NoCBench: A Benchmarking Platform for Network on Chip

Suman K. Mandal, Nikhil Gupta, Ayan Mandal, Javier Malave, Jason D. Lee and Rabi N. Mahapatra
Texas A&M University, College Station, TX 77843
{skmandal, ngupta, ayan, jmalave, jdlee, rabi}@cs.tamu.edu

ABSTRACT
This paper describes NoCBench, a benchmarking platform for evaluating the performance of Network-on-chip enabled Systems-on-chip. NoCBench includes an initial set of standardized processing cores, NoC components, and application benchmarks for system-level design exploration and analysis. It uses the NoCSim network on-chip simulator as the core simulation engine to execute these models and applications. We also present a simple case study to demonstrate the capability of the simulation environment.

Categories and Subject Descriptors
I.6.7 [**Simulation Support Systems**]: Simulation Environment, B.4.4 [**Performance Analysis and Design Aids**]: Simulation.

General Terms
Measurement, Performance, Verification.

Keywords
Network on Chip, Benchmarks, Simulator, NoCSim

1. INTRODUCTION
Network-on-Chip (NoC) based designs have garnered significant attention from both researchers and industry over the past several years. The analysis of these designs has focused on broad topics such as NoC component micro-architecture [7][[12][19], fault-tolerant communication, and system memory architecture. The effective analysis of these system parameters requires a simple, yet powerful network-on-chip simulator which supports a large set of system components and application benchmarks.

In this paper, we present NoCBench, a system analysis platform which includes standardized NoC components and application benchmarks for rapid prototyping of NoC-enabled systems-on-chip (SoC). We use NoCSim as the core simulation engine to execute NoCBench models and applications. Using this platform, NoC and system researchers can easily plug-and-play existing components, or create new components, into a full SoC to analysis the effect of system design choices on application performance.

A benchmark designed for NoC-based systems needs to quantify the performances of the following: 1) functional and storage blocks with corresponding network interface (cores), 2) the interconnect infrastructure (routers and links), and 3) the fully integrated system. NoC provides the underlying communication infrastructure that allows effective integration of functional, I/O, and storage blocks. Latency, throughput, reliability, energy dissipation, and silicon area requirements characterize such communication-centric interconnect fabrics.

Currently, network simulators like Noxim[1], developed in SystemC [15], intend to explore the design space spanned by the different parameters of a NoC for the analysis and evaluation of a large set of quality indices including delay, throughput, energy consumption using synthetic traffic loads. However, current NoC simulators like Noxim do not support benchmark analysis, which is necessary to validate the superiority of one's design compared to another for specific applications. Other NoC simulators like Garnet [14] are focused heavily on application execution, and therefore attempt to abstract away as many network details as possible for efficient execution. In most ease of reconfigurability and flexibility are sacrificed for simulator performance.

The main contribution of our work is the design of an integrated simulation environment with an initial set of standardized NoC components and cores. It provides a single, cohesive simulation environment capable of integrating different embedded cores and network components for simulating them as a system is developed. Additionally, many design choices independent of system composition, such as network topology, can be easily explored.

The rest of the paper is organized as follows: Section 2 provides a brief overview of related work. In Section 3, we present the NoC architecture. Section 4 presents the proposed NoCBench platform followed by a case study in Section 5. Finally, Section 6 concludes the paper.

2. RELATED WORK
Traditionally, researchers have used custom simulators that can simulate traffic for use in Network on Chip research. Grecu et al. [2] brought the need for Network-on-Chip benchmarking into perspective and suggested compelling features of a NoC benchmarking environment. Although the

authors proposed a large set of parameterized reference inputs for the NoC benchmarks: i.e. a) NoC functional cores composition (number of Processing Elements, number and size of memories, number of I/Os) b) Interconnect architectures c) Data communication requirements. There is no evidence of building such a platform for benchmarking. NoCBench is a step forward towards achieving an integrated platform for NoC design and verification.

Nostrum is flexible NoC Simulator defining a 2D mesh topology, [4]. It focuses on communication primitives implementing protocol stack for link layer, network layer and session layer along with a buffer-less, loss-less switch.

Garnet [14] is a network-on-chip performance simulator which is compatible with the GEMS [22] multiprocessor framework with Simics [16]. It can be interfaced with the Orion [17] network-on-chip power modeler when necessary. Garnet provides two modes of operation: a detailed "fixed pipeline" mode, and a high-level "flexible-pipeline" mode. The fixed-pipeline mode models the micro-architectural details of the on-chip router, while the flexible-pipeline mode allows the user to parameterize the number of router pipeline stages, and simply delays network traffic by that many cycles per router. Although Garnet is an excellent tool due to its accuracy and inter-operability with other system-level simulators, it does not provide full end-to-end models of the NoC (e.g. CNI, other infrastructure IP) within a single framework. Additionally, only 2D-mesh topologies can be modeled with the Garnet simulator.

The main shortcoming of all the simulators discussed above is, they either simulate the network or the cores. We still need a comprehensive system simulation and benchmarking platform. NoCBench is the first step towards bridging the gap by providing a full system simulation environment equipped with cores and network components.

3. SYSTEM ARCHITECTURE

3.1 System on Chip

Modern embedded systems are often composed of many IP blocks including processors, memory blocks, DSPs and controllers. Traditionally these blocks have been connected using direct wiring or on chip buses. However, with growing integration and shrinking device size and increased chip size, wiring delays are becoming significant. Also, as we pack more cores and connect with buses, contention becomes a major bottleneck. To address these network on chip has emerged as a solution for communication on the chip [12][18]. Details of the Network on Chip architecture is discussed here.

3.2 NoC Architecture

Network on Chip architecture is inspired by the asynchronous communication paradigm of packet switched computer networks. With increased on chip wiring delay synchronized communication between the far ends of a large system on chip is impractical. Network on Chip tries to address this issue by introducing a globally asynchronous locally synchronous (GALS) paradigm. In NoC, communication takes place between cores using information packets. Packets are generated at the source and carried to the destination by intermediate routers. At the destination, the packet is decomposed into data and processed by the receiver. Clearly NoC architecture will have the following components. A Core to Network interface, Routers and Links. The following sections discuss the functional and architectural details of these components.

3.2.1 Processing Cores

Processing cores are the working components of the system on chip. Depending on the application performance requirements, the core can vary widely from a simple CPU and a collection of controllers and DSPs to very powerful processor cores with cache memory blocks. Cores can be designed independently of the communication infrastructure to minimize development cost. However, communication protocols and structures are specific to each processing core. This requires an intermediate entity which interfaces the processing core to the network-on-chip. This entity performs communication translation and other, secondary functions. This is called CNI or core network interface.

3.2.2 Core Network Interface (CNI)

The Core Network Interface, or CNI, is the functional equivalent of a network card in standard multi-computer systems [8]. The CNI connects the IP cores to the network. Its primary job is to translate the raw information generated by the core into packets to be transmitted through the network and the reverse process for packets coming in from the network. In addition, a CNI can also perform system management tasks such as power management, fault detection, core test support, and system reconfiguration. The block diagram of an example CNI is given in Figure 1

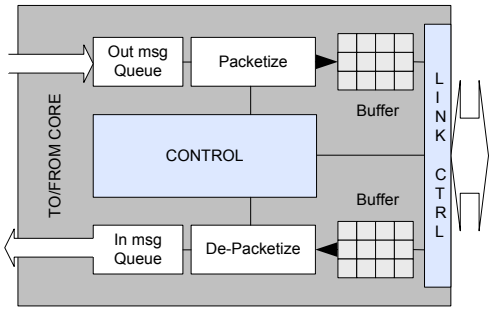

Figure 1: A Simple Core Network Interface

The CNI architecture assumed in this research provides simple traffic translation to enable inter-core communication across the network-on-chip. For more

complex systems, other functionality may be added to this base model as necessary. Other researchers present a thorough examination of additional functionality that may be added [8][9].

3.2.3 Router

The on-chip router is the most salient component in a Network on Chip. Similar to standard computer networks, the router's job is to efficiently route packets through the network. A router mainly consists of the input channels to receive the packets, output channels for sending, a crossbar for switching and a routing logic for doing the routing. Generally the input and output channels are buffered. A simple router block diagram is illustrated in Figure 2.

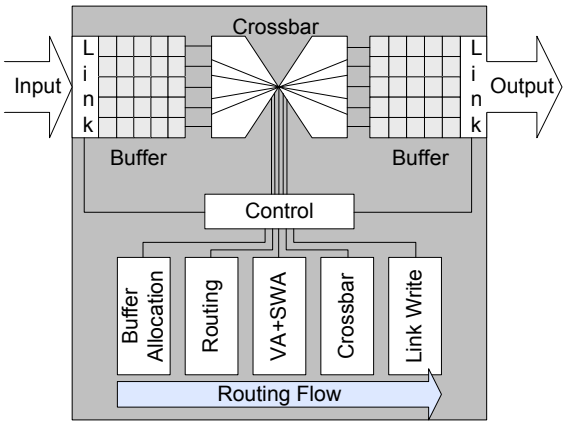

Figure 2: A Simple 5 Cycle Router Pipeline

The router architecture used in this research follows the standard five stage pipelined, wormhole router proposed by various researchers [7][11][19]. In wormhole routing each packet is further split into units called flits. Each port has a set of input buffers, and each incoming flit is first stored in the appropriate input buffer. If the incoming flit is a head flit, then a routing decision must be made so that the entire packet is forwarded to the appropriate output port. Flits are then assigned an output virtual channel, and the crossbar switch is allocated for each flit based on utilization and fairness policies. Finally, the flit traverses the crossbar switch and is sent through the appropriate output port.

3.2.4 Links

Links connect the routers and the cores via CNI. Due to the structure of NoC, the links are generally shorter compared to bus or dedicated wiring. Hence, often the links can be of higher bit width and 64, 128 even 256 bit wide links are common. For NoC and system benchmarking purposes, all data flowing through links is assumed to require one cycle of latency.

3.3 NoC Topologies and Routing

A NoC can have different topologies and routing algorithms depending on application needs and available resources. We will discuss the most common and widely used topologies and routing algorithms in the following sections.

3.3.1 Topology

The most commonly used topologies in NoC research are 2D Mesh and Torus topologies. However, other interesting topologies such as butterfly, fat-tree, and Gaussian networks have also been proposed [20]. The mesh and torus topologies are favored due to their regular structure and planar geometry and simplicity. Figure 3 illustrates the three common topologies, namely Torus, Mesh and Irregular.

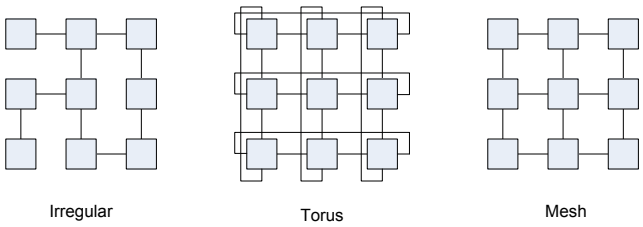

Figure 3: Common Network Topologies

System designers must select the appropriate topology for their expected application requirements, based on topological metrics such as link density and network diameter.

3.3.2 Routing

Routing in NoC depends on the topology. It can be table lookup based where each router has next hop information for every destination in the network or it can be geometry based like dimension ordered or XY routing. The third type of routing is called source routing, where the sender specifies all the intermediate nodes in the packet. Each routing technique differs in performance with respect to adaptability, average hop count, and flit header storage requirements. The performance of the system can vary greatly depending on the routing technique used. In this work, we analyze the table based and geometry based routing techniques.

4. THE NoCBench PLATFORM

The NoCBench platform consists of NoCSim, a flit accurate network on chip simulator plus a set of synthetic and real world benchmark applications. Both the simulator and the benchmark applications can be configured according to system needs. The simulator can support a variety of architectures as discussed in Section 3. Also, the benchmarks can be configured for the given architecture and system specification. In this section we describe the NoCSim simulation environment and the benchmarks in detail.

4.1 NoCSim Simulator

NoCSim is a flit accurate network on chip simulator written in SystemC [21]. It makes use of the SystemC simulation engine and behavioral level network component library for fast simulation. Figure 4 shows the organization of the NoCSim simulator in NoCBench. The Simulator consists of the following main units: the NoC generator, the component library, and the simulation engine.

4.1.1 Network Generator

This module reads a configuration file specified in XML format and generates the NoC using SystemC library modules. The configuration includes the topology, specification of parameters for the CNI, Router and Links. It can also specify fault simulation parameters, and the configuration can be extended according to the need for specification of additional properties. An example configuration can be found in Section 5.2.

4.1.2 Network Component Library

The network component library is the core of the simulation system. In this library we have modeled the components of the NoC using SystemC. The router and the CNI can be configured to implement popular peak power management schemes like PowerHerd [7] and PC [11]. The system level model allows for fairly accurate and much faster simulation compared to detailed RTL simulation. **Table 1** provides the details for the available modules in the component library.

4.1.3 Configurable Core Library

The configurable Core library provides SystemC models of IP cores. We have two types of core models in this library in the current version of the NoCBench Tool.

Synthetic Cores: These cores can generate communication based on statistical traffic distributions or synthetic task graphs with communication. These types of cores do not generate meaningful traffic in the network and do not necessarily perform meaningful communication. However, these cores can be configured to produce network traffic that follows the expected high-level behavior in the network, which allows for generalized performance benchmarking of NoC components. Statistical random number generators are used to mimic traffic distributions. Examples of statistical traffic patterns include uniformly distributed network traffic and self-similar traffic models.

Table 1: Configurable Parameters of The Simulator

Component	Configuration parameters
CNI	Flit injection rate, message queue length.
Router	Number of ports, buffer lengths, number of virtual channels.
Routing	XY routing, table based routing, source routing.
Link	Bit widths.

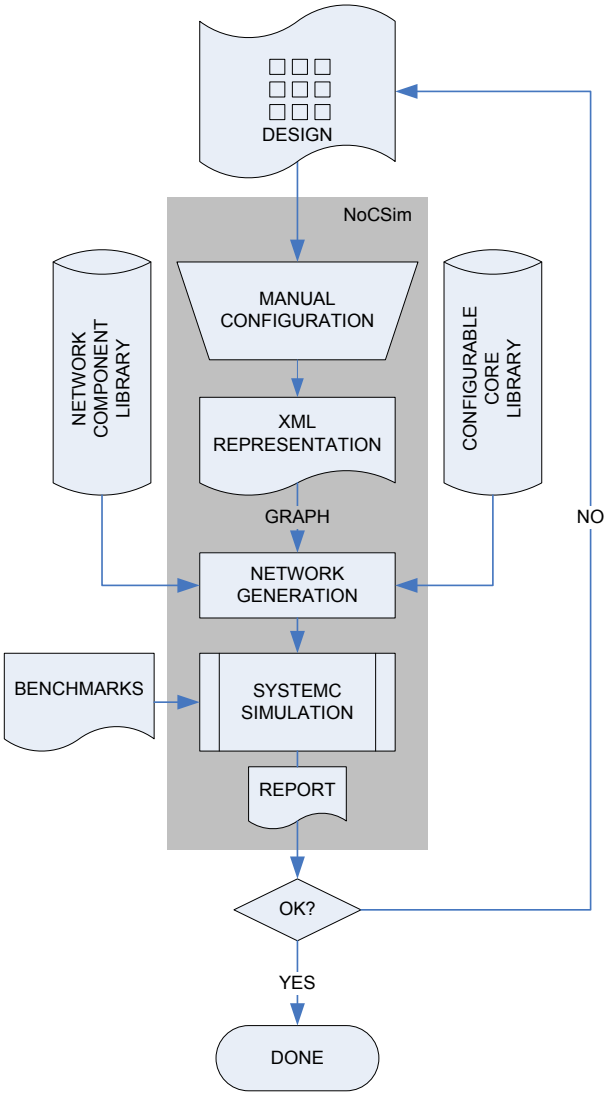

Figure 4: NoCBench Execution Overview

Real Cores: These are SystemC wrapped cores, typically implemented in either C++/C. The most common examples of these core implementations are instruction set simulators (ISS) for different processing cores. They also include cache and memory models.

Currently, we have a simple RISC CPU modeled in SystemC. This model is a simple pipelined in-order processor that partially implements the RISC2000 ISA with a 3-stage pipeline, a MMX-type module, and a floating point unit. We call it SimpleRISC core [Figure 5]. The core has a structural, SystemC top module while the internal functions are implemented in a functional C++ model. This significantly increases the simulation speed. The processor core acts as a master providing an OCP (Open Core

Protocol [13]) compliant port for communication with the CNI.

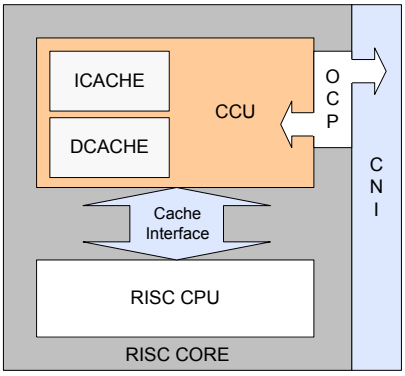

Figure 5: The SimpleRISC Core along with the CNI

This port is connected internally to a soft arbiter that handles incoming/outgoing requests. This soft arbiter serves for future expansions (e.g. Interrupt Control).

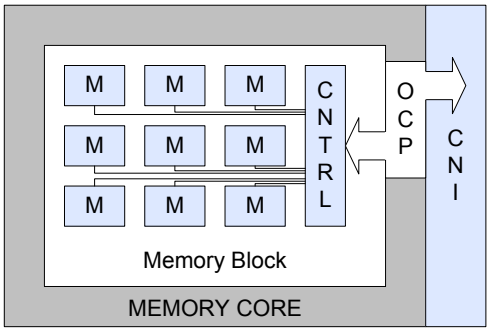

Figure 6: On Chip Memory Core with CNI

The SimpleRISC core also consists of a Cache Control Unit (CCU) shown in Figure 5. CCU implements a L1 instruction and data cache for the processor. The cache is a detailed L1 model developed in C++ which provides accurate measurement of latency and is a cycle-accurate simulation of cache behavior.

In addition to SimpleRISC, NoCBench consists of an on-chip memory model and an L2 cache model. Both the L2 model and the on chip memory model are soft implementations of their respective blocks. The modularized nature of the models allow the user to readily experiment with memory and cache sizes and other characteristics to discover the most efficient implementation for a given system. Figure 6 shows the block diagram of the memory model included in this version of the NoCBench tool.

4.1.4 Simulation Engine

The NoCSim simulation environment is compatible with any SystemC simulation engine. In our evaluation we have used the OSCI SystemC simulation engine. It is capable of very fast and accurate simulation in any platform and this is open source.

4.2 Benchmarks

The design of low-latency, high-bandwidth, low-power and area-efficient NoC is extremely complex due to the conflicting nature of these design objectives. The NoC must be co-designed with other chip components and its design must be evaluated with a total system perspective. Classic benchmarks are application oriented and do not exploit communication intensive architectures. Application sets with a heterogeneous nature, with respect to computation and communication, running on a NoC-based system are better candidates for validation.

Benchmarks are now regularly used by compiler companies to improve not only their own benchmark scores, but real application performance. The use of application driven workload is essential to compare different designs and evaluate the system effects on performance characteristics of the NoC. The variability of injection rates and traffic patterns in real applications provide an excellent opportunity for implementing adaptive hardware in NoC.

Benchmarks are designed to mimic a particular type of workload on a component or system. "Synthetic" benchmarks do this by specially-created programs that impose the workload on the component. "Application" benchmarks, instead, run actual real-world programs on the system. While application benchmarks usually give a much better measure of real-world performance on a given system, synthetic benchmarks still have their use for testing individual components, like a hard disk or networking device.

Synthetic benchmarks cover the following aspects in a NoC: 1) Packets and Transactions (Delay, bandwidth, jitter, power consumption of individual packets and routing, switching, buffering, flow control of the network) 2) Congestion (Arbitration, buffering, flow control) 3) Temporal and Spatial Distribution (burst traffic scenarios, hot spot pattern detection) 4) Quality of Service (guaranteed throughput and latency). 5) Network Size (scalability of communication network)

Application benchmark programs are modeled as communication task graphs exhibiting control and data dependencies. The processing cores mimic the real application as the generated data traffic matches the actual implementation of the application. The following metrics are available for analyzing the performance: 1) Application Run Time (Completion time of the application). 2) Application Throughput (The minimum period at which the input processing core injects data).

In case study, presented in Section 5, we use matrix multiplication to benchmark the cores and the NoC. This

Table 2: System Configuration Details for the 2 Processor Design

	Instruction Cache			Data Cache		
	Cache size	**Line size**	**Associativity**	**Cache size**	**Line size**	**Associativity**
CPU 1	8K	8 bytes	8 way	16k	8 bytes	8 way
CPU 2	4k	8 bytes	8 way	16k	8 bytes	8 way

matrix multiplication application creates sufficient network traffic for evaluating the network performance. Further, it exhibits a high degree of locality in instruction and data references. This makes it suitable to study the effects of varying the cache parameters.

5. CASE STUDY

To evaluate the efficacy of the proposed simulation platform we shall present a case study. First we will show the design of a NoC based system, followed by the configuration steps involved to simulate that design using the proposed platform. Then we present some of the simulation results as a glimpse of what the platform can provide.

5.1 System Design

For illustration purpose, we have designed a system consisting of 2 processing cores and 2 memory cores [Figure 7]. The memory is shared between the two processors and can be custom set. We connected the processors and the memory in a 2x2 2-Dimensional mesh topology. To exploit the nature of the globally asynchronous nature, we have set the operating frequency of the processor at 2 GHz and the NoC operating frequency at 1 GHz. Other relevant detail about the setup has been summarized in Table 2.

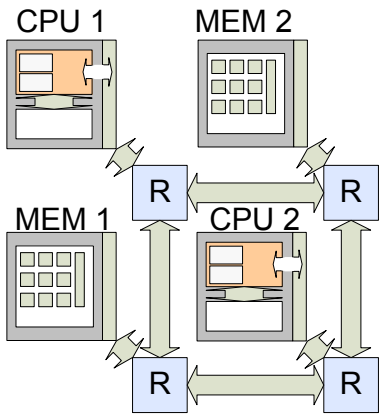

Figure 7: An Example 2 CPU 2 Memory Network On Chip Configuration

We have divided the memory address space of both the processors equally among the two memory cores. However, they do not share any data in this design. Sharing of data would require coherence protocols which are not yet implemented in this design.

5.2 System Configuration

NoCSim can be configured to simulate a given design using a simple and straight forward XML configuration file. In the configuration file all the relevant details of the system design can be specified. The configuration required to simulate this design has been given in Figure 8.

5.3 Sample Results

NoCSim models all the components of the system using SystemC and hence has control over the complete system. Unlike other simulators, in NoCSim, along with network design parameters one can simulate architectural behaviors and get data on those. Table 3-5 illustrate the sample simulation run results.

Table 3: Cache Results

Hit Rate	67%
Average Access Latency	309 ns

Table 4: Simulation Speed

Cycles Per Second	67k

Table 5: Network Results

Average Hop Latency	15 cycle
Average End Latency	50 cycles
Average Power	26 mW

The power has been evaluated using the power model described in [11].

6. CONCLUSION

In this work, we have described the NoCBench framework using NoCSim, a SystemC-based network-on-chip simulator. NoCBench allows fast and accurate simulation of Network on Chip based SoC designs and is capable of simulating the full system. Hence NoCBench enables quick validation of design of each component of the system and the overall performance. An example case study illustrates the ease of simulation.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE Simulator SYSTEM "config.dtd">
<Simulator cycles="1000000" maxresponsetime="1000" quietmode="false">
<PowerModel tgpb="10000" pgpb="8" ebuffread="76" ebuffwrite="80" ecrossbar="83" elink="6"/>
<Network nvcs="3" buflen="8">
        <Node type="PElement" name="pe1">
         <PElement type="ocp" infifolen="8" outfifolen="8">
           <CNI msgqlen="4" reorder="false" type="ocp"></CNI>
           <CORE type="risc"></CORE>
           <ADDRSPACE>  /* Defines the address maps for this core */
            <SEGMENT base="0x00000000" range="0x000fffff" target="mem1"></SEGMENT>
            <SEGMENT base="0x00100000" range="0xffffffff" target="mem2"></SEGMENT>
           </ADDRSPACE>
         </PElement>
        </Node>
        <Node type="PElement" name="pe1"> /* Similar to pe1 with different name and address map */ </Node>
        <Node type="PElement" name="mem2">
                <PElement type="ocp" infifolen="8" outfifolen="8" >
                        <CNI msgqlen="4" reorder="false" type="ocp" flitinjectionrate="10"></CNI>
                        <CORE type="memory"></CORE>
                </PElement>
        </Node>
        <Node type="PElement" name="mem1"> /* Same as mem1 */ </Node>
        <Node type="Router" name="router1"><Router type="generic" ports="3" infifolen="8" outfifolen="8"/>      </Node>
        /* Similar declaration for other 3 routers */
        <Link src="pe1" dst="router1"/>
        /* Similar declaration for rest of the connections */
</Network>
</Simulator>
```

Figure 8: Sample Configuration for the System used in case study

To further enhance the usability of NoCBench, several steps will be taken that includes development of additional core library components that can allow a more diverse system setup. Secondly, a rich set of applications will be compiled for using in the validation process. This will enable the study of NoC performance in a realistic scenario. In addition, a more intuitive interface is being developed for seamless configuration and result analysis purposes.

7. REFERENCES

[1] http://noxim.sourceforge.net/

[2] Grecu, C.; Ivanov, A.; Pande, R.; Jantsch, A.; Salminen, E.; Ogras, U.; Marculescu, R., "Towards Open Network-on-Chip Benchmarks," *First International Symposium on Networks-on-Chip, 2007. NOCS 2007.*, vol., no., pp.205-205, 7-9 May 2007

[3] L. Chunho, M. Potkonjak, and W. H. Mangione-Smith, "MediaBench: a tool for evaluating and synthesizing multimedia and communications systems," in *Thirtieth Annual IEEE/ACM International Symposium on Microarchitecture, 1997*. 1997, pp. 330-335.

[4] Zhonghai Lu, Rikard Thid, Mikael Millberg, Erland Nilsson, and Axel Jantsch. NNSE: Nostrum network-on-chip simulation environment. In *Swedish System-on-Chip Conference (SSoCC'03)*, April 2005.

[5] P. Guerrier and A. Greiner, "A generic architecture for on-chip packet-switched interconnections," in *Proc. Design, Automation and Test in Europe Conference and Exhibition (DATE)*, Paris, 2000, pp. 250-256.

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

[7] L. Shang, L.-S. Peh, and N. K. Jha, "PowerHerd: a distributed scheme for dynamically satisfying peak-power constraints in interconnection networks," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Designs,* vol. 25, pp. 92-110, 2006

[8] P. Bhojwani and R. Mahapatra, "Core network interface architecture and latency constrained on-chip communication," in Proc. IEEE Intl Symposium on Quality Electronic Devices (ISQED), San Jose, 2006, pp. 358-363

[9] P. Bhojwani, J.D. Lee, and R.N. Mahapatra, "SAPP: Scalable and Adaptable Peak Power Management in NoCs," *Proceedings of Intl. Symposium on Low Power Electronics and Design* (ISLPED), 2007, pp. 340-345.

[10] P. Bhojwani, R. Mahapatra, E. J. Kim, and T. Chen, "A heuristic for peak power constrained design of network-on-chip (NoC) based multimode systems," in *Proc. IEEE 18th Intl Conf on VLSI Design*, 2005, pp. 124-129.

[11] Y. Jin, E. J. Kim, and K. H. Yum, "Peak Power Control for a QoS Capable On-Chip Network," in *Proc. Intl. Conf. on Parallel Processing*, 2005, pp. 585-592.

[12] B. Towles and W. J. Dally, "Route packets, not wires: on-chip interconnection networks," in *Proc. ACM/IEEE Design Automation Conference (DAC)*, 2001, pp. 684-689.

[13] OCP International Partnership. Open core protocol specification, release 2.2, Jan 2007.

[14] A. Kumar, N. Agarwal, Li-Shiuan Peh, N.K. Jha, "A system-level perspective for efficient NoC design," in *Proceedings IEEE International Symposium on Parallel and Distributed Processing, (IPDPS) 2008.* pp. 1-5.

[15] P.R. Panda, "SystemC - a modeling platform supporting multiple design abstractions," in *The 14th International Symposium on System Synthesis, 2001*, pp. 75-80.

[16] P.S. Magnusson, M. Christensson, J. Eskilson, D. Forsgren, G. Hallberg, J. Hogberg, F. Larsson, A. Moestedt, B. Werner, "Simics: A full system simulation platform," in *Computer*, vol.35, no.2, pp.50-58, 2002

[17] H.-S. Wang; X. Zhu; L.-S. Peh; S. Malik, "Orion: a power-performance simulator for interconnection networks," in *Proceedings. 35th Annual IEEE/ACM International Symposium on Microarchitecture, 2002. (MICRO-35)*, pp. 294-305.

[18] S. Borkar, "Thousand core chips: a technology perspective." in *Proceedings of the 44th Annual Conference on Design Automation*. DAC '07, pp. 746-749

[19] Peh, L.-S.; W.J. Dally, "A delay model and speculative architecture for pipelined routers," in *The Seventh International Symposium on High-Performance Computer Architecture, 2001. HPCA*, pp.255-266.

[20] P.P. Pande, C. Grecu, A. Ivanov, R. Saleh, G. De Micheli, "Design, synthesis, and test of networks on chips," *IEEE Design & Test of Computers,* vol.22, no.5, pp. 404-413, 2005.

[21] NoCSim, http://codesign.cs.tamu.edu/nocsim

[22] M. M. K. Martin, D. J. Sorin, B. M. Beckmann, M. R. Marty, M. Xu, A. R. Alameldeen, K. E. Moore, M. D. Hill, and D. A. Wood, "Multifacet's general execution-driven multiprocessor simulator (gems) toolset," *SIGARCH Comput. Archit. News*, vol. 33, no. 4, pp. 92-99, 2005.

[23] F. Li, C. Nicopoulos, T. Richardson, Y. Xie, V. Narayanan, M. Kandemir. "Design and Management of 3D Chip Multiprocessors Using Network-in-Memory". International Symposium on Computer Architecture, June, 2006.