Network Interface Buffer Elimination

EECS 578 Final Report

Qilu Guo, Jing Ji, Jiong Xue
Department of Electrical and Computer Engineering
University of Michigan, Ann Arbor, MI 48109, USA
{qiluguo, silverji, xuejiong} @umich.edu

***Abstract*—Aggressive transistor scaling continues to increase integration capacity with each new technology node. In recent five years, however, the performance gap between Moore's Law and the art-of-design technology is getting larger, and area and power concerns are drawing much more attention, especially for network-on-chip design. Much attention has been paid on routers, and few on network interface. However, the storage of the data flits in the network interface causes an area overhead and therefore consumes more power. In this project, we propose eliminating buffers in the network interface and preserving the data in the cache instead to reduce overall area. From the simulation result, the proposed design reduces the area of the network interface by 6 times without degrading the performance.**

***Keywords*—Network interface, network-on-chip, area saving, buffer elimination**

I. Introduction

Multiprocessor architectures and platforms have been introduced to keep up with the Moore's law [1]. The general design trend in processor development has moved from dual- and quad-core processor chips to ones with tens and even hundreds of cores [2]. Network-on-Chip (NoC) is a general-purpose on-chip communication concept that offers high throughput, which is the basic requirement to deal with complexity of modern systems. All links in NoC can be simultaneously used for data transmission, which provides a high level of parallelism and makes it attractive to replace the typical communication architectures like shared buses or point-to-point dedicated wires [3]. However, all of these advantages come with a high expense in both area and power.

Many efforts have been put on routing algorithm optimization. Devices with different purposes have different requirements for routing algorithms. These various communication topologies for NoC architecture developed so far include mesh, torus, ring, butterfly, octagon and irregular interconnection networks [4]. It has no doubt that they all have their own strengths and many researchers have exploited them such as [5], [6] and [7].

Router design enhancement has been paid much attention as well. The router contains buffers that consume 64% of the total node leakage power [8]. It is not a recent issue to enhance buffer management and most researchers have proposed methodologies that focus on buffer full utilization and optimization of buffer decoupling [9][10].

It should be observed also that in realistic NoC architectures, the network interface (NI) plays a significant role in determining overall NoC area, and a reduction in area means lower cost and less power consumption. Few researches have been made on NI area reduction. In [11], a TV companion chip was redesigned with a NoC as the interconnect fabric, and a 78% of increase in chip area was proved to come from the NIs. Among current few researches, some proposed ideas on network interface sharing, like [12]. However, [12] involves replication of the buffering resources in the NI, thus leading to an increase of the area, which hardly justifies this design choice [13]. Another approach delves into NI design and proposes reducing computation complexity [14]. It can make more sense with advanced communication technology, but that is not what our purpose is for the general approach we are going to propose in this paper.

This paper focuses on reducing NI area by eliminating data buffers stored in NI and this approach can be supported with caches individual to each processor within a multiprocessor network. The rest of the paper is organized as follows. Section II presents the implementation of our design and Section III evaluates the experimental results on area and performance with the baseline design. Section VI introduces the future work of our project. Finally, Section V concludes the paper.

II. Proposed Approach

Instead of storing the data in the network interface, we propose to preserve the data in the cache. Hence, we modify the communication scheme to allow direct data transmission between the cache and the router. Thus, the network interface only stores the head and the tail flits of the packet. Fig. 1 shows the high-level architecture of the proposed design.

The proposed design mainly involves modifying three components in an original NoC: network interface, router and cache. The rest of the session gives a brief introduction to these components and the transmission protocol between them.

A. Network Interface

The network interface stores two pairs of head and tail flits, corresponding to "send" and "receive" process respectively. The "send" process refers to sending data out of the cache to the router and the "receive" process is the opposite. The head

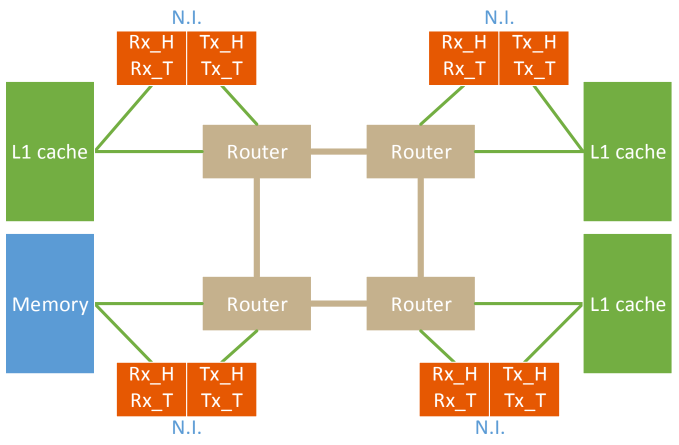

Fig. 1. **High-level architecture of the proposed design.** The network interface only stores the head and the tail flits of the packet and the data are preserved in the cache. Also, the data are allowed to transmit directly between the cache and the router.

flit contains essential information when transmitting packets. It is composed of (1) source and destination coordinates within the network; (2) the number of data flits in one packet; and (3) the memory address of the leading data flits. In addition, the head flit contains several bits for identifying the type of the packet as well, such as one bit indicating whether it is a data request or eviction, and another one indicating whether it is an acknowledgement packet or not. The tail flit contains the same information as the head one except that the most significant bit is set to 0 to identify that it as a tail. The composition of the head/tail flit is shown in Fig. 2. The number of data flits in the packet is related to the packet type. If it is an acknowledgement or data request packet, the flit number is 0; otherwise, the flit number is 4.

63	62	61	[60:39]	38	37	36	35	[34:32]	[31:0]
head/tail	req/evict	ack	22'b0	src_x	src_y	dest_x	dest_y	flit #	addr

Fig. 2. **Composition of the head/tail flit.** The flit cosists of (1) whether it is a head or tail; (2) whether it is a data request or eviction; (3) whether it is an acknowledgement packet or not; (4) source and destination coordinates within the network; (5) the number of data flits in one packet; and (6) the memory address of the leading data flits.

B. *Router*

At the beginning of this project, we planned to use the Open Source Network-on-Chip Router RTL design from Stanford University [15], which is a parameterized RTL implementation of the state-of-art VC router. However, we did not choose to use their design for two reasons: (1) We do not need to include virtual channels to resolve deadlock for our 2x2 mesh network. Instead, we implemented the routing algorithm as deterministic X-Y routing. (2) Since our design requires modifications to the router, it takes a lot of efforts to understand the original RTL codes and modify them in order to support communication with the cache. Therefore, we decided to design and implement the router ourselves.

For the router we designed, it applies packet-based flow control, and is store and forward to be specific. It means that the head flit waits at the router until the entire packet is received before being forwarded to the next hop. Although store and flow is not the state-of-art design choice for routers and per-hop routing latency is large, it is not a concern for this project as long as the baseline also uses the same flow control. It is more straightforward to implement store and forward control flow as well. The routing algorithm is circuit-based deterministic X-Y routing. In order to for the router to have direct communication with the cache, data ports and other control signal ports are augmented to the baseline router design and more control logic is added.

C. *Cache*

The L1 cache is implemented as an N-way write-back, write-allocate cache. When a read or write instruction comes, there are three conditions for the cache: cache hit, cache miss on an invalid cache line and cache miss on a valid cache line. The cache handles these conditions using the finite state machine shown in Fig. 3.

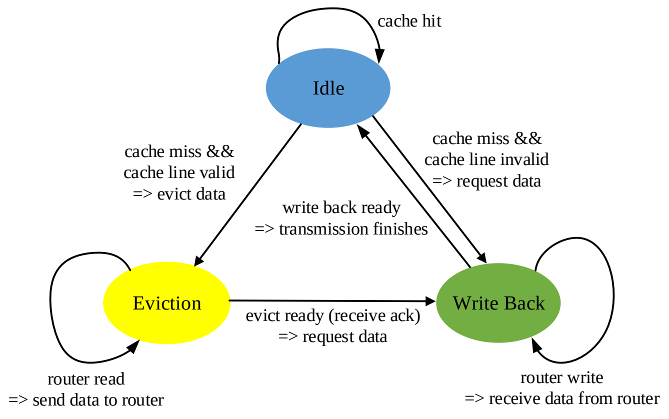

Fig. 3. **Finite state machine of the cache.** Intially, the cache is in idle state and stays in the state unless a cache miss happens. If a cache miss happens on an invalid cache line, the cache will enter the “Write Back” state waiting for the requested data to be retrieved. If a cache miss happens on a valid cache line, the cache will enter the “Eviction” state to evict the cache line first and then go to the “Write Back” state. The cache is blocked until the data transmission completes.

If it is a cache hit, the cache will transmit the data to the processor for a read or store the data to the corresponding cache line for a write to complete the execution. If a cache miss happens on an invalid cache line, the cache will enter the state waiting for the requested data to be retrieved. If a cache miss happens on a valid cache line, the cache will evict the cache line first and then wait for the requested data. The cache is blocked until the data transmission completes.

The implementation of the cache and the cache controller are completely the same for the proposed design and the baseline except that the read and the write ports are connected to the router instead of the network interface.

D. *Data Transmission Protocol*

Since data do not need to be stored in the network interface anymore, the data transmission protocol is modified to support transmission of data between the cache and the router directly. If it is a cache hit, the network interface and the channel for this core in the router will be kept in the idle state. However, if a cache miss happens, data start to transmit within the network.

When a cache miss happens on an invalid cache line, the cache will send a data request to the network interface asking for data from the memory. The network interface generates the head and tail flits containing the information of the request and then sends them to the router when the router is available. Once the router receives the tail, it passes the request to the destination node. The data flow of sending a data request packet is shown in Fig. 4.

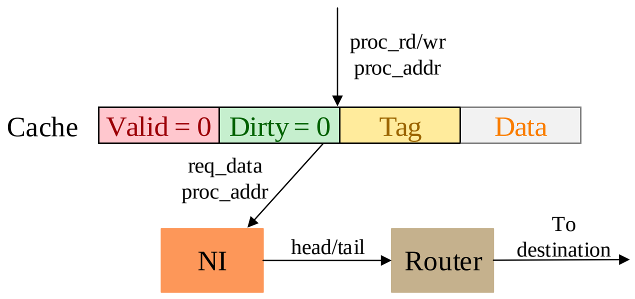

Fig. 4. **Data flow of sending a data request packet.** The cache sends data request to the network interface first. Then, the network interface generates the head and tail flits containing the information of the request and sends them one by one to the router which passes them to the destination node when the tail is received.

When the destination router receives the request, it will give the head and tail flits to the network interface. The network interface generates the new head and tail flits according to the received head flits and then sends the new ones to the router. After the router receives the new head flit, it reads the data flits one by one from the cache and completes the packet on receiving the tail from the network interface. Extracted from the received head, the address of the data flits is provided by the network interface. Later, the router sends the packet containing the requested data back to the source node.

After the source router receives the packet, it first sends the head flit to the network interface. The network interface then deconstructs the head flit to obtain the data address and signals the cache to receive data from the router. Once the cache receives all data, the router sends the tail to the network interface to signal the end of the process. The data flow of receiving a data packet is shown in Fig. 5.

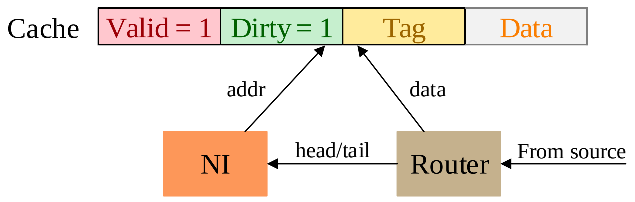

Fig. 5. **Data flow of receiving a data packet.** The router first sends the head flit to the network interface. The network interface then deconstructs the head flit to obtain the data address and signals the cache to receive data from the router. Once the cache receives all data, the router sends the tail to the network interface to signal the end of the process.

Fig. 6 summarizes the whole data transmission process when a cache miss happens on an invalid cache line.

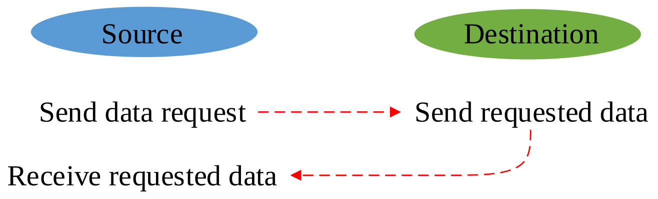

Fig. 6. **Cache miss on an invalid cache line.** The source node sends packet to the destination node to request data. Then the desitination node sends the corresponding data accordding to the request packet to the source node.

When a cache miss happens on a valid cache line, eviction of that cache line is needed first. The cache sends an eviction request and the corresponding eviction address to the network interface. Then, the network interface constructs the head and tail flits based on the eviction address. Once the head flit is successfully sent to the router, the network interface signals the cache to start sending data flit by flit to the router. After the router receives all the data, it signals the network interface to send the tail to it. When the router gets the whole packet, it starts to send the packet to the destination. The data flow of sending an evicting data packet is shown in Fig. 7.

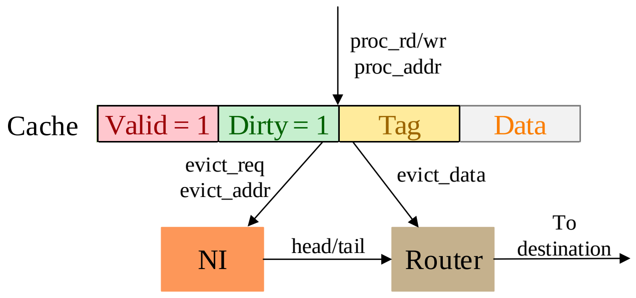

Fig. 7. **Data flow of sending an evicting data packet.** The cache sends an eviction request and the eviction address to the network interface. Then, the network interface constructs the head and tail flits. The router first receives the head from the network interface and then asks the cache for data. When the tail arrives, it sends the whole packet to the destination.

The destination node receives the packet according to the process indicated in Fig. 5 first. When the receiving process completes, the network interface constructs an acknowledgement packet and transmits it to the source through routers. Once receiving the acknowledgement, the evicted cache line is invalidated and a data request packet is sent as shown in Fig. 4. The following process is the same as the cache miss on an invalid cache line. The overall process for this situation is shown in Fig. 8.

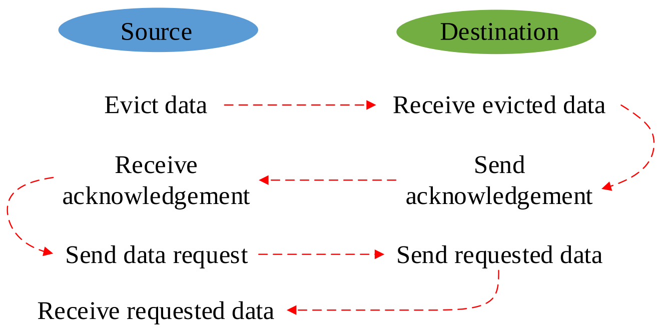

Fig. 8. **Cache miss on a valid cache line**. The source node first sends evicted data to the destination node which sends an acknowledgement back to signal successfully reception to the source. The following steps is the same as Fig. 4.

The acknowledge signal is implemented to ensure that the evicted data are invalidated from the cache only after they have been written back to the memory. In this way, the data will not be lost if the transmission fails.

III. Evaluation

The goal of our evaluation is to determine the area and performance of our design compared with the baseline. This session includes a brief overview of the baseline design, area evaluation and performance evaluation.

A. Baseline

All evaluations are conducted comparing to the baseline within the same network. In this baseline design, the network interface stores data flits in its buffer, and is in charge of transferring data flits to and from the router. The router thus has no communication channel with the cache, and the cache connects only to the network interface as well. No major changes are made to the cache between the baseline and our design. At current stage, we do not include acknowledgement in the baseline design for simplicity. Comparison results between the baseline and our design are shown in table 1.

TABLE 1. COMPARISION RESULTS

Components	Baseline	Our Design
Cache	N-way write-back, write-allocate cache Blocked until the transmission completes	
Network Interface	Store head, data and tail flits	Only store head and tail flits
Router	Communicate only with the network interface	Communicate with both the network interface and the cache

B. Area Evaluation

To compare the area of the baseline and our design, we synthesized the modules using Synopsys Design Compiler to obtain the actual area estimations. The technology that this compiler uses is 130 nm. We did not change the L1 cache module at all and only the connection is changed, e.g. the output data port is connected with the network interface in the baseline and it is connected with the router in our design. Thus, the area of the cache is not changed from the baseline to our design. The synthesized area results for the network interface and the router are shown in table 2.

TABLE 2. SYNTHESIZED AREA RESULTS

Area Evaluation		
module	baseline (μm2)	our design (μm2)
router	512,292	518,037
network interface	179,558	31,518

From the table, we can see that the area overhead of the router is extremely small. The reason is that the storage of the data flits takes up most of the area and adding ports and control logic to the baseline router does not make a huge impact on the overall router area. Another observation is that the area of the network interface is reduced by almost six times. This is achieved by network interface buffer elimination and it successfully proves the concept of our design. The last point is that we believe that the power consumption will be reduced as the overall area decreases. We do not have the tool for the power measurement for now, so we leave it as a future work.

Overall, the reduction of the area in the network interface is far greater than the increase of the area in the router. Hence, there is a reduction in the total area for our design.

C. Performance Evaluation

Besides of the area evaluation, we want to see whether the design will cause any performance overhead or not. Since our design does not include any processor cores, traces are needed as the inputs for the caches. We obtained the memory access traces and cycle delays between memory accesses by running test cases on the EECS 470 project. Then we injected memory accesses and delays between these accesses to both our design and the baseline and measured the total execution cycles.

First, we injected memory accesses with delay cycles between them for three cores with the same test case mapping to different portions of memory and the associativity of the cache is 4-way. The result is shown in Fig. 9. From the figure, it is clear to see that our design does not result in any performance overhead and instead it improves the performance a little. The performance improvement ranges from 0.28% - 1.54%, with the average being 0.61%. The direct communication between the L1 cache and the router without going through the network interface reduces the packet injection and packet reception latency. Take evicting four flits of a cache line as an example. The baseline requires all the flits sent to the network interface before going to the router while the router of our design gets the body flits directly from the cache, which saves four cycles for the packet injection.

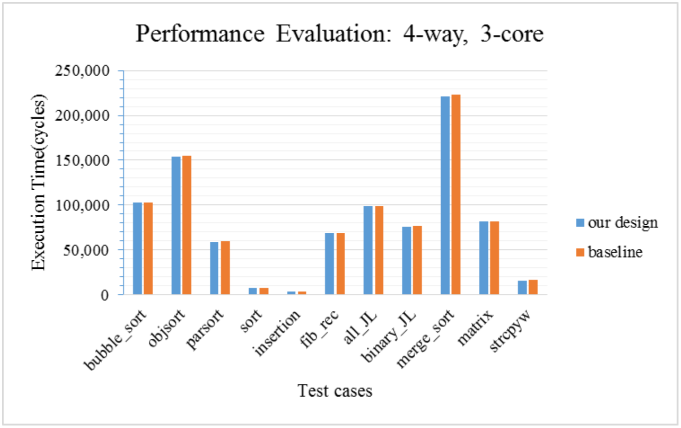

Fig. 9. **Results from running on 3 cores with 4-way associative cache.** Our design has a slight performance improvement over the baseline. The average performance improvement is 0.61%.

Next, we changed the associativity of the cache from 4-way to 2-way and the result is shown in Fig 10. The observation is similar. The execution cycles increase for some test cases due to the increased evictions resulting from the low associativity. This time, the performance improvement ranges from 0.43% - 0.77%, with the average of 0.16%. The reason behind this is that more evictions exist for 2-way associative cache and an eviction saves a few cycles, so more evictions results in greater performance improvement.

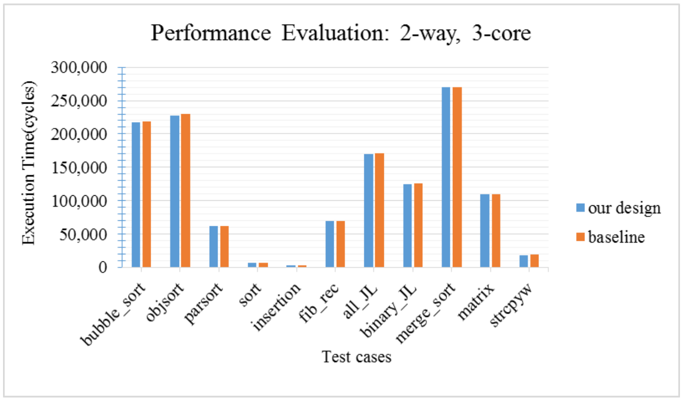

Fig. 10. **Results from running on 3 cores with 2-way associative cache.** The average performance improvement is 0.77%. The increase of the performance improvement is due to the increased evictions.

Last, we want to show how our design performs when a single core runs. The test cases are picked among those that have a large number of memory accesses. The result is shown in Fig. 11. Our design still outperforms the baseline. The average performance improvement is 4.91%. The performance improvement for one core is far greater than the performance improvement for three cores. The reason is that for three cores, when one core is not issuing memory accesses, the latencies of memory accesses from other cores can be hidden during this memory idle window.

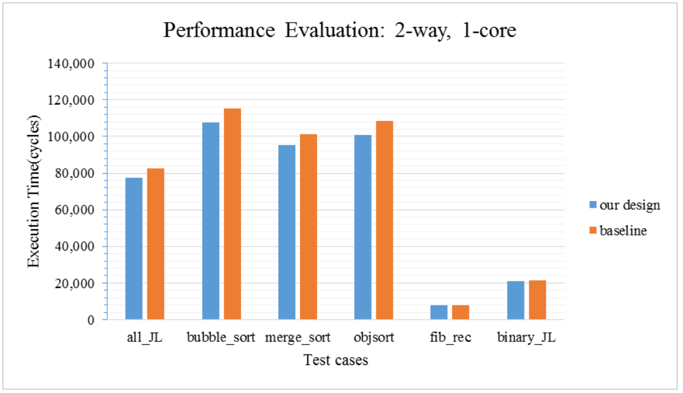

Fig. 11. Running on 1 core with 2-way associative cache. We pick test cases with many memory accesses. The average performance is 4.91%, far greater than the performance improvement running on 3 cores due to the latency hiding in the 3 cores.

To summarize, our design has no performance overhead and instead has a small performance improvement. The improvement exists due to the use of blocking cache. If the non-blocking cache is used, extra performance overhead would be introduced. In that case, our design might have a small performance overhead compared to the baseline.

IV. FUTURE WORK

The future work can be explored in the following directions.

(1) We implemented the simplest 2x2 mesh to prove the feasibility of the proposed approach, so the design can be expanded to larger network in the future.

(2) The proposed approach can only detect the error for evicted data loss, so more effort can be put on checking more types of error such as misrouted packet, and implemented recovery mechanism to improve the reliability.

(3) For simplicity, the flow control of the proposed router is store and forward, which can be changed to wormhole to improve the performance in the future.

(4) To improve the performance further, the proposed design can be modified to apply to the non-blocking cache.

V. CONCLUSION

As the performance gap between Moore's law and the art-of-design technology gets larger, multiprocessor architectures and platforms have been drawing much more attention. Network-on-chip is a general-purpose on-chip communication concept, which is the basic requirement to deal with complexity of modern system. Both area and performance are critical issues and the network interface influences a lot within the network.

In this paper, we propose a method that can reduce the network interface by 6x compared to the baseline design by eliminating data buffers. It is also worth notice that this change of design for network interface does not degrade the overall performance according to our evaluation. More memory access intensive test cases can be conducted to have a more thorough analysis on our design. At this stage, we conclude that by eliminating the data buffer in network interface and preserving all data flits in cache, the area of the network interface is decreases by 6 times while the performance does not degrade.

ACKNOWLEDGMENT

We would like to thank Professor Valeria Bertacco for her instructions and support that guide us through the research process. We would also like to express our deepest thank to our GSI Doowon Lee for his kind help that always comes in time.

REFERENCES

[1] Ankur Agarwal, Cyril Iskander, Ravi Shankar. Survey of Network on Chip (NoC) Architectures and Contributions. Journal of Engineering, Computing and Architecture, vol 3, 2009

[2] Rakesh, K., Timothy, G.M., Gilles, P., Rob, V.D.W.: The Case for Message Passing on Many-Core Chips. Multiprocessor System-on-Chip, pp. 115–123 (2011)

[3] Latif, K., Seceleanu, T., & Tenhunen, H. (n.d.). Power and Area Efficient Design of Network-on-Chip Router through Utilization of Idle Buffers. 2010 17th IEEE International Conference and Workshops on Engineering of Computer Based Systems.

[4] J.Dally and B. Towles, Principles and Practices of Interconnection Networks, Morgan Kaufmann, 2004.

[5] D. Kim, Manho Kim, and G.E. Sobelman, “CDMA-based NoC architecture”, Proc. IEEE Conference on Circuits and Systems, vol. 1, pp. 137-140, 2004.

[6] A. Adriahantenaina, H. Charlery, A. Greiner, L. Mortiez, and C.A. Zeferino, “SPIN: a scalable, packet switched, on-chip micro-network, Proc. IEEE Conference on Design, Automation and Test, pp. 70-73, 2003.

[7] F. Karim A. Nguyen, and S. Dey, “An interconnect architecture for networking systems on chips”, IEEE Journal on Micro High Performance Interconnect, vol. 22, issue 5, pp. 36-45, Sept 2002.

[8] Xuning Chen and Li-Shiuan Peh. Leakage power modeling and optimization of interconnection net- works. Proceedings of International Symposium on Low Power Electronics and Design, pp. 9095, 2003.

[9] Ying-Cherng Lan, Shih-Hsin Lo, Yueh-Chi Lin, Yu- Hen Hu, Sao-Jie Chen. BiNoC: A bidirectional NoC architecture with dynamic self-reconfigurable channel. Proceedings of 3rd ACM/IEEE International Sympo- sium on Networks-on-Chip (NoCS), pp.266-275, May 2009.

[10] M. Coenen et. all. A buffer-sizing algorithm for net- works on chip using TDMA and credit-based end-to- end flow control. Proceedings of the 4th international conference on Hardware/software codesign and system synthesis (CODES+ISSS), pp.130-135, October 2006.

[11] F.Steenhof et al., Networks on Chips for high-end consumer- electronics TV system architectures, DATE, 2006, pp.148-153.

[12] A.Radulescu, J.Dielissen, K.Goossens, E.Rijpkema, P.Wielage, An Efficient On-Chip Network Interface Offering Guaranteed Services, Shared-Memory Abstraction, and Flexi- ble Network Configuration, DATE 2004, pp.873-883.

[13] Ferrante, A., Medardoni, S., & Bertozzi, D. (n.d.). Network Interface Sharing Techniques for Area Optimized NoC Architectures. 2008 11th EUROMICRO Conference on Digital System Design Architectures, Methods and Tools.

[14] Kim et al., Solutions for Real Chip Implementation Issues of NoC and Their Application to Memory-Centric NoC, Int. Symp. on Networks-on-Chip, pp.30-39, 2007.

[15] Open Source Network-on-Chip Router RTL, http://nocs.stanford.edu/cgi-bin/trac.cgi/wiki/Resources/Router