Parametrizable NoC Emulation Framework for Performance Evaluations

Jaya Suseela and Venkatesan Muthukumar
Dept. of Electrical and Computer Engineering, University of Nevada Las Vegas, Las Vegas, NV-USA

Abstract: Specific parameters for Network on Chips (NoCs), such as topology, switching method, and packet sizes, have a huge impact on performance of NoCs. Cycle and bit accurate simulation and emulation are necessary to evaluate and validate the performance of the NoC system. The goal of this work is to develop an open platform, synthesizable NoC framework that would evaluate such performance metrics as area, power, latency, and congestion for various design explorations. The NoC framework developed is completely parameterizable, where the designer can evaluate various design space explorations like topology, PE architecture, switching and routing algorithms, packet size, and error correction, by modifying the configuration file. The proposed NoC framework has been evaluated for various congestion scenarios, and the results are discussed.

Keywords: NoC, Router, Implementation, Cycle and bit accurate, Openrisc.

1 Introduction

Networks on Chips (NoCs) have been proposed as a promising solution to complex on-chip communication problems. However, many challenging research problems remain unsolved at all levels of design abstraction, such as design exploration of NoC architecture for applications; scheduling and mapping algorithms; evaluation of switching, topology or routing algorithms for efficient execution of applications; and optimizing communication costs, area, energy, and so forth. A solution to solving the above problems calls for the development of a synthesizable, parameterizable NoC framework that would evaluate and implement these problems and algorithms with minimum ease and flexibility.

The main contribution of our work is the implementation of a parameterizable cycle accurate NoC framework. The framework helps us to: (1) explore the architectural design space faster (2) evaluate and choose efficient NoC architecture from a range of switching techniques and topologies, with regard to latency, area and power; and (3) evaluate different architecture, topologies, switching, and routing algorithms extensively with various regular traffic patterns and application-oriented traffic. Moreover, the design is fully synthesizable and has been implemented in a field-programmable gate array (FPGA).

A brief summary of existing NoC simulators and emulators are presented here. Orion [7] and LUNA [8], two NoC simulators especially developed for power simulation of on-chip interconnection networks, do not consider computational cores. FAST [6] is a functionally accurate NoC simulator limited to IBM's proprietary Cyclops-64 architecture. SICOSYS [13] is a general-purpose interconnection network simulator that captures essential details of low-level simulation. RSIM simulates shared-memory multiprocessors and uniprocessors built from processors that aggressively exploit instruction-level parallelism (ILP). RSIM, which is execution-driven, models state-of-the-art ILP processors, an aggressive memory system, and a multiprocessor coherence protocol and interconnect, including contention at all resources. NoC simulators such as NNSE [9], Noxim [10], and NIRGAM [11] have flexibilities in configuring parameters of on-chip networks and are capable of obtaining performance metrics; however, these simulators are based on SystemC and are not synthesizable. XPIPES [19] consists of parameterizable network building blocks that can be composed at instantiation time; the parameterizable factors are the network interface, switches, and links.

The ability of a network to efficiently disseminate information depends largely on the topology. Mesh and Torus are the most commonly used topologies. The WK-recursive networks [16] are a class of recursively scalable networks that offer a high degree of regularity, scalability, and symmetry.

The NoC framework with WK-recursive topology is shown in Figure 1.

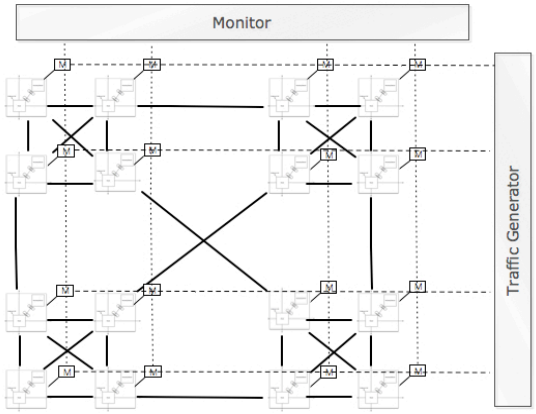

Figure 1: NoC Framework with WK Topology

In this paper, Section 2 explains briefly the implementation details, and Section 3 details the communication flow in the NoC framework. Performance analysis of switching

networks and topologies from simulation and synthesis results also are presented in Section 3. Section 4 provides the results and conclusions of this work.

2 NoC Framework Implementation

The proposed NoC framework consists of five main modules: i) the Processing Architecture, ii) the Communication Infrastructure, iii) a Communication Paradigm, iv) the Monitor, and v) the Traffic Generation Module.

The Processing Architecture module consists of a Processing Element (PE) and Network Adapter (Core Network Interface) module. The Communication Infrastructure consists of network topology and a routing node. The Communication Paradigm describes the switching techniques and routing algorithms employed in the NoC Communication Infrastructure. The Monitor module includes two sub modules: a) a Node monitor, which monitors the activities in a routing node, and b) an NoC monitor, which monitors the communication within the framework.

2.1 Processing Architecture

The processing element (PE) in the framework can be a master PE or a slave PE. Only master PEs can initiate a message transfer. Slave PEs respond to the requests from the master PE either by sending back the requested signals/data or by saving the received information. In our framework, UART, TIMER, Instruction/Data Memory and slave processors are considered as slave PEs, and the master PEs and slave processors are capable of performing computational operations.

Each master PE or slave processor consists of one OpenRisc 1000 (OR1K) [1] processor that communicates to an Instruction memory (IMEM) through a Wishbone bus. OR1K is a 32-bit load and stores an ARM9-based RISC embedded processor with 5 state pipelines; it has a maximum clock frequency of 250MHz. The OR1K processor shows better performance per clock cycle than MicroBlaze in the Stanford benchmark [4], and therefore is considered a more efficient architecture than the MicroBlaze architecture [3]. The architecture defines several features that are quite useful for networking and embedded computer environments. Most notable are the 32/64-bit architecture, the Programmable Interrupt Controller, several instruction extensions, a 2/3 Level Cache, a configurable number of general-purpose registers, a configurable cache and TLB sizes, dynamic power management support, and space for user-provided instructions. IMEM is a Block Ram with an 8-bit data bus and a 32-bit address bus. The instructions to be executed by the core are loaded in IMEM. The Wishbone clock frequency can be equal to an OR1K or an OR1K/2 clock frequency.

The Network Adapter (NA) interfaces the PEs with the network. Its main function is to generate and process packets to and from the PEs. The NA component on the master side is called the Core Interface (CI); the Network Adapter on the slave side is called a slave network interface (NI).

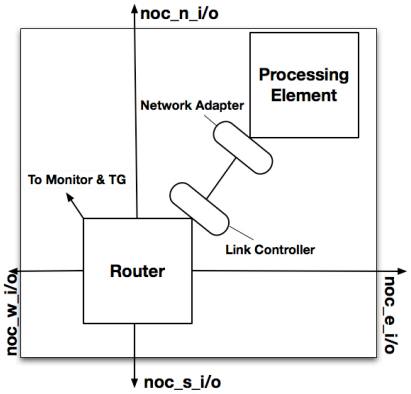

Figure 2: Routing node

2.2 Communication Infrastructure

The communication infrastructure consists of a routing node and network topology. The routing node (shown in Figure 2) consists of a link controller and a router. The link controller (LC) provides an interface between the NA and the NoC. Its main function is to match the NA clock rate with that of the network topology. Routing nodes run at four times the frequency of PEs. Synchronization registers are used to match clock rates between the slow PE and fast routing nodes. First-in first-out (FIFO) buffers are also added in the LC to store data packets from the network before transmitting to adjacent PEs.

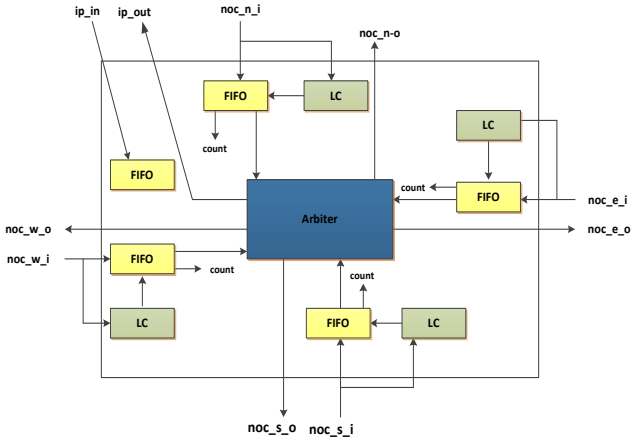

Figure 3: Router Architecture

The Network Router is responsible for the transfer of packets between nodes. Each router consists of two main components: input buffers and an arbiter. Each router (shown in Figure 3) has five input and five output ports. There exist four inputs/outputs from/to the four cardinal directions (North, East, South and West), and one from/to the PE. To

prevent deadlocks, the input buffer implements the virtual channel concept.. The Virtual Channel (VC) identifier module determines which VC should be used based on occupancy of the input buffer. VC identifier polls the buffer count of each VC and directs the incoming packet to the least occupied VC. The Switch Identifier module chooses a packet from each VC in a round-robin manner and sends it to the output port based on the routing signals obtained from the arbiter. The arbiter implemented is FSM-based, and consists of the routing table with the shortest path to the destination PE.

2.3 Communication Paradigm

In order to forward the message/packet, the implemented NoC framework can choose either the Store and Forward (SF) switching technique or the Wormhole (WH) switching technique. In SF switching, the message can be sent either as packets or in the form of flits. Each flit is contains 25 bits. When the message is transmitted as flits, each routing node will wait until the entire message is received before processing the HEADER. The end of the message/packet is determined by the TAIL flit. In Wormhole routing, the message is transmitted as soon as the HEADER is available. The path is determined from the HEADER as it moves through the network. The remaining flits follow the same path. The path is disconnected when the TAIL flit is received. For the Torus and Mesh topologies, the implementation uses an XY routing algorithm with store and forward switching. For WK-recursive topology the framework uses the adaptive routing algorithm with wormhole switching.[18].

2.4 Monitor Module

Every routing node in the NoC is connected to a "Node monitor," which connects to a top-level monitor called the "NoC monitor." The main function of the NoC monitor is to collect information from individual Node monitors regarding the traffic. The Node monitors generate control information based on the buffer conditions of that router node. The Node monitor uses a few ON/OFF signals, such as FAIL, FULL and ALMOST FULL, to communicate with the NoC monitor.

2.5 Traffic Generator Module

The Traffic Generator (TG) module is responsible for generating different traffic distribution in the network. The TG can generate mainly three different types of traffic: 1) uniform traffic, where packets are send at equal intervals of time; 2) hotspot traffic [2], where the cores either receive packets at a higher rate than the rate they can process or else generate packets at a higher rate than the destination can process; and 3) sporadic traffic, where each core generates a burst of packets.

3 NoC Framework Communication Flow

This section explains the control and data flow sequence followed during the transfer of packets between PEs. Each packet can contain a variable number of flits. Every packet consists of a header and a tail flit. Data and address flits are optional. The framework allows packing of a variable number of data flits into a single packet, which further reduces latency during burst mode data transfers. The packets are classified as request packets, which have an optional address field; and response packets, which have a data field. The address field defines the local memory address in the destination PE. This allows the slave PE to have its own unique memory address space. The packet format is shown in Figure 4.

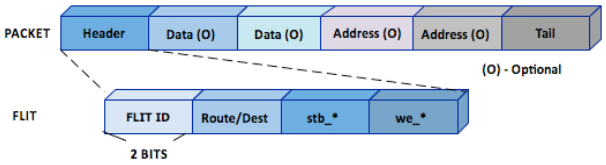

Figure 4: Packet Format

The header flit can consist of either only the source and destination address or the entire route. Two bits in each flit are used as a packet ID, which determines the type of packet (00-Header, 01-Data, 10-Address, 11-Tail). In the data and address flits, the first three bits next to the packet ID determines the order of the data/address flits. Every flit contains a control bit (stb_*), which determines the validity of packet. The we_* bit signifies if the packet has to perform a read or write operation. Typically, each data or address flit contains 16 bits of data or address information. The tail flit contains parity bits for every data or address flit. A parity bit is a bit that is added to ensure that the number of bits with the value of "one" in a set of bits is even or odd. Parity bits are used as the simplest form of error detecting code (ECC). The type of ECC (odd or even parity) used in tail flit is parameterizable.

Every channel/link in the network is full duplex, i.e., two messages can travel simultaneously on the link in opposite directions. A channel/link is said to be congested if its associated router buffer is completely full or partially full (parameterizable). A READY and SEND signal are used to communicate between adjacent routers. Whenever the channel buffer is partially full, the monitor informs every adjacent router about a possibility of congestion.. When the channel buffer of the router is full, the monitor flags the congestion by setting the READY signal to low.

The communication flow in the proposed NoC framework, as shown in Figure 5, can be summarized as follows. Let us assume a scenario where the master PE wants to write data to

a specific address location on a slave PE (D-MEM). The master PE initiates the transfer by activating the Network Adapter (NA), which validates the memory address and assembles the packet. In source routing, the NA uses a routing table to determine the route. Once the packet is ready, the Link Controller stores the packet in the input buffer of the router. The router arbiter, based on the route information and the availability of input buffers in the adjacent routers, determines the output port and forwards the packet.

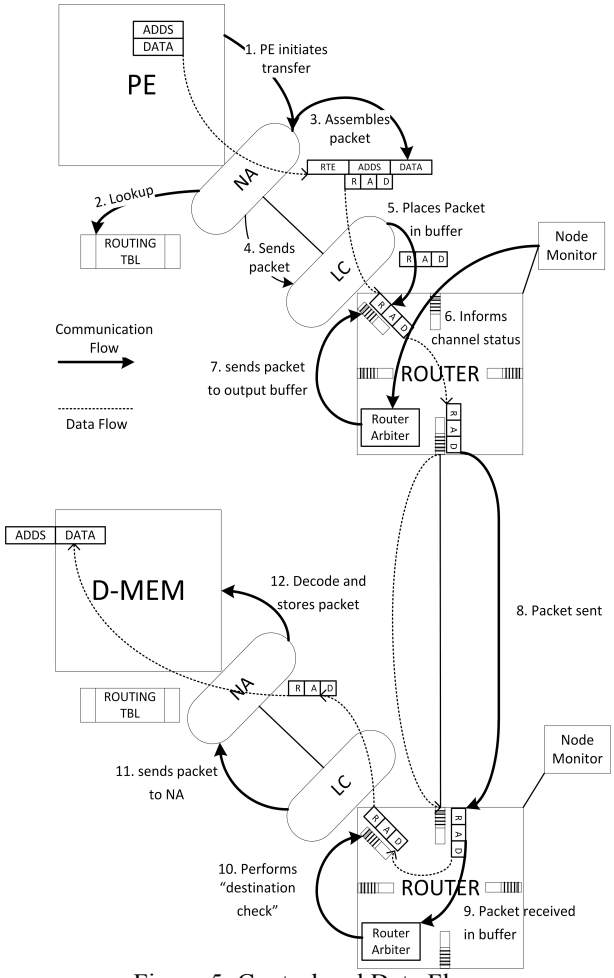

Figure 5: Control and Data Flow

When the packet reaches the input buffer of the adjacent router, the router arbiter performs a "destination check" on the packet to determine if the packet has reached its destination. If the packet had reached its destination, the router arbiter sends the packet to the network adapter. The network adapter decodes the packet into the D-MEM address and the data, and performs the write operation. If the packet has not reached its destination, the router arbiter forwards the packet to the next router in a similar manner as the master router arbiter. In case of channel congestion, the router stores the packet in its buffer until the congestion is eliminated.

4 Evaluations and Results

The parameterizable NoC framework that was developed was implemented in Verilog HDL. The input to the design is a configuration file that determines the topology (torus, mesh, or WK-recursive), switching (Store and Forward or Wormhole), size, depth and the number of virtual channel buffers, size of packet, and type of ECC (odd or even parity). The parameters are static, which means the configurations cannot be modified during simulation or after synthesis. Also, different architectural components of the framework can operate at different frequencies to mimic real-world applications and real-time traffic. These characteristics can be modeled by modifying the NoC configuration file. Table 1 summarizes the most important features that can be parameterized in the framework.

Table 1: Parameterizable features in the NoC framework

NETWORK TOPOLOGIES	LINEAR, MESH, TORUS, WK-RECURSIVE
Channel Width/Packet size	22-80 bits
Flow Control	ON/OFF
Switching mode	SF and Wormhole
Routing Algorithm	XY for Mesh, Torus, Simplex Routing Algorithm for WK-recursive
Buffering	Input Buffering, Single or Multiple Virtual Channels
Network Synchronization	Synchronous
Virtual Channels	Depth, size and number of virtual channels are parameterizable
Traffic Patterns	Uniform, Sporadic, Hotspot

A packet flow/traffic can be initiated either by the Traffic Generator (TG) or by the master PE. OR1k has a GNU tool chain, including the GCC compiler and the GNU debugger. The application software can be loaded in the external instruction memory of the corresponding master PE. The processor uses on-chip RAM to execute a bootloader, in which the cache, stack, and MMU are enabled and initialized. After initialization, the master PE interacts with other PEs in framework, thereby executing the application. The complete emulation framework is presented in Figure 6.

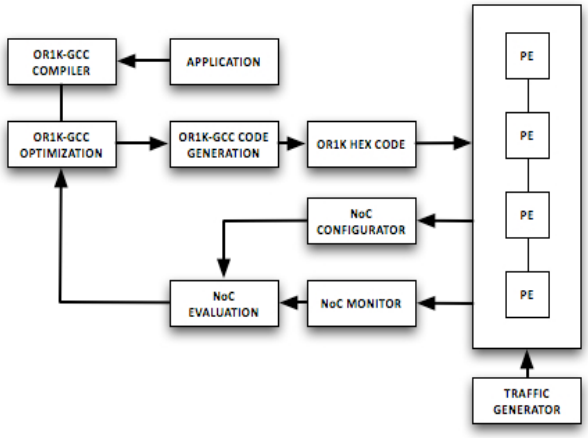

Figure 6: The complete NoC Emulation Framework

The NoC network clock (Router node) is operated at 1GHz. Framework consists of several master and slave PEs. Master PEs (OR1K+IMEM) operates at 250MHz. The slave PEs, which includes: TIMER, UART, and normal data memory unit operate at 125MHz, a D-MEM operates at 250MHz (Slow Mode) or 500MHz (Fast Mode). Congestion scenarios (hot-spot) are simulated when a faster PE sends a series of packets to a slower PE. Regular traffic scenarios are simulated when a faster/slower PE sends a series of packets to a faster PE. Also traffic scenarios for single and multiple hop transfers are modeled. Figure 7, shows the different traffic scenarios modeled in the proposed NoC framework.

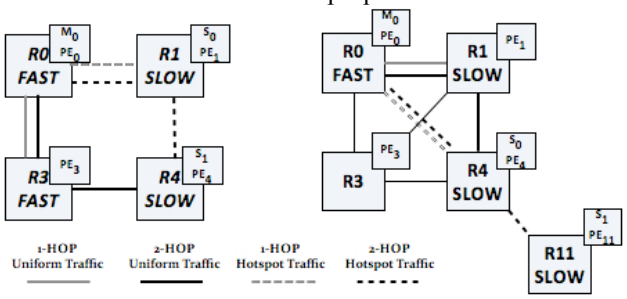

Figure 7: Congestion Traffic Models

In order to evaluate the flexibility and diversity of the proposed NoC framework, the following design variations are considered. First, the 3x3 Torus topology with SF switching and XY routing algorithm is evaluated. Second, the WK(2,4) topology with Wormhole(WH) switching and minimum routing algorithm is evaluated. Evaluation of WK topology includes support for burst data transfer. A master sending multiple data to same destination can be packed into a single packet in WH, hence further reducing latency.

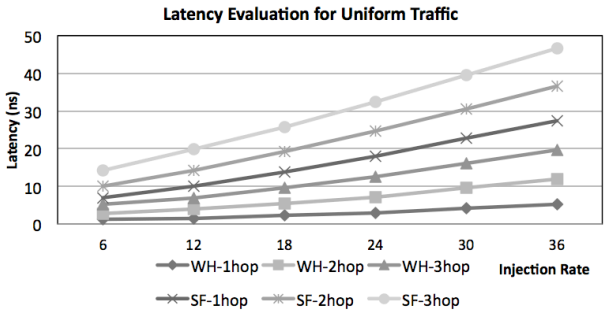

Figure 8: Latency Evaluation of Torus Topology

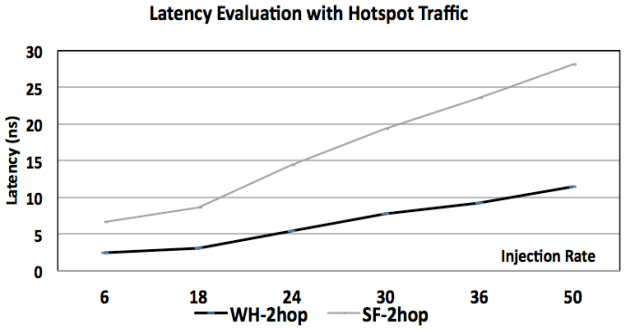

Figure 9: Latency Evaluation of WK Topology

The performance metrics of these design variations include: latency vs injection rate (packets/clock cycle), logic area (number of slices) and power (in watts). The designs were also evaluated for performance metrics under normal and hotspot traffic conditions. Figure 8 shows the latency for Torus and WK topologies for uniform traffic, and Figure 9 shows the latency for hotspot traffic for different hops. The design was synthesized for Virtex4, using Xilinx ISE 11.1. Table 2 shows the area and static power comparisons of different NoC architecture components for the Torus and WK topologies.

Table 2: Area and Power of Torus and WK

	Area (Slices)		Power (W)	
Block/Module	WH	SF	WH	SF
PE-OR1K	5992	5992	0.38216	0.38216
Router	1670	1778	0.17106	0.19048
Core Interface	65	543	0.17948	0.16633
Network Interface	37	69	0.16183	0.16893

5 Conclusion

A synthesizable NoC framework was developed using Verilog HDL. The framework is parameterizable, and has been used as a tool for design space exploration of various topology (Torus and WK), switching techniques (SF and WH), and network traffic (uniform and hot-spot). The performance metrics of the design space exploration include latency, area, and power. By using the proposed framework, researchers will be able to evaluate and compare various novel NoC architectures and algorithms with accurate performance, power, and area parameters, and hence facilitate the determination of system-level performance.

6 References

[1] OpenRisc 1000 architecture Manual from <opencores.com>

[2] A. Kumar et al; "Toward Ideal On-Chip Communication Using Express Virtual Channels". IEEE Micro Vol 28, Issue 1, January- February 2008, pp 191-202.

[3] D.Mattsson and M. Christensson , "Evaluation of Synthesizable CPU Cores" master's thesis 2004.

[4] https://benchmark.stanford.edu

[5] Krishnan Srinivasan, Karam S. Chatha, and Goran Konjevod "Linear-Programming-Based Techniques for Synthesis of Network-on-Chip Architectures". In *Proceedings of the IEEE International Conference on Computer Design* (ICCD '04). IEEE Computer Society, Washington, DC, USA, 422-429.

[6] Juan del Cuvillo et al.: FAST: A Functionally Accurate Simulation Toolset for the Cyclops64 Cellular Architecture. MoBS'05 Workshop in conjunction with ISCA'05, 2005.

[7] Hangsheng Wang et al.: Orion: A Power-Performance Simulator for Interconnection Networks. In Proceedings of MICRO 35, 2002.

[8] Zhonghai Lu, Rikard Thid, et al.: NNSE: Nostrum network-on-chip simulation environment. Design, Automation and Test in Europe Conference, 2005.

[9] Noxim. http://sourceforge.net/projects/noxim, 2008.

[10] Lavina Jain et al.: NIRGAM: A Simulator for NoC Interconnect Routing and Application Modeling. Design, Automation and Test in Europe Conference, 2007.

[11] CellSim. http://pcsostres.ac.upc.edu/cellsim, 2007.

[12] V. S. Pai et al., "RSIM: Rice Simulator for ILP Multiprocessors," SIGARCH Comput. Archit. News, vol. 25, no. 5, p. 1, 1997.

[13] V. Puente et al., "Sicosys: An integrated framework for studying interconnection network performance in multiprocessor systems," Parallel, Distributed, and Network-Based Processing, Euromicro Conference on, vol. 0, p. 0015, 2002.

[14] Y. Hu, H.Chen, Y.Zhu, A. A. Chien and C. Cheng, "Physical Synthesis of Energy-Efficient Network-on-Chip Through Topology Exploration and Wire Style Optimizations," Design (ICCD), pp.111-118, 2005.

[15] K. Srinivasan and K.S. Chatha, "A technique for low energy mapping and routing in network on chip architectures". In Proceedings of the 2005 international symposium on Low power electronics and design (ISLPED '05). ACM, New York, NY, USA, 387-392.

[16] G. D. Vecchia and C. Sanges, "A recursively scalable network VLSI implementation," Future Generation Computer Systems, 4(3) 235-243, 1988.

[17] D.Rahmati, A.Kiasari, S.Hessabi, H.Sarbazi-Azad, "A Performance and Power Analysis of WK-Recursive and Mesh Networks for Network-on-Chips", IEEE International Conference on Computer Design (ICCD 2006), San Jose, CA, USA, Oct. 2006.

[18] Della Vecchia, G., Sanges, C.: A Recursively Scalable Network VLSI Implementation. Future Generation Computer Systems 4(3), 235–243 (1988).

[19] M. Dallosso et al., "Pipes: A Latency Insensitive Parameterized Network-on-chip Architecture for Multi-Processor SoCs," pp. 536-539, Proc. Int'l Conf. Computer Design, 2003.