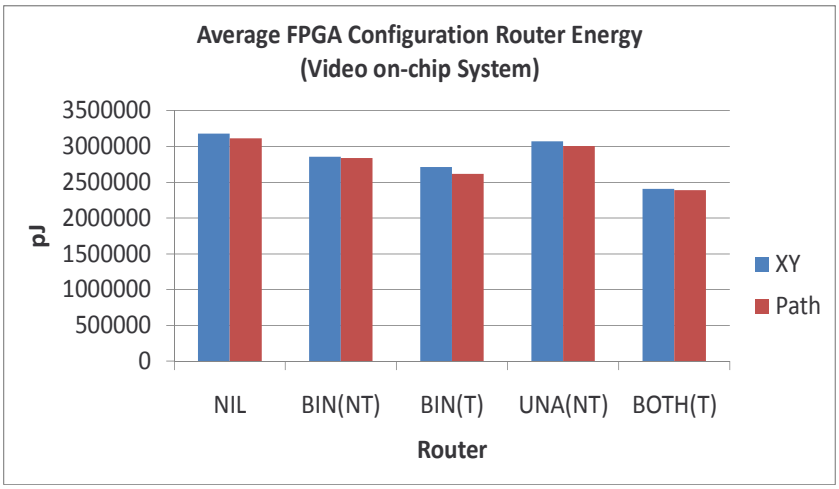


Figure 6.12 Video System FPGA Configuration Router Energy Consumption

The adaptive routers reduce the average FPGA configuration router energy consumption. The multicast routers perform better than the unicast router in terms of router energy consumption. FPGA IP reconfiguration in the adaptive multicast router supporting both binary and unary codes with code transformation function has the lowest router energy consumption.

6.3 Address-Data Decoupling Experiment

Address-Data FIFO Decoupling allows multicast routers using *Wormhole Routing* without having multicast deadlock. The routers can support packets which are longer than the virtual

channel. In this section, the router performances between routers using *Virtual Cut-through Routing* and *Wormhole Routing* are compared.

6.3.1 Synthetic Traffics

In the synthetic traffics experiment, a dual-coded adaptive multicast router with transformation function and an XY-routing multicast router are tested. Each router is tested using *Virtual Cut-through Routing* and *Wormhole Routing*. The routers using *Wormhole Routing* have the address-data FIFO decoupled virtual channels. Every packet in the routers using *Virtual Cut-through Routing* has a length of 10. Every packet in the routers using *Wormhole Routing* has a length of 20.

6.3.1.1 Maximum Throughput

Table 6.6 Virtual Cut-Through and Wormhole Routing Comparison (Maximum Throughput (Data Flits))

Router	Random	Transpose	Transpose2
XY(VCT)	905058	679126	679560
XY(Wormhole)	927716	695256	718888
Path(VCT)	963166	855494	786414
Path(Wormhole)	974072	961516	915640

The maximum throughputs of the four tested routers are shown in Table 6.6 and Fig. 6.13. *Wormhole Routing* can store more data into a single packet than *Virtual Cut-through* Routing. It increases the maximum throughput in both routers using *XY-Routing* and *Path-Based Adaptive Routing*.

6.3.1.2 Average Packet Latency Per Flit

Table 6.7 and Fig. 6.14 show the average packet latency per data flit. The data is recorded when the number of arrival data flits is around 551176. Wormhole router reduces the average packet latency of both routers using *XY-Routing* and *Path-Based Adaptive Routing*.

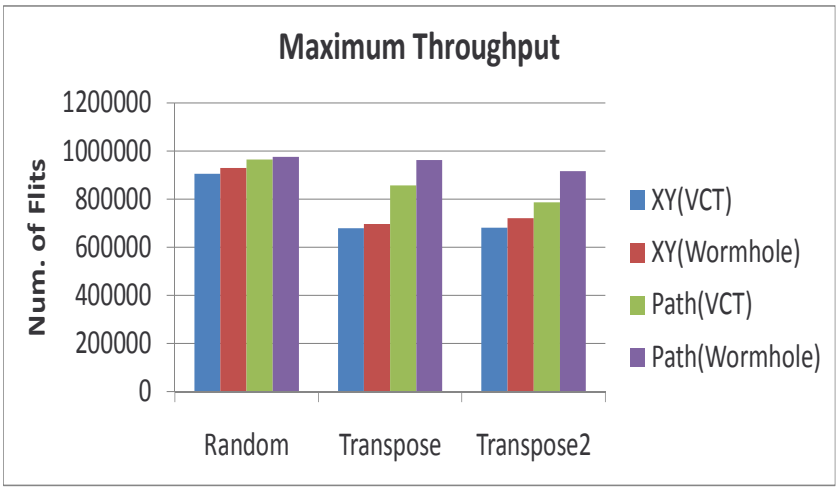


Figure 6.13 Virtual Cut-Through and Wormhole Routing Comparison (Maximum Throughput (Data Flits))

Table 6.7 Virtual Cut-Through and Wormhole Routing Comparison (Average Packet Latency Per Data Flit (Cycles))

Router	Random	Transpose	Transpose2
XY(VCT)	11.47	12.78	12.75
XY(Wormhole)	6.9	7.74	7.73
Path(VCT)	11.77	12.33	12.36
Path(Wormhole)	6.97	7.3	7.35

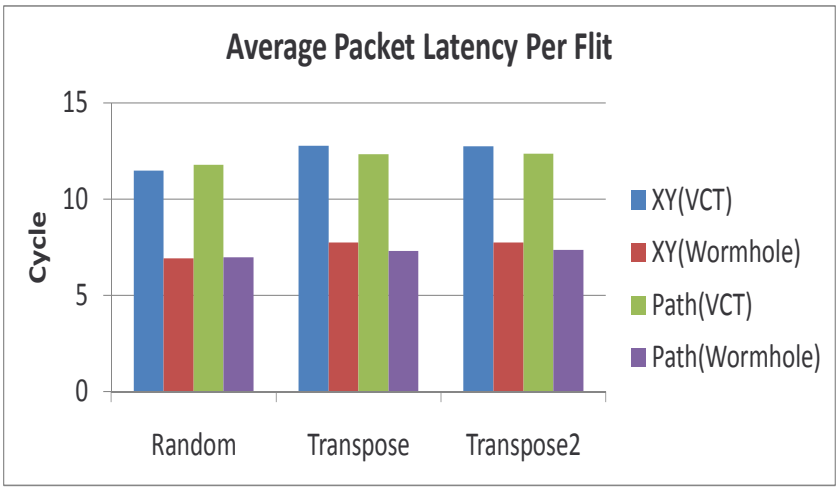


Figure 6.14 Virtual Cut-Through and Wormhole Routing Comparison (Average Packet Latency Per Data Flit (Cycles))

6.3.1.3 Average Router Energy Per Flit

Table 6.8 Virtual Cut-Through and Wormhole Routing Comparison (Energy Consumption per Data Flit (pJ))

Router	Random	Transpose	Transpose2
XY(VCT)	354.62	373.35	373.25
XY(Wormhole)	338.46	356.49	356.47
Path(VCT)	353.05	368.34	368.04
Path(Wormhole)	335.16	350.25	350.17

Table 6.8 and Fig. 6.15 compare the energy consumption per data flit. The data is recorded when the number of arrival data flits is around 551176 during the simulation. *Wormhole Routing* reduces the energy consumption per data flit of both routers using *XY-Routing* and *Path-Based Adaptive Routing* by reducing the number of packet headers.

6.3.2 Video System on FPGAs with 2D Mesh Network

In this section, the video on-chip system performances of the five adaptive routers in section 6.2.1 are compared. The video system used in this experiment is discussed in Chapter 7.

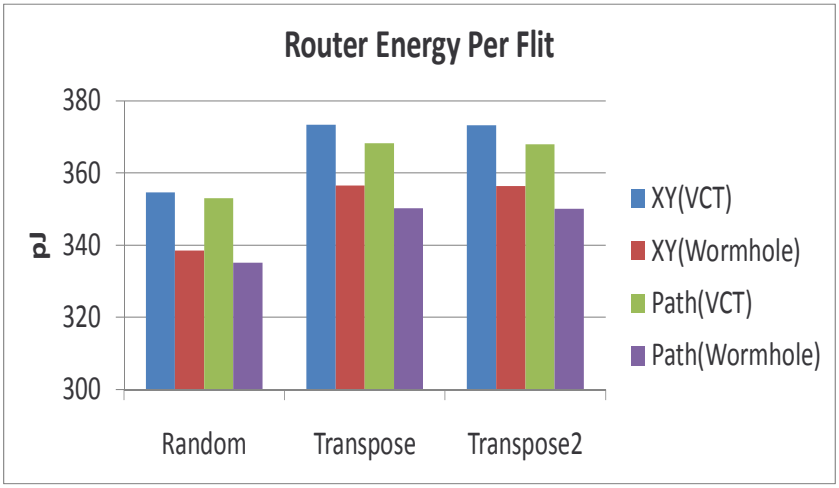


Figure 6.15 Virtual Cut-Through and Wormhole Routing Comparison (Average Packet Latency Per Data Flit (Cycles))

All multicast routers using *Wormhole Routing* have address-data FIFO decoupling virtual channels. Every packet in the routers using *Virtual Cut-through Routing* has a maximum packet length of 10. Every packet in the routers using *Wormhole Routing* has a maximum packet length of 20. A datum longer than the maximum packet length will be sent by multiple packets.

A co-processing request queue for the video system is randomly generated. The configuration bitstream size of each CLB group is 2.2 Mbits which equals 18,000 128-bit packets. The runtime (RT) and configuration time (CT) are measured in cycles and the energy is measured in nJ. Configuration time is the elapsed time interval between the system issuing reconfiguration signals until all tile reconfiguration is complete. Runtime is the time interval from when all tile reconfiguration is complete to when all data is processed. Note that runtime excludes configuration time.

The average co-processor runtime and the average FPGA configuration time are shown in Table 6.9, Fig. 6.16 and Fig. 6.17.

Table 6.9 Virtual Cut-Through Router and Wormhole Router Comparison (Video on-chip system using FPGA)

Router	Avg. Run Cycles	Avg. Run Cycles	Avg Conf. Cycles	Avg Conf. Cycles
	VCT	Wormhole	VCT	Wormhole
NIL	496585	469192	71792	57728
BIN(NT)	498072	462294	64417	41324
BIN(T)	494126	461907	48724	34502
UNA(NT)	515521	475190	71161	49608
BOTH(T)	486276	463397	43991	33712

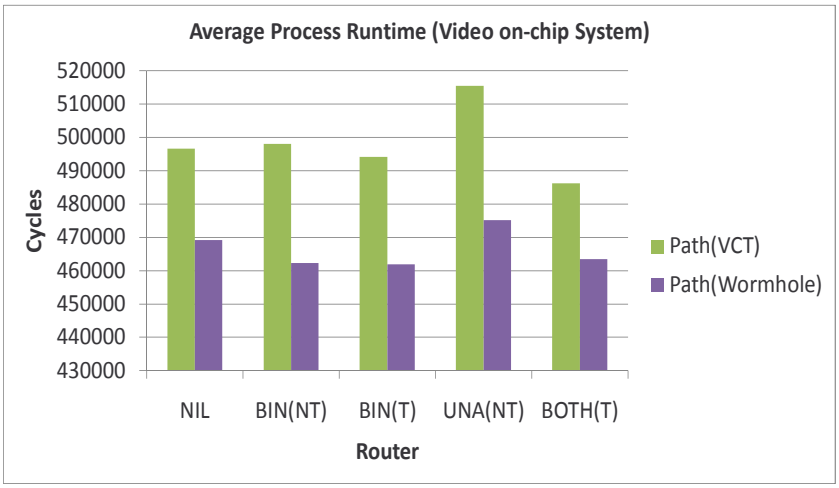


Figure 6.16 Average Video System Process Runtime Comparison

6.3.2.1 Co-processor Runtime

The routers using *Wormhole Routing* have lower average co-processor runtime than the routers using *Virtual Cut-through Routing*. As the number of data flits in a packet increases, the overhead of each data flit decreases. This reduces the network load. Then, packet latency also decreases because of the less congested network. As a result, the co-processor runtime is reduced.

6.3.2.2 FPGA Reconfiguration Time

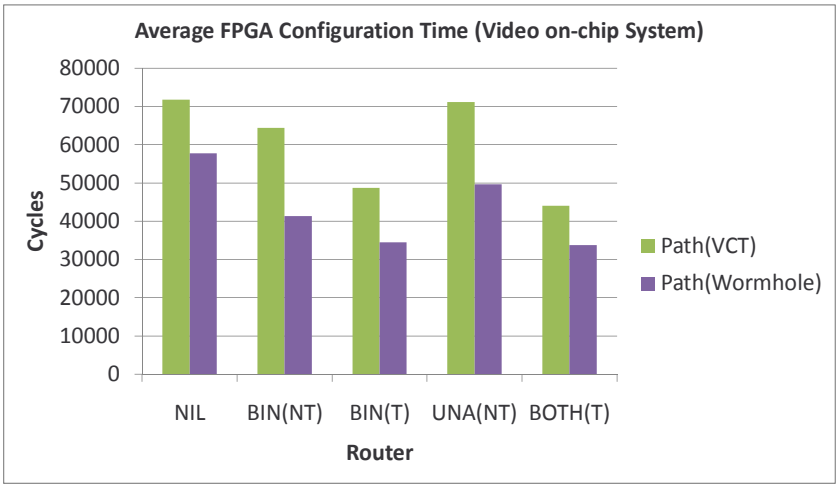


Figure 6.17 Video System FPGA Configuration Time Comparison

The routers using *Wormhole Routing* have lower average FPGA tile reconfiguration time than the routers using *Virtual Cut-through Routing*. As the number of data flits in a packet increases, the overhead of each data flit decreases. This reduces the network load. Then, packet latency also decreases because of the less congested network. As a result, the FPGA configuration time is reduced.

6.3.3 Hardware Implementations

Six routers are implemented to compare their chip spaces and clock cycles. The six routers are:

1. *XY unicast router (XY-NIL)*
2. *Binary-coded XY router with code transformation function (XY-BIN(T))*
3. *Dual-coded XY Router with code transformation function (XY-BOTH(T))*
4. *Path-based adaptive unicast Router (Path-NIL)*
5. *Binary-coded Path-based adaptive router with code transformation function (Path-BIN(T))*
6. *Dual-coded Path-based adaptive router with code transformation function (Path-BOTH(T))*

The routers are written in VHDL and are synthesized by Cadence RTL Compiler using TSMC $65nm$ standard cell library. The unicast router has three unicast virtual channels on each input port. The multicast router has two unicast channels and one multicast channel on each input port. The data bus width is 128-bit. The *Wormhole* router's virtual channel length is 9. The memory component is implemented using flip-flop standard cells. The size of the adaptive router's observation window is 5×5. Table 6.10 shows the details of the hardware implementation.

The chip space of the multicast routers are 100 $M\lambda^2$ higher than the unicast routers because of the extra 5-head FIFOs and the multicast maps. The chip space of the multicast routers with code transformation is 25 to 30 $M\lambda^2$ higher than the multicast routers without code transformation. The *Route Computation* stage of *Path-based Adaptive Router* is longer than *XY-router* because of the more complex route computation. By limiting the observation window size to 5×5, the *Route Computation* path would not become the critical path. Then, the clock in *Path-based Adaptive Unicast Router* can run as fast the clock in *XY Unicast Router*. *Address-Data FIFO Decoupling* lengthens *Switch Allocation* stage because of the *Body-to-Tail* flit modification. The router clock increases from $1151ps$ to $1264ps$.

Table 6.10 Area and Cycle Time Data

Router	Multicast Support	Area ($M\lambda^2$)	RC (ps)	VA (ps)	SA (ps)	ST (ps)
XY	NIL	208.42	375	503	1151	257
XY	BIN(T)	308.36	575	503	1264	257
XY	BOTH(T)	333.45	597	503	1264	257
Path	NIL	216.10	843	503	1151	258
Path	BIN(T)	328.35	1016	503	1264	258
Path	BOTH(T)	358.11	1027	503	1264	258

CHAPTER 7. An On-chip System Built from Many FPGAs with a 2D Mesh Network

This chapter discusses the operation of an on-chip system built from many FPGA tiles and a 2D-Mesh on-chip network. The protocol in the on-chip system is explained and a sample application which could be used in the system is given.

7.1 Motivation

There are many ways to use an FPGA. A notable use is as a co-processor to speed up (accelerate) the intensive portion of a program. Lysecky *et al.* [31] propose *Warp Processors* to accelerate a program by converting software binary instructions into FPGA circuit configuration bitstream. A CPU stores the data going to be processed in a memory space which is the input buffer of the co-processor. Then, CPU notifies the configured FPGA to process the data. After the data has been processed, the FPGA stores the processed data in a memory space which is the output buffer of the co-processor. In this system, data travels from the CPU to the FPGA and back to the CPU.

Depending on the co-processor logic size and internal structure, a co-processor can be implemented using multiple FPGA tiles. For a co-processor with many stages, each stage can be built from a single FPGA tile. *SCORE* [7] is a co-processor system that supports multiple FPGAs co-processing. In *SCORE*, the data travels through multiple FPGA tiles before going back to the CPU. *SCORE* contains four FPGA tiles connected by a bus.

As the fabrication technology improves, more FPGAs can be put onto a single chip. With the increased number of on-chip system modules, the bandwidth of a bus can no longer meet the increased communication demands from the FPGAs. Therefore, a many-FPGA on-chip

system needs a native 2D mesh network to prevent the communication infrastructure from becoming the system bottleneck [28].

7.2 Architecture of an on-chip FPGA system with a 2D Mesh Network

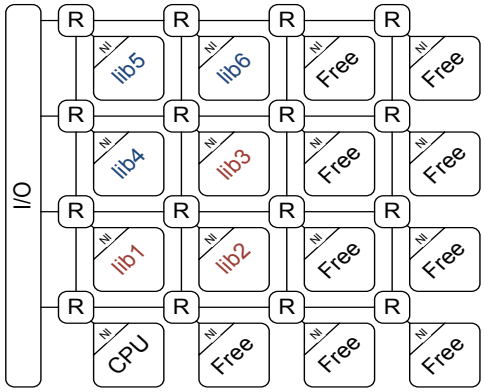


Figure 7.1 Many-FPGA on-chip system Layout

A many-FPGA on-chip system (Fig. 7.1) contains many CLB Groups and a CPU on the bottom left corner. They are connected by a 2D mesh network. Each CLB Group can be considered as an FPGA tile and be reconfigured independently as a module to support a co-processor. A CLB Group (Fig. 7.2) contains:

1. A set of input buffers to store the data from other CLB Groups or I/O.
2. A set of output buffers to store the processed data going to other CLB Groups or I/O.
3. An address table to store the physical addresses of the ancestor modules and descendant modules of the CLB Group.
4. A set of Block RAMs.
5. A set of Configuration Logic Blocks for data processing.

The CPU is responsible for modules placements and CLB Group reconfigurations. A co-processor (accelerator) (Fig. 7.3) is composed of several modules. Each CLB Group can be

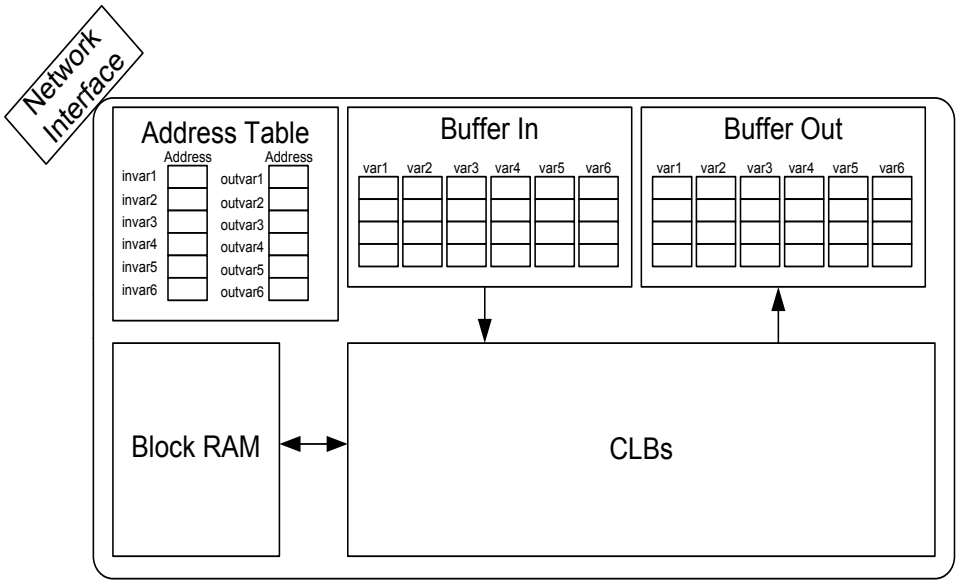


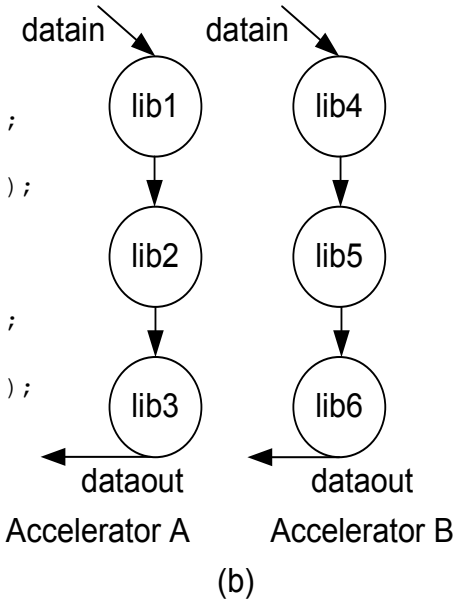
Figure 7.2 CLB Group

configured as a single module to support the co-processor. The data comes from the I/O, processed by several modules and then goes back to the I/O. The I/O tiles are located on the left side of the chip. Each I/O tile handles the input and output of all CLB Groups in its row.

When the system is implemented using the $16nm$ technology, a $112mm^2$ chip contains 400 FPGA tiles. These FPGA tiles are connected by a 20×20 2D Mesh network. Each FPGA tile and its router occupy $66.14K\lambda \times 66.14K\lambda$ of chip space. The configuration bitstream size of each FPGA tile is about $2.44Mbits$.

```
void acceleratorA(){
 for (i=0;i<1000;i++){
   lib1(datainto1[i],data1to2[i]);
   lib2(data1to2[i],data2to3[i]);
   lib3(data2to3[i],data3toout[i]);
}
void acceleratorB(){
 for (i=0;i<1000;i++){
   lib4(datainto4[i],data4to2[i]);
   lib2(data4to2[i],data2to5[i]);
   lib5(data2to5[i],data5toout[i]);
}
```

(a)



(b)

Figure 7.3 Co-processors (a) Code, (b) Dataflow Graph

7.3 Protocols

The on-chip FPGA system supports ten types of packets listed in Table 7.1.

Table 7.1 Packet Type

	Packet Type	Src.	Dest.	Description
1	Data	I/O CLB G. CLB G.	CLB G. CLB G. I/O	The data to be processed or processed data going back to I/O
2	Data Ack	CLB G.	CLB G.	Ack the sender that the data is received
3	Reconfigure	CPU	CLB G.	Set up the CLB Group's address table
4	Configuration Bigstream Request	CPU CPU	I/O CLB G.	Ask the I/O or CLB Group to send bitstream to the reconfiguring CLB Groups.
5	Configuration Bitstream	I/O CLB G.	CLB G. CLB G.	Configuration Bitstream from I/O or CLB Group to the CLB Group to be reconfigured
6	Configuration Complete	CLB G.	I/O	Notify the CPU that the reconfiguration is complete
7	Start	CPU	CLB G.	Ask the CLB Group to start requesting data from the I/O
8	Data Request	CLB G.	I/O	Ask the I/O to provide Data
9	Buffer Request	CLB G.	CLB G.	The ancestor CLB Group requests the buffer in the descendant CLB Group
10	Buffer Ack	CLB G.	CLB G.	Answer the ancestor CLB Groups with the number of available buffers

7.4 Co-Processor Computing Model

In this section, a co-processing scenario for an on-chip FPGA system is illustrated. Fig. 7.4(a) shows the co-processor structure and Fig. 7.4(b) shows the current chip layout. When a program needs co-processing, it puts a request into a system co-processing queue. The CPU checks the queue and services the requests in a first come first serve basis. When there are enough free CLB Groups to support a request, the CPU starts co-processing. It picks a CLB Group for each co-processor module. (The placement algorithms' details are given in Chapter 8.) Then, the system reconfigures the CLB Groups to form a co-processor.

In this scenario, the CPU has decided to map co-processor modules *M1*, *M2*, *M3* and *M4* to Tile (2,2), Tile (2,3), Tile (3,2) and Tile (3,3) respectively. It sends a $Reconfigure$ packet to each tile to modify their address tables. From the table, Tile (2,2) knows that it will be configured to module *M1*. *M1* is the level 0 node in the data flow graph and it has two descendants. Its descendant *M2* is located at Tile (2,3) and its descendant *M3* is located at

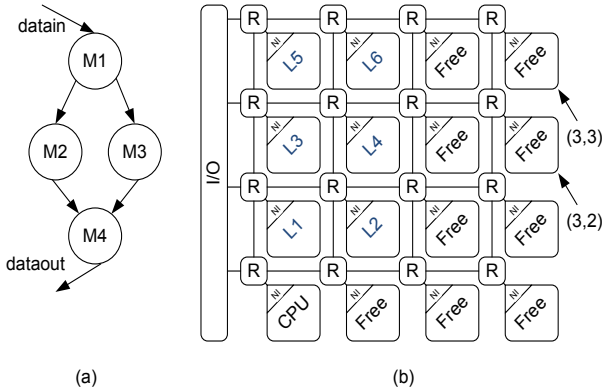


Figure 7.4 (a) Co-processor Structure (b) Current Chip Layout

Tile (3,2).

Because configured modules *M1*, *M2*, *M3* and *M4* do not exist on the chip at this time, the configuration bitstreams have to be fetched from the I/O. The CPU sends four *BS Request* packets to the I/O tiles. These *BS Request* packets ask the *I/O* tiles to send *M1* configuration bitstream to Tile (2,2), *M2* configuration bitstream to Tile (2,3), *M3* configuration bitstream to Tile (3,2) and *M4* configuration bitstream to Tile (3,3). Once Tile (2,2), Tile (2,3), Tile (3,2) and Tile (3,3) have received the configuration bitstream, they start the CLB Group reconfiguration. When the reconfiguration is complete, the CLB Group sends a *Configuration Complete* packet back to the CPU. Once the CPU has received all the *Configuration Complete* packets from the reconfiguring CLB Groups, the CPU sends a *Start* packet to the head (root) of the co-processor (*M1* at Tile (2,3)). Then, module *M1* sends a *Data Request* packet with a memory address to the I/O to ask for the input data. The amount of the requested data depends on the number of available input buffers in module *M1*. The *I/O* then sends some *Data* packets back to module *M1*. Module *M1* starts processing the data received from the *I/O* once the data packet has arrived. It also sends a *Data Ack* packet to the *I/O* to acknowledge the *I/O* that it has received the data. It keeps processing the input data from the input buffers and producing the output data to the output buffers. After several rounds of data processing, the number of input data in the input buffers will drop below a threshold level. When this

happens, module *M1* sends another *Data Request* packet to the *I/O* to ask for more data. When the number of output data in module *M1*'s output buffers is higher than a threshold level, module *M1* sends a *Buffer Request* packet to its descendant modules *M2* and *M3* asking for the number of available input buffers. Modules *M2* and *M3* respond with two *Buffer Ack* packets to inform module *M1* about the number of available input buffers. Module *M1* sends an appropriate amount of processed data to the descendants *M2* and *M3* for further processing. Modules *M2* and *M3* send two *Data Ack* packets back to *M1* once they have received the data before they start data processing. The leaf node of the co-processor data flow graph sends the processed data back to the *I/O*. The CPU keeps track of the number of processed data. When all input data have been processed, the CPU removes the co-processor from the system by setting all CLB Groups allocated to this co-processor *FREE*. Then, it checks if there are sufficient free CLB Groups for the next co-processor request.

7.5 On-chip Video FPGA System

In this section, a video server hosted on an on-chip FPGA system is discussed. This video server is proposed by Herveille *et al.* which is publicly available on *opencore.org* [22]. The system contains 23 modules. The details of each module are shown in Table 7.2. Six types of MPEG co-processor could be built using these 23 modules. The six MPEG co-processors are:

1. MPEG4 CABAC encoder (Fig. 7.5)
2. MPEG4 CABLC encoder (Fig. 7.5)
3. MPEG4 CABAC decoder (Fig. 7.6)
4. MPEG4 CABLC decoder (Fig. 7.6)
5. MPEG2 encoder (Fig. 7.7)
6. MPEG2 decoder (Fig. 7.8)

Table 7.2 Video Server Modules

	Module Name	Cycles	Inputs Width (bits)	Outputs Width (bits)
1	I-prediction	256	256/256	288/256
2	P-prediction	256	256/256	288/128/256
3	Core Transform	80	288/288	128
4	DCT	64	128	128
5	Quantise	64	128	128/128/128
6	IDCT	64	128/128	128
7	Dequantise	48	128	128
8	Inverse Transform	128	128	256
9	Reconstruction	128	256/256/256	256/256
10	CABAC	80	128	240
11	CAVLC	32	128	120
12	ToByteB	16	240	64
13	ToByteV	80	120	32
14	Arithmetic (ARI)	8	128/128	128
15	Motion Estimator	256	128	128
16	Motion Compensator	256	128/128	128
17	VLC	16	128	64
18	FBYTEV	32	64	240
19	CAVLD	40	240	128/128
20	FBYTEB	16	32	120
21	CABAD	80	120	128/128
22	VLD	16	64	128
23	MC	68	128/128	128

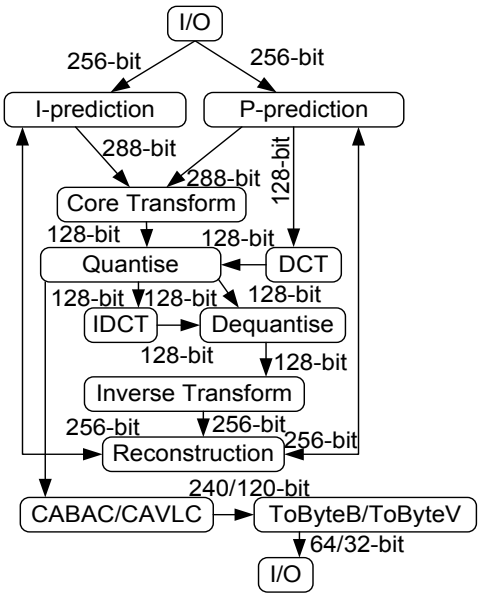


Figure 7.5 MPEG4 Encoder (M4EB/M4EV)

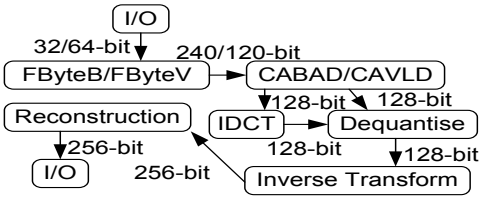


Figure 7.6 MPEG4 Decoder (M4DB/M4DV)

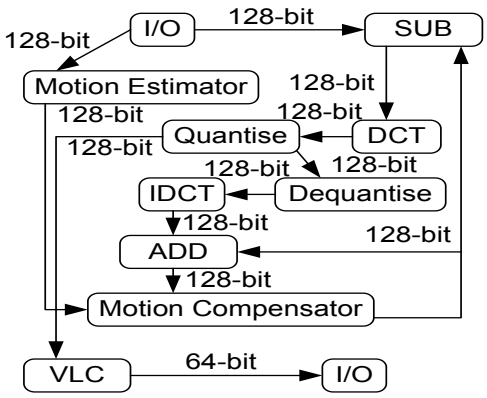


Figure 7.7 MPEG2 Encoder (M2E)

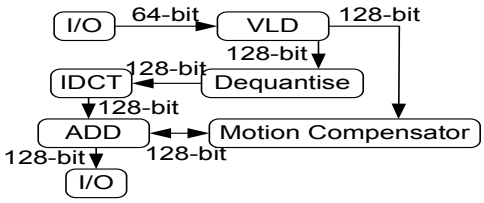


Figure 7.8 MPEG2 Decoder (M2D)

CHAPTER 8. A Basic Co-processor Placer

8.1 Motivation

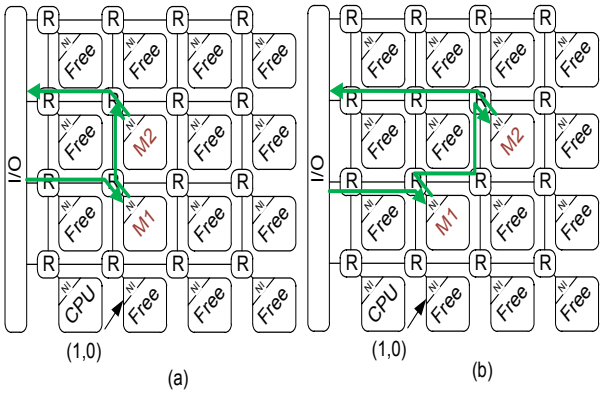


Figure 8.1 Placement Comparison

In a native 2D mesh network supporting FPGA on-chip system, co-processor modules' locations affect system performance. When the system modules are closely placed, network energy consumption can be reduced and system throughput can be increased. Fig. 8.1 compares two co-processor placements. A co-processor in this system requires two modules, *M1* and *M2*. The data travels from the *I/O* to *M1*, from *M1* to *M2* and back to the *I/O*. In Fig. 8.1(a), *M1* is placed on location (1,1) and *M2* is placed on location (1,2). The number of hops of each data stream is 3. In Fig. 8.1(a), *M1* is placed on location (1,1) and *M2* is placed on location (2,2). The number of hops of each data stream is 5.

Placing modules closer can reduce communication energy because each extra packet hop traversed by a packet increases energy consumption. The co-processor performance increases because its data packets experience less congestion when the number of hops is lower. In

addition, system performance increases because the decreased demands of the co-processor's packets lessen the link competition with other packets.

8.2 Algorithm

This section details the basic placement algorithm. The placer uses this algorithm to place a co-processor on free CLB Groups. The co-processor is considered to be a data flow graph. Each node in the data flow graph is a module supporting the co-processor. Each module belongs to a IP type. A co-processor can start once all its modules are placed on the CLB groups which are configured to the corresponding IP types. Each co-processor has a module placement order list. The CPU places the co-processor's modules following this list. The list is created by sorting the edges in the data flow graph according to their communication volume. Two modules connected by edge e with the highest communication volume $Comm(e)$ are placed first. In these two modules, the module m with higher communication volume $Comm(m)$ is placed first. The communication volume of a module m is
$Comm(m) = \sum_{i=1}^{\#parent(m)} Comm(m, parent(m,i)) + \sum_{j=1}^{\#children(m)} Comm(m, children(m,j))$.

The basic co-processor placement algorithm contains four main steps.

1. Look for some revivable tiles to reuse them.
2. Look for some free tiles to place the co-processor's modules.
3. Place the modules on the selected free tile.
4. Decide the configuration bitstream sources.

The algorithm notation is shown in Table 8.1. The algorithm starts with a search for revivable tiles. Revivable tile is a CLB group which is idle (Free) but has been configured to an IP type needed by the co-processor being placed. Using revivable tiles reduces tile reconfiguration time and reconfiguration energy. The center of gravity method is used to reduce the Manhattan Distances between the mapping modules to minimize network communication energy. CG_{base} is computed based on the required IP types and the number of available revivable tiles (Line 3, Algorithm 1).

1: //Compute Mapping Center of Gravity
2: **if** $\sum_{i \in IP_g} |TileR_{g,i}| > 0$ **then**
3: $CG_{base} = \frac{\sum_{i \in IP_g} \sum_{j \in TileR_{g,i}} loc(j) \times |IP_{g,i}|}{\sum_{i \in IP_g} |TileR_{g,i}| \times |IP_{g,i}|}$;
4: **else**
5: $CG_{base} = \frac{\sum_{s \in List_{free}} loc(s)}{|List_{free}|}$;
6: **end if**
7: //Revivable Tiles Selection
8: $num\_freetile\_needed = 0$;
9: **for all** $i \in IP\_g$ **do**
10: **if** $|IP_{g,i}| \geq |Tile\_R_{g,i}|$ **then**
11: Move all $Tile\_R_{g,i}$ from $List_{free}$ to $List_{revive}$;
12: $num\_freetile\_needed += (|IP\_g, i| - |Tile\_R_{g,i}|)$;
13: $Rtoken(i) = (|IP\_g, i| - |Tile\_R_{g,i}|)$;
14: **else**
15: Sort $Tile\_R_{g,i}$ according to their MDs from the CG_{base} in ascending order;
16: Move the first $|IP\_R_{g,i}|$ $Tile\_R_{g,i}$ from $List_{free}$ to $List_{revive}$;
17: **end if**
18: **end for**
19: **if** $num\_freetile\_needed > num\_freetile$ **then**
20: return "Not enough free tile to run the co-processor";
21: **end if**
22: //Reconfiguration Tiles Selection
23: $CG_{revive} = \frac{\sum_{s \in List_{revive}} loc(s)}{|List_{revive}|}$;
24: Sort $List_{free}$ according to their MD from the CG_{revive};
25: Move the first $num\_freetile\_needed$ tiles from $List_{free}$ to $List_{new}$;
26: //Place Graph's Module
27: **for all** $Module\ t \in List_g$ **do**
28: **if** $RToken(IPtype(t))! = 0$ **then**
29: $List_{candidate_t} = List_{new} + List_{revive_{IPtype(t)}}$;
30: **else**
31: $List_{candidate_t} = List_{revive_{IPtype(t)}}$;
32: **end if**
33: **if** Some t's neighbor modules are mapped **then**
34: $CG_{neigh} = \frac{\sum_{m \in mapped\_neigh(t)} loc(m)}{|mapped\_neigh(t)|}$;
35: **else**
36: $CG_{neigh} = \frac{\sum_{m \in neigh(t)} CG_{IPtype(m)}}{|neigh(t)|}$ where $CG_{IPtype(m)} = \frac{\sum_{p \in List_{candidate_m}} loc(p)}{|List_{candidate_m}|}$;
37: **end if**
38: Map t on the tile $k \in List_{choices}$ with the smallest MD from CG_{neigh};
39: **if** $k \in List_{new}$ **then**
40: Add k to $List_{reconfigure_{IPtype(t)}}$;
41: $Rtoken(IPtype(t)) - -$;
42: **end if**;
43: Remove k from either $List_{new}$ or $List_{revive_{IPtype(t)}}$;
44: **end for**
45: //Bitstream Source Selection
46: **for all** $i \in IP\_g$ **do**
47: **if** $List_{reconfigure_i}$!=NULL **then**
48: Compute $CG_{List_{reconfigure_i}}$
49: **if** $\exists$ on-chip bitstream **then**
50: $Src_{bitstream} = Tile\_R_{g,i}$ with the smallest MD from $CG_{List_{reconfigure_i}}$;
51: **else**
52: $Src_{bitstream} = I/O$ close to $CG_{List_{reconfigure_i}}y$;
53: **end if**
54: **end if**
55: **end for**

Algorithm 1 Basic Runtime Placement Algorithm

Table 8.1 Notation

n	Number of tiles
t_g	Number of modules in the graph g
IP_g	Set of IPtype required by the graph g
$\|IP_{g,i}\|$	Number of IPtype i required by the graph g
$TileR_{g,i}$	Revivable tile configured as IPtype i which is required by the graph g
CG_R_i	Center of gravity of all $TileR_{g,i}$
$CG_{proposed}$	Proposed center of gravity for the placement
CG_{revive}	Center of gravity of all selected revivable tiles
CG_{free}	Center of gravity of all free tiles
CG_{neigh}	Center of gravity of all neighbor modules
MD	Manhattan distance
$neigh(t)$	A set of neighbor modules of the module t
$List_g$	Sorted graph's module list according their communication volumes in descending order
$List_{revive_i}$	List of selected $TileR_{g,i}$
$List_{free}$	List of free tiles
$List_{new}$	List of selected free tiles to be used
$List_{candidate_t}$	List of placement candidate of module t
$loc(x)$	Location of the tile x

If the number of module with IP type t needed by the co-processor is k, the weighting of each tile configured to t is k. If no revivable tiles could be used for the placement (*e.g.* the system has just started up), CG_{base} is equal to CG of all free tiles.

The number of revivable tiles of a certain IP type can be higher than, equal to or lower than those needed by the co-processor being placed. If there are more revivable tiles of a certain IP type than needed, the revivable tiles which are closer to CG_{base} would be selected. The availability of the revivable tiles may not fulfill the needs of the co-processor. Therefore, some idle tiles have to be reconfigured to the desired IP types. Tokens ($Rtoken$) are given to those IP types for claiming reconfigurable free tiles at the placement stage.

Some on-chip free tiles are configured to the desired IP types for the co-processor. CG_{revive} which is the center of gravity of all selected revivable tiles in the previous stage is computed. The required number of free on-chip tiles is selected. Those selected tiles are the tiles which are closest to CG_{revive}. At the placement stage, each module is placed into the chosen tiles. The modules are stored in a list $List_g$. These modules in $List_g$ are sorted according to their communication volume in descending order. The module with the highest communication volume is placed first. Modules are placed either on the revived tiles or on the free tiles (if

$RToken$ is available). If the mapping module's neighbors are mapped, a weighted CG_{neigh} of those mapped neighbors is computed based on the communication volume between the mapping module and those neighbors. If none of its neighbors is mapped, CG_{neigh} will equal CG of its neighbors' IPtype revivable tile and free tiles. A Module will be placed on the available tile which is close to CG_{neigh}. Finally, bitstream sources are selected to reconfigure the tiles. As the router has native multicast support [25], the bitstream for all modules with the same IP type is fetched from a single source. Bitstream is fetched from an on-chip tile if there exists a tile with a cached version of the required bitstream. Otherwise, the bitstream will be fetched from the I/O. $CG_{List_{reconfigure}}$ of tiles requiring the same type of bitstream is computed. The bitstream source is the tile closest to $CG_{List_{reconfigure}}$ and with the required bitstream.

8.3 Experiments

8.3.1 Experimental Setup

A co-processing request queue for the video system is randomly generated. The configuration bitstream size of each CLB group is 2.2 Mbits which equals 18,000 128-bit packets. The runtime (RT) and configuration time (CT) are measured in cycles and the energy is measured in nJ. Configuration time is the elapsed time interval between the system issuing reconfiguration signals until all tile reconfiguration is complete. Runtime is the time interval from when all tile reconfiguration is complete to when all data is processed. Note that runtime excludes configuration time.

The simulation parameters are shown in Table 8.2.

8.3.1.1 Comparison Between Random Placement and CG Placement

In this experiment, placement qualities of random co-processor placement algorithm and basic co-processor placement algorithm proposed in this chapter are compared. The result is shown in Table 8.3. The average router energy consumption per co-processing, the number of co-processor executions within the simulation time and the average co-processors runtime are

Table 8.2 Baseline simulation parameters

Grid Size	20×20
Video System Simulation Cycles	$20,000,000$
Num. Unicast Virtual Channels on each port	1
Num. Multicast Virtual Channels on each port	2
Num. Samples	5
Flit Width	$128 - bit$
Num. of Jobs Request/Co-processor	2000
Path-Based Adaptive Routing observation window	5×5

recorded. The system using basic placement algorithm consumes 16% less data router energy than random placement algorithm. Basic placement algorithm allows co-processors to run 6% faster and complete 10% more co-processor executions than random placement algorithm.

Table 8.3 Average Bitstream and Data Network Energy per Co-processing

Placement	Random	Basic
#Exec.	1204	1325
Avg. RT	568135	533439
Avg. Co-processor Data Energy	141226 nJ	117979 nJ

8.3.1.2 Comparison Between Reviving Tiles and not Reviving Tiles

The goal of reviving a tile is to reduce the tile configuration time and router configuration energy. In fact, an algorithm without searching for revivable tiles has its own advantages. Table 8.4 compares the number of executions, average co-processing runtime and average co-processor configuration time between *revival (R)* algorithm and *non-revival (NR)* algorithm. Reviving configured tiles reduce co-processor configuration time. Co-processing runtime is also reduced because of the decreased network load. As data packets and configuration bitstream packets share the same network, data packets benefit from lower traffic congestion. As a result, the number of co-processor executions during the same simulation cycles is higher in *revival (R)* algorithm than in *non-revival (NR)* algorithm. Table 8.5 shows the data and the configuration

Table 8.4 Comparison between non-reviving (NR) and reviving (R) the pre-existing tiles

Acc.	#Exec.	Avg. RT (NR/R)	Avg. CT (NR/R)
M4EB	197/210	763549/702158	47335/31989
M4EV	184/202	716915/697708	51909/33407
M2E	212/243	692512/620156	51997/37402
M4DB	215/241	336451/307686	48734/37437
M4DV	185/214	333652/303519	48593/38963
M2D	211/241	621426/564318	48237/35412

bitstream energy consumptions per executed co-processor of both *revival* algorithm and *non-revival* algorithm. Although the *revival* algorithm can reduce the configuration bitstream energy consumption, it does increase the data energy consumption. It indicates that the co-processor modules placed using *revival* algorithm are not located as close as those being placed using *non-revival* algorithm. *Non-Revival* algorithm leads to higher placement quality because it has more flexibility in tile selections. The *revival* algorithm is forced to use the already configured tiles which might be located far away. If the amount of data to be processed by the co-processor is large, the *non-revival* algorithm is better in reducing energy consumption.

Table 8.5 Placement Comparison

Algorithms	Conf. Bitstream Energy	Data Energy
Revival	113298 nJ	9492 nJ
Non-Revival	92434 nJ	23884 nJ

8.3.1.3 Tile Idling

If the system waits to service the co-processing request until the chip has enough CLB groups, the placement quality is usually low. It is because the modules can only be placed on a limited number of free tiles regardless of how far they are separated. It results in high data network energy consumption. Leaving some tiles idle increases the flexibility of the CLB Groups selection. It leads to a high quality placement.

Table 8.6 shows the effects of idling the CLB group on the number of co-processing executions, average co-processor runtime and router energy consumption of the data packet using *no-reviving algorithm*.

The router energy consumption of data packets decreases when the number of idle tiles increases because the co-processor modules are placed close to each other. The number of co-processor executions decreases when the number of idle tiles increases because the tiles are not being utilized largely. The average co-processor runtime decreases when the number of idle tiles increases because the network is less congested.

Table 8.6 Idle Tiles Result

#Idle Tiles	#Exec.	Avg. RT	Data Energy
0	1204	576684	92434 nJ
10	1201	573902	89371 nJ
20	1203	557800	84798 nJ
30	1182	546649	81029 nJ
40	1164	543187	79665 nJ
50	1145	539223	78183 nJ
60	1113	536646	77688 nJ
70	1086	535033	76508 nJ
80	1051	533943	76490 nJ

CHAPTER 9. A Co-processor Placer For The Sharable CLB Groups

9.1 Motivation

In Chapter 8, each CLB Group is limited to support only one module in the co-processor. Within each co-processor, the module with the highest processing time is the bottleneck module of the co-processor. It is a key factor in determining the co-processor's throughput. This bottleneck module creates idle time on other modules within the same co-processor. The bottleneck module's ancestors have to wait for the bottleneck module to accept the newly generated data while the descendants have to wait for the data generated by the bottleneck module. In this chapter, an algorithm is proposed to allow different co-processors to share a CLB Group to increase the CLB Group utilization rate and increase the throughput of the system.

9.2 CLB Group Sharing

To support CLB Group sharing, additional buffer sets are added to the CLB Groups (Fig. 9.1). Each co-processor module mapped to the CLB Group occupies one buffer set.

Fig. 9.2 shows an example of module sharing. Two co-processors are to be mapped onto the chip. Module *lib1* processing time (PT) is ten cycles. Module *lib2* processing time is two cycles. Module *lib3* processing time is twelve cycles. Without CLB Group sharing, six CLB Groups are needed. If each CLB Group can be shared by two co-processors, two logical *lib1* modules can be mapped into the same physical CLB Group (Tile 1,1)), two logical *lib2* modules can be mapped into the same physical CLB Group (Tile 1,2)) and two logical *lib3* modules can be mapped into the same physical CLB Group (Tile 2,2)). Therefore, these two co-processors can be supported by three CLB Groups.

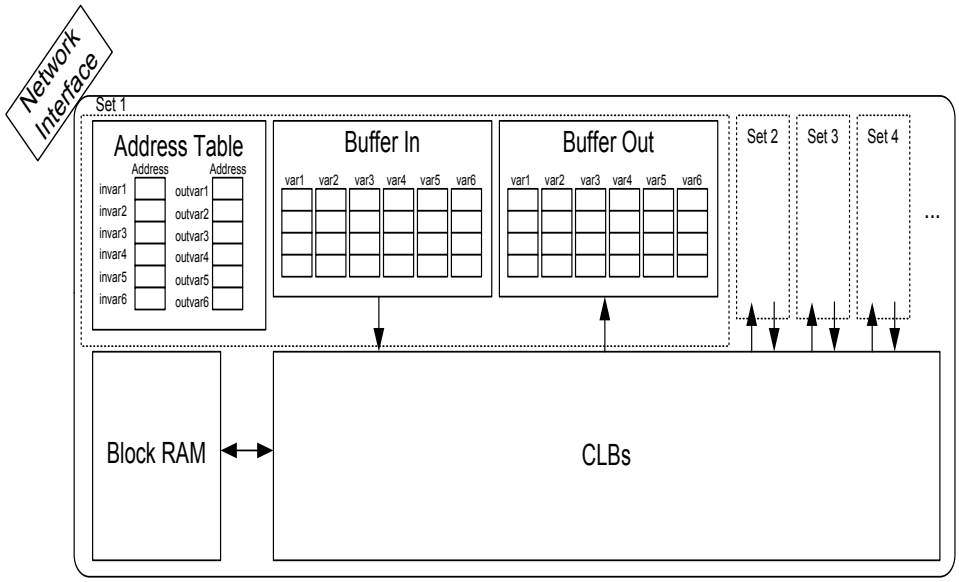


Figure 9.1 Sharable CLB Group Architecture

9.3 Full Tile Sharing and Bottleneck Aware Sharing

In this section, two different ways to share a CLB Group are discussed. They are *Full Tile Sharing* and *Bottleneck Aware Sharing*.

The basic co-processor placer in Chapter 8 is modified to support CLB Group sharing. The name of this modified algorithm is *Sharable CLB Group Placer (SCGP)*. In *SCGP*, revivable tile is considered to be a sharable tile with all the buffer sets available. At the first stage of the algorithm, the number of available IP slots (buffer sets) for sharing is found. Each tile can be shared by at most Max_S IP modules due to the limited number of buffer sets. In Full Tile Sharing (*FTS*), a module is shared as long as it has free buffer sets. The drawback of *FTS* is that it could slow down the co-processor greatly due to bottleneck module sharing. When the co-processor's bottleneck module is shared, the co-processor throughput decreases. In the example shown in Fig. 9.2, when all three modules are shared by the two co-processors, the *lib1*, *lib2* and *lib3* module processing time double. The bottleneck processing time of each co-processor becomes 24. To avoid this problem, bottleneck aware sharing (*BAS*) is used. (The details of *BAS* are shown in Algorithm 2, 3). In *BAS*, not all of the available

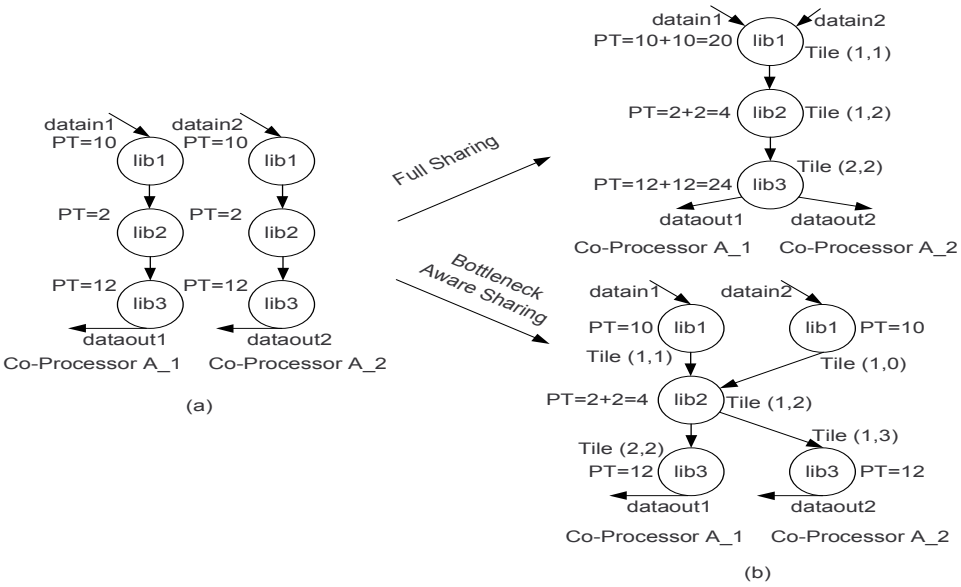


Figure 9.2 Module Sharing (a) Two Co-processor Graphs, (b) Two Mapped Co-processor Graphs

buffer sets are used to share the CLB Group. Instead, a CLB Group will be shared only if the additional sharing does not affect the bottleneck of the co-processors which are currently sharing the CLB Group. When the number of CLB Group sharers increases by one, the tile (T) processing time ($PT(T)$) will increase by the processing time ($PT(i)$) of the IPtype i. To avoid increasing the bottleneck processing time of the co-processors acc which are currently occupying the CLB Group, the placer can only share the CLB Group to $(BN(acc) - PT(T)) \div PT(i)$ additional modules. In the example, the *lib2* CLB Group is shared. Even though the *lib2* CLB Group's average processing time is doubled, the new processing time is lower than the original bottleneck processing time. Therefore, no new bottlenecks are created for the co-processors. The *lib1* CLB Group will not be shared even though it is not the co-processor bottleneck because if the *lib1* CLB Group is shared, *lib1* module will become the new bottleneck for both co-processors and the processing time will be 20. Similar to the previous algorithm, a module can either be placed on a free CLB Group or already configured CLB Group with its desired IP type. After a module has been placed on the tile that needs reconfiguration, the

remaining slots (buffer sets) will be added to the $List_{share_{IPtype(t)}}$ for sharing.

Table 9.1 Notation

n	Number of tiles
t_g	Number of modules in the graph g
IP_g	Set of IPtype required by the graph g
$\|IP_{g,i}\|$	Number of IPtype i required by graph g
$TileR_{g,i}$	Revivable tile configured as IPtype i which is required by the graph g
$TileS_{g,i}$	Sharable tile configured as IPtype i which is required by the graph g
CG_R_i	Center of gravity of all $TileR_{g,i}$
$CG_{proposed}$	Proposed center of gravity for the placement
CG_{revive}	Center of gravity of all selected revivable tiles
CG_{free}	Center of gravity of all free tiles
CG_{share}	Center of gravity of selected sharable slots
CG_{neigh}	Center of gravity of all neighbor modules
MD	Manhattan distance
$neigh(t)$	A set of neighbor modules of module t
$List_g$	Sorted graph's module list according their communication volumes in descending order
$List_{revive_i}$	List of selected $TileR_{g,i}$
$List_{free}$	List of free tiles
$List_{new}$	List of selected free tiles to be used
$List_{candidate_t}$	List of placement candidate of module t
$List_{share_i}$	List of selected IP i slots
Max_S	Number of buffer sets in a CLB Group
S_i	Set of sharable slots of IP i
$loc(x)$	Location of tile or slot x

9.4 Experiments

9.4.1 Experimental Setup

A co-processing request queue for the video system is randomly generated. The configuration bitstream size of each CLB group is 2.2 Mbits which equals 18,000 128-bit packets. The runtime (RT) and configuration time (CT) are measured in cycles and the energy is measured in nJ. Configuration time is the elapsed time interval between the system issuing reconfiguration signals until all tile reconfiguration is complete. Runtime is the time interval from when all tile reconfiguration is complete to when all data is processed. Note that runtime excludes configuration time.

The simulation parameters are shown in Table 9.2. The result is shown in Table 9.3 and Table 9.4.

1: //Sharable Tiles selection
2: **for all** $i \in IP_g$ **do**
3: **for all** tile $T_{i,j} \in TileS_{g,i}$ **do**
4: $\#shareslot_{T_{i,j}}$=Max_S;
5: **for all** $T_{i,j}$'s sharer graph g **do**
6: **if** (BN(acc)-PT($T_{i,j}$)) div PT(i) < $T_{i,j}$'s slotleft **then**
7: $\#shareslot_{T_{i,j}}$=(BN(g)-PT($T_{i,j}$)) div PT(i));
8: **end if**
9: **end for**
10: Add $T_{i,j}$'s sharable slots to S_i;
11: **end for**
12: **end for**
13: **if** $\sum_{i \in IP_g} |S_i| > 0$ **then**
14: $CG_{base} = \frac{\sum_{i \in IP_g} \sum_{s \in S_i} loc(s)}{\sum_{i \in IP_g} |S_i|}$;
15: **else**
16: $CG_{base} = CG_{free}$;
17: **end if**
18: //Sharable Tiles Selection
19: $\#tiles\_needed$=0;
20: **for all** $i \in IP_{g,i}$ **do**
21: **if** $|IP_{g,i}| \geq |S_i|$ **then**
22: Move all S_i to $List_{share(i)}$;
23: $Rtoken(i)$=$\lceil(|IP_{g,i}| - |S_i|)$ div (BN(g) div PT(i))$\rceil$;
24: $\#tiles\_needed$+=$Rtoken(i)$;
25: **else**
26: Sort $s \in S_i$ according to their MDs from the CG_{base} in ascending order;
27: Move the first $|IP_{g,i}|$ $s \in S_i$ from S_i to $List_{share_i}$;
28: **end if**
29: **end for**
30: **if** $\#tiles\_needed > |Tile_{Free}|$ **then**
31: return "Not enough free tile";
32: **end if**
33: //Reconfigurable Tile Selection
34: CG_{share}=$\frac{\sum_{i \in IP_g} \sum_{s \in List_{share_i}} loc(s)}{\sum_{i \in IP_g} |List_{share_i}|}$;
35: Sort $List_{free}$ according to their MDs from the CG_{share};
36: Move the first $\#tiles\_needed$ tiles from $List_{free}$ to $List_{new}$;

Algorithm 2 Bottleneck Aware Sharing Placement Algorithm for Sharable CLB Groups Part 1

1: //Place Graph's Modules
2: **for all** $Module\ t \in List_g$ **do**
3: **if** $RToken(IPtype(t))! = 0$ **then**
4: $List_{candidate_t}$=$List_{new}$+$List_{revive_{IPtype(t)}}$;
5: **else**
6: $List_{candidate_t}$=$List_{revive_{IPtype(t)}}$;
7: **end if**
8: **if** Some t's neighbor modules are mapped **then**
9: CG_{neigh}=$\frac{\sum_{m \in mapped\_neigh(t)} loc(m)}{|mapped\_neigh(t)|}$;
10: **else**
11: CG_{neigh}=$\frac{\sum_{m \in neigh(t)} CG_{IPtype(m)}}{|neigh(t)|}$ where $CG_{IPtype(m)}$=$\frac{\sum_{p \in List_{candidate_m}} loc(p)}{|List_{candidate_m}|}$;
12: **end if**
13: Map t on the tile $k \in List_{choices}$ with the smallest MD from CG_{neigh};
14: **if** $k \in List_{new}$ **then**
15: Add k to $List_{reconfigure_{IPtype(t)}}$;
16: Add ((BN(g) div PT(i))-1) entries of k to $List_{share_{IPtype(t)}}$;
17: $Rtoken(IPtype(t)) - -$;
18: **end if**
19: Remove k from either $List_{new}$ or $List_{share_{IPtype(t)}}$;
20: **end for**
21: //Bitstream Source Selection
22: **for all** $i \in IP\_g$ **do**
23: **if** $List_{reconfigure_i}$!=NULL **then**
24: Compute $CG_{List_{reconfigure_i}}$
25: **if** $\exists$ on-chip bitstream **then**
26: $Src_{bitstream}$=$Tile\_R_{g,i}$ with the smallest MD from $CG_{List_{reconfigure_i}}$;
27: **else**
28: $Src_{bitstream}$=I/O close to $CG_{List_{reconfigure_i}}y$;
29: **end if**
30: **end if**
31: **end for**

Algorithm 3 Bottleneck Aware Sharing Placement Algorithm for Sharable CLB Groups Part 2

Table 9.2 Baseline simulation parameters

Grid Size	20×20
Video System Simulation Cycles	$20,000,000$
Num. Unicast Virtual Channels on each port	1
Num. Multicast Virtual Channels on each port	2
Num. Samples	5
Flit Width	$128 - bit$
Num. of Jobs Request/Co-processor	2000
Path-Based Adaptive Routing observation window	5×5
Num. CLB Group Buffer Sets	$1 - 4$
Sharing Methods	*FTS* and *BAS*

Table 9.3 Full Tile Sharing (FTS)

# buffer sets	1	2	3	4
# Co-processing. Exec.	1325	1671	1787	1806
Avg. RT	533439	814311	1140823	1435264
Data Energy/Co-processing.	117979	156761	154981	153398
Bitstream Energy/Co-processing.	9473	2676	1682	1266
Avg. CLB Group active cycles	5564423	6628319	7012392	7093083
Avg. IO active cycles	2232097	2810802	3052118	3118726

Table 9.4 Bottleneck Aware Sharing (BAS)

# buffer sets	1	2	3	4
# Co-processing. Exec.	1325	1852	2094	2205
Avg. RT	533439	600098	660437	671230
Data Energy/Co-processing	117979	138597	138113	136026
Bitstream Energy/Co-processing	9473	4135	3290	2989
Avg. CLB Group active cycles	5564423	7501767	8447646	8882855
Avg. IO active cycles	2232097	3056009	3476610	3647329

9.4.2 Number of Sharers

In both *FTS* and *BAS*, the number of co-processor executions increases as the number of CLB Group buffer sets increases. The CLB Group and I/O also become more active. The router energy consumed by the configuration bitstream per co-processor execution decreases because CLB Group module sharing decreases the reconfiguration needs. The router energy consumed by the data packets increases. It implies reduced placement quality. The co-processor runtime increases because more co-processors are running on the system at the same time. The increased number of packets traversed between the modules increases data packet latency.

9.4.3 Comparison between Full Tile Sharing (*FTS*) and Bottleneck Aware Sharing (*BAS*)

Table 9.3 and Table 9.4 show that *BAS* has higher number of co-processor executions than *FTS*. The co-processor runtime in *BAS* increases because more co-processors are running on the system at the same time. The increased number of packets traversed between the modules increases data packet latency. This increase is smaller than *FTS* because *BAS* avoids bottleneck module sharing. The increases in the average CLB Group active cycles and the average *I/O* active cycles indicate that *BAS* leads to a more effective use of the CLB Groups and the *I/O*s. The router energy of the configuration bitstream in the system using *BAS* is higher than *FTS*. It is because *BAS* performs less aggressive sharing by sharing the non-bottleneck modules only. Therefore, the CLB Group chosen by *BAS* are more distant.

CHAPTER 10. Polymorphic Modules Placement

10.1 Motivation

FPGA can be configured as a specific module such as Discrete Cosine Transform (DCT) module or Advanced Encryption Standard (AES) module to speed up a specific function. It can also be configured as a soft microprocessor, *e.g.* Xilinx Microblaze, to handle a broad range of functions. The maturity of C-to-HDL technology allows code written in C to be translated into binary image for the soft microprocessor and FPGA bitstream for FPGA fabric. Therefore, the future on-chip system can choose a module running in either software mode or hardware mode to support a co-processor. Modules running in software mode usually provide lower throughput and consume more energy compared with the modules running in hardware mode. However, when the co-processor's expected runtime is short and there exists a configured microprocessor on the chip, running the modules in software mode allows the co-processor to start sooner without waiting for the tile reconfiguration. The module is ready to support the co-processor once the module instructions arrive at the soft processor L1 cache (configured by the block RAM). This characteristic allows the co-processor to complete its execution in a short time.

10.2 Throughput Expectation

The co-processor throughput ($Tput$) depends on three factors:

1. The number of tiles waiting to be configured (N_c)
2. The number of jobs to be processed (N_j)
3. The bottleneck of the data flow graph (T_b)

When an on-chip system runs a co-processor, some selected CLB groups are configured to support the co-processor's modules. The reconfiguration time R is closely related to the number of tiles waiting to be configured. The expected reconfiguration time $E(R)$ is a function of the number of tiles waiting to be reconfigured (N_c). $E(R) = f(N_c)$ where $f(N_c)$ can be obtained by profiling.

The co-processor throughput $Tput$ can be modeled by

$Tput = (f(N_c) + T_b \times (N_j))/N_j = f(N_c)/N_j + T_b$.

If the number of jobs to be processed by the co-processor is high, reconfiguration time can be ignored. The throughput is determined by the bottleneck of the co-processor only. Therefore, mapping the co-processor's bottleneck module in hardware mode can maximize the co-processor throughput. Note that the non-bottleneck modules in this co-processor could be mapped into a soft processor as long as they do not become the new bottleneck. If the number of jobs to be processed is small, reconfiguration time affects the co-processor's throughput. In this case, mapping the co-processor's bottleneck in software mode may increase the throughput by saving the long reconfiguration time.

A polymorphic co-processor can run in multiple modes. Each mode has different combination of hardware and software modules. It allows the system to choose the most suitable mode to execute in.

Polymorphic placement algorithm is very similar to the Algorithm 2 and 3 in Chapter 9. The main difference is that when the polymorphic placer places a polymorphic co-processor, it tries to map the modules of each co-processor mode to the CLB Groups. The mode with the highest expected throughput $E(Tput)$ will be executed.

10.3 Experiment

In this section, a new soft processor IP is added to the system. This IP supports any modules running in software mode. The processing time of the module running in soft processor mode is 2 times longer than the module running in hardware mode.

A co-processing request queue for the video system is randomly generated. The placer

places the video system into the on-chip FPGA system polymorphically. The configuration bitstream size of each CLB group is 2.2 Mbits which equals 18,000 128-bit packets. The runtime (RT) and configuration time (CT) are measured in cycles and the energy is measured in nJ. Configuration time is the elapsed time interval between the system issuing reconfiguration signals until all tile reconfiguration is complete. Runtime is the time interval from when all tile reconfiguration is complete to when all data is processed. Note that runtime excludes configuration time.

The simulation parameters are shown in Table 10.1. Each co-processing request has K data to be processed. K is a random number which ranges from 1 to Max_d where Max_d is $200, 500, 1000$. For each parameter Max_d, we run the experiment in polymorphic mode (Poly) and hardware only mode (Hw). The results are shown in Table 10.2.

Table 10.1 Polymorphic On-Chip System Simulation Parameters

Grid Size	20×20
Video System Simulation Cycles	$20,000,000$
Num. Unicast Virtual Channels on each port	1
Num. Multicast Virtual Channels on each port	2
Num. Samples	5
Flit Width	$128 - bit$
Max. Num. of Jobs Request/Co-processor	$200, 500, 1000$
Path-Based Adaptive Routing observation window	5×5
Num. CLB Group Buffer Sets	4
Sharing Methods	BAS

The introduction of soft processor IP increases system throughput. This increase is more significant when Max_d is small. The placer prefers mapping some modules in software mode to save configuration time to increase the throughput if the chip has configured a soft processor. When $Max_d = 200$, 44% of the co-processors use soft processor modules. The throughput advantage diminishes when Max_d increases because the effect of the reconfiguration time on the throughput becomes smaller. Running IP in hardware mode can provide higher throughput.

The introduction of soft processor decreases the number of CLB Group reconfigurations

because soft processor is capable of handling a broad range of IP functions. The average configuration time per co-processor execution and the average router energy for the configuration bitstream per co-processor execution, hence, decrease.

The existence of soft processor does not affect the placement quality. The average data energy per co-processor execution stays the same.

Co-processors using some soft processor mapped modules run slower than co-processors using all hard IPs because the IP running in software mode has higher processing time than in hardware mode.

Table 10.2 Polymorphic Module Placement

Max_d+Mode	200Hw	200Poly	500Hw	500Poly	1000Hw	1000Poly
# co-processing	16128	20385	9415	10521	5565	5751
# data processed	1611326	2053698	2339892	2651544	2795923	2875082
# All HW Mapping	16128	11353	9415	9653	5565	5472
# Some SW Mapping	0	9082	0	868	0	279
# configuration	9212	4122	4980	3705	2341	1977
avg. conf. time/co-processing	10893	5095	11134	8166	10223	8544
avg. conf. bitstream energy/co-processing	1379	526	1253	813	975	815
avg. data energy/co-processing	6764	6748	16815	16307	33021	33245
avg RT/data. (All HW)	613	510	432	398	366	370
avg RT/data. (Some SW)	NIL	734	NIL	455	NIL	541

CHAPTER 11. Conclusions

Future on-chip system will contain a large number of IPs. The on-chip system needs a high bandwidth 2D mesh on-chip network to provide the communication between these modules. In this thesis, we proposed an advanced router to improve the on-chip network performance. The router supports adaptive routing, unary and binary multicast code and on-the-fly code transformation to increase the network throughput and reduces the on-chip router energy consumption. A novel approach is proposed to make the router free from multicast deadlock.

On-chip FPGA system is one of the on-chip systems which requires a 2D mesh network. In this system, modules location affects system performance. In the second part of the thesis, we propose an algorithm to place co-processor modules as close as possible. Building on the algorithm, we develop two algorithms to support CLB Group sharing and to support a Polymorphic Co-processor. The advanced router and the algorithms can increase the throughput and reduce the network energy consumption of the on-chip system.

Bibliography

[1] Teraflops research chip, February 2007. `http://www.intel.com/pressroom/kits/Teraflops/index.htm`.

[2] Hideharu Amano. A survey on dynamically reconfigurable processors. *IEICE Transactions on Communications*, E-89-B(12):3179–3187, 2006.

[3] Giuseppe Ascia, Vincenzo Catania, Maurizio Palesi, and Davide Patti. Implementation and analysis of a new selection strategy for adaptive routing in networks-on-chip. *IEEE Trans. Comput.*, 57(6):809–820, 2008.

[4] J. Bainbridge and S. Furber. Chain: a delay-insensitive chip area interconnect. *Micro, IEEE*, 22(5):16–23, Sep/Oct 2002.

[5] T. Bjerregaard and J. Sparso. A router architecture for connection-oriented service guarantees in the mango clockless network-on-chip. *Design, Automation and Test in Europe, 2005. Proceedings*, pages 1226–1231 Vol. 2, March 2005.

[6] Tobias Bjerregaard and Shankar Mahadevan. A survey of research and practices of network-on-chip. *ACM Comput. Surv.*, 38(1):1, 2006.

[7] Eylon Caspi, Michael Chu, Randy Huang, Joseph Yeh, John Wawrzynek, and Andre DeHon. Stream computations organized for reconfigurable execution (SCORE). In *FPL*, pages 605–614, 2000.

[8] Chi-Ming Chiang and Lionel M. Ni. Multi-address encoding for multicast. In *PCRCW '94: Proceedings of the First International Workshop on Parallel Computer Routing and Communication*, pages 146–160, London, UK, 1994. Springer-Verlag.

[9] Andrew A. Chien and Jae H. Kim. Planar-adaptive routing: low-cost adaptive networks for multiprocessors. *J. ACM*, 42(1):91–123, 1995.

[10] Ge-Ming Chiu. The odd-even turn model for adaptive routing. *IEEE Trans. Parallel Distrib. Syst.*, 11(7):729–738, 2000.

[11] Pat Conway and Bill Hughes. The amd opteron northbridge architecture. *IEEE Micro*, 27(2):10–21, 2007.

[12] Matteo Dall'Osso, Gianluca Biccari, Luca Giovannini, Davide Bertozzi, and Luca Benini. xpipes: a latency insensitive parameterized network-on-chip architecture for multi-processor socs. In *ICCD '03: Proceedings of the 21st International Conference on Computer Design*, page 536, Washington, DC, USA, 2003. IEEE Computer Society.

[13] W. J. Dally and C. L. Seitz. Deadlock-free message routing in multiprocessor interconnection networks. *IEEE Trans. Comput.*, 36(5):547–553, 1987.

[14] William Dally and Brian Towles. *Principles and Practices of Interconnection Networks*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2003.

[15] William J. Dally. Performance analysis of k-ary n-cube interconnection networks. *IEEE Trans. Comput.*, 39(6):775–785, 1990.

[16] International Technology Roadmap for Semiconductors. *Executive Summary*. http://www.itrs.net/Links/2007ITRS/, 2007 Edition.

[17] Mark Gebhart, Bertrand A. Maher, Katherine E. Coons, Jeff Diamond, Paul Gratz, Mario Marino, Nitya Ranganathan, Behnam Robatmili, Aaron Smith, James Burrill, Stephen W. Keckler, Doug Burger, and Kathryn S. McKinley. An evaluation of the trips computer system. *SIGPLAN Not.*, 44(3):1–12, 2009.

[18] Christopher J. Glass and Lionel M. Ni. The turn model for adaptive routing. *SIGARCH Comput. Archit. News*, 20(2):278–287, 1992.

[19] K. Goossens, J. Van Meerbergen, A. Peeters, and P. Wielage. Networks on silicon: Combining best-effort and guaranteed services. In *In Proc. Design, Automation and Test in Europe Conference and Exhibition (DATE)*, pages 423–425, 2002.

[20] P. Gratz, B. Grot, and S.W. Keckler. Regional congestion awareness for load balance in networks-on-chip. *High Performance Computer Architecture, 2008. HPCA 2008. IEEE 14th International Symposium on*, pages 203–214, Feb. 2008.

[21] Pierre Guerrier and Alain Greiner. A generic architecture for on-chip packet-switched interconnections. In *DATE '00: Proceedings of the conference on Design, automation and test in Europe*, pages 250–256, New York, NY, USA, 2000. ACM.

[22] Richard Herveille and Andy Henson. Video compression systems. `http://www.opencores.org`.

[23] D. Kanter. The common system interface: Intel's future interconnect. http://www.realworldtech.com/page.cfm? ArticleID=RWT082807020032.

[24] Faraydon Karim, Anh Nguyen, and Sujit Dey. An interconnect architecture for networking systems on chips. *IEEE Micro*, 22(5):36–45, 2002.

[25] Ka-Ming Keung and Akhilesh Tyagi. Breaking adaptive multicast deadlock by virtual channel address/data fifo decoupling. In *NoCArc '09: Proceedings of the 2nd International Workshop on Network on Chip Architectures*, pages 11–16, New York, NY, USA, 2009. ACM.

[26] Jongman Kim, Chrysostomos Nicopoulos, and Dongkook Park. A gracefully degrading and energy-efficient modular router architecture for on-chip networks. *SIGARCH Comput. Archit. News*, 34(2):4–15, 2006.

[27] S. Kumar, A. Jantsch, J.-P. Soininen, M. Forsell, M. Millberg, J. Oberg, K. Tiensyrja, and A. Hemani. A network on chip architecture and design methodology. *VLSI, 2002. Proceedings. IEEE Computer Society Annual Symposium on*, pages 105–112, 2002.

[28] Shashi Kumar, Axel Jantsch, Mikael Millberg, Johny Oberg, Juha-Pekka Soininen, Martti Forsell, Kari Tiensyrja, and Ahmed Hemani. A network on chip architecture and design methodology. *isvlsi*, 00:0117, 2002.

[29] Ming Li, Qing-An Zeng, and Wen-Ben Jone. Dyxy: a proximity congestion-aware deadlock-free dynamic routing method for network on chip. In *DAC '06: Proceedings of the 43rd annual Design Automation Conference*, pages 849–852, New York, NY, USA, 2006. ACM.

[30] X. Lin, P. K. McKinley, and L. M. Ni. Deadlock-free multicast wormhole routing in 2-d mesh multicomputers. *IEEE Trans. Parallel Distrib. Syst.*, 5(8):793–804, 1994.

[31] Roman Lysecky, Greg Stitt, and Frank Vahid. Warp processors. *ACM Trans. Des. Autom. Electron. Syst.*, 11(3):659–681, 2006.

[32] Srinivasan Murali and Giovanni De Micheli. Sunmap: a tool for automatic topology selection and generation for nocs. In *DAC '04: Proceedings of the 41st annual conference on Design automation*, pages 914–919, New York, NY, USA, 2004. ACM.

[33] P.P. Pande, C. Grecu, A. Ivanov, and R. Saleh. Design of a switch for network on chip applications. In *Circuits and Systems, 2003. ISCAS '03. Proceedings of the 2003 International Symposium on*, volume 5, pages V–217–V–220 vol.5, May 2003.

[34] Li-Shiuan Peh and W.J. Dally. A delay model for router microarchitectures. *Micro, IEEE*, 21(1):26–34, Jan/Feb 2001.

[35] Faizal Samman, Thomas Hollstein, and Manfred Glesner. Planar adaptive router microarchitecture for tree-based multicast network-on-chip. In *NoCArc '08: International Workshop on Network on Chip Architectures*, 2008.

[36] Karthikeyan Sankaralingam, Ramadass Nagarajan, Haiming Liu, Changkyu Kim, Jaehyuk Huh, Doug Burger, Stephen W. Keckler, and Charles R. Moore. Exploiting ilp, tlp, and dlp with the polymorphous trips architecture. *SIGARCH Comput. Archit. News*, 31(2):422–433, 2003.

[37] Karin Strauss, Xiaowei Shen, and Josep Torrellas. Uncorq: Unconstrained snoop request delivery in embedded-ring multiprocessors. In *MICRO '07: Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 327–342, Washington, DC, USA, 2007. IEEE Computer Society.

[38] Steven Swanson, Ken Michelson, Andrew Schwerin, and Mark Oskin. Wavescalar. In *MICRO 36: Proceedings of the 36th annual IEEE/ACM International Symposium on Microarchitecture*, page 291, Washington, DC, USA, 2003. IEEE Computer Society.

[39] Steven Swanson, Andrew Schwerin, Martha Mercaldi, Andrew Petersen, Andrew Putnam, Ken Michelson, Mark Oskin, and Susan J. Eggers. The wavescalar architecture. *ACM Trans. Comput. Syst.*, 25(2):4, 2007.

[40] Michael Bedford Taylor, Walter Lee, Saman Amarasinghe, and Anant Agarwal. Scalar operand networks: On-chip interconnect for ilp in partitioned architectures. In *HPCA '03: Proceedings of the 9th International Symposium on High-Performance Computer Architecture*, page 341, Washington, DC, USA, 2003. IEEE Computer Society.