

國立交通大學

電子工程學系 電子研究所碩士班

碩士論文

單晶片多處理器系統的通訊交換器設計

A Switch Design for Multi-Processor System on Chip

研究生：黃保瑞

指導教授：周景揚 博士

中華民國九十三年七月

單晶片多處理器系統的通訊交換器設計

A Switch Design for Multi-Processor System on Chip

研究生：黃保瑞

Student : Pao-Jui Huang

指導教授：周景揚 博士

Advisor : Dr. Jing-Yang Jou

國立交通大學
電子工程學系 電子研究所碩士班
碩士論文



A Thesis
Submitted to Department of Electronics Engineering
College of Electrical Engineering and Computer Science
National Chiao Tung University
in partial Fulfillment of the Requirements
for the Degree of
MASTER OF SCIENCE
in

Electronics Engineering

July 2004

Hsinchu, Taiwan, Republic of China

中華民國九十三年七月

單晶片多處理器系統的通訊交換器設計

研究生：黃保瑞

指導教授：周景揚博士

國立交通大學

電子工程學系 電子研究所碩士班



隨著半導體製程的不斷進步，十年後，IC 工程師將有可能在單一個晶片上整合上百個運算元件。此時，各個元件間的通訊將會成為影響系統效能的一大關鍵。IC 設計工程師將需要一個能考慮通訊效能的系統設計方法。在這篇論文中，我們提出了一個適用於單晶片上多處理器的通訊架構。經由適當的設定，這個架構將可以提供不同的資料交換機制。由於我們的通訊架構具有能預測通訊效能的特性。系統設計者可以利用我們的架構在設計初期便分析系統效能以作出更好的決定。相關的實驗也顯示我們的架構能夠有效的傳遞資料。

A Switch Design for Multi-Processor System-on-Chip

Student : Pao-Jui Huang

Advisor : Dr. Jing-Yang Jou

Department of Electronics Engineering

Institute of Electronics

National Chiao Tung University



ABSTRACT

Driven by the advance of semiconductor technology, it is possible to integrate hundreds of processing elements on a single chip in the next decade. At the moment, communication between the components will become the limiting factor for system performance and a communication-driven system design methodology will be needed. In this thesis, we propose an on-chip communication infrastructure for multi-processor system-on-chip. By appropriate configuration, the network can work as circuit switching, packet switching, and dedicated bus. System designers can also benefit from our framework to analyze the system performance and make better decisions at higher level

because our platform exhibits predictable performance. The experiments of performance evaluation show that the communication fabrics can efficiently transfer data within system.



Acknowledgements

I would like to express my sincere gratitude to my advisors, Professor Jing-Yang Jou for his suggestion and guidance throughout the course of this thesis. I am also indebted to Cheng Yeh Wang and Lin Yu Ling for their great help on my research. Special thanks to all members in the EDA lab and my friends in Mountain Club for their friendship. Finally, I would like to show my appreciation to my family and Mei Hsuan Chen for their love and encouragement.



Contents

摘要..... i

Abstract..... ii

Acknowledgements..... iv

Contents..... v

Lists of Tables..... vii

Lists of Figures..... viii

Chapter 1 Introduction 1

 1.1 Technology trend 2

 1.2 On-chip network 3

 1.3 Communication-driven system design methodology 4

 1.4 Related works 6

 1.5 The focus of this thesis 7

Chapter 2 Preliminaries 9

 2.1 Topology 9

 2.2 Switching strategy 13

 2.3 Routing algorithm..... 16

 2.4 Transaction protocol 18

Chapter 3 Our Platform and Switch Design..... 19

 3.1 What network we need 20

 3.2 Switch architecture 22



3.3	Transaction	30
3.4	Transaction protocol.....	33
3.5	Round robin scheduling.....	35
3.6	Performance.....	37
3.7	Design overview	39
Chapter 4	Experimental Results.....	42
4.1	Definition.....	43
4.2	Experiment	43
4.3	Synthesis report	47
Chapter 5	Conclusions and Future Work	48
Vita.....		53



List of Tables

Table 1 : Histograms of normalized latency under different injection rate	44
Table 2 Histograms of normalized latency under different buffer size of virtual channel	46
Table 3 : Synthesis report	47



List of Figures

Figure 1 : Protocol stack of inter-network.....	4
Figure 2 : Communication-driven system design methodology [4]	5
Figure 3 : Orthogonal network topology	10
Figure 4 : Other direct network topology	11
Figure 5 : Indirect network topology	12
Figure 6 : Deadlock situation	15
Figure 7 : Virtual channel	16
Figure 8 : A taxonomy for routing algorithms [7]	18
Figure 9 : A 2-D mesh switched network with 2x3 nodes.....	22
Figure 10 : Switch architecture.....	23
Figure 11 : Basic transmission procedure.....	24
Figure 12 : Switch interface	24
Figure 13 : Input stage of switch port.....	25
Figure 14 : Output stage of switch port	26
Figure 15 : Memory duty diagram.....	27
Figure 16 : Ack controller.....	28
Figure 17 : Organization of memory hierarchy	29
Figure 18 : Buffer naming rule	29
Figure 19 : Path transaction procedure	32
Figure 20 : Transaction protocol between switches	33

Figure 21 : Output arbitration..... 35

Figure 22 : Round robin scheduling 36

Figure 23 : Bandwidth sharing example..... 37

Figure 24 : Bandwidth guaranteed transmission path 39

Figure 25 : Assigning different path to avoid traffic congestion..... 40

Figure 26 : Fault tolerance..... 41



Chapter 1

Introduction



As the semiconductor technology advances, SoCs in the next decade are expected to integrate hundreds of computing elements on a single chip to obtain more computing power. However, designers encounter some new problems: wire delay becomes the limiting factor for the signal delay, communications between computing components become the bottleneck of system performance, and system design becomes more difficult because of more and more components integrated together. Moreover, it is observed that traditional design flow is incapable of solving these problems. Designers need not only new system architectures but also a new design methodology to conquer these problems and to reduce the time to market.

1.1 Technology trend

As predicted by *International Technology Roadmap of Semiconductors (ITRS)*, it is possible to integrate multi-billion transistors on a single chip within ten years [9]. The chip fabricated by 50nm technology can work at around 10GHz or faster. At the moment, wire delay will dominate the signal delays [16]. The gate delay of a transistor is scaled down linearly, whereas wire delay remains constant with scaling. Although larger wire delay can be managed with wire pipelining techniques, it is unavoidable for designers to deal with the problem of timing uncertainty.

Clock synchronization in future system is another problem for the system designers. Because the clock skew is not negligible any more, synchronizing all components on the chip with single clock will become almost impossible. The fact that global wire which spans the whole chip, like the clock signal, may conduct signals with latency that exceeds the clock cycle will also make the system synchronization problem more serious. The *globally-asynchronous, locally synchronous (GALS)* technique may be the most possible solution [17].

Performance is still the most important issue. Traditional shared-medium network architecture, like AMBA bus [13], is the most convenient architecture in current SoC integration. Such kinds of architectures can support broadcast transmission, and cost low implementation overhead. However, high contentions among masters caused by simultaneous requests degrade the system performance and make extra power

consumptions. These problems make designers discard share-bus architecture and search for some new communication architectures. Some studies propose different solutions like routing packet without wire [2], or using different topologies for specific applications [10]. There are still open problems in selecting a suitable architecture for different application domain.

Another important issue is the time to market. It becomes more complex to design a system due to integrating more components. Traditional design flow is not sufficient to conquer this problem. The design trend is toward system level design. The impact that communication becomes bottleneck of system performance also influences the system design methodology. A communication-driven design methodology should be considered.



1.2 On-chip network

It seems a feasible way to solve these new problems by applying similar concepts that are maturely developed in other fields. Some researchers adapt the layer method from traditional inter-network to manage the on-chip communication [4][8]. They view the interconnections between on-chip components as micron-network and apply the layer method to build the on-chip communication infrastructure. Figure 1 is the protocol stacks paradigm adapted from inter-network; with bottom up construction, the layers spans increasing design abstraction level [4]. System designers and architecture designers can work together to implement a system under different abstraction levels. Also, this

method maximizes the ability of reusing components.

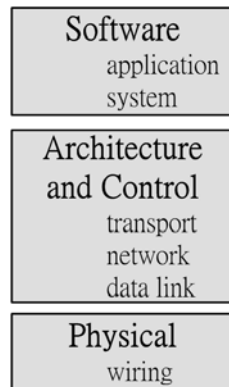


Figure 1 : Protocol stack of inter-network

Although the researches of constructing inter-network are well-studied, there are some differences between communication infrastructure of SoC and wide area network. There are some unique features of SoC network: less non-determinism of communication among applications, more strict constraints of power consumption, less memory space for on-chip system and the physical issue of fabrication.

1.3 Communication-driven system design methodology

Driven by the advances of semiconductor technology, future SoCs will accelerate the capacity and complexity of a system. It is possible for designers to integrate hundreds of functional components within the same die size to obtain more computing power. At this scale, it is believed that SoC will be implemented by using pre-designed components in a plug-and-play fashion [11]. In such system integration approach, communications among computing components becomes the most critical factor and make it more

complex to design the system and to predict the system performance. It is an urgent issue to balance the communication and computation power over the whole system.

Here, we introduce a communication-driven system design methodology, as shown in Figure 2, as a new design flow.

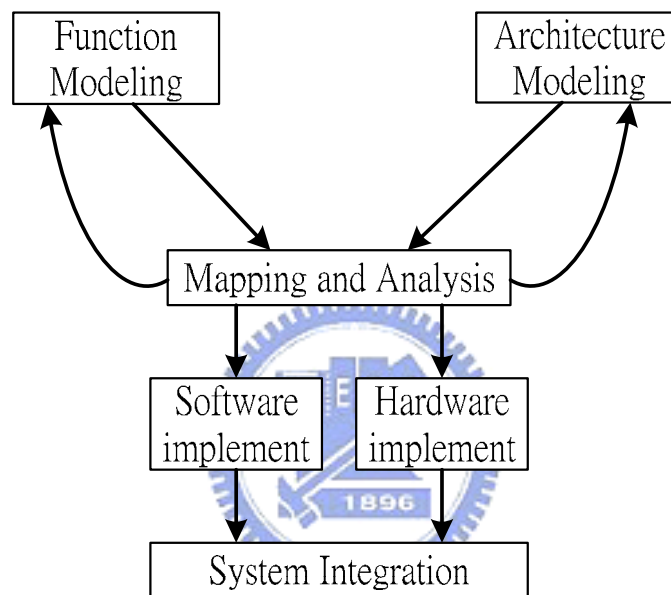


Figure 2 : Communication-driven system design methodology [4]

In this flow, we first separate the system design into two parts: functional modeling and architecture modeling. Functional modeling contains application modeling, task partition and job scheduling. Designers can collect rough system information by profiling the applications. The responsibility of architecture modeling is providing a library that contains the simulation models of various computation, memory components and communication fabrics as implementation choices. According to the profiling information of applications, system designers will make decision to select appropriate

platform.

After function and architecture modeling, system designers will map and allocate the tasks scheduled onto the platform which they chose in architecture modeling. By performance evaluation of these implementation examples, designers can get more detailed information of the whole system and make better design trade-off. With this method, system designer can refine the implementation of design at higher level and reduce the times of try and error.

Traditional design flows will follow after system designers decide the details of implementation.

1.4 Related works



There have been various studies in this field. We present some of them which are strongly related to our work.

1.4.1 Communication-based design flow

The design methodology for communication-based design is proposed in [3]. In this paper, researchers propose a “network-on-chip” approach to partition the communication into layers to maximize the reuse and provide programmers with an abstraction of the underlying communication framework. The OSI Reference Model is adapted to the layer approach and is demonstrated with a reconfigurable DSP example.

1.4.2 Network architecture

The analysis of why the shared bus, which dominates the system integration now, will not meet the performance requirement of future system is proposed in [1]. They present an alternative interconnection in the form of switching network. Such technique is well used in parallel computing, but is also suitable for heterogeneous communication between on-chip processors.

The technique of mapping applications onto targeted communication platform and analyzing system performance are studied in [14] and [15]. In their experiments, they discussed the relative strengths and weakness of the considered architectures for system design.

Some studies address the network fabric design. A circuit switching architecture, the SoCBUS, is proposed in [5]. SoCBUS has very good properties in providing guaranteed bandwidth, and is suitable to build the real time system. However, it is not suitable for general purpose computing that exhibits random traffic patterns. A hybrid router design between packet-switching and circuit-switching was proposed in [6]. It exhibits the property of both switching techniques but still suffers the lack of channel utilization.

1.5 The focus of this thesis

Based on the promise that communication will become the bottleneck of future system performance and on-chip communication will be treated as micron-network. We

propose a novel network platform and related infrastructure for on-chip communication in this thesis. System designers can also benefit from our framework to analyze the system performance and make better decisions at higher level because our platform exhibits predictable performance.

The rest of this thesis is organized as follows. In Chapter 2, we introduce basic network concepts. In Chapter 3, we highlight the requirements of future network and present details of our platform design and a novel switch design for on-chip communication. We prove the correctness of our platform and study some design space explorations in Chapter 4. Finally, we give the conclusion and future work in Chapter 5.



Chapter 2

Preliminaries



In this chapter, we introduce some basic concepts and related issues about constructing network. This chapter provides background knowledge of our platform and switch design in Chapter 3.

2.1 Topology

The word “topology” defines how the nodes are interconnected by channels and is usually modeled by a graph [7]. The nodes include communication fabrics, bridges and processors. Major network topologies can be categorized as *direct* network and *indirect*

network. In direct network, nodes are connected directly with each other by the network. In indirect network, nodes are connected by one or more intermediate node switches. The switching nodes perform the routing and arbitration operations. Because of different performance requirements and cost trade-off, many different network topologies are designed for specific applications [11]. We are going to give a brief description of some of the popular network topologies.

2.1.1 Direct network topologies

1. Orthogonal

A network topology is orthogonal if and only if nodes can be arranged in an orthogonal n -dimensional space. The most popular direct networks are k -ary n -dimensional mesh, k -ary n -dimensional cube and the hypercube, as shown in Figure 3. Such kinds of topologies exhibit the properties of regularity and symmetry.

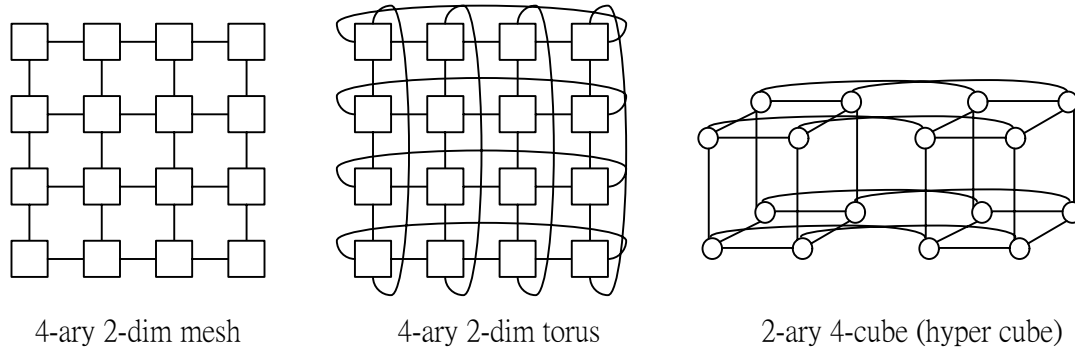


Figure 3 : Orthogonal network topology

2. Other direct network topologies

In addition to these topologies defined above, there are many other topologies that have been proposed with different properties, as shown in Figure 4. The cube-connected-cycles topology is proposed as an alternative way to orthogonal topologies to reduce the degree of each node. Tree topology provides the advantage of low implementation cost, in which each of these nodes on the topology is in turn connected to a disjoint set of descendants. A star graph is proposed to minimizing the network diameter of cube-connected cycles. However, it need more complex routing algorithm.

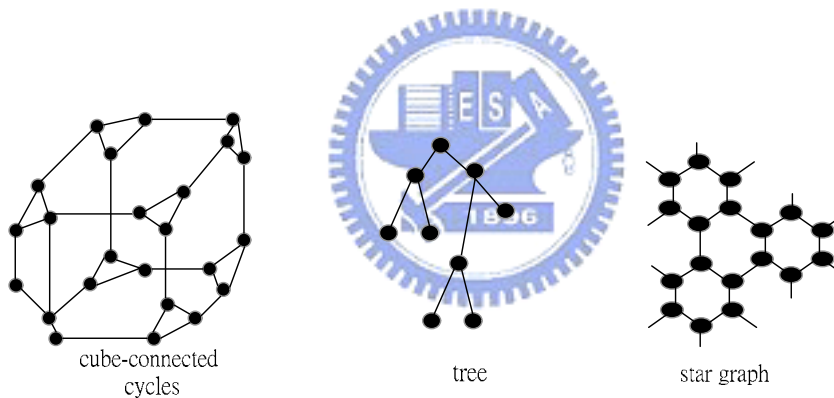


Figure 4 : Other direct network topology

2.1.2 Indirect network topologies

1. Crossbar networks

Crossbar networks allow any node in the system to communicate with any other node directly, as shown in Figure 5. In such way, several processors or memories can communicate simultaneously without contention. The disadvantage of crossbar networks

is the cost, and has been traditionally used in small-scale system [11].

2. Multi-stage interconnection network

Multi-stage interconnection networks (MIN) connect the input nodes to output nodes through switch stages, which are crossbar network. The number of stages and connections between switch stages determine the routing capability of the networks. Depending on the interconnection scheme employed between two adjacent nodes, various MINs have been proposed.

Fat-tree is one classical topology of MIN. A fat-tree network can provide multiple data paths from source node to destination nodes depending on the path usage. As shown in Figure 5, the latency is directly proportional to the depth of the tree.

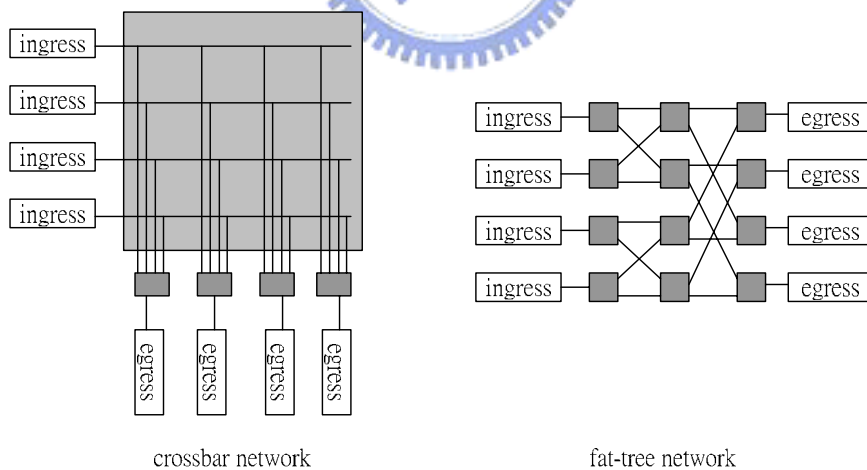


Figure 5 : Indirect network topology

Among these topologies, 2-D mesh is considered as the most suitable topology for on-chip network because the 2-D mesh has the advantages of an acceptable wire cost,

reasonably high bandwidth, and that it is easy to group components on plane.

2.2 Switching strategy

Switching strategy is defined as the method used to exchange data between network components. Common switching strategies can be classified into two categories: *connection-oriented* and *connection-less*.

Connection-oriented switching technique is widely used in telecommunication. It is also named *circuit switching* because the connection from source to destination is built before data transmission. Once the connection established, data from source to destination can be transmitted with guaranteed bandwidth and will be delivered without any contention. With this advantage, we can employ it to build a real time system. This strategy is advantageous when data transmission is long and few.

Alternative to connection-oriented switching strategy, connection-less switching strategy partitioned data into several packets before transmission. The routing and transmission of packets are handled by network fabrics individually. Without any reservations of the channel bandwidth, it provides more efficient bandwidth utilization. Common types of communication-less switching include *store-and-forward*, *virtual-cut-through* and *wormhole switching*.

Store-and-forward switching technique is named because each packet transmitted in network is completely buffered at each intermediate node before it is forwarded to the

next node. The header information of each packet is extracted by the intermediate switch to determine the output destination over which the packet is to be forwarded. Different from the circuit switching, store-and-forward switching is advantageous when the messages are short and frequent. However, the implementation of store-and forward switching is expensive because a switch should have enough buffer size to hold a whole packet.

Unlike the store-and-forward switching, that switch should hold the whole packet before it is forwarded to next switch, virtual-cut-through switching can start the transmission as soon as the routing decision of packet is determined and the output channel is free. Actually, the packet doesn't even have to be stored at the output buffer and can cut through to the input of the next switch before the complete packet is received at current switch. In the absence of blocking, virtual-cut-through switching performs better than store-and-forward switching because the packet is effectively pipelined through successive switches. If the header of packet is blocked on a busy output channel, virtual-cut-through switching will hold the complete message in the switch and behaves like store-and-forward switching.

The requirement to buffer whole packet in the switches makes it difficult to construct a faster and smaller switches. In wormhole switching, packets are pipelined through the network like virtual-cut-through switching. However, the buffer requirements with switches are reduced over that for virtual-cut-through switching. If the packet is blocked in the network, the buffer in the switch doesn't have the capability of buffering the

whole packet; the blocked packets will occupy buffers in several switches. This degrades the network performance because the packet blocked by other packets will occupy buffers in these switches on part of its transmission path, similarly blocking other packets. Moreover, it often causes deadlock problem to happen. A deadlock situation is the network state that some packets cannot advance toward their destination because the buffers requested by them are full. As shown in Figure 6, all the packets involved in a deadlocked configuration are blocked forever [7].

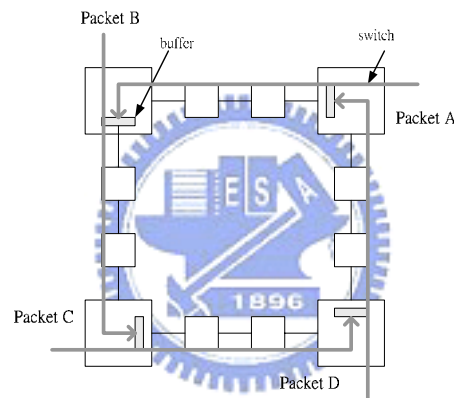


Figure 6 : Deadlock situation

Virtual channels are originally introduced to solve the problem of deadlock in wormhole switching. The key idea is to multiplexing the physical channel to support several virtual channels. Logically, each virtual channel is operating as if a distinct physical channel operates at lower bandwidth. By providing two virtual channels at output channel at each switch in Figure 7(virtual channels), all the packets blocked in the switches continue to make progress with half the channel bandwidth as shown in Figure 7(virtual channels solve the deadlock problem). This technique can not only solve

deadlock problem but also improve network throughput.

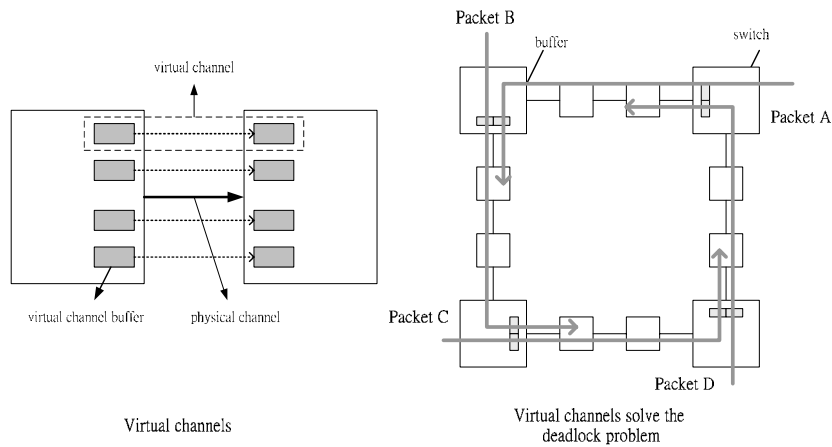


Figure 7 : Virtual channel

2.3 Routing algorithm

Routing algorithms determine the path followed by each packet. Figure 8 presents a taxonomy of routing algorithms that are classified according to several criteria. Routing algorithms can be first classified according to the number of destinations. Packets may have only one destination or be broadcasted to multiple destinations. Routing algorithm can also be classified according to the place where the routing decisions are made. The decision can be made centralized at the source (centralized routing), be determined in a distributed manner while across the network (distributed routing), or hybrid schemes. Moreover, routing algorithms can be classified according to the way they are implemented. The most popular ways consists of either looking at a routing table or executing a routing algorithm in software and hardware based on finite state machine. In

both cases, they can be either deterministic or adaptive according to whether the packet transmitted between a given source/destination pair is supplied with the same path. Adaptive routing can also be classified according to their progressiveness as progressive and backtracking. Progressive routing moves the header forward, reserving a new channel at each routing operation. Backtracking allow the header to backtrack while it is blocked. Backtracking routing algorithms are mainly used for fault tolerance. In the scope of adaptive routing, routing algorithms can be classified according to the distance of routing path as profitable or misrouting. Profitable routing algorithms always deliver the packet closer to the destination across the network, while misrouting algorithms may send packet away from the destination. The last taxonomy is according to the number of paths as completely adaptive or partially adaptive.



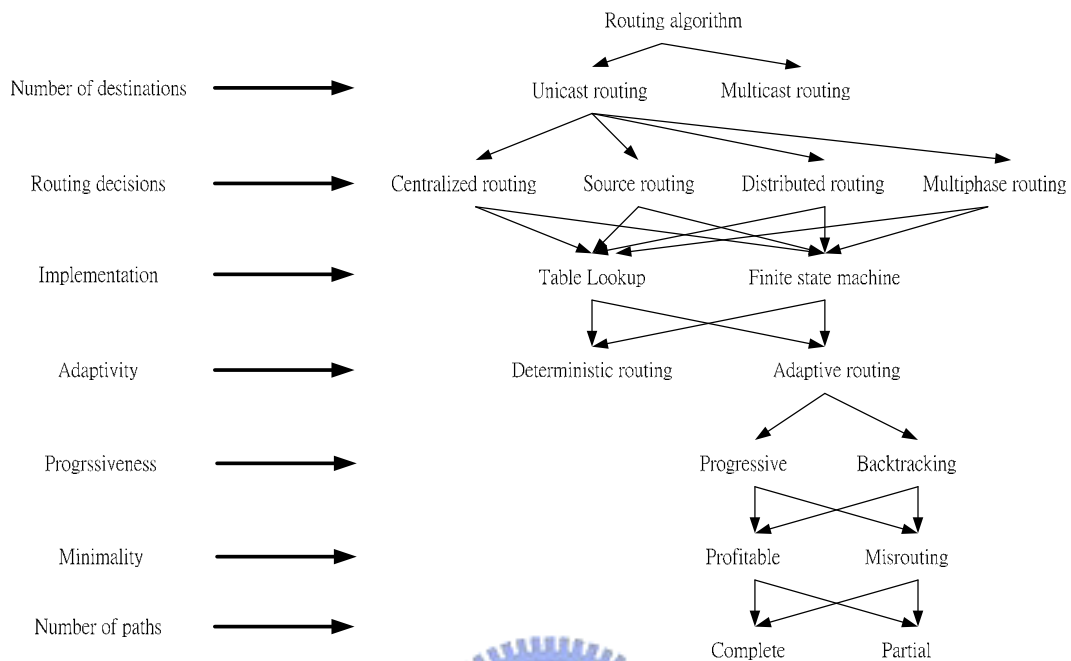


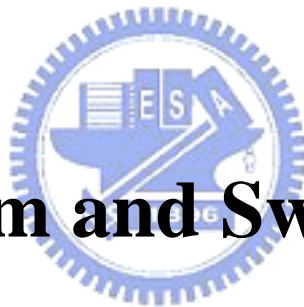
Figure 8 : A taxonomy for routing algorithms [7]

2.4 Transaction protocol

In network or telecommunication field, protocol is an agree-upon format or a set of rules for transmitting data between two devices. The protocols determine the type of error detection or error correction to be used, the data compression methods, or handshaking convention between sending device and receiving device. It not only defines how senders and receivers execute the communication transactions, but also determines how data flows across the network. For on-chip communication, different protocol options greatly influence the reliability and power consumption issues.

Chapter 3

Our Platform and Switch Design



In this chapter, we will describe our platform and switch design in detail. First, we will remark what future network infrastructure should provide for communication-driven system design methodology. Following, we will present a complete description of our platform and switch design. We will also present how to use the switch to transmit messages between components through illustrations. After that, we will review how our platform meets these requirements of constructing future on-chip network.

3.1 What network we need

When it comes to constructing the network infrastructure for future system on chip, the hardest problem is to meet the various communication requirements in different application domains. Some applications, such as Software Defined Radio and MPEG codec, can be thread paralleling processing and they just need local and fixed communication bandwidth. For other applications, there may be irregular traffic load among communication channels. Here we summarize some basic concepts what future communication infrastructure should provide in communication-driven system design flow.

1. Efficient communication

When we consider constructing a network infrastructure for system on chip, the first task is to balance computing power and communication capability. If we implement a system by integrating some powerful processing elements that coordinate with other components, yet we only provide a poor communication infrastructure. The first problem is that messages transmitted between these components will waste unnecessary time on transmitting. Lots of jobs assigned to processing elements will be postponed because the data needed is delayed. This is a serious problem which not only makes processing elements idle to degrade the whole system performance, but also make extra power consumption

while processing elements wait for data. Thus, we must provide a network infrastructure that meets high network utilization criticism.

2. Guaranteed throughput

For some applications, real time requirement is the critical issue. Circuit switching may be a good choice for such kinds of applications because it provides the transmission with guaranteed throughput.

3. Fault tolerance capability

Even with the advance of semiconductor technology, it still cannot be promised to fabricate a perfect chip without any error on the chip. This problem becomes worse in deep-submicron era. When it comes to integrating several processing elements on a chip, there is better chance that we will find some manufacturing faults in it. There may be faulty fabrics in memory, wrong connections between components, or breaking down processing elements. These issues can be rare but unavoidable. Future network infrastructure should provide some mechanisms such that the whole system still works smoothly with faulty components on it.

In this chapter, we propose a novel platform and switch design as a feasible solution to these network requirements and as the network infrastructure for the future communication-driven system design methodology.

3.2 Switch architecture

3.2.1 System Scheme

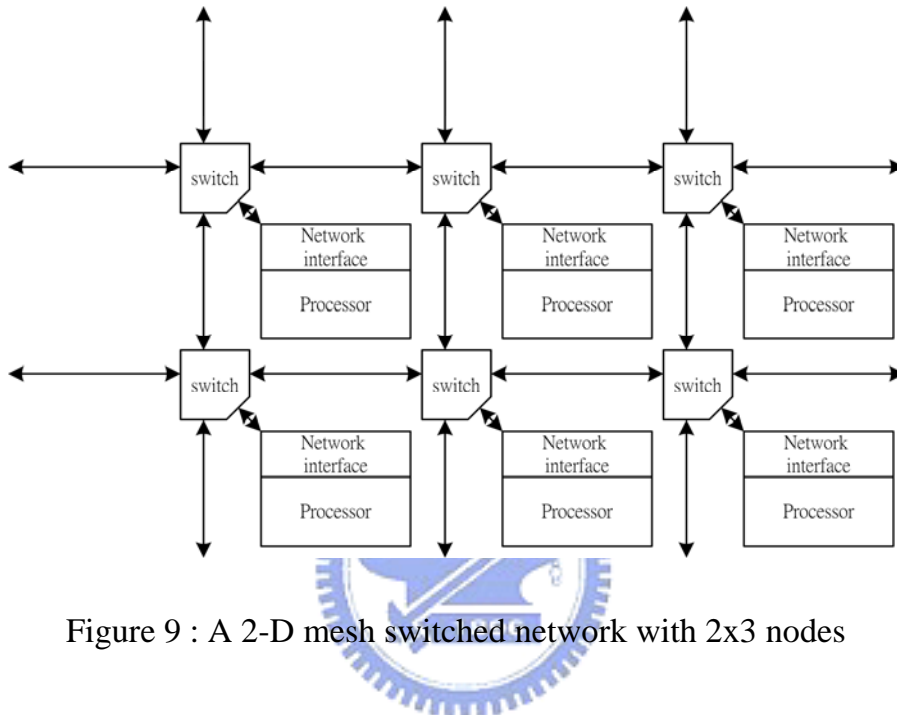


Figure 9 : A 2-D mesh switched network with 2x3 nodes

Our platform uses a 2-D mesh topology to organize on-chip components, as shown in Figure 9. The main reason for selecting the two dimensional mesh is its acceptable wire cost, and that it is easy to group components on plane [5][11]. In our platform, the network is composed of 5-ports switches. Processors use network interface to communicate within network.

The architecture of 5-ports switch is shown in Figure 10. The switch has four ports connecting to neighboring switches and one port connecting to local processing element. Each port is composed of input and output stage, which is shown in Figure 13 and Figure 14.

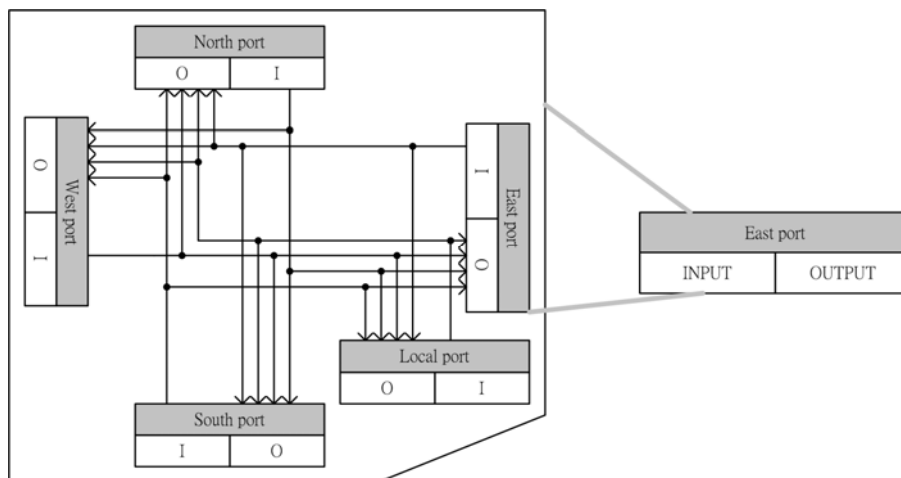


Figure 10 : Switch architecture

The basic transmission procedure is illustrated in Figure 11. Suppose that a packet is sent into current switch from the neighboring switch at west direction and will be delivered forward the neighboring switch at east direction. In current switch, the packet will be received by input stage of west port first and be stored in memory of output stage of east port. Once the output channel of east port which is connecting to the neighboring switch is available, the output stage of east port in current switch will send the packet to the next switch soon.

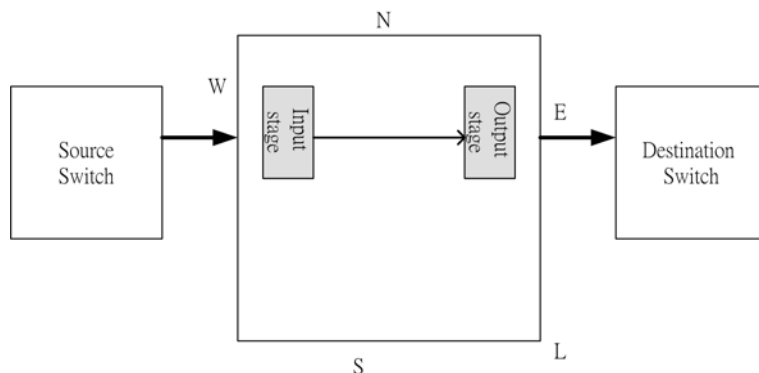


Figure 11 : Basic transmission procedure

The interface of switch is composed of input and output channel. Each channel contains Address-line, Data-line and Ack-line. We show that in Figure 12. The Address-line delivers the input or output address of the packet. The Data-line delivers data transmitted. And the Ack-line feeds acknowledgement back to source switch or processing elements to report the result of transmission. Output channel and input channel are complementary to each other.

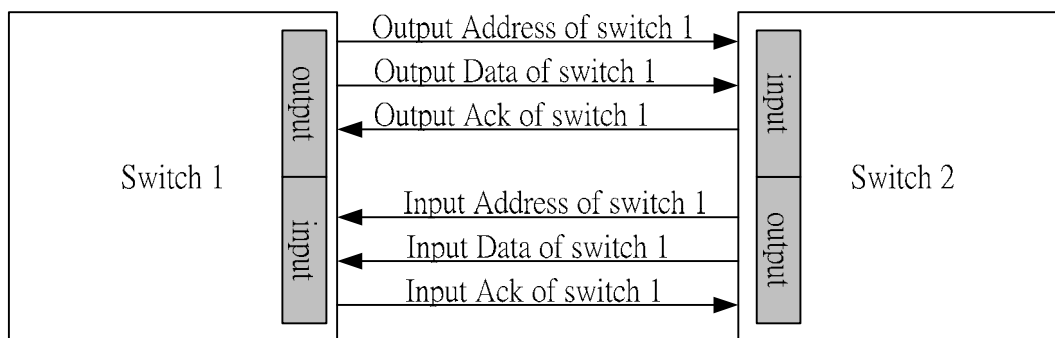


Figure 12 : Switch interface

After basic introduction of switch architecture, we explain the architecture of input, output stage and the organization of memory hierarchy in detail.

3.2.1.1 Input stage

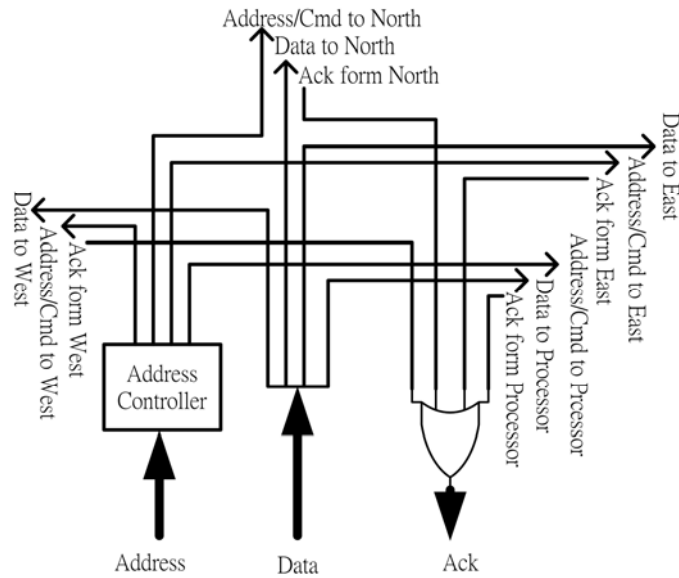


Figure 13 : Input stage of switch port

The main duty of input stage of switch port is as follows:

- (1) **Address controller** extracts packet address from input Address-line to decide where to store input data.
- (2) Dispatch input data on input Data-line to the buffer which stores the data.
- (3) Collect output acknowledgement from ack-controller of other output stages and deliver acknowledgement signal on input Ack-line.

3.2.1.2 Output stage

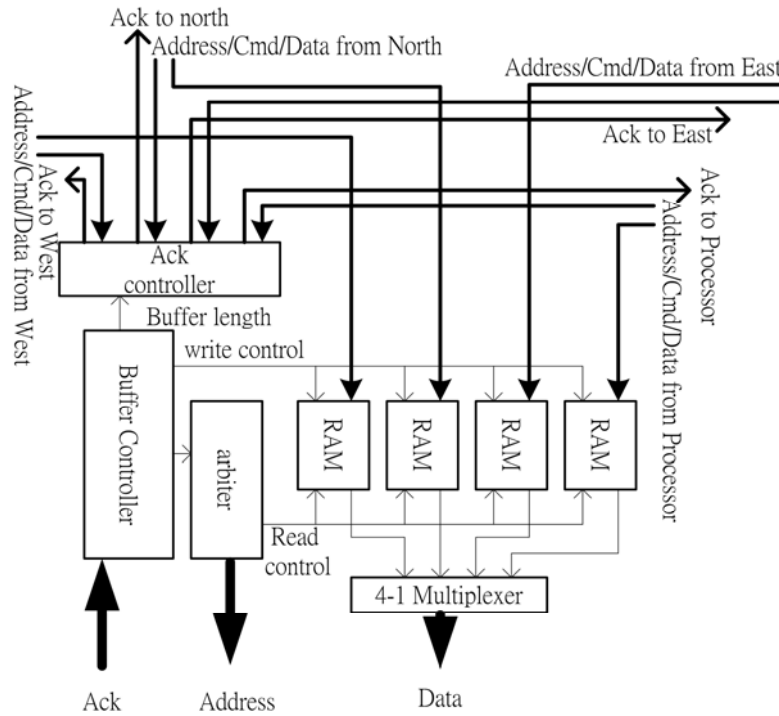


Figure 14 : Output stage of switch port

The output stage is composed of following elements:

(1) There are four memory modules in each direction, which are called **RAM** in Figure 14. They are all one-read/one-write memory architecture. These four memory modules store input data which is received by other four input stages separately. Take Figure 15 as an example. The data of these packets, which comes from north direction and will make east turn in current switch, will always be received by input stage of north port and then be stored in RAM-N of output stage of east port.

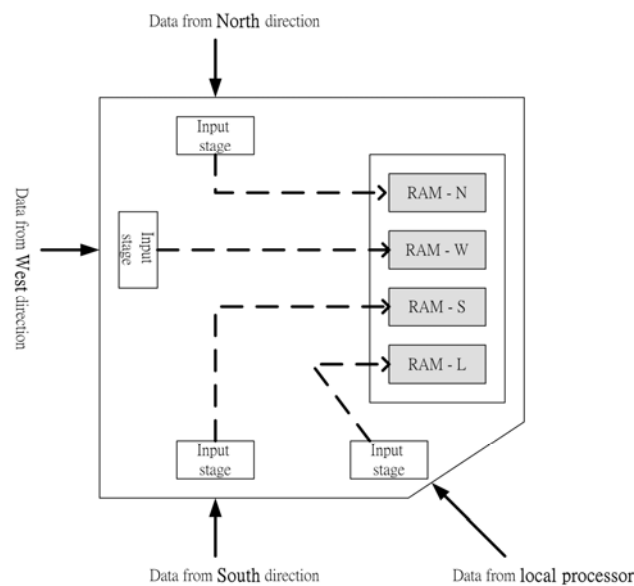


Figure 15 : Memory duty diagram

(2) **Buffer controller** records the size and status of the buffers in the switch, and asks **Arbiter** to grant channel privilege.

(3) **Ack controller** checks status of the buffer which is indicated by input address, and responses acknowledgement according to the status. For example, in Figure 16, a packet from south direction is transmitted across current switch to east direction. The neighboring switch at south direction will first notify the Ack-controller of east port of current switch to check whether there are available buffer space to store the data or not. The Ack-controller will response acknowledgement as a result. After receiving acknowledgement, the neighboring switch at south direction will know whether the data which is transmitted at this transmission is successfully received by the switch or not.

(4) **Arbiter** use weighted round robin scheduling to grant channel privilege.

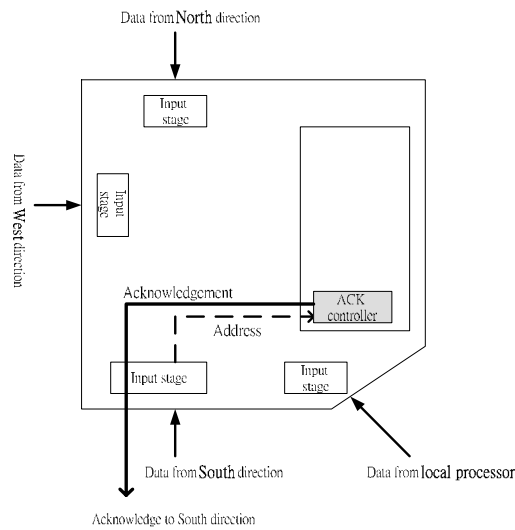


Figure 16 : Ack controller

3.2.2 Organization of internal buffers

After brief description of our memory architecture in subsection 3.2.1.2(1), we present the implementation details in this section. Each memory module, which is called RAM in Figure 14, can be partitioned into several buffers to provide necessary virtual channels. As illustrated in Figure 17, we partition each memory module into 4 buffers, resulting total 16 data buffers in this port, which means a capacity of 16 virtual channels to route packets. The size of memory will influence the flexibility of partition. For example, a memory module which has 32 words can be partitioned to two buffers with 16-words, four buffers with 8-words, or even eight buffers with 4-words.

We must highlight that the partition of memory is reconfigurable independently not only in each switch but also in each port even after fabrication. Memory partition can be used to trade-off between flexibility of routing packets and communication performance.

Assume that there are applications that need lots of long-distance transmissions in our platform. It will become difficult to route all the packets if we only provide few virtual channels in each switch. On the contrary, smaller buffer size causes higher failing rate at transmission and degrades communication performance.

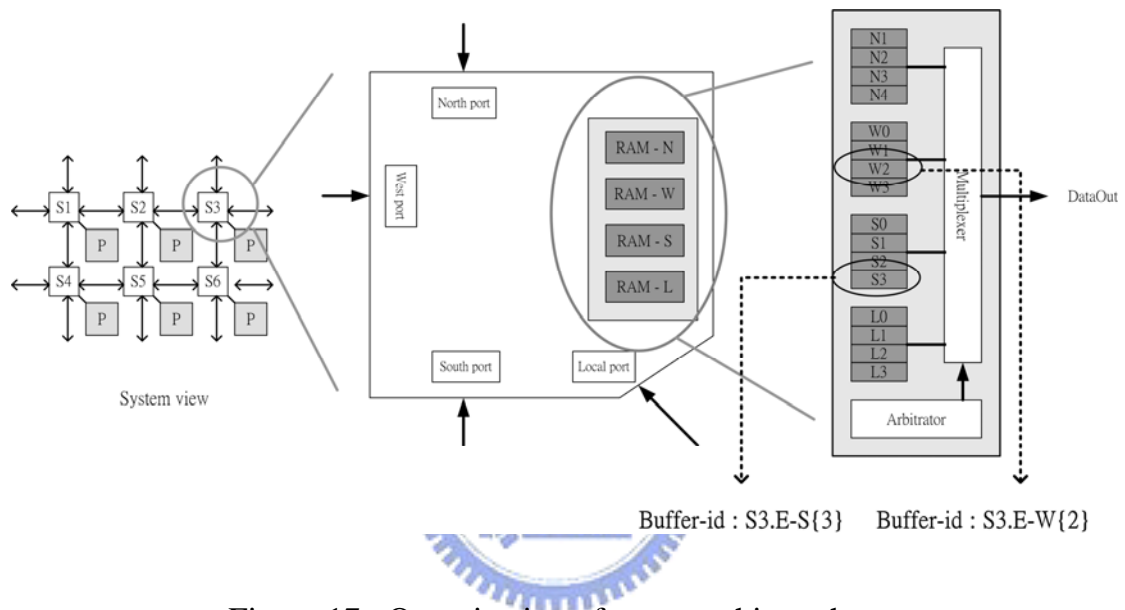


Figure 17 : Organization of memory hierarchy

The organization of memory hierarchy is illustrated in Figure 17. In our platform, we will give each buffer in the switch a buffer-id. Figure 18 illustrates the meaning of expression.

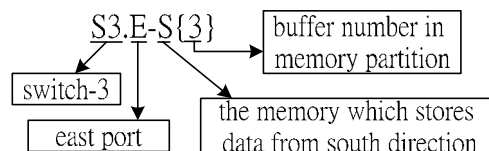
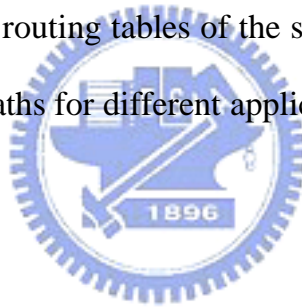


Figure 18 : Buffer naming rule

For example, the buffer-id S3.E-S{3} is the identification of the buffer that is the third buffer of the memory module, which stores data from south direction at east port of switch 3.

In addition to the space for each buffer to store input data, there is another memory space, called routing table, for each buffer to record a unique buffer-id as output address. The data stored in current buffer will be sent to the buffer which is identified with this buffer-id at next successful transaction. As you can figure, the buffer with this unique buffer-id must be one of the buffers of the neighboring switch, which is connected to current buffer. By configuring the routing tables of the switches in our platform, we can provide the needed transmission paths for different applications.



3.3 Transaction

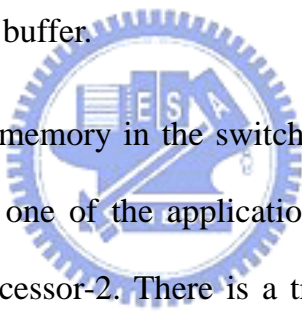
After a detail explanation of our switch architecture and organization of memory hierarchy, we describe how to form a transmission path and explain transaction procedure with a distinct illustration here.

3.3.1 Path configuration

In the system design flow that we introduced in section 1.3, after mapping the applications onto associated processing elements, we have to provide all transmission paths that the applications need. Assume that the system designers have decided all the transmission paths of routing packets. The next thing we should do is to configure our

platform to form these paths. For each transmission path, we will look for one available buffer in each switch along the path and reserve these buffers to form a dedicated virtual channel. Among these buffers that form this dedicated virtual channel, we will repeatedly assign the buffer-id of the succeeding buffer for current buffer as output address. By configuring the routing table of the buffers, we can set up this the transmission path.

Unlike the packet switching, which should decode the packet address, search space to store the data, and compute the routing path of the packet. We simplify the duty of switch by the routing table of each buffer.



In Figure 19, assume that each memory in the switch is partitioned into four buffers with eight words. Supposing that one of the applications in our platform will deliver messages from processor-1 to processor-2. There is a transmission path from network interface of processor-1 through switch-1 and switch-2 to processor-2. The transmission path is established by configuring routing table of buffers in these two switches. First, we assign buffer-id: $S1.S-P\{2\}$ for processor 1 as source buffer-id when it want to send message to processor-2. Secondly, assign buffer-id: $S2.L-N\{1\}$ for buffer: $S1.S-P\{2\}$ as output address and assign a memory address for buffer: $S2.L-N\{1\}$ as output address. This buffer chain which is composed of the network interface of processor-1, the buffer $S1.S-P\{2\}$, the buffer $S2.L-N\{1\}$ and the network interface of processors-2 will form a dedicated transmission path for packets from processor-1 to processor-2.

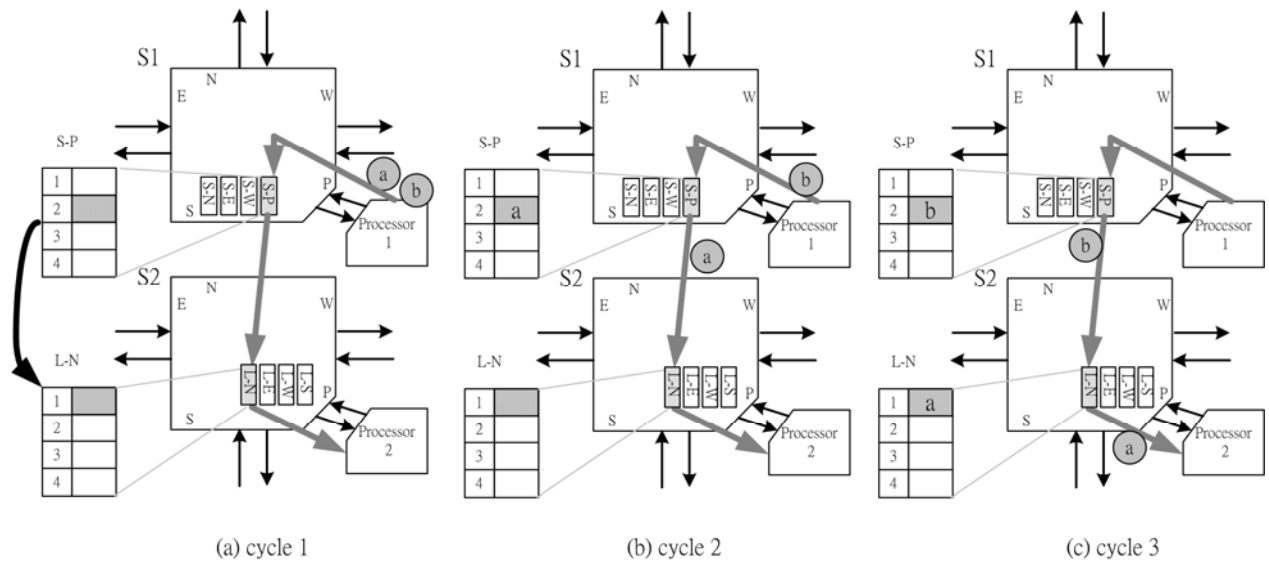


Figure 19 : Path transaction procedure

3.3.2 Transaction procedure

After the description of setting up transmission path, we illustrate our transaction procedure in Figure 19. Assume that processor-1 send a packet with 2 words to processor-2. We mark the words as grey circle-{a} and grey circle-{b} in Figure 19. First, processor-1 sends word-{a} to input stage of local port of switch-1 and switch-1 stores word-{a} in buffer: S1.S-L{2}, as shown in Figure 19(a). In Figure 19(b), the output stage of south port of switch-1 send data-{a} to input stage of north port of switch-2 and switch-2 stores data-{a} in buffer: S2.L-N{1}. At the same time, processor-1 can send data-{b} to switch-1 and switch-1 stores data-{b} in buffer: S1.S-L{2} as it did to word-{a}. This shows that we allow pipelined transactions. In the last step, output stage of local port of switch-2 sends data-{a} to processor-2 and finishes the transmission of word-{a}. Word-{b} will arrive at processor-2 with the

same procedures.

3.4 Transaction protocol

In this section, we explain the detail of transaction procedure between neighboring switches. In Figure 20, we show the interface diagram between two neighboring switches. Routing table is a mapping between buffers of east port of switch-1 and buffers of switch-2. It records the buffer-id of switch-2 as output address of output buffer in east port of switch-1. For example, the output address of S1.E-N{1} is S2.E-W{1} in Figure 20.

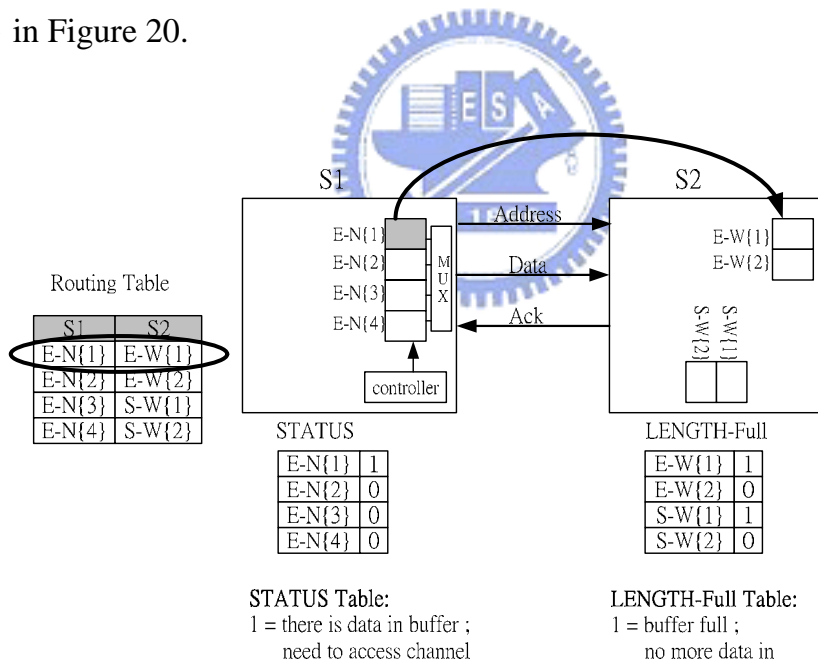


Figure 20 : Transaction protocol between switches

Assume that at this moment, the buffer S-1.E-N{1} in switch-1 stores data that will be delivered to buffer S-2.E-W{1} in switch-2. We will show the detailed transaction

procedure between these two switches. This is a simple example but shows clearly our procedure. Our procedure is divided into four steps: channel privilege arbitration, output address transmission, data transmission, and acknowledgement. Each step will be finished in one clock cycle. And these transactions can be executed in pipelined manner to increase throughput.

The detailed transaction is as follows:

Cycle 1: buffer $S1.E-N\{1\}$ notes the controller that it wants to access the output channel and urges controller to grant channel privilege to it.

Cycle 2: buffer $S1.E-N\{1\}$ sends address ' $S2.E-W\{1\}$ ' on the Address-line. This address indicates that this transaction tries to send data into buffer $S2.E-W\{1\}$.

Cycle 3: In switch-1, buffer $S1.E-N\{1\}$ sends data on Data-line. In switch-2, the Ack-controller sends the acknowledgement (false/true) according to the status (full/available) of buffer $S2.E-W\{1\}$ on Ack-line. At the same time, buffer $S2.E-W\{1\}$ stores data on input Data-line if it still has memory space, else discards the data.

Cycle 4: according the acknowledgement on Ack-line, buffer $S1.E-N\{1\}$ in switch-1 will decide whether to keep the data or erase data it stores. If the acknowledgement is true, it means that this is a successful transmission. Buffer $S1.E-N\{1\}$ will erase the data that has been transmitted successfully.

3.5 Round robin scheduling

At the output stage of switch port, we implement the arbiter with round robin scheduling technique to decide which virtual channel can get the privilege to access output channel and transmit data.[7] In this way, these transmission paths that deliver data in the same port of switch will use equal bandwidth of output channel. This technique will avoid the starvation for accessing channel. Note that only these virtual channels that are really active to transmit data will be scheduled. We won't guarantee channel privilege to these buffers that don't transmit data. This will increase the channel utilization and prevent unnecessary power consumption.

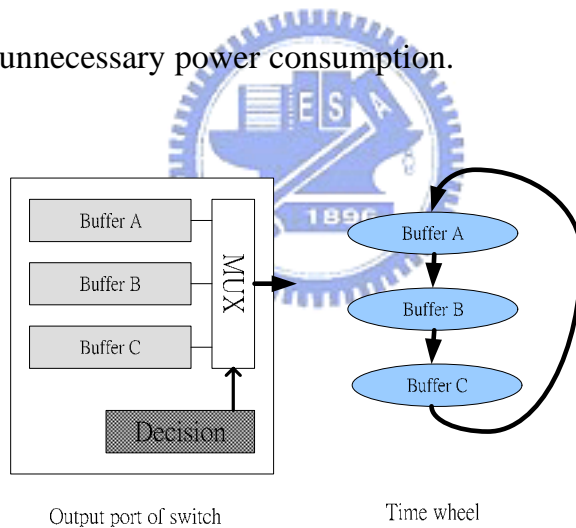


Figure 21 : Output arbitration

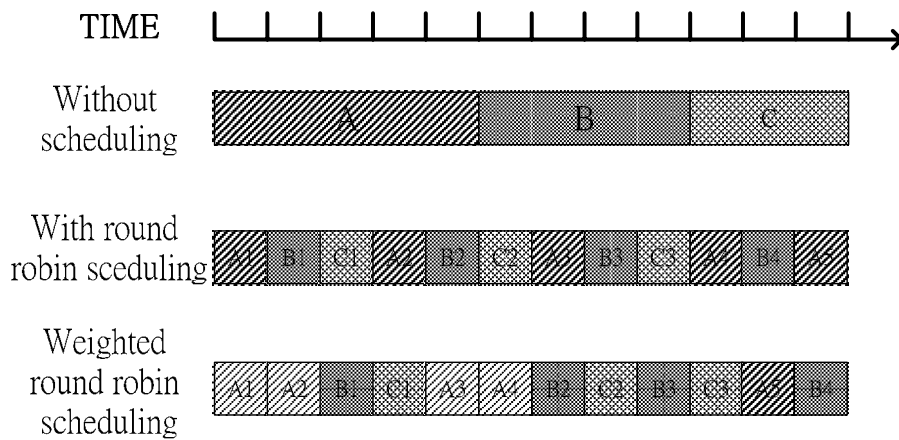


Figure 22 : Round robin scheduling

In Figure 21, there are three messages delivered by three virtual channels separately in output port. We identify these three messages as A, B, C. If we don't use a round robin scheduling to transmit data, a short message may be possibly postponed a very long time before it is transmitted. We show this situation in Figure 22 (without scheduling). With a round robin scheduling technique, each message can equally share the channel bandwidth, as shown in Figure 22 (with round robin scheduling).

Moreover, there may be different communication requirements for different applications. We sometimes need to provide larger bandwidth for some transmission paths. By assigning different weight to each transmission path, we can provide flexible bandwidth. We show this in Figure 22 (weighted round robin scheduling). In this example, buffer A has twice bandwidth than buffer B and buffer C.

3.6 Performance

For each transmission between on-chip components, we will reserve buffers in those switches which are on the transmission path to form a virtual channel connection. With such dedicated channel and round robin scheduling, we can guarantee the minimum bandwidth of each transmission. For example, consider that in east port of switch in Figure 23, there are three paths sharing the bandwidth. We assign them with the same weight. This means that for each transmission path, they are all guaranteed to use at least one third bandwidth in this switch.

We make a simple expression:

$$local\ guaranteed\ minimum\ bandwidth = \frac{channel\ bandwidth}{sum\ of\ weight\ of\ paths\ using\ this\ channel}$$

(3-1)

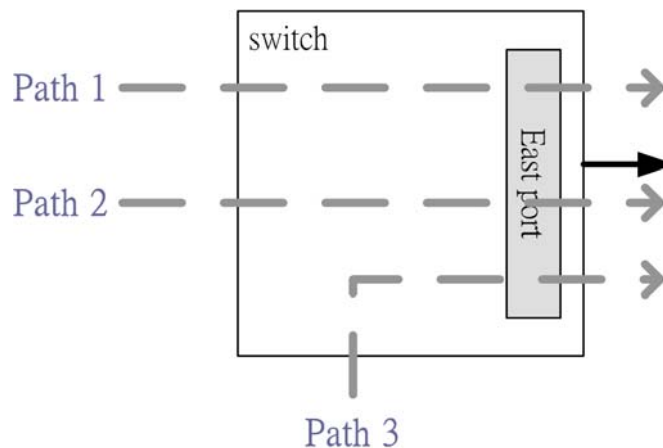


Figure 23 : Bandwidth sharing example

If the transmission intervals of these three paths are not overlapped, the switch will

provide higher bandwidth to these paths that are transmitting data.

The example in Figure 23 and equation (3-1) describes only local guaranteed bandwidth expression in one switch. When it comes to the guaranteed bandwidth of whole transmission path, we need to trace all the local guaranteed bandwidth in those switches along this transmission path, and choose the smallest guaranteed bandwidth as the minimum bandwidth of this transmission path. Equation (3-2) and (3-3) describe the expression.

$LGBW_i = \text{local guaranteed bandwidth of } i\text{-th switch on transmission path}$

(3-2)

$\text{Guaranteed bandwidth of transmission path}$

$$= \min_{i \in \text{switch on the transmission path}} (LGBW_i)$$

(3-3)

Assume we provide all the channels of our platform with the same bandwidth, called standard channel bandwidth (scb), in Figure 24. There is a transmission path from P1 to P6 through switches S1, S2, S3 and S6. The related LBW of these switches are shown in the figure. The guaranteed bandwidth of this path is equal to the smallest local guaranteed bandwidth of these switches, which is $LGBW_{S3}$.

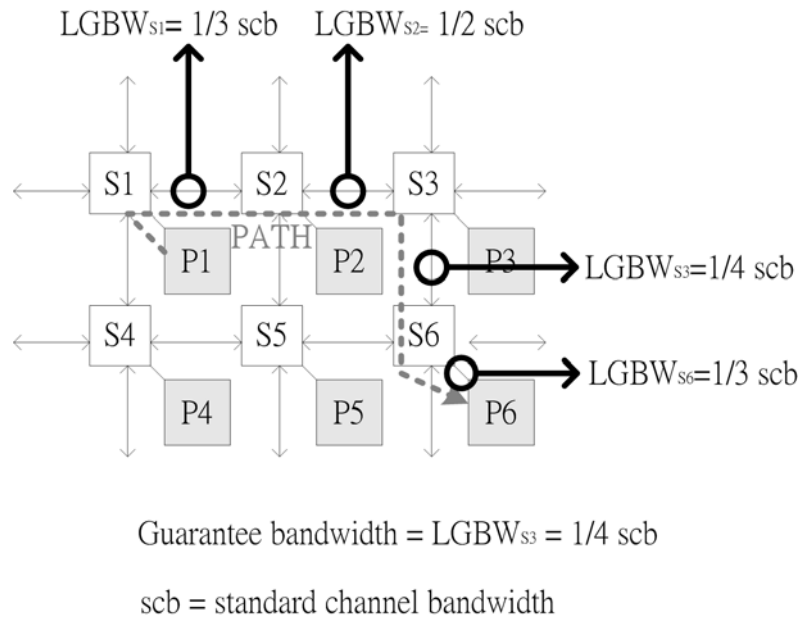


Figure 24 : Bandwidth guaranteed transmission path



3.7 Design overview

In section 3.1, we summarized the requirements that a future communication infrastructure should provide: efficient communication, guaranteed throughput and fault tolerance capability. We explain how to use our platform to meet these requirements.

1. Efficient communication

By profiling the applications in the communication-driven system methodology, we can get some statistics information about communication traffic of our system. Moreover, we will know the constraints requirements that different applications need. To avoid the traffic congestion between these transmission paths that need larger bandwidth, we can alleviate the network load

by assign different paths for them. Figure 25 is an example. In (a), if both transmission paths use the channel connection from S2 – S3 – S6, they will make traffic contention to degrade the system performance. We can assign different paths for them in (b) to avoid the overlapping of transmission paths and get better performance. With proper setting of dedicated transmission paths, we can provide a efficient communication environment.

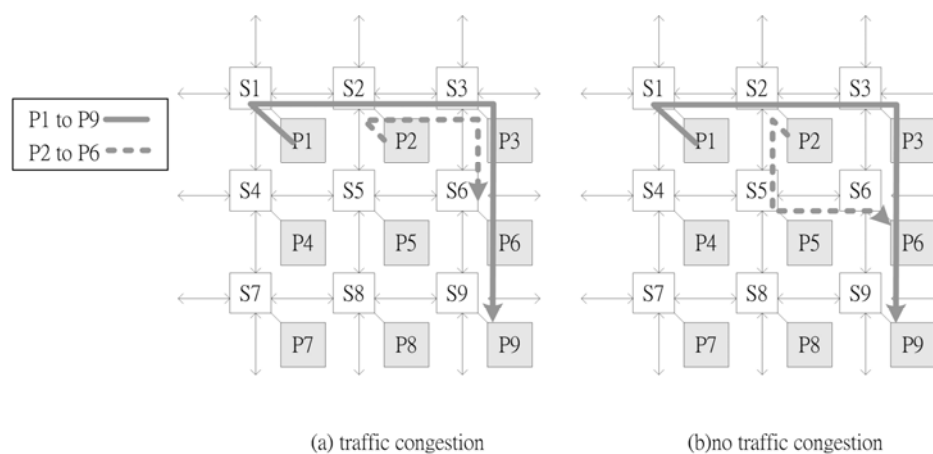


Figure 25 : Assigning different path to avoid traffic congestion

2. Guaranteed throughput

With dedicated virtual channel and round robin scheduling, we can provide guaranteed bandwidth. In section 3.6, we express the equations. In this way, we can implement a hard real time system.

In addition, the advantage of providing a network infrastructure with guaranteed throughput is on system modeling. The system designer can predict

the worst cast of transmission and estimate the system performance at higher levels.

3. Fault tolerance capability

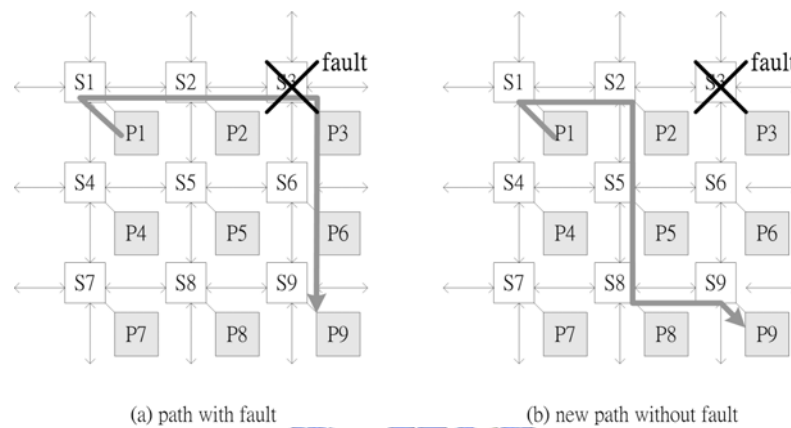


Figure 26 : Fault tolerance

With different path assignments, we can easily avoid to use the faulty components. In Figure 26 (a), originally the transmission path from P1 to P6 will use the faulty switch, S3. By assigning another transmission path, we can still transmit data from P1 to P6 correctly and solve this problem. This example shows that we provide a flexible environment to overcome fabrication faults.

Chapter 4

Experimental Results



To verify the functionality and evaluate the communication performance of our platform, we use a 2-D mesh topology with 4-by-4 nodes as our platform. With routing path configuration, we allow each processing element to communicate with other processing elements. Because we only consider evaluating the communication performance of our platform, individual processing element only provides the function that generates random traffic here. We write a random pattern generating model to replace the original processing element. This pattern generator can generate packets with random length from random source to random destination between random intervals.

We implemented our switch design by both Verilog HDL and cycle accuracy C++ model. Verilog version is for traditional cell-based design flow implementation, and C++ model is for platform evaluation in system design flow.

4.1 Definition

We define some terminologies to be used in our experiments.

Latency: The time elapsed from when the packet transmission is initiated until the packet is received at the destination node.

Maximum Latency: ~~#~~The predicted worst case latency under guaranteed bandwidth.

Normalized Latency: The transmission latency divided by the maximum latency.

Injection Rate: ~~#~~The actual bandwidth of communication traffic divided by the guaranteed bandwidth of the transmission path.

4.2 Experiment

4.2.1 Functionality of our platform

We verify that our platform can guarantee the minimum bandwidth for each transmission. In this experiment, we provide each buffer in the intermediate switch with 2 words and evaluate the performance of our platform under different injection rate. Table 1 shows histogram of the results.

First, we observe that even under high injection rate (injection rate = 1), packets in the network are still delivered to the destination within maximum latency (normalized latency < 1). The result shows that our platform guarantees the performance even at worst-case.

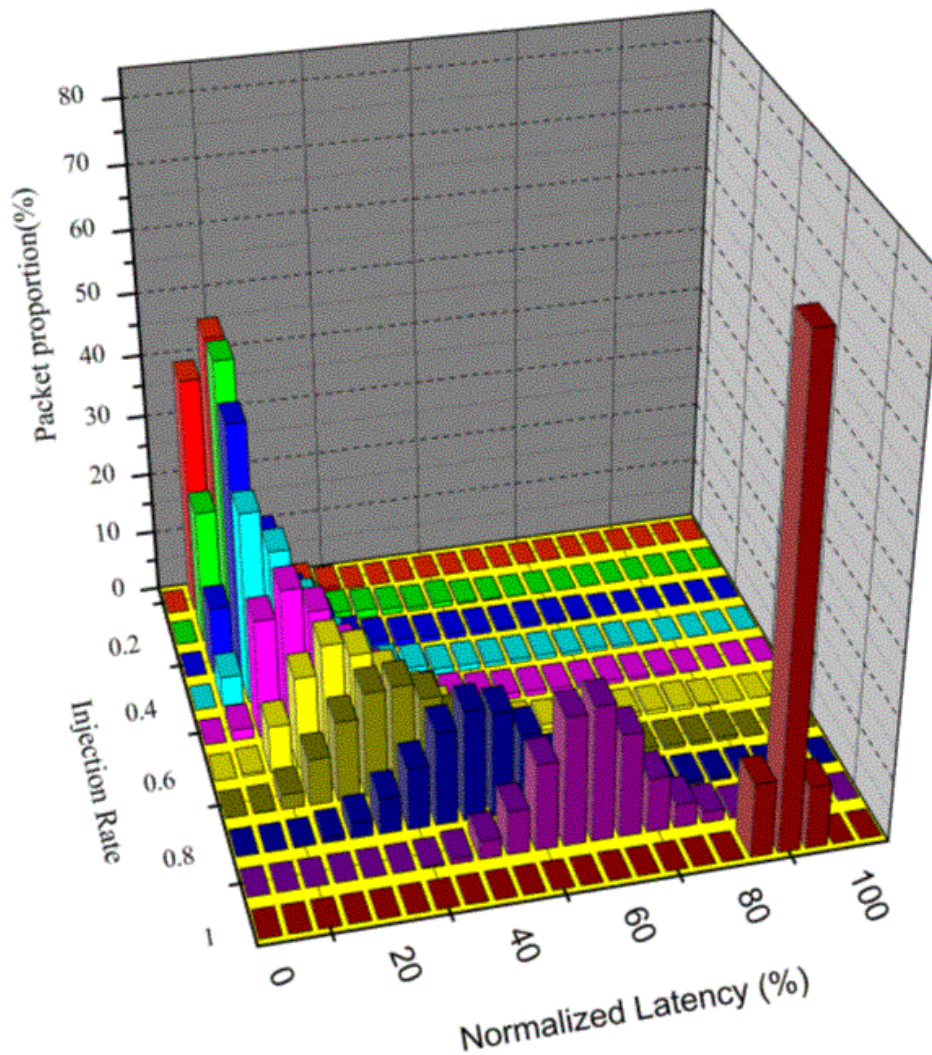


Table 1 : Histograms of normalized latency under different injection rate

Secondly, in this table, the histogram of the normalized latency shifts to low as

injection rate decreases. It indicates that the average latency decreases as injection rate decreases. This trend implies that if we can well control the communication to decrease the traffic load, we can transmit packets faster.

This histogram also shows the trend that the pattern proportion at high injection rate and low injection rate are more centralized than these at medium injection rate. The reason for very high injection rate is that almost all of the packets are transmitted with guaranteed bandwidth and transmission path hardly use extra bandwidth. That makes the normalized latency at high injection rate close to maximum latency. The reason for low injection rate is similar. Almost all of the packets can be delivered without contention and transmission paths can use almost full channel bandwidth. Only few packet transmissions will overlap and share the same channel bandwidth. That makes the average latency of transmissions at low injection rate close to minimum latency. On the contrary, these transmissions at medium injection rate will sometimes come up against contention and sometimes be transmitted without blockings. This property makes that the actual bandwidths of these transmission paths vary from guaranteed bandwidth to full channel bandwidth. It will cause the latency of transmissions become uncertainty. That's why the packet proportions at medium injection rate are widely distributed.

4.2.2 Different communication quality of our platform

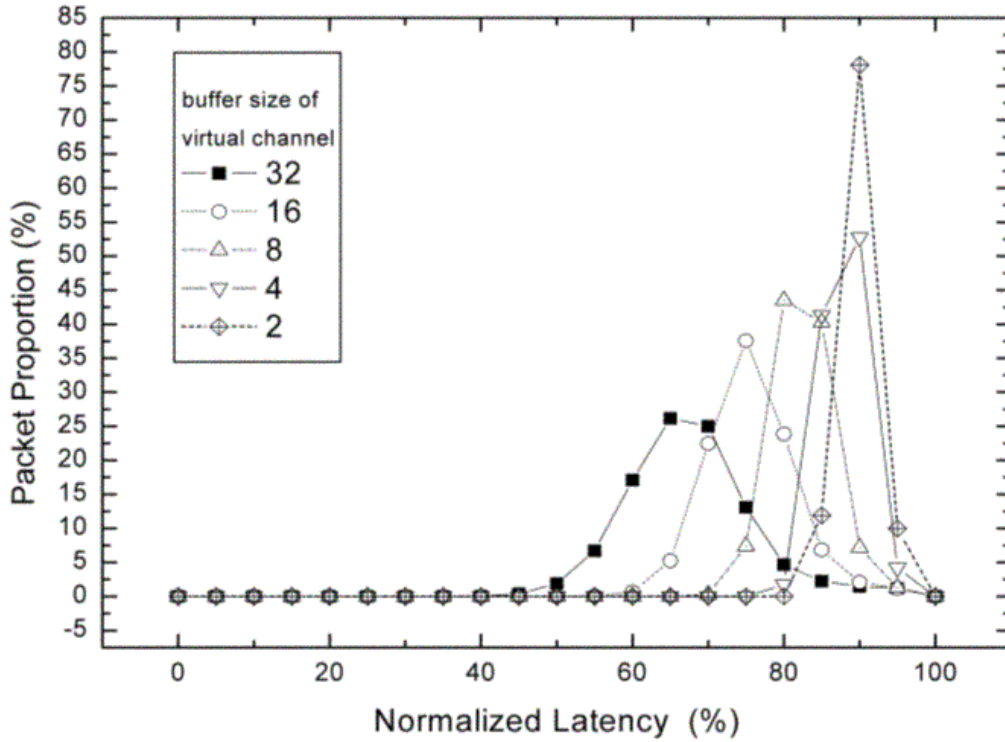


Table 2 Histograms of normalized latency under different buffer size of virtual channel

Because different applications may need different communication quality, we should provide different transmission bandwidth. By deciding appropriate buffer size of virtual channel in our switch, we can provide necessary bandwidth.

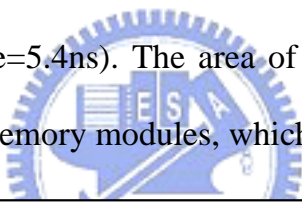
In Table 2, we show the histograms of normalized latency under different buffer size of virtual channels. A trend is clear observed that the average normalized latency decreases as the buffer size increases. This means that if the system on our platform

needs some transmission paths that need large channel bandwidth, we can simply reconfigure the partition of memory in these switches on these transmission paths to accomplish these demands.

4.3 Synthesis report

We have implemented a switch design in Verilog and have synthesized it. We implement the queue in the switch with 32 words, and each word is four Bytes.

We use TSMC 0.25um technology. The synthesis report shows that our switch design can work at 185MHz (clock cycle=5.4ns). The area of our switch design is about 3.5 mm². The main area is used on memory modules, which is about 2.7mm².

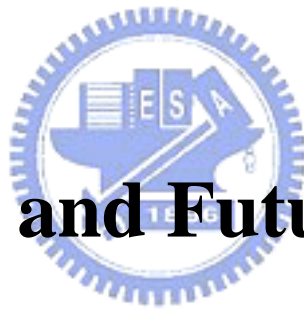


[Technology]	
TSMC .25um 1p 4m	
[Area] (unit: um ²)	
Combinational area:	529199
Non-combinational area:	2986619 (RAM module ~270000)
Total cell area:	3515726
[Timing]	
Clock cycle:	5.4ns (185MHz)

Table 3 : Synthesis report

Chapter 5

Conclusions and Future Work



We introduce a communication-driven design methodology and related communication infrastructure. By considering computing, communication, and memory at the same time, we proposed a novel switch design for on-chip communication. This infrastructure can be configured as circuit-switching, packet-switching or dedicated bus for different applications. With dedicated virtual channels and round robin scheduling, we can guarantee the minimum channel bandwidth for transmission paths. By using pipeline bus as basic communication mechanism, the system can transfer data in pipeline fashion to increase performance. The experimental results indicate that our platform can guarantee the bandwidth and efficiently transmit data.

There are still some problems unsolved. Deciding the optimal buffer size in the switches is the trade-off between flexibility and cost. It depends on application features and needs the top down design flow to optimize it. The selection of transmission paths will be another problem to balance the communication and computing power.



Reference

- [1] Pierre Guerrier and Alain Greiner, "A generic architecture for on-chip packet-switched interconnections," *Proceedings of the conference on Design, Automation and Test in Europe*, 2000.
- [2] William J. Dally and Brian Towles, "Route packets, not wires: on-chip interconnection networks," *Proceedings of the Design Automation Conference*, 2001.
- [3] M. Sgroi, M. Sheets, A. Mihal, K. Keutzer, S. Malik, J. Rabaey, and A. Sangiovanni-Vincentelli, "Addressing the system-on-chip interconnect woes through communication-based design," *Proceedings of the Design Automation Conference*, 2001.
- [4] Luca Benini and Giovanni De Micheli, "Networks on chips: a new SoC paradigm," *IEEE Computer magazine*, 2002.
- [5] Daniel Wiklund and Dake Liu, "SoCBUS: switched network on chip for hard real timing embedded systems," *International Parallel and Distributed Processing Symposium*, 2003.
- [6] E. Rijpkema et al., "Trade offs in the design of a router with both guaranteed and best-effort services for networks on chip," *Proceedings of the conference on Design, Automation and Test in Europe*, 2003.

- [7] Jose Duato, Sudhakar Yalamanchili, and Lionel Ni, *Interconnection Networks: an engineering approach*, Morgan Kaufmann, 2003.
- [8] Axel Jantsch , Hannu Tenhunen , *Network on Chip* , KLUWER Academic Publishers , 2003.
- [9] <http://public.itrs.net/>
- [10] Marco E. Kreutz, Luigi Carro, Cesar A. Zeferino, Altamiro A. Susin, “Communication Architecture for System-on-Chip”, *Proceedings of Integrated Circuits and Systems Design*, 2003.
- [11] Terry Tao Ye, On-chip, *Multiprocessor communication network design and analysis*, Stanford University, 2003.
- [12] Halsall, Fred, *Data communications, computer networks, and open systems* , Addison-Wesley, 1992.
- [13] www.arm.com
- [14] Kanishka Lahiri, Anand Raghunathan, Sujit Dey, “Efficient Exploration of the SoC Communication Architecture Design Space”, *Proceedings of the IEEE/ACM international conference on Computer-aided design* , 2000.
- [15] Kanishka Lahiri, Anand Raghunathan, Sujit Dey, “Evaluation of the Traffic-Performance Characteristics of System-on-Chip Communication

Architectures”, *Proceedings of the The 14th International Conference on VLSI Design*, 2001.

[16] S. Y. Kung, *VLSI Array Processors*, Prentice-Hall, 1988.

[17] W. Dally and J. Poulton, *Digital Systems Engineering*, Cambridge University Press, 1998.



Vita

Pao-Jui Huang was born in Changhu on May 30, 1980. He received the B.S degree in Electronics Engineering from National Tsing Hua University in June 2002. From September 2002 to July 2004, he was a graduate student of Professor Jing-Yang Jou in the institute of Electronics, National Chiao Tung University. His research was related to Electronic Design Automation (EDA). He received the M.S degree in Electronics Engineering from National Chiao Tung University in July 2004.

