A System Level Network-on-chip Model with MLDesigner

Ankur Agarwal, Rabi Shankar, A. S. Pandya and Young-Uhg Lho, *Member, KIMICS*

Abstract—Multiprocessor architectures and platforms, such as, a multiprocessor system on chip (MPSoC) recently introduced to extend the applicability of the Moore's law, depend upon concurrency and synchronization in both software and hardware to enhance design productivity and system performance. With the rapidly approaching billion transistors era, some of the main problem in deep sub-micron technologies characterized by gate lengths in the range of 60-90 nm will arise from non scalable wire delays, errors in signal integrity and non-synchronized communication. These problems may be addressed by the use of Network on Chip (NOC) architecture for future System-on-Chip (SoC). We have modeled a concurrent architecture for a customizable and scalable NOC in a system level modeling environment using MLDesigner (from MLD Inc.). Varying network loads under various traffic scenarios were applied to obtain realistic performance metrics. We provide the simulation results for latency as a function of the buffer size. We have abstracted the area results for NOC components from its FPGA implementation. Modeled NOC architecture supports three different levels of quality-of-service (QoS).

Index Terms—Network On Chip, System On Chip, embedded system, Quality of Service, MLDesigner

I. INTRODUCTION

Advances in semiconductor technologies have led to continual and rapid transistor scaling; this will soon facilitate the integration of billion transistors on a small size integrated circuit. Product development time will increase from the current two years to five years or more, due to the increasing hardware complexity and the concomitant increasing sophistication of the applications demanded by the consumer. Traditional SoC systems may integrate one or two general processors, with a few slaves as application specific processors (ASPs). Such a system is divided into several independent functional blocks, which are then designed, most likely as dedicated hardware engines (ASPs) or as tasks on one or two processors. These functional blocks synchronize their activities by means of local operating system(s) on the processor(s) and an on-chip communication bus such as AMBA[1]. With increasing user demands for computationally extensive applications on an embedded system, such as, multimedia, real-time video communication, and 3-D video gaming, this model is no longer viable. Current Designs cannot be separated so clearly into hardware engines and software processes. QoS requirements for real-time performance, mobility, and flexibility (to address various consumer preferences) are too complex to be addressed so simplistically.

Moore's law predicts that a chip in 2010 will have more than four billion transistors operating in the multi GHz range[2]. This is mainly due to the near-exponential decrease in the transistor size enabling faster transistor switching times and more densely packed integrated circuits. Such computation power has posed some challenges which include the disparity in transistor and wire speeds, and increased power dissipation, leading to a decrease in the area of the chip which can be utilized with a single clock cycle[3]. Under such considerations, a single-processor implementation will not suffice[4]. This has resulted in exploitation of multi-core architectures, thus driving the development of increasingly complex MPSoCs[3]. Such MPSoC platforms provide a new avenue for designers to build real-time systems. The consequences of this trend imply a shift in concern from computation and sequential algorithms to addressing concurrency, synchronization and communication issues in every aspect of hardware and software co-design and development. As the technology scaling works better for transistors than for interconnecting wires[5], there is an increasing disparity between the wires and the transistors in terms of power consumption and latency. Thus, bus based communication becomes a bottleneck[6]. It is estimated that in the 50 nm technology, global wire delays will amount to 6-10 clock cycles[5]. On a billion transistor chip, it would not be possible to send a global signal across the chip in real-time. As a result, achieving synchronization onto the system will be very difficult, if not impossible. Moreover a bus based SoC or MPSoC does not offer the required amount of reuse in order to meet the time-to-market requirements. Technology scaling has also had some unwanted side effects. This includes cross coupling, noise, and transient errors[6]. This will continue to impact the productivity of system architects and designers.

Manuscript received February 7, 2008; revised May 3, 2008. Ankur Agarwal, Rabi Shankan and A. S. Pandya are with the Dept. of Computer Science and Engineering, Florida Atlantic University, Boca Raton, FL-33431, USA.

Young-Uhg Lho is with the Dept. of Computer Education, Silla University, Busan, 617-736, Korea (Tel: +82-51-309-5570, Email: yulho@silla.ac.kr)

NOCs can improve design productivity by supporting modularity and reuse of complex cores, thus, enabling a higher level of abstraction in the architectural modeling of future systems[7]. No delays are experienced for accessing the communication infrastructure, since multiple outstanding transactions originated by multiple cores can be handled at the same time, resulting in more efficient network resource utilization. However, given a certain network dimension (that is, the number of instantiated switches), large latency fluctuations for packet delivery could be experienced as a consequence of network congestion. Consequent delay and uncertainty would be unacceptable in real-time systems. There are two possible solutions to this problem: (1) Over-dimensioning the network to optimize behavior even for worst case scenarios; (2) Providing priority levels to the traffic. The second solution is a cost-effective solution in terms of power consumption. In this way, real-time requirements can be met by providing priority to real-time traffic. Such real-time traffic can be guaranteed its delivery to its destination by either reserving some paths for real-time data, or implementing a priority-based scheduling criterion.

In this paper we present a system level model of the our NOC, called ONOC, after our project entitled "One Pass to Production," where our goal is to develop a seamless integrated methodology for developing a mobile embedded system . Our model has been tested for different loads and distribution patterns.

II. BACKGROUND

Researchers have suggested various network topologies including a star based NOC[8], a tree based implementation of NOC[9], mesh torus and ring[10]. New algorithms have been proposed to reduce power consumption while securing cost optimization[11]. NOC architectures use packet switched networks. New and efficient router architectures have evolved for the NOC [12]. These routers will be responsible for routing the entire traffic across the chip and have to interface with switches and resources in the NOC architecture. Various routing algorithms have been proposed for the NOC environment. Most of the researchers have suggested static routing algorithms and have performed communication analysis for NOC based on static behavior of the processes, thus obtaining a static routing for NOC. Siebenborn et. al. use communication dependency graph (CDG) to analyze inter-process communication[13]. Hu et. al. provide graph based application decomposition and mapping strategies for NOC[14]. Madsen et. al. provide a real-time operating system (RTOS) architecture for NOC[15]. They provide RTOS-based application scheduling techniques for use on a NOC. Based on routing strategies, various router implementations have also been proposed. Wolkotte et. al. proposed circuit switched router architecture for NOC [16]. Dally and Towles propose a router architecture for packet based switching[17]. Albenes and Susin provide a wormhole based packet forwarding scheme[18]. Design of a virtual channel is another important aspect of NOC. A virtual channel splits a single channel into two channels, virtually providing two paths for the packets to be routed over. Bjerregaard and Sparso implement virtual channel router using asynchronous circuit techniques[19]. Another important aspect of NOC is the design of interconnects. Brager et. al. propose transmission line based design of interconnects for NOC [20]. Morgenshtein et. al. provide a comparative analysis for serial and parallel links for interconnect implementation[21]. Need for implementation of fault tolerance, error detection, and error correction techniques is not certain for on chip implementation. QNOC implementation of NOC argues that the communication strategies for on chip network may be considered reliable [22] while Bertozzi et. al. propose error detection and correction schemes for data on NOC link[23]. [24] proposed an integrated NOC implementation.

III. ONOC Model

We have modeled all the key NOC building blocks (we call them "classes") in MLDesigner, a system level design and modeling environment. These classes were identified from a high level system specification. They are: Producer (P), Consumer(C), Input Buffer (IB), Scheduler (S), Output Buffer (OB) and Router (R). We modeled ONOC to support three priority levels for data traffic: High, Mid and Low. High priority supports control signals and have a short packet (single flit packet). Mid priority supports real-time traffic on the system, while Low priority supports non-real time block transfers of data packets. In this section, we discuss the specific implementation of our ONOC classes.

A. Producer Class

Producer comprises of a resource and a network interface (NI). Producer generates the required traffic pattern and packetizes the data into flits. A Flit is the smallest unit of communication supported in ONOC. The customizable parameters for Producer are: (1) The distribution pattern of the data; (2) Amount of data generated as a function of time (throughput); (3) Priorities of the generated data - High, Mid or Low; and (4) The source and the destination addresses for the data (route information). We used three statistically generated traffic distribution patterns - uniform, exponential and Bernoulli. Producer outputs a flit when buffAvail is asserted by Buffer. A flit is time stamped at the time of its generation. The time stamp is used to determine latency involved in delivering the flit. The source and destination address fields of the flit header are updated at this time. Fig. 1 shows the flit header: It has fields for its priority, time stamp, x-direction of source address, y-direction of the source address, x-direction of the destination address, and the y-direction of the destination address. The priority of this flit is governed as per a statistical distribution block. For example, in case of a uniform distribution pattern,

every third flit will be a high priority flit. Once the new flit has its time stamp, source and destination address and priority fields updated, it is then forwarded to the output through dataOut.

Priority	Time Stamp	Source Address (X)	Source Address (Y)	Destination Address (X)	Destination Address (Y)

Fig 1. Flit Header

B. Input Buffer Class

This stores the incoming flits, generates the proper handshaking signals to communicate with Scheduler and forwards the flits to the next Router. Data forwarding path has been implemented based on "request-grant" signaling approach. Incoming flits corresponding to all the priority levels (High, Mid and Low) are stored in a common buffer.Let *buffSize* represent all the available space in Input Buffer. The *buffAvail* signal indicates whether there is space available in Input Buffer. The stored data flits are forwarded to Router based on their priority. High priority flits are forwarded before the Mid or Low priority flits, etc. The availability of data flits in Input Buffer for further transmission is indicated by *shortDataInBuff* and *dataInBuff* signals. The *shortDataInBuff* signals that a High priority flit is stored in the input buffer. Similarly, presence of Mid and Low priority data flits stored in Input buffer is indicated by *dataInBuff*. If a grant comes in response any of these requests (*nodeGrantFast* for *shortDataInBuff* or *nodeGrantSlow* for *dataInBuff*), the flit stored in the buffer is forwarded to the corresponding Router. A Flit is not removed from Input Buffer until a confirmation (via *confirm*) is received from Scheduler. If *confirm* is not received, the data flit is not removed from Input Buffer; however, it will be queued for later forwarding. Fig. 2 shows the MLD implementation of the Input Buffer class. Initial condition in Input Buffer is generated by "*init*" signal. It defines the space available to store flits.

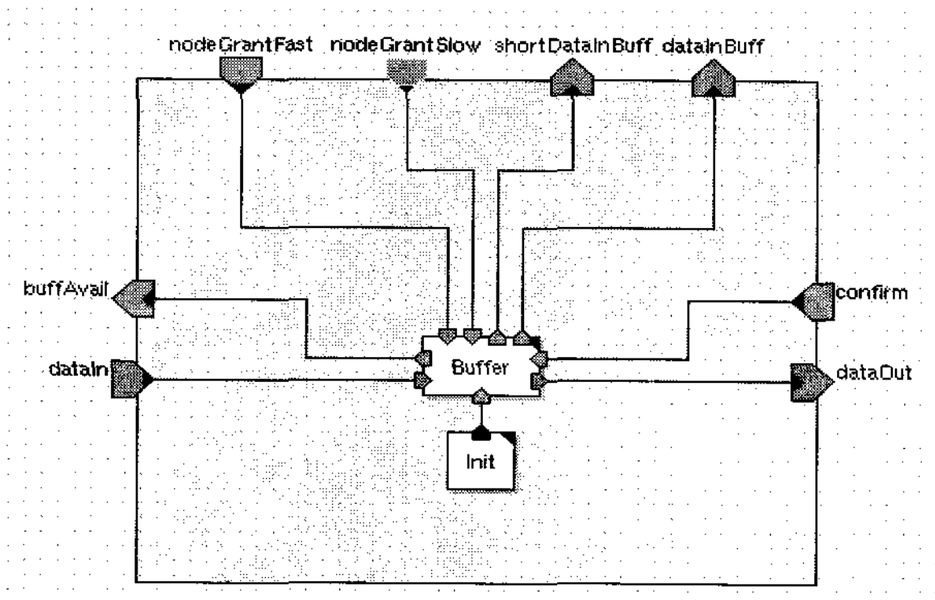


Fig. 2. MLDesigner Implementation of the Buffer Class

C. Scheduler Class

Scheduler handles the control part of the Router Class. It schedules the incoming requests for data transmission to the next node, by checking for the availability of the output data path and by arbitrating the requests from various Input Buffers associated with it. The data and control part of a node have been separated (Router class handles the data part and Scheduler class handles the control signals) in order to manage concurrency issues and make the design more scalable and reusable. Fig. 3 shows the MLD implementation of the Scheduler class. The model shown is a simplified model to better identify its connections. A Scheduler is connected to five instances of Input Buffer (one for each direction in 2-D mesh network and fifth buffer for the local Producer class connected through an NI) and, similarly, five instances of Output Buffer on the output side. Scheduler accepts the requests from a Input Buffer via *shortDataInBuff* or *dataInBuff* signals and allocates the data path to one of these requests by asserting *nodeGrantFast* or *nodeGrantSlow* respectively. The data path allocation is based on the availability of an Output Buffer and the route. Priority-based Round-Robin scheduling algorithm is used for handling multiple requests from various Input Buffers. Router informs Scheduler the physical output path for flit transmission via *outputPort*. Availability of the data path is acknowledged by assertion of *confirm*.

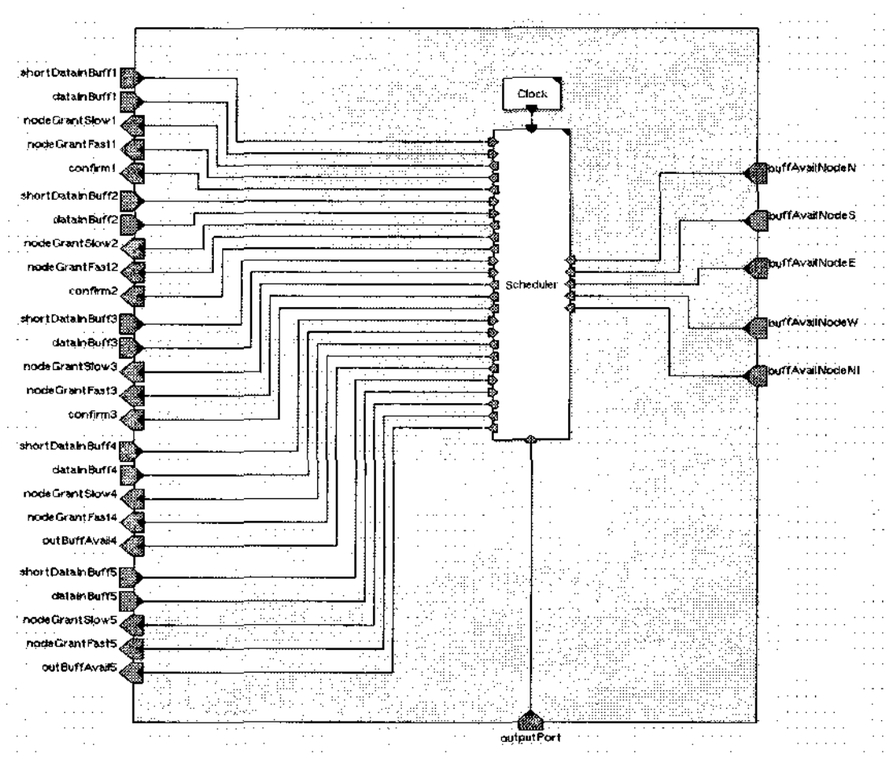


Fig. 3. MLD Implementation of Scheduler Class

For modeling Scheduler functionality in a system level environment, request and output path availability tables were implemented inside Scheduler. These tables are updated by their corresponding signals. This takes care of the credit flow control process in the network. In order to maintain priority based round robin discipline, a separate memory space was allocated for individual signals in the request table. Thus, Scheduler first allocates the requests corresponding to High priority data request signals and then serves the Mid and Low priority data requests. The available credits for the Output Buffers are updated in an output path availability table.

D. Router Class

Router determines the output path and handles the actual data transfer on the implemented backbone. Fig. 4 shows the MLDesigner implementation of Router class and Fig. 5 shows Router's internal connections.

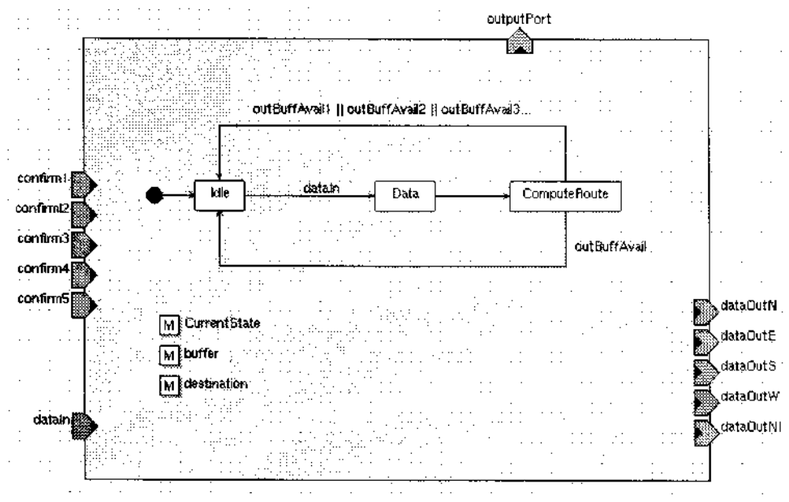


Fig. 4. MLD Implementation of Router Class

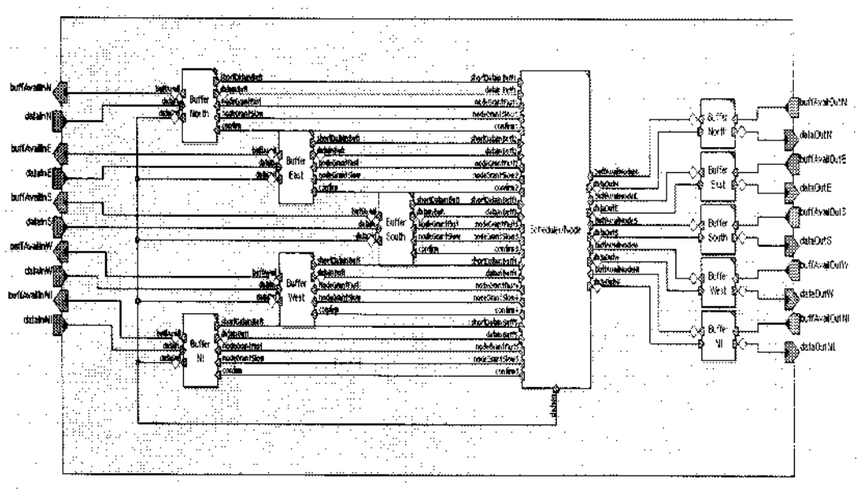


Fig. 5. Internal Details of Router Class

Dimension order routing protocol was implemented in the Router class for determining the output path. A turn-prohibition method was employed for avoiding the deadlock/livelock situation in the network. Upon receipt of data, Router extracts the destination information and determines the physical output port for transmitting the data. This output port address is sent to Scheduler, which determines the availability of this port. Upon its availability, data flits are then forwarded to the corresponding Output Buffer for this port.

E. Output Buffer Class

Output Buffer accepts the incoming flits from Router and forwards these flits to the Input Buffer of the next node. It is implemented in the form of the two concurrently executing state machines. Fig. 6 show the Output Buffer model.

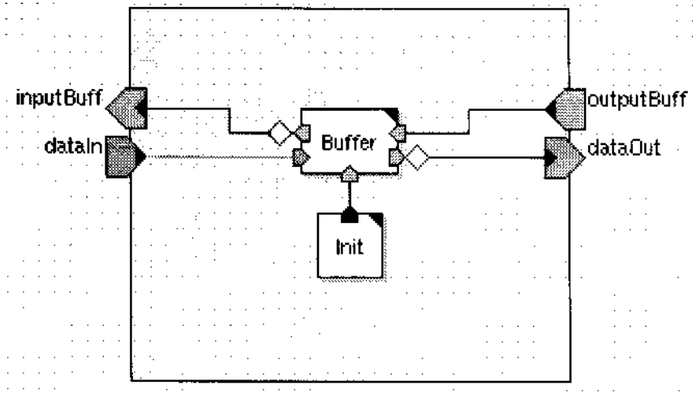


Fig. 6. MLD Implementation of Output Buffer Class

The received flits are stored in the Output Buffer memory. The input state machine accepts and stores the data flits while there is available memory in OutputBuffer. Availability of Buffer space in Output Buffer is signaled by *buffAvail*. The output state machine senses the request for transmitting data from Output Buffer of the next Router via *outputBuffAvail* of that Output Buffer and forwards the data flit, if that signal was asserted.

F. Consumer Class

Consumer comprises of a computing resource and a network interface (NI). Consumer accepts data flits, strips off the header information, and forwards the remainder of the data to its internal computing resources.

G. System Simulation

Consumer comprises of a computing resource and a network interface (NI). Consumer accepts data flits, strips off the header information, and forwards the remainder of the data to its internal computing resources. Fig. 7 shows the 4×4 mesh network we developed for our ONOC example. We studied various simulation platforms such as Ptolemy, Modeling Environment for Software hardware (MESH), OPNET and MLD, to identify the right simulation environment. We determined that MLD was suitable for our work because of its support for multiple MOCs (Models of Computation). We have modeled our classes and their instances using three different domains: finite state machine (FSM), synchronous data flow (SDF) and discrete events (DE). We performed the top level simulation of the overall model in the DE domain; this executes the simulation substantially faster than otherwise.

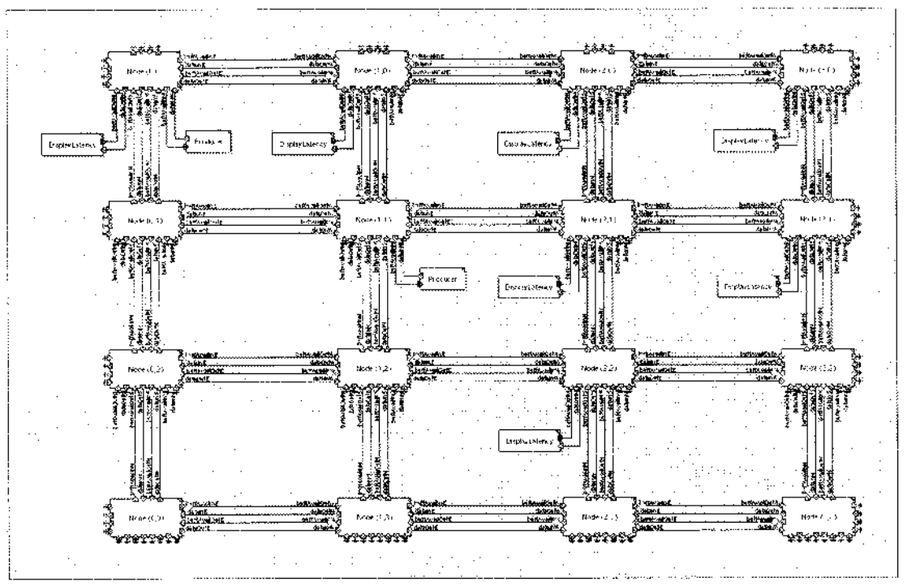


Fig. 7. A 4×4 mesh network for ONOC

IV. SIMULATION RESULTS

In a real-time embedded system, it is must that a system responds to the real-time requirements. Our underlying architecture must make sure to deliver real-time data and required control signals for this data processing in timely manner.

Consequently, we must select scheduling criteria that can perform this task. Fig. 8 shows High Priority data latency result against buffer sizes for different scheduling criteria. It can be seen that FCFS and RR may not always deliver High priority data packets in real-time

bounds. Thus, we should use either PBRR or PB scheduling criteria for our NOC architecture. We used Low priority data latency results for finally choosing the scheduling criteria between PBRR and PB scheduling. Fig. 9 shows Low priority data latency against buffer sizes for PBRR and PB scheduling criteria. PB scheduling delivers the High priority data and Mid priority data in timely manner. However, it has higher latency for low priority data. Thus, we recommend using PBRR as the scheduling criteria for NOC architecture.

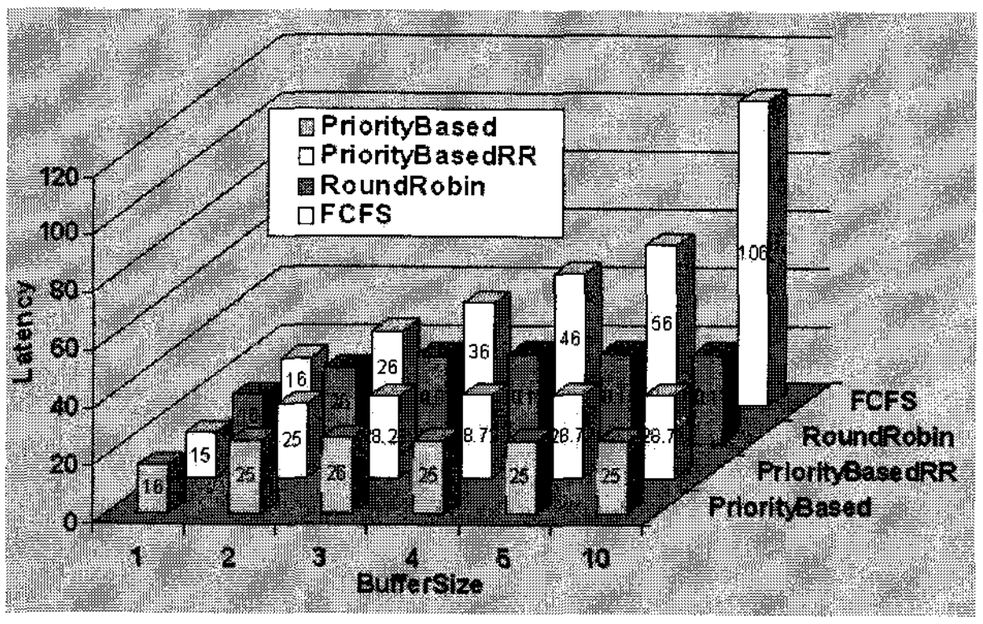


Fig. 8. High Priority Data Latency Vs Buffer Size For Different Scheduling Criteria

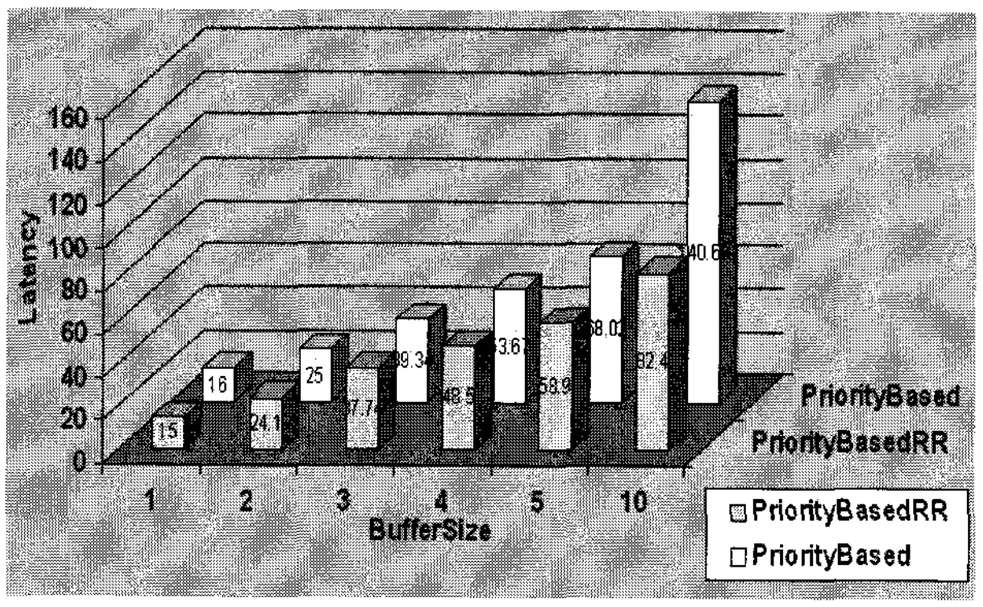


Fig. 9. Low priority Data Latency for PB and PBRR Scheduling

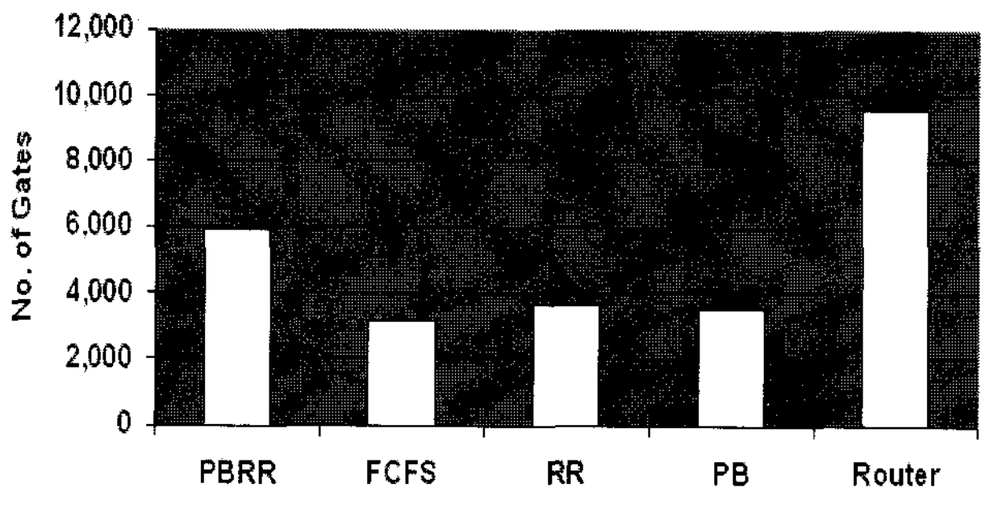


Fig. 10. Number of Gates for ONoC Classes

We implemented all the NoC classes with Field Programmable Gate Arrays (FPGA) to extract the total NoC area and area information of each component. Fig. 10 shows area result for different scheduling criteria (RR, PB, FCFS, and PBRR), and Router. Router handles the actual data transfer, thus has dedicated lines for sending and receiving data bits. Thus, Router takes more number of gates than Scheduler.

V. CONCLUSION

We have developed a methodology to realistically model and optimally design the communication backbone of an NOC. The model is built with reusable and customizable building blocks which are abstract enough to facilitate quick analysis. The modeling environment chosen supports multiple models of computation, which helps with both fast model building and simulation. As an example of model's use, we chose three different traffic patterns and three priority levels to show that the communication backbone can be designed to optimally meet different latency requirements even when the traffic patterns are markedly different.

ACKNOWEDGEMENT

We would like to acknowledge iDEN, Motorola for funding this research work.

REFERENCES

[1] G. Desoli, E. Filippi, "An Outlook on the Evolution of Mobile Terminals: From Monolithic to modular multi-radio, multi-application platforms", IEEE magazine on Circuits and Systems, vol. 6, No. 2, pp. 17-29, 2006.

[2] L. Benini and G. De Micheli. Networks on chip: a new SOC paradigm, IEEE Computer, vol. 35 No. 1, pp.70-78, 2002.

[3] J. Ahmed Meine, Wolf Wayne, Multiprocessor System-On-Chips. Morgan Kaufamann Publisher, 2005.

[4] A. Agarwal, R. Shankar, "A Layered Architecture for NOC Design methodology", IASTED Conference on parallel and Distributed Computing and Systems, pp. 659-666, 2005.

[5] A. Hemani, A. Jantsch, S. Kumar A. Postula, J. Öberg, M. Millberg, D. Lindqvist, Network on Chip: an architecture for billion transistor era, Proc. of IEEE NorChip Conference, pp. 8-12, November 2000.

[6] D. Bertozzi and L. Benini, "Xpipes: A network-on-chip architecture for gigascale system-on-chip," IEEE Circuits and Systems., vol. 4, no.1 pp. 18-31, 2004.

[7] A. Jantsch and H. Tenhunen. Networks on Chip (Kluwer Academic Publisher, 2003.

[8] D. Kim, M. Kim, G.E. Sobelman, "CDMA-based network-on-chip architecture", IEEE Asia-Pasific Conference on Circuits and Systems, vol. 1, pp. 137-140, December 2004.

[9] A. Adriahantenaina, H. Charlery, A. Greiner, L. Mortiez, C.A. Zeferino, C.A., "SPIN: a scalable, packet switched, on-chip micro-network, IEEE Conference and exhibition on, Design, Automation and Test in Europe, pp. 70-72, 2003.

[10] P. Pratim Pande, C. Grecu, M. Jones, A. Ivanov, R. Saleh, "Performance evaluation and design trade-offs for network-on-chip interconnect architectures", IEEE transactions on Computers, vol. 54, Issue 8, pp. 1025-1040, Aug. 2005.

[11] P. Bhojwani, R. Mahapatra, Jung Kim Eun, T. Chen, "A heuristic for peak power constrained design of network-on-chip (NoC) based multimode systems", IEEE International Conference on VLDI Design, pp.124-129, 2005.

[12] D. Rostilav, V. Vishnyakov, E. Friedman, R. Ginosar, An asynchronous router for multiple service levels networks on chip, 11th IEEE international symposium on asynchronous circuits and systems, pp.44-53, March 2005.

[13] A. Siebenborn, O. Bringmann, W. Rosenstiel, "Communication analysis for network-on-chip design", IEEE International conference on Parallel Computing in Electrical Engineering, oo. 315-320, Sepetmber 2004.

[14] J. Hu, R. Marculescu, "Energy-aware Communication and task scheduling for network-on-chip architectures under real-time constraints", IEEE Conference on Design, Automation and Test, vol. 1, pp. 234-239, Febrary 2004.

[15] J. Madsen, S. Mahadevan, K. Virk, M. Gonzalez, "Network-on-chip modeling for system-level multiprocessor simulation", IEEE 14th Conference on Real Time System, pp. 265-274, 2003.

[16] P.T. Wolkotte, G.J.M. Smit, G.K. Rauwerda, L.T. Smit, "An Energy-Efficient Reconfigurable Circuit-Switched Network-on-Chip", 19th IEEE International Conference on Parallel and distributed Processing, pp. 155-163, 2005.

[17] W. J. Dally, B. Towles, "Route Packets, Not Wires: On-Chip Interconnection Networks", IEEE Conference on Design and Automation, pp. 684-689, June 2001.

[18] C. Albenes Zeferino Frederico G. M. E. Santo, Altarniro Amadeu Susin, "ParlS: A Parameterizable Interconnect Switch for Networks-on-Chips", ACM Conference, pp. 204-209, 2004.

[19] T. Bjerregaard, J. Sparso, "Virtual channel designs for guaranteeing bandwidth in asynchronous network-on-chip", IEEE Proceedings on Norchip Conference, pp. 269 – 272, November 2004.

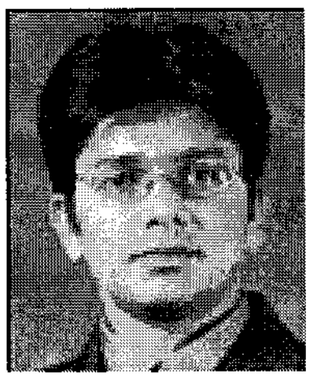
[20] A. Barger, D. Goren, A. Kolodny, "Design and modelling of network on chip interconnects using transmission lines", 11th IEEE International Conference on Electronics, Circuits and Systems, pp. 403-406, December 2004.

[21] A. Morgenshtein, I. Cidon, A. Kolodny, R. Ginosar, "Comparative analysis of serial vs parallel links in NoC", IEEE International Proceedings on System-on-Chip, pp. 185-188, November 2004.

[22] E. Bolotin, I. Cidon, R. Ginosar, and A. Kolodny, "QNoC: QoS architecture and design process for network on chip", Journal of Systems Architecture, December 2003.

[23] D. Bertozzi, L. Benini, G. De Micheli, "Error control schemes for on-chip communication links: the energy-reliability tradeoff", IEEE Transaction on Computer-Aided Design of Integrated Circuits and Systems, vol. 24, Issue 6, pp. 818-831, June 2005.

[24] S. J. Lee, K. Lee, S.J. Song, H.J. Yoo, "Packet-switched on-chip interconnection network for system-on-chip applications", IEEE Transaction on Circuits and Systems II, vol. 52, Issue 6, pp. 308-312, June 2005.

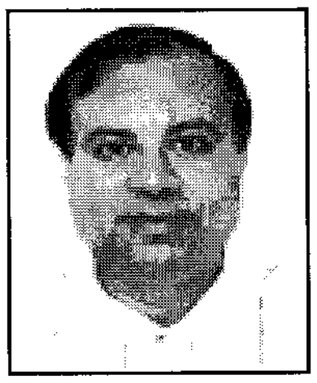


Ankur Agarwal

He is an assistant professor at the Computer Science and Engineering Department, Florida Atlantic University. He pursued his MS and Ph.D. in computer engineering from Florida Atlantic University. He also holds two post graduate diplomas in VLSI design and real-time embedded system design. He has earned his bachelor of engineering from Pune University, India in year 2000. His research areas are concurrency modeling, system level design, network-on-chip, real-time-operating system and VLSI design.

Rabi Shankan

He is a professor at the Computer Science and Engineering Department, Florida Atlantic University. He received his undergraduate education at the Karnataka university, Dharwar, India. He earned his M.S. and Ph.D. in Computer Science from the University of Wisconsin, Madison. His research areas are engineering productivity, concurrency and software decomposition.



A. S. Pandya

He is a professor at the Computer Science and Engineering Department, Florida Atlantic University. He received his undergraduate education at the Indian Institute of Technology, Bombay. He earned his M.S. and Ph.D. in Computer Science from the Syracuse University, New York. He has worked as a visiting Professor in various countries including Japan, Korea, India, etc. His research areas are VLSI implementable algorithms, Applications of AI and Image analysis in Medicine, Financial Forecasting using Neural Networks.

YoungUhg Lho
(Corresponding author)

He received the B.S., M.S. and Ph.D degrees in Dept. of Computer Science from Busan National University, Busan, Korea, in 1985, 1989, and 1998, respectively. From 1989~1996, he was with the Electronics and Telecommunications Research Institute (ETRI), Daejeon, Korea. Since 1996, he has been with the Dept. of Computer Education, Silla University, where he is now Professor. His research interests include NOC, real-time system, ubiquitous computing, embedded system, multimedia system, parallel and distributed system, intelligent system and computer education.