International Conference on Information and Communication Technology
ICICT 2007, 7-9 March 2007, Dhaka, Bangladesh

GPNOCSIM – A GENERAL PURPOSE SIMULATOR FOR NETWORK-ON-CHIP

Hemayet Hossain[1], Mostak Ahmed[2], Abdullah Al-Nayeem[3], Tanzima Zerin Islam[4], and Md. Mostofa Akbar[5]

[1]Department of Computer Science, University of Rochester, Rochester, NY-14627, USA.
[2, 5]Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology, Dhaka, Bangladesh
[3]Department of Computer Science and Engineering, United International University, Dhaka, Bangladesh
[4]Research and Development, Commlink Info Tech Ltd., Dhaka, Bangladesh
E-mail: hossain@cs.rochester.edu[1], mostak@grameenphone.com[2], alnayeem@gmail.com[3], tz.islam@gmail.com[4], mostofa@cse.buet.ac.bd[5]

ABSTRACT

Network-on-Chip (NoC) has gained considerable attention over the last few years as a paradigm for implementing communication among the system components embedded in a single chip. As such, there has been an increased need for defining and developing simulation software for carrying out simulation of NoC architectures. In this paper, we present gpNoCsim, a JAVA based general-purpose network simulator for NoC, which is built upon the object-oriented modular design of the NoC architecture components. Here we demonstrate the use of our proposed simulator by simulating several existing well-known architectures. We have also provided the guidelines on how to simulate future architectures. Finally, we have validated the outputs of gpNoCsim with the studies of existing NoC architectures.

Keywords: NoC, SoC, Wormhole Switching

1. INTRODUCTION

As technology scales and chip integrity grows, on-chip communication is playing an increasingly dominant role in System-on-Chip (SoC) design. To meet the performance and design productivity requirements, Network-on-Chip (NoC) [1,2] has been proposed as a solution to provide better modularity, scalability, reliability and higher bandwidth compared to bus-based communication infrastructures. Design decisions about NoC architectures are typically taken on the basis of simulation before resorting to implementation since it is cheaper and more flexible.

Only a few simulators have been developed for simulating NoC architectures. While researchers have developed their own simulators to simulate specific NoC architectures, these simulators are not applicable for other architectures. For example, the simulation performed in [3] only evaluates Mesh and Butterfly Fat Tree architectures. The only general purpose NoC simulator that we have found is the one presented in [4], which is developed in SystemC. However, gpNoCsim is a JAVA based general purpose simulator which supports evaluation of different existing architectures for on-chip networks: Mesh, Torus, Butterfly Fat Tree, etc. Moreover its modular design helps it to incorporate new architectures. Here we are providing full documentation on how and where to make changes in order to simulate new topology, new design of switch, new routing algorithm etc. The source code and the manual can be downloaded from www.buet.ac.bd/cse/research/group/noc/.

The rest of the paper is structured as follows. Modeling of NoC in gpNoCsim is discussed in Section 2. We have described the flow diagram of the simulator in Section 3 and the implementation details in Section 4. Section 5 delineates the methodology of simulating a new architecture using gpNoCsim. Then we describe the performance evaluation parameters and provide validation of our simulator in Section 6. Finally, we conclude the paper in Section 7 with our contribution and future research works.

2. MODELING OF NOC IN GPNOCSIM

2.1 Interconnect Architectures

In the forthcoming billion-transistor era, Network-on-chip--as the most promising communication paradigm of System-on-Chip--implies the communication between numerous heterogeneous

semiconductor intellectual property (IP) blocks. These IP blocks communicate with each other through the switches. Although the on-chip interconnection topologies have root in the interconnection architectures of high-performance parallel computing model, few of the interconnection architectures—for example, Mesh, Butterfly Fat Tree, Torus, etc—are applicable for the SoC design. gpNoCsim provides the designers with the flexibility to simulate different network configurations of NoC.

2.2 Switch Structure

In gpNoCsim, each switch has a number of ports to connect with its adjacent switches and IP blocks. Each port includes two sets of buffers –- input and output, a router and a controller. The router decodes header and determines the output port. The controller is partitioned into two disjoint controllers, one for input channel, and the other for output channel. The arbiter is implemented in round-robin fashion.

2.3 Switching Technique

gpNoCsim uses the wormhole switching technique [5]. It also uses virtual channels in the input and output port. The first flit, i.e., header flit, of a packet contains routing information. Header flit decoding enables the switches to establish a path and subsequent flits simply follow this path in a pipelined fashion. As a result, each incoming data flit of a message packet is simply forwarded along the same output channel, as the preceding data flit and no packet reordering is required at destination. If a certain header flit faces a busy channel, subsequent flits also have to wait at their current locations.

2.4 Traffic Configuration

In gpNoCSim, the users can define different types of traffic distribution by selecting destinations either with uniform or with biased probability. Inter-arrival time of message generation event in a node is exponentially distributed. Similarly, the number of flits in a message can be either fixed or exponentially distributed. Flit-size can be configured in the simulator and the encoding of the header flit can be changed according to the topology in question.

3. FLOW DIAGRAM OF THE SIMULATOR

Steps of the flow diagram in Fig. 1 are as follows:
1. Input parameters are read from the input file *nocSimParameter.txt*.
2-3. If there exists any more network configuration in the input file, then program flow transfers to step 4. Otherwise the simulation is terminated.
4-5. The network is created and the performance parameters are initialized.
6. If the simulation runs are completed, program flow transfers to step 18.
7-9. For each simulation run, the network and the statistical counters are initialized and the initial events of the network are set.

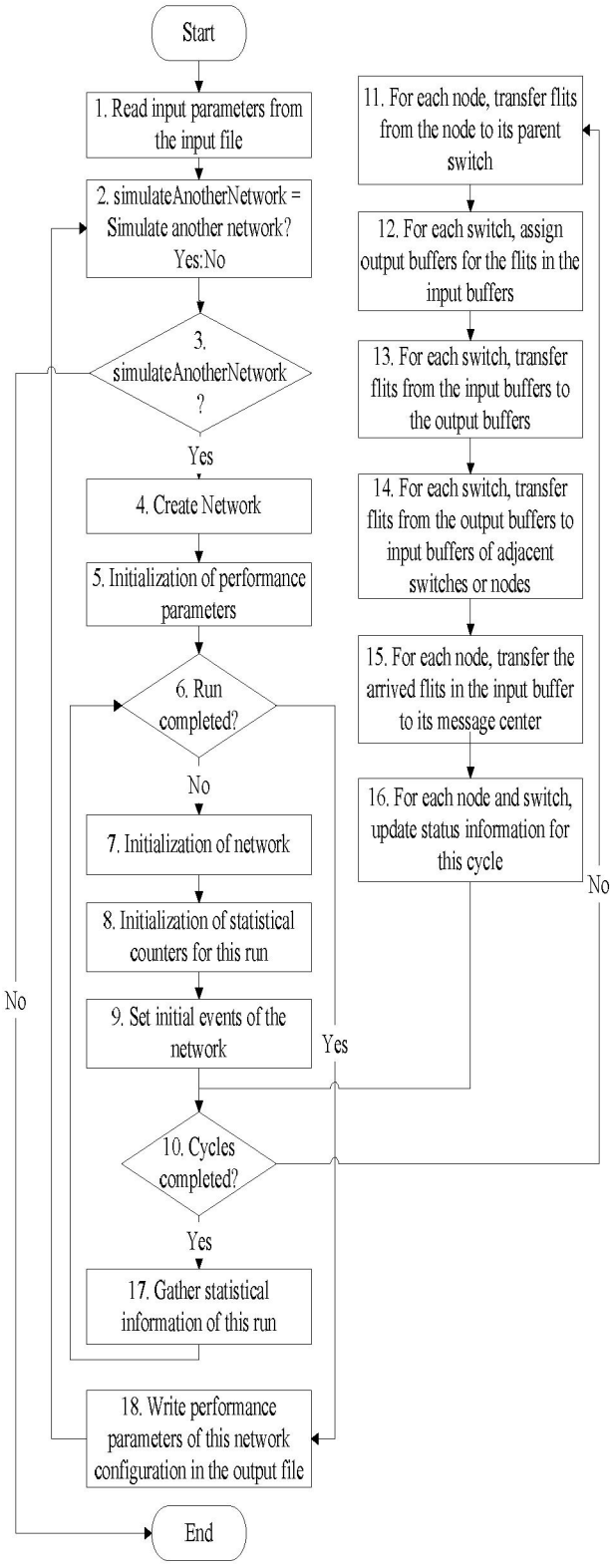

Fig. 1 Flow diagram of gpNoCsim

10. If all the cycles of this run are completed, then simulation flow transfers to step 17. Otherwise it transfers to step 11.
11. For every node, outgoing traffic is moved from its message center to its parent switch, if possible.
12. For every switch, path request of the incoming traffic is updated.
13. For every switch, incoming traffic is moved from the input buffer to the output buffer, if possible.
14. For every switch, outgoing traffic is moved from the output buffer of the switch to the input buffer of the adjacent switches or nodes.
15. For every node, incoming traffic is moved from the input buffer of the node to the message center of that node.
16. For every node and switch, status and flags are updated at the end of the cycle.
17. The statistical parameters are updated for this simulation run. Then the flow transfers to step 6.
18. After all the runs are completed, performance parameters of this network configuration are written to the output file and the flow jumps to step 2.

3. IMPLEMENTATION DETAILS:

To design or expand a network, at first a user has to form the whole topology and make decisions on each device's characteristics, which can be done using a configuration file in ASCII - *nocSimParameter.txt*. Then the performance of the designed network can be analyzed and predicted.
A brief description of different components (classes) of the simulator is given below.

- *Controller* - The Controller class dictates the total flow of execution of the simulator.
- *Network Manager* - NetworkManager class creates and initializes new networks and corresponding objects of HelpingUtility and StatisticalData class (described later).
- *Network* - This class builds the networks and connects all nodes and switches.
- *Switch* - Switches are the objects of the network where switching of flits from various input ports to output ports are performed (routing).
- *Router* - This interface provides the declaration of the method to find out the output link of a switch to reach the destination.
- *Node* – The Node class checks for free virtual channels, assigns the virtual channels to a message and forwards incoming messages to message center of the node.
- *NodeTraffic* - The NodeTraffic abstract class defines the necessary methods for setting up traffic configuration for the generated messages in a node.
- *Link Controllers* - Each input link controller has a separate FIFO that buffers the input packets before delivering them to the output ports. When a new header flit is received the input link controller determines the output port. On the other hand, the output link controller adds, removes data to and from the output buffer.
- *Virtual Channel Buffers* - InputVCBuffer and OutputVCBuffer classes define data structures of input and output buffers and functionalities for storing and managing flits of a particular port.
- *Flit* - It defines different flit-related member variables, such as flit type, source IP address, destination IP address, flit data, generation time and last service time.
- *Statistical Counters* – The StatisticalData class computes various performance parameters.
- *Helping Utility* - This class defines methods for initializing random seed, generating random numbers and reading data from the input file.

4. SIMULATOR SETUP

To simulate a new architecture, following changes need to be made:

- To change the architecture of the switch, one has to build a new class with the name - NewArchSwitch [for example – MeshSwitch] that implements the Switch interface and overrides all the methods necessary to build the switch.
- To change the routing algorithm, one has to create a new class, e.g. NewArchRouter, that implements the getDestination method of the Router interface.
- NodeTraffic interface defines all the necessary methods related to traffic generation. In our simulator, ConcreteNodeTraffic class is the default implementation of this interface. To redefine the traffic pattern one has to implement own version of NodeTraffic interface and supply it as a parameter of Node class.
- To change message encoding, one has to modify the Flit encoding in createHeaderFlit and createDataFlit methods of ConcreteNodeTraffic class and adjust the encoding to support different address formats of various architectures.
- To change the distribution of message generation, one has to modify the setNextMsgGenTime method in the class ConcreteNodeTraffic.
- To change the distribution of packet length, one has to modify the getMessageSize method in ConcreteNodeTraffic class.

- To change traffic pattern, one has to modify the getDestination method in ConcreteNodeTraffic class.
- To add new performance parameters, the StatisticalData class needs to be modified.

5. PERFORMANCE EVALUATION:

gpNoCsim is able to model and evaluate both the existing and future chip architectures. It gives output in terms of a number of performance metrics: throughput, latency (average packet delay), link utilization, buffer utilization, average hop count. Here we discuss on only latency and throughput of Mesh and Butterfly Fat Tree topologies.

5.1 Latency

Latency is defined as the number of clock cycles required for complete transfer of a packet from the source node to the destination node on average.
From the following Fig. 3, we find that all topologies have the increasing latencies with the decrease of average inter message generation time, because a decrease in inter message generation time steps-up the network load. In the case of increased network load, more flits are coming to the switches, thus contention for outgoing link increases. As a result buffer delay increases which consequently increases the latency of the network. Such behavior has also been confirmed in [6].

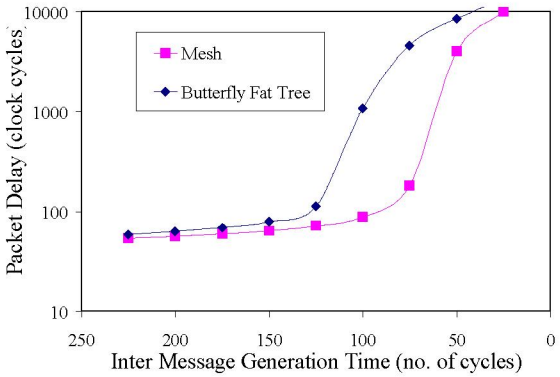

Fig. 2 Packet Delay Vs Inter Message Generation Time (Network Load).

5.2 Throughput

We define throughput as follows:

$$\frac{\text{Total number of flits received at their destinations}}{\text{(Number of IP blocks) x (Total Time in Cycles)}}$$

From Fig. 3, throughput initially increases with the increment of virtual channels. But after increment of virtual channels to a certain level the increase rate of throughput is very low for these two topologies for certain network load. The results concur with that in [6].

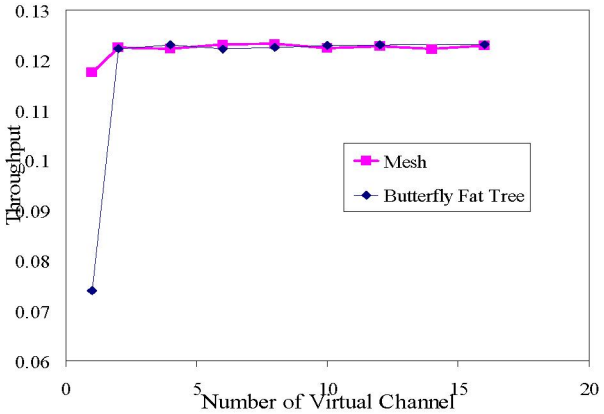

Fig. 3 Throughput Vs Number of Virtual Channels

6. CONCLUSION AND FUTURE WORKS:

In this paper, we have presented a general-purpose simulation software for the NoC architectures and described its modular software architecture, along with the necessary guidelines for the NoC designers to use the simulator for their architectures.
Followings are our suggested research plans for the future versions of gpNoCsim.

- We have implemented only uniform traffic distribution. Application specific traffic patterns can be implemented in the future versions of the simulator by making necessary changes in the implementation of NodeTraffic interface.
- Power consumption equations can be considered for further study.

REFERENCES

[1] S. Kumar, A. Jantsch, J.-P. Soininen, M. Forsell, M. Millberg, et. al, "A network on chip architecture and design methodology," in Proc. of IEEE Computer Society Annual Symposium on VLSI, USA, pp. 117-124, April 2002.

[2] L. Benini and G. De Micheli, "Networks on chips: A new SoC paradigm," IEEE Computer, vol. 35, pp. 70-78, Jan 2002.

[3] P. Pande, C. Grecu, M. Jones, A. Ivanov and R. Saleh, "Performance Evaluation and Design Trade-offs for MP-SoC Interconnect Architectures," IEEE Transactions on Computers, vol. 54, no. 8, pp. 1025-1040, August 2005.

[4] Z. Lu, R. Thid, M. Millberg, E. Nilsson and A. Jantsch, "NNSE: Nostrum Network-on-Chip Simulation Environment," in Swedish System-on-Chip Conference, Stockholm, Sweden, April 005.

[5] J. Duato, S. Yalamanchili, and L. Ni, "Interconnection Networks – An Engineering Approach," Morgan Kaufman, 2002.

[6] P. P. Pande, C. Grecu, A. Ivanov, and R. Saleh, "High-Throughput Switch-Based Interconnect for Future SoCs," in Proceedings of 3rd IEEE International Workshop on System-on-Chip for Real-Time Applications, Calgary, Canada, pp. 304-310, June 30-July 2, 2003.