Cycle Stealing Buffers and Physical Channel Management Scheme for Wormhole Based On-chip Networks

Jwo-An Lin, Yung-Chou Tsai, Yarsun Hsu
Department of Electric Engineering, National Tsing Hua University, Hsinchu, Taiwan
Email: yshsu@ee.nthu.edu.tw

Abstract

With the improvement of chip manufacture process, a single chip may contain many processor cores and functional units. These cores and functional units communicate with each other through an on-chip interconnection network. Therefore, a key issue in the design of multi-core chip is how to construct a low latency high bandwidth on-chip interconnect.

In this paper, a cycle stealing buffer and a physical channel management scheme are proposed for the design of on chip networks. The cycle stealing buffer can reduce the number of cycles in reading and writing a network buffer. The physical channel management scheme can efficiently multiplex and arbitrate the physical channel among many virtual channels. These two methods can be adopted in wormhole based networks to improve both latency and throughput. In this paper, by use of simulation, we study the feasibility and benefits of these two methods using a wormhole based ring network.

Index Terms— on chip network, cycle stealing buffers, physical channel management scheme

1. Introduction

With the advance on electronic systems, the demand for more computing power has never stopped. Although the performance of processors has doubled in approximately every three-year span from 1980 to 1996, the complexity of applications has continuously driven the development of even faster processors. However, boosting the performance of processors becomes very complex and expensive, only a few companies all over the word can afford it. Therefore multi cores had been proposed as an alternative approach.

In this approach, several processor cores and functional units can cooperate to solve a large problem. Therefore, designing high performance interconnections becomes a critical issue to exploit the performance of multi-core system [1].

Thanks to the evolution of integrated circuit technology, a single chip may contain a large set of processing units and numerous IPs. This makes multi processor system on chip (MP-SoC) possible and provides integrated solutions to challenging design problems in lots of domains, such as telecommunication, multimedia and consumer electronics [2].

A critical issue for MP-SoC design is the interconnection between processing units and heterogeneous functional units [3]. Efficient interconnection is necessary for these elements to cooperate with each other. MP-SoC was developed for high-performance computation, such as image processing. For example, “Emotion Engine” proposed by Sony [4], and “Cell Processor” proposed by IBM [5], where on chip interconnection efficiency is the key to the overall system performance.

For an interconnection design in MP-SoC, synchronization with a single clock can be extremely difficult. Though gate delays scale down with technology, global wire delays typically increase exponentially or, at best, linearly by inserting repeaters. Even after repeater insertion, the delay may exceed the limit of one global clock cycle [6]. Even more, in ultra-deep submicron processes, more than 80 percent delay will come from the interconnection wires [7].

In the last decades, one of the most frequently used on-chip interconnection is the shared medium bus; this interconnection architecture serializes requests from any master units connecting to the bus, forces each transaction to complete before next transaction can begin. As a result, the interconnection efficiency decreases severely when the number of masters increases, this limits the number of processing units and functional IP blocks that can be connected to a bus and thereby limits the system scalability. Several solutions for such case were proposed, based on splitting the bus into many separate local segments. By introducing a hierarchical architecture and the concept of global asynchronous and local synchronous (GALS), modules in a particular segment can exchange data independent of modules in other segments at a locally defined speed, and access the global bus through self-timed interface [9] [10]. However this bus-based system has the inherent limitations, as all attached devices must share the bandwidth of the bus. Also, the performance degrades due to the bus parasitic capacitance and the complexity of arbitration.

To overcome these problems, a network-based interconnection approach was proposed. In this approach, the communication among units can take place in the form of some pre-defined logical units, such as message or packet. With the assist of network components, such as router, switch … etc, units connected to the network can exchange their information. This approach resembles the network of multi-processor system, and is well known as "On-Chip Networks".

Unlike other networks, such as internet or multi-computer cluster system network, on-chip network has much more design constraints due to the physical characteristic of a chip. It is not feasible to think that on-chip network equates to porting the Transmission Control Protocol/ Internet Protocol (TCP/IP) to silicon or achieving an on-chip Internet, due to the high latency and complexity of TCP/IP. [11]

For on-chip network design, the network must be simple enough. Complex network may lead to complex design issues, therefore increase the hardware complexity of each network component. With the increase on hardware complexity, network area and power consumption also increase, thereby reducing the overall network performance. Moreover, an on chip network must conform to on chip wiring constraints. Long wring paths may lead to huge parasitic resistance and cause a large wire delay. Crossover wirings in different metal layers may lead to parasitic capacitance and cause signal crosstalks A design tradeoff between hardware cost and network performance must be handled carefully. Therefore, in this paper, the cycle stealing buffers and the physical channel management scheme are proposed for wormhole on-chip networks. These methods can be easily adopted in wormhole on-chip networks which improves the overall network performance under the chip design constraints.

We describe our cycle stealing buffer design and physical channel management scheme in section 2 and 3 respectively. In section 4, we adopt these two methods in the design of a ring based network. We then describe the simulation environment in section 5. Section 6 and 7 show the simulation results using these methods. Finally we conclude our study in section 8.

2. Cycle stealing buffers

As we mentioned earlier, on-chip network has much more constraints than conventional network designs. Most of them come from the physical characteristic of a chip. Under these constraints, designers must optimize the network carefully. Otherwise the optimization may incur a huge cost on hardware and become infeasible.

Since network buffers are necessary components for every network node, their designs will severely influence the overall network performance. We can estimate that a small performance improvement in network buffer will induce a large performance improvement in overall network.

In today's network designs, most of them adopt the first in first out (FIFO) architecture as their storage buffer. In this section, we will briefly discuss the conventional FIFO architecture and its behavior. Then, the cycle stealing buffers are proposed, which improves the network performance with only a small extra cost.

2.1 Conventional One Port FIFO

A conventional one port FIFO shown in Fig. 2.1 is composed of two data pointers (read and write), some necessary control elements and lots of storing units. This module includes one bidirectional bus for data transfer, two status output signals (full and empty) which show the FIFO state and four control signal inputs (clock for synchronization, reset for buffer refreshment, and read/write for data access control).

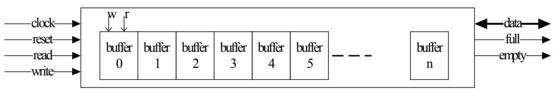

Fig. 2.1 Block diagram of a conventional one port FIFO

This architecture can implement a first-in-first-out storage function using the smallest number of IO pins and control signals [13]. If the hardware area constraint is severe and the performance requirement is not critical, this FIFO architecture may be a good solution for NoC buffers and can be easily adopted.

2.2 Conventional Two Port FIFO

As described in the previous section, a conventional one port FIFO can be easily adopted as storage units in on-chip networks. However, this architecture has a limitation that only one data command, read or write can be processed in every cycle. As a network buffer, this disadvantage will become a problem because it can't store and forward data information at the same time. Packets flowing in the network may be blocked frequently, waiting for the FIFO to finish its previous data transfer command. Lots of cycles are wasted due to this IO resource limitation, thus it becomes important to solve this problem. Therefore we add another data port, that is, two data ports in this module, one for data input, another for data output as shown in Fig. 2.2.

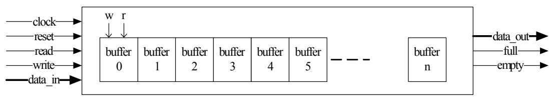

Fig. 2.2 Block diagram of a conventional two port FIFO

While the FIFO is not full or empty, user can write data into the FIFO buffer and read data from the FIFO simultaneously. No extra delays on contention for the data bus resource and the chance for packets to be blocked decreases. Without other conditions, packets can flow trough the network in a pipeline fashion.

2.3 Conventional FIFO Behavior

In this section, we would like to discuss the conventional FIFO behavior when switching from a full state to a non-full state and its shortcoming.

Fig. 2.3 shows a FIFO read/write example. In this example, we are going to discuss the behavior of a conventional FIFO design when switching from a full to a non-full state.

In cycle 1, a write event is declared by asserting the write signal and the data input. However, the FIFO is full and the data is not written into the FIFO in this cycle. In cycle 2, a read event is declared by asserting the read signal. Because the write process is not finished in cycle 1, the write signal remains high. In this cycle, data_0 is successfully read, therefore the data output is data_0. After this read process, the FIFO has one empty space, and the full signal is also reset. Since the FIFO is not full, the write event declared in cycle 1 can write data into the FIFO in cycle 3.

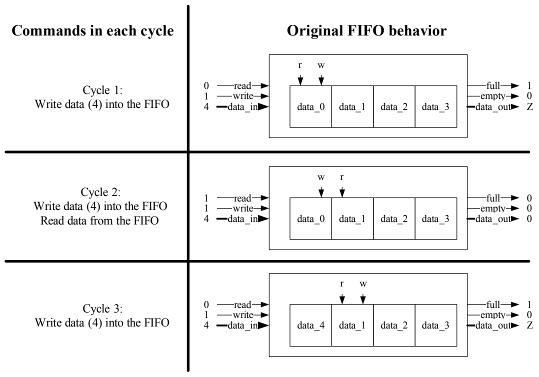

Fig. 2.3 An example of conventional FIFO behavior

2.4 Cycle Stealing Buffer Behavior

From the above example, we can recognize that the FIFO has to change its state from a full state to a non-full state in cycle 2, and then the data can be written in cycle 3. With this FIFO behavior, it induces an extra cycle delay every time when the buffers are full.

In order to fix this problem, we proposed a new design concept, which eliminates the extra cycle delay as Fig. 2.4 shows.

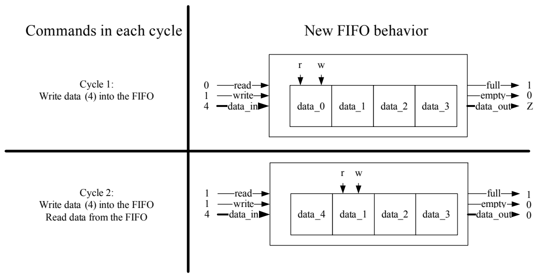

Fig. 2.4 The behavior of a cycle stealing buffer

The same as in previous example, a write event is declared in cycle 1, and the data is not written into the FIFO since the FIFO is full. In cycle 2, there is a new read event declared, so there are two events: a write and a read ready to be processed. Instead of waiting for the state transition from a full state to a non-full state, we can let both events be processed at the same cycle. That is, to read out the old data (data_0), and then write in the new data (data_4), just like what a single Flip Flop works. By using this concept, we can reduce the extra cycle wasted on an un-necessary state switching.

Although we have increased the buffer design complexity slightly, the buffer is still less complex than other network components and will not impact the network operation frequency.

3 Physical channel management scheme

The most expensive resources in network design are physical channel bandwidth associated with communication links and buffer space associated with each channels. Therefore, most current network designs use wormhole switching technique to efficiently use these expensive resources.

In wormhole switching networks, once a packet occupies the buffer of a physical channel, no other packets can access the physical channel. If this packet is blocked, the physical channel resource will be wasted. In order to increase the physical channel efficiency in wormhole network, a physical channel may be divided into several logical or virtual channels [14]. Logically, these virtual channels operate as distinct physical channels with a lower speed. Physically, packets at different virtual channels share the resource of that physical channel. With this method, a blocked packet stored in a certain virtual channel's buffer won't occupy the whole resource of the physical channel.

Virtual channels can also be used for deadlock prevention algorithm. By virtualizing the escape channels [15], which break the cyclic channel dependencies, the number of physical channels can be reduced and no extra cost incurs on inter-node wiring when deadlock prevention algorithm is adopted.

In the following section, the conventional virtual channel implementation scheme is briefly described first. Then we will propose a physical channel management scheme to efficiently manage the physical channel resource among multiple virtual channels in wormhole on-chip networks using virtual channels.

3.1 Conventional virtual channel multiplexing

A conventional virtual channel multiplexing is the flow control that can be easily adopted in wormhole based on-chip networks using virtual channels. It equally assigns the physical channel resource to those virtual channels across it. Each virtual channel uses the physical link for one clock cycle only and has to wait until all other virtual channels have used the link. These virtual channels share the physical channel in a round robin fashion. This method can be described by a selection function:

[Owner ID = (cycle count) mod (n)]

In this selection function, n represents the total number of the virtual channels and each of these virtual channels can operate as 1/n speed of the physical channel. Since the transfer speed of each virtual channel is fixed, no extra control is needed in this implementation method.

To implement the conventional virtual channel multiplexing, a multiplexer and a de-multiplexer are added on different end of a physical channel and a selector is added to implement the selection function, as Fig. 2.1 shows.

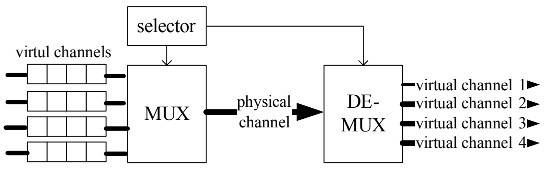

Fig. 3.1 Virtual channel implementation block diagrams

In conventional virtual channel multiplexing, although the flexibility and efficiency of physical channels has increased; a blocked packet stored in a certain virtual channel's buffer won't occupy the whole resource of the physical channel. However, the speed of each virtual channel also decreases significantly according to the number of virtual channels [1]. This decrease in speed will become a bottleneck for using virtual channel.

To solve this problem, we need to analyze the transfer state of virtual channels in different kind of situations.

3.2 The transfer state of virtual channels

First, we assume an ideal case that all of the virtual channels across one physical channel are ready to send packet, and these packets will not be blocked in the future. In this ideal case, the utilization rate of the physical channel is 100%, thus the speed of these virtual channels across it can not be further increased.

However, the ideal case mentioned above is just a sparse event. Most of the time, not every virtual channel across the physical channel is ready to transfer data through it. Some of the virtual channels may not buffer any packet; or it may buffer a packet which will be blocked in the future. Under these circumstances, if we just equally assign the physical channel resource to every virtual channel, the resource may be assigned to a virtual channel which is not ready to transfer data through it, causing a waste on physical channel resource. Therefore, how to assign the physical channel resource among these virtual channels becomes an important issue.

Instead of equally sharing the physical channel resource, we can make a resource arrangement according to the statistic traffic information. We can assign more usage time to these virtual channels with higher traffic loads. However, this pre-determined resource arrangement method may just meet the requirement of a certain traffic pattern. When the applied traffic changes, it may perform worse than the conventional virtual channel multiplexing.

3.3 Physical channel management scheme

Since the pre-determined resource arrangement method is not a good solution, we try to adaptively arrange the physical channel resource according to the run time traffic information about each virtual channel and to make the utilization rate as high as possible. However, collecting the global run time traffic information is not an easy problem. The time needed to process and apply this information on resource assignment may also cause the traffic information to be out of date, making the resource assignment not suitable for the current traffic load. In addition, the hardware needed to implement this method will also become a bottleneck.

For these reasons, we propose an idea to collect the run time local traffic information, and immediately apply it to the resource assignment method. Since the processing time is short, this resource arrangement should be much more efficient.

The run time local traffic information can be collected by viewing the state of each virtual channel and its destination buffer. If there isn't any packet in a virtual channel's buffer, we can conclude that this virtual channel will not send a packet through the physical channel in the next clock cycle; therefore there is no need to assign the physical channel resource to it. Besides, if a packet exists, but the destination buffer is full and blocked, this virtual channel cannot send a packet through the physical channel in the next clock cycle and therefore there is no need to assign physical channel to it either. By not assigning physical channel resource to those virtual channels that have no data to transfer and those their destination buffers are full and blocked, we can prevent unnecessary wastes on physical channel resource.

In addition, by viewing the buffer state of each virtual channel, the physical channel management scheme can also assign more physical channel resource to those virtual channels with higher traffic loads. This also reduces the congestion problem due to hot spot and improves the overall data transfer efficiency.

4 Applying these methods to wormhole based ring network

In this section, the cycle stealing buffers and the adaptive physical channel management scheme are adopted in a wormhole based ring network design as shown in Fig. 4.1. We use ring network for our initial study because it has been used in IBM's Cell processor[5].

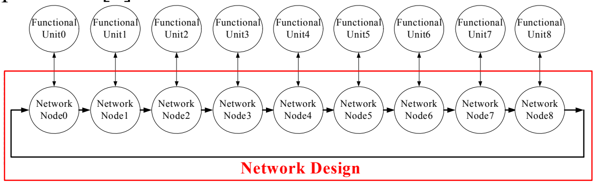

Fig. 4.1 A rough diagram of ring topology network

4.1 Network node design in wormhole based ring topology networks adopting cycle stealing buffers

In order to break the deadlock anomaly resulting from the possible cyclic channel dependencies in a ring topology network, an extra escape channel is added to connect the neighbor nodes [15]. In addition, there are two physical channels connecting to each node, one for data injection from functional units, another for data consumption by functional units.

In this network design, we assume a buffer is associated with each channel for storing messages in transmit and an extra buffer for data injection.

In each network node, we use a router to determine the path for the packets stored in different buffers and an arbiter to decide which packets will be granted to use the physical channel. These modules can co-operate to maintain the path for each incoming packet and support simultaneous access of different channels. The block diagram of the network node is shown in Fig. 4.2.

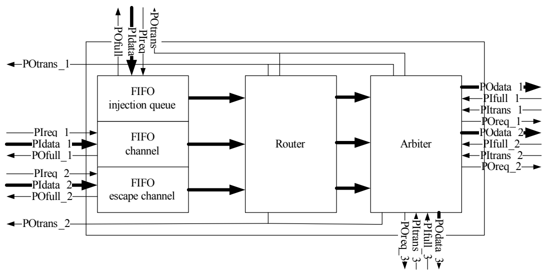

Fig. 4.2 A network node block diagram for ring topology network adopting cycle stealing buffers

The IO ports of this block diagram are described in Table

4.1.

Table. 4.1 IO port description of Fig. 4.1

IO Ports	description
PIreq	shows a write request from functional unit
PIreq_1	shows a write event from channel 1
PIreq_2	shows a write event from escape channel
PIdata	data input from functional unit
PIdata_1	data input from input channel 1
PIdata_2	data input from input escape channel
PIfull_1	shows a full state from right hand side node FIFO channel, which is the destination FIFO for output channel 1
PIfull_2	shows a full state from right hand side node FIFO escape channel, which is the destination FIFO for output escape channel
PIfull_3	shows a full state from functional unit buffer, which is the destination FIFO for output physical channel to functional unit
PItrans_1	shows a transfer state from next state's FIFO channel, which is the destination FIFO for output channel 1
PItrans_2	shows a transfer state from next state's FIFO escape channel, which is the destination FIFO for output physical channel 2
PItrans_3	shows a transfer state from functional unit buffer, which is the destination FIFO for output physical channel 3
POfull	declare a full state of FIFO injection queue
POfull_1	declare a full state of FIFO channel
POfull_2	declare a full state of FIFO escape channel
POtrans	declare a read state of FIFO injection queue
POtrans_ 1	declare a read state of FIFO channel
POtrans_ 2	declare a read state of FIFO escape channel
POreq_1	declare a write request to channel
POreq_2	declare a write request to escape channel
POreq_3	declare a write request to functional unit
POdata_1	data output to channel 1
POdata_2	data output to escape channel
POdata_3	data output to functional unit

By using the state signals (PItrans, PIfull …etc) from the neighbor node on the right hand side, the current node can determine the state of the destination buffer. If this buffer is transferring data, no matter it is full or not, a granted buffer at current node can start transmitting data through the physical channel to the destination buffer on the next node.

Since the buffer state at current node may also influence the data transfer of the left hand side neighbor node, which wants to send data to current node. Therefore, output state signals (POtrans, POfull …etc) describing the state of each buffer, must be sent to the left hand side node too. Using this network node architecture, we can realize the cycle stealing buffer concept proposed earlier.

4.2 Network nodes adopting cycle stealing buffers and physical channel management scheme

To realize the physical channel management scheme in a ring topology network, we virtualize the escape channels which break the cyclic dependency and add two modules in each network node as Fig. 4.3 shows.

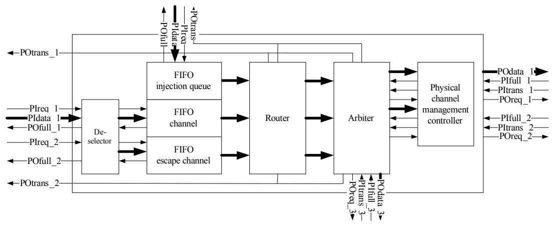

Fig. 4.3 A network node design for ring topology network adopting cycle stealing buffers and physical channel management scheme

The physical channel management controller can select one of the virtual channels which can transfer data through the physical channel, and skip those unable to transfer. Thus, the physical channel resource won't be wasted, and the utilization rate of physical channel can be higher than just equally assigning the physical channel resource to every virtual channel across it. In addition, the controller can also assign more physical channel resource to virtual channels having more ready-to-send packets on their buffers.

The de-selector is a de-multiplexer device, which can pass the incoming data to their destination buffer using the information from sender's physical channel management controller module.

In this network node architecture, we have encountered some problems. Most of them come from the irregular transfer state of network buffers resulting from adaptive physical channel resource management. A buffer sharing more physical channel resource with others will have longer transfer time across the physical channel. To prevent data miss or duplication due to the irregular transfer state of network buffers, we have added some control signals and functions in each network component, balance the data transfer among all network buffers to prevent data miss or duplication. We also consider this irregular transfer state of network buffers in our adaptive resource management scheme and maintain the physical channel efficiency as high as possible.

5 Simulation environment & performance matrix

5.1 Simulation environment

To verify our network design and measure the network performance, we need a simulation environment that can provide cycle accurate RTL level behavior model, which describes the detail behavior of each network component. In addition, we also need a simulation environment that can support statistic measurement analysis. Therefore a hardware description langue, like Verilog, can not satisfy our need. Although it can describe the detailed behavior of a hardware component in a cycle accurate way, it can not easily support complex statistic measurement analysis.

Besides Verilog, a high level programming language, such as C++, can't describe the detailed behavior of a hardware component, thus, these languages won't satisfy our need either. For these reasons, we try to build up the simulation environment using SystemC.

SystemC is a C++ class library and a methodology that can be used to effectively create a cycle-accurate model for software algorithms, hardware architecture, interfaces of SoC and system-level designs. The SystemC class library provides necessary constructs to model system architecture including hardware timing, concurrency, and reactive behavior that are missing in standard C++ [16]. SystemC also supports C++ library which provides us a convenient environment for statistic measurement analysis on network performance. Due to these characteristics, we adopt SystemC to set up our simulation environment to implement, verify, and measure our network design.

Methods about how higher level layer protocols are implemented are less critical on the network performance than other network design issues. So, in our simulation environment, we will assume that data have already finished their high level layer protocols, like data encryption or data packetization. With this point of view, an abstract class called 'Flit' was built, which contains the necessary information for network simulation such as message number, packet number, flit injection time, …etc. With this abstraction, we can build up a packet class, and a message class in different size.

For high level system abstraction, we let the Flit size equal to the Phit size. By definition: a flit is the logical unit of information that can be transferred across a physical channel in a single cycle. We can also abstract the physical channel width to one flit size.

After creating these system level abstractions, we are going to design a traffic generator module that can generate network traffic in these abstract formats. For a general case study on network performance, we model a traffic generator which generates messages using a Poisson process with an adjustable injection rate at each node This helps us to achieve a fast and general simulation [2].

5.2 Performance metrics

For network performance evaluation, designers have to define the network performance matrix. In this section, we will briefly describe the performance matrix for our network performance evaluation.

1. <u>*Accepted traffic*</u> ***(Flits)****:* total flit numbers that can travel from source to destination during the simulation period.
2. <u>*Accepted traffic per node*</u> ***(Flits)****:* accepted traffic divided by total network node numbers in network.
3. <u>*Throughput*</u> ***(Flits)****:* the maximum amount of accepted traffic.
4. <u>*Flit latency*</u> ***(cycles)****:* cycles needed from flit injection to flit consumption.
5. <u>*Average flit latency for node n*</u> ***(cycles)****:*

$$\left[\frac{\sum_{k=0}^{flit\_number} Cycle\_needed\_from\_flit\_injection\_to\_consumption_k}{total\_number\_of\_flits\_accepted\_by\_node\_n}\right]$$

6. <u>*Overall average flit latency*</u> ***(cycles)****:*

$$\left[\frac{\sum_{n=0}^{node\_number} average\_flit\_latency\_for\_node_n}{node\_number}\right]$$

7. <u>*Applied load for generator n*</u> ***(Flits/cycles)****:*

$$\left[\frac{total\_number\_of\_flits\_generated\_by\_generator\_n}{total\_simulation\_cycles}\right]$$

Using these definitions, the applied load value shows the injection rate of the generator. For example, applied load of 1.0 means that the generator will generate a flit in each cycle; applied load of 0.5

means that the generator will generate a flit in every two cycles.

6 PERFORMANCE EVALUATION OF CYCLE STEALING BUFFERS

6.1 Performance comparisons between two ring networks adopting different buffer architectures

In this section, we would like to show the performance comparisons between two ring topology networks adopting different buffer architectures. One of the architecture is the conventional two ports FIFO, the other is the cycle stealing buffer architecture.

In these performance evaluations, we let other network issues, such as topology, routing, switching method…etc be fixed, and try to make fair comparisons between two different buffer architectures. Those fixed network issues are described in Table. 6.1.

Table. 6.1 Fixed network parameters for performance evaluations between two ring networks adopting different buffer architectures.

	Message Size	Packet Size	Flit size	Number of nodes
Fixed value	1 packet	4 Flits	1 Phit	9

	Topology	Switching	Routing	FIFO size
Fixed value	Ring	Wormhole	Deadlock prevention	4 Flits

	Arbitration method	Simulation Cycle	
Fixed value	Weighted round robin	100000	

The simulation results about average flit latency for the networks adopting different buffer architectures in different applied load per node are shown in Fig 6.1.

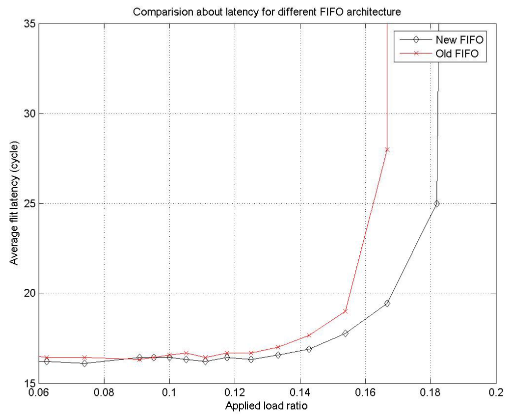

Fig. 6.1 Average flit latency comparison between two ring networks adopting different buffer architectures

In this figure, we can see that the average flit latency of the network adopting cycle stealing buffer architecture is smaller than the network adopting conventional FIFO architecture, especially when the applied load grows larger. With a very small applied load, the two designs have about the same average flit latency. This is expected since each buffer in the networks will never be full.

As a result of average latency reduction, we can expect that the maximum accepted traffic will also improve. These simulation results about accepted traffic for the ring network adopting different buffer architectures are plotted in Fig. 6.2.

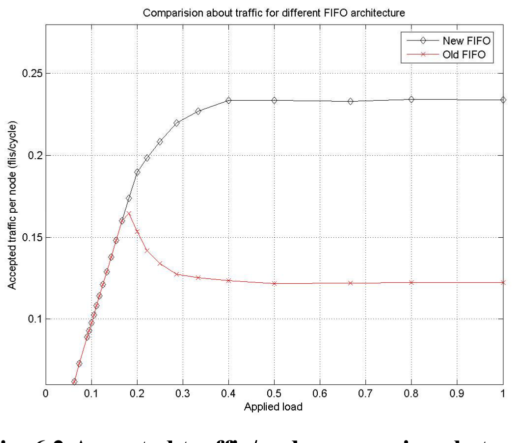

Fig. 6.2 Accepted traffic/node comparison between the ring networks adopting different buffer architectures

From this figure, we can see that there is a large gap in the maximum accepted traffic per node between these two different buffer architectures as the applied load increases. The network adopting conventional FIFO architecture peaks at a smaller applied load than the network adopting cycle stealing buffer architecture. In addition, when the applied load is further increased, the accepted traffic starts decreasing and finally reaches a saturation value smaller than the maximum accepted traffic However, this problem won't happen in the network adopting the cycle stealing buffer architecture.

7 PERFORMANCE EVALUATION OF PHYSICAL CHANNEL MANAGEMENT SCHEME

In this section, we would like to show the performance differences between two different networks adopting different methods to realize multiple virtual channels across one physical channel.

7.1 Performance evaluation between different ring networks adopting different methods to realize multiple virtual channels across one physical channel

First, we would like to show the performance evaluations of ring networks adopting different methods to realize multiple virtual channels across one physical channel. One method is to equally assign the physical channel resource to each virtual channel in round robin. The other is the physical channel management scheme described previously. In these two cases, only one physical channel is used to connect neighbor nodes. For comparison purpose, we also simulate a ring network using two physical channels to break deadlock due to cyclic dependence. In this case, escape channel is not virtualized. This provides the best scenario at the expense of an extra physical channel.

For a fair simulation, some network issues are fixed and described earlier in Table. 6.1.

The simulation results about average flit latency in different applied loads are shown in Fig. 7.1.

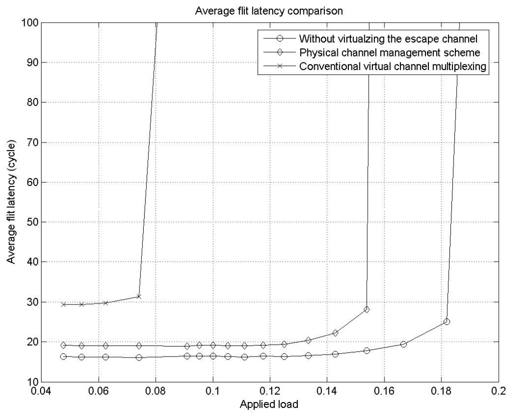

Fig. 7.1 Average flit latency of ring networks adopting different methods to realize multiple virtual channels across one physical channel

In this figure, we can see that by adopting the physical channel resource management scheme, the average flit latency performance (the middle curve in Fig.7.1) is very close to the network without virtualing the escape channel which uses two physical channels instead of one physical channel (the curve on the right). In addition, the performance is also much better than the case just equally assigning the physical channel resources to each virtual channel across it (the curve on the left in Fig.7.1).

The accepted traffic per node in different applied load can also be plotted in Fig. 7.2.

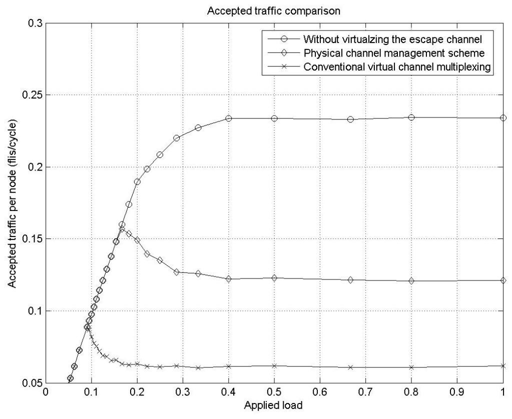

Fig. 7.2 Accepted traffic/node of ring networks adopting different methods to realize multiple virtual channels across one physical channel

In this figure, the maximum value of the accepted traffic in the ring network adopting the physical channel management scheme is smaller than the network without virtualing the escape channel which uses two physical channels instead of one. However, the performance is much better than just equally assign the physical channel resource to each virtual channel across it (only one physical channel is used to connect neighbor nodes).

Since the network adopting the physical channel management scheme uses only one physical channel, and the network without virtulizing the escape channel uses two physical channels, it is reasonable to expect that the accepted traffics per node using physical channel management scheme is smaller. However, the network peaks at a smaller applied load. When we increase the applied load further, the accepted traffic starts decreasing and finally reaches a steady state value smaller than the maximum accepted traffic.

Therefore, we can conclude that under the same amount of physical channel resources, using the physical channel management scheme can certainly improve the overall network performance.

8 CONCLUSIONS

In this paper, the cycle stealing buffer architecture is proposed, which eliminates the waste cycles in a read/write event. From simulation results, we clearly prove that this buffer architecture can improve the network performance with only a small extra cost on hardware.

In addition, the physical channel management scheme is proposed, which manages how one physical channel resource can be shared by multiple virtual channels effectively. This scheme can improve the physical channel data transfer efficiency and reduce the congestion problem due to a hot spot.

From simulation results, we also prove that the network adopting physical channel management scheme performs much better than the network adopting conventional virtual channel multiplexing. And the performance is just slightly smaller than the network without virtualing the escape channel which uses two physical channels instead of one physical channel. Therefore, we can conclude that the physical channel management scheme is an efficient solution for multiple virtual channels to share one physical channel resource.

Acknowledgements

This work is supported by National Science Council, Taiwan under grant 96-2220-E-007-032 and ITRI under grant of 62-96032

REFERENCES

[1] J. Duato, S. Yalamanchili, and L. Ni, Interconnection Networks – An Engineering Approach. Morgan Kaufmann, 2002.

[2] L .Benini and G. De micheli, "Networks on Chips: A New SoC Paradigm," Computer, vol.35, no.1,pp. 70-78,Jan.2002.

[3] P. Magatshack and P. G. Paulin, "System-on-Chip beyond the Nanometer Wall," Proc. Design Automation Conf. (DAC), pp.419-424, June 2003.

[4] M. Oka and M. Suzuoki, "Designing and Programming the Emotion Engine,"IEEE Mirco, Vol. 19, No.6, November-December 1999, pp. 20-28.

[5] D. Pham, et al., "Overview of the Architecture, Circuit Design, and Physical State Circuits of a first generation Cell processor", IEEE Journal of Solid State Circuits Vol. 41, No.1, January 2006,pp 179-196.

[6] M.A. Horowitz et al., "The Future of Wires," Proc. IEEE,vol.89, no.4, pp. 490-504, Apr. 2001.

[7] D. Sylvester and K. Keutzer, "Impact of Small Process Geometries on Microarchitectures in Systems on a Chip," Proc. IEEE, vol.89, no. 4, pp.467-489, Apr.2001.

[8] Kyeong Keol Ryu, Eung Shin, and Vincent J. Mooney, "A Comparison of Five Different Multi processor SoC Bus Architectures," Digital Systems, Design, 2001. Proceedings. Euromicro Symposium on 4-6 Sept. 2001 pp.202-209

[9] J. Plosila, T. Seceleanu, and P. Liljeberg, "Implementation of a self-timed segmented bus," Design & Test of Computers, IEEE Vol. 20, Issue 6, Nov.-Dec. 2003 pp.44 – 50

[10] J. Plosila, P. Liljeberg, J. Isoaho, "Pipelined on-chip bus architecture with distributed self-timed control," Signals, Circuits and Systems, 2003. SCS 2003. International Symposium on Vol. 1, 10-11 July 2003 pp.257 – 260

[11] L. Benini and G. De Micheli, Networks on chip – Technology and Tools. Morgan Kaufmann, 2006.

[12] L. De Coster,N. Dewulf, and C. T. Ho,"Efficient Multi-Packet Multicast Algorithms on Meshs with Wormhole and Dimension-Ordered Routing," Proc. 1995 Int'l Conf. Parallel processing, Vol.3 IEEE CS Press, Los Alamitics, Calif.

[13] Michael D.Ciletti, Modeling, synthesis and rapid prototyping with the Verilog HDL. Prentice Hall, 1999.

[14] W. J. Dally, "Virtual-channel Flow Control," IEEE Trans. On Parallel and Distributed systems, March, 1992.

[15] W. J. Dally and C. L. Seitz, "Deadlock-Free Message Routing in Multiprocessor Interconnection Networks," IEEE Transaction on Computers, Vol. 36, No. 5, May 1987, pp.547-553.

[16] SystemC User's Guide Version 2.0