Ocin_tsim - DVFS Aware Simulator for NoCs

Subodh Prabhu*, Boris Grot†, Paul V. Gratz* and Jiang Hu*
camsin_group@listserv.tamu.edu
http://www.ece.tamu.edu/∼ocintsim/
* Department of Electrical and Computer Engineering, Texas A&M University
† Department of Computer Sciences, The University of Texas at Austin

***Abstract*—Networks-on-Chip (NoCs) are a general purpose, scalable replacements for shared medium wired interconnects offering many practical applications in industry. Dynamic Voltage Frequency Scaling (DVFS) is a technique whereby a chip's voltage-frequency levels are varied at run time, often used to conserve dynamic power. Various DVFS-based optimization techniques have been proposed; however, few have been implemented, in part due to the resources required to validate architectural decisions through prototyping. As a result, designers are faced with a lack of insight into potential power savings or performance gains at early architecture stages. This paper proposes a DVFS aware NoC simulator with support for per node power-frequency modeling to allow the fine-tuning of such optimization techniques early on in the design cycle. The proposed simulator also provides a framework for benchmarking various candidate strategies to allow selective prototyping.**

***Index Terms*—NoC (Network on Chip), network simulator, DVFS (Dynamic Voltage Frequency Scaling), power modeling**

I. Introduction

Today's dominant approach to scaling compute performance while mitigating wire delays and power consumption is through the use of single chip multi-processors (CMPs). Similarly, system-on-chip (SoC) designs have recently emerged in the embedded market as a means to reduce power consumption and costs by integrating a diverse set of many IP blocks onto a single chip. In both cases, with rising core and IP block counts, communication between cores has become a major challenge as bus-based and ad-hoc interconnects have been observed to not scale [1, 2]. In response, researchers have proposed packet-based networks-on-chip (NoCs) as a structured and scalable alternative [2, 3]. For instance, Intel achieved tera-flop performance at 5 GHz using an 80-core substrate with a mesh interconnect [4, 5]. As NoCs are still an emerging field, it has enjoyed a considerable amount of recent research activity.

One active area of work has focused on dynamically varying operating voltage and frequency levels to achieve a balance between power and performance [6]. This technique, referred to as DVFS, is used in some recent SoC designs [7]. Alternative techniques using voltage/frequency islands for IP blocks, sleep transistors and clock gating have also been proposed [8, 9, 10]. W. Kim, et. al. have proposed per-core DVFS techniques for use in fine-grain power management [11].

This document introduces "Ocin_tsim" (On Chip Interconnection Network Timing Simulator), a cycle-accurate microarchitectural Network-On-Chip (NoC) simulator. Ocin_tsim supports per-node DVFS modeling for early identification of optimum power-performance savings possible given a proposed DVFS policy. In traditional NoC-based designs, power savings estimation and verification is delayed until design prototyping via FPGA or silicon implementation [12]. As indicated in Figure 1, a large number of design candidates may exist in the planning stages. Once established, power, performance, and area objectives will subsequently reduce the possible design space. At this point, evaluation via simulation can provide a rapid, quantitative basis for refining a limited number of design candidates into a few good prototype choices. It also enables exploring a wider design space in cases where system objectives have not been defined or are relatively flexible.

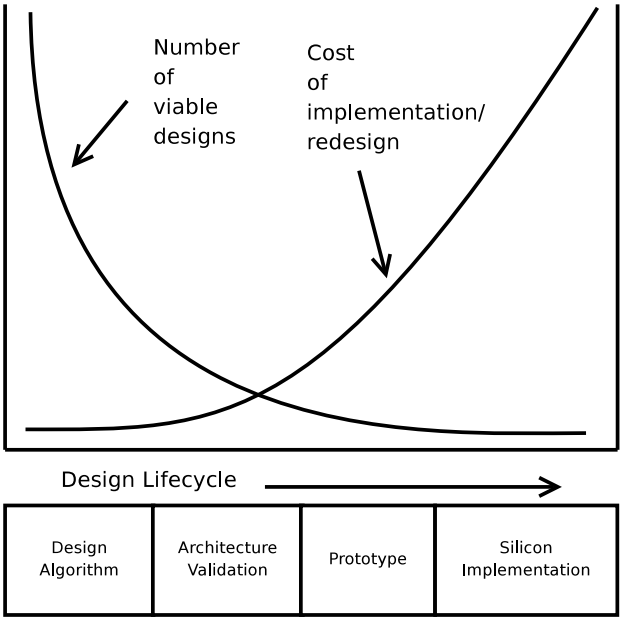

Fig. 1. Cost of redesign/implementation and viable number of potential design candidates starting from algorithm stage to silicon implementation stage. Architectural simulators work much higher up the chain and can potentially weed out several inefficient design ideas.

Existing NoC simulators either do not model DVFS or assume global DVFS [13, 14, 15, 16, 17]. As a result, many architectural optimizations can be validated only for performance benefits. To the best of our knowledge, this simulator is the first to accurately model the power and performance of an NoC utilizing per-core DVFS.

Our work on the Ocin_tsim simulator makes the following contributions:

1) A novel DVFS-aware NoC simulation environment with power-performance calculations for each node, necessary for modeling multiple voltage-frequency regions dynamically managed at run-time.
2) Support for modular integration of power models allowing users to plug-in their own power models for new process technologies and custom circuit designs.
3) Support for modeling and benchmarking of arbitrary interconnect topologies and router microarchitectures.
4) Support for visualizing network state to enhance understanding of time-varying behavior. Both conventional (text-based) and visual output modes are available and can be easily customized to a variety of needs.

The rest of the paper is organized as follows. Section II discusses prior work, Section III covers the implementation of Ocin_tsim and various decisions made in the design process. In Section IV we present the internals of simulator, while Section V covers potential use cases of our simulator for some standard designs. Sections VI and VII present summary and future work. Finally we present acknowledgments in Section VIII and some detailed simulator information in Appendices A- C.

II. Related Work

Existing literature landscape in NoC architectural simulators at architectural stage can be classified into the following broad classes:

Class A: Micro-architectural simulators
: These simulators compute actual physical quantities of an NoC such as network latencies [13, 14, 15], power consumption [14, 15, 16], and noise immunity [18]. While being exhaustive, they are computationally intense and limited in precision by models employed.

Class B: Abstract simulators
: Class B simulators model only a specific section/layer of network such as task scheduling, packet arbitration, packet management and/or packet routing and generalize the results for complete interconnect network. [17]. Abstract simulators provide high simulation performance and can be effective at proving the underlying optimization without the complexities of an actual NoC implementation. These simulators rely upon generalized assumptions about the characteristics of network traffic and often prove inaccurate at estimating the performance of realistic workloads.

The previous division is by no means exhaustive. Our simulator lies squarely in Class A and hence is different from all Class B simulators in that it directly models network latencies and dynamically estimates the power consumption for a given NoC configuration. Our simulator is different from other simulators in Class A in three respects: 1) it allows each node to operate in independent voltage and frequency domains as in a per-core DVFS setup, 2) it uses Orion system of power modeling [19] to compute both static and dynamic power consumption and 3) it provides a graphic representation of dynamic network traffic to allow an intuitive visualization.

III. Simulator Implementation

In this section, we discuss the implementation details of Ocin_tsim. This section presents an eagle's eye view of the simulator, reserving more detailed discussion for subsequent sections.

A. Preliminaries

Ocin_tsim was born out of the need for a tool that could quickly and efficiently validate or reject NoC architectural propositions employing DVFS. Existing simulators, as listed in Section II, were found lacking for such a purpose. As previously discussed, NoC power consumption and packet latency are highly dependent upon upon routing function, DVFS policy and network traffic. Therefore, detailed microarchitectural simulation of the proposed NoC under a given workload is required to provide an accurate measurement of the system's performance and power characteristics.

The goals of Ocin_Sim's design are as follows:

Flexible design
: The simulator was designed for use in early stage, NoC architectural research where numerous designs must be evaluated. By supporting run-time configurability, Ocin_tsim enables rapid evaluation of diverse workloads and network parameters. Such efficiency is not possible with compile-time configurable simulators.

Extensibility
: The simulator has a modular architecture that facilitates extensibility via user-defined pluggable extensions. Sample components for visualization and power modeling are included in the standard distribution of the simulator and may easily be replaced with custom models.

Fast performance
: As with all architectural simulators, an important challenge lies in simulating more cycles in less wall-clock time. One of the benefits of having a modular simulator architecture is that it enables suppression of unused components to improve simulator performance. For example, in timing-only mode, Ocin_tsim achieves faster simulation speeds by disabling power modeling and visualization modules.

B. Organization

Modular structure: To support our aims, we chose a modular, object-oriented design style to implement the simulator. Each logical component was implemented as a separate C++ class. Such a structure not only supports our extensibility goal by being amenable to modular modifications, but also supports our flexibility goal by allowing use of inheritance and polymorphism to implement runtime configuration.

As an example of how this modular structure can be exploited, consider how per-node frequency support was added to the simulator. Initially Ocin_tsim supported a single, universal clock frequency across all the nodes. Simulation context was provided to each node by a simulator object every simulation

step thereby evaluating and thus proceeding the simulation. Therefore to implement dynamic, per-node frequency assignment, we simply had to constrain the number of simulation steps when a given simulator object would be evaluated. By including this count of cycles as a runtime configurable parameter, we were able to configure each node to be evaluated at different intervals thereby implementing multiple clock frequencies.

Top-level structure: Figure 2 shows a structural organization as well as syntactical description of nodes in a simple 2x2 mesh-based NoC. As the diagram shows, each node of an NoC is modeled in Ocin_tsim as an object containing four different module instances. We briefly discuss the role of each of these nodes below:

- **Router model:** The router module models the flow of packets through the given router's user-defined microarchitecture. This module contains a simulator timing object which synchronizes the packet traversal at an interval set by its clock frequency.
- **Power model:** Each node also carries an Orion [19] instance that computes the per-node power consumption based on actual run-time activity. The original Orion release was modified to allow per-node voltage-frequency selection. These values may be changed at runtime for fully dynamic voltage and frequency modeling. Other power models can also be easily adapted for use with our simulator.
- **Monitor and visualization:** Ocin_tsim contains monitoring modules for data collection and subsequent post processing using built-in statistics and/or visualization blocks. Monitor modules can be used to report statistics on various network elements such as packet routing, arbitration logic, link bandwidth, FIFO usage and other router statistics. The visualization module uses the "GD" standard graphics library to capture the selected monitor statistic into a series of images [20]. In complex, time-varying environments such as those found in NoCs, graphical representations can often present a more efficient way to gain an intuitive grasp of network performance bottlenecks.
- **I/O model:** Ocin_tsim contains injector and ejector modules which mimic traffic generation and reception by the local processor or IP block attached to each router. While injector and ejector modules mimic various possible ways in which traffic may be induced across the NoC, an instance of I/O module in each node controls their behavior on a per-node basis.

Figure 2 shows a sample node declaration in the network configuration file. Runtime configuration via simple text files allows the description of complex NoC topologies via simple declarative statements. The configuration files are discussed in detail in Appendix B.

Traffic generation and routing: Ocin_tsim supports several traffic generation, routing, and port selection functions. As with other simulator fields, all of these are runtime configurable. Packets can be generated either in random or bimodal mode (alternate between two different packet sizes). The simulator supports a variety of synthetic traffic patterns including bit complement, transpose, bit reverse and hotspot. In addition, Ocin_tsim has a trace-driven packet generation mode to support network-level replay of simulated applications. Similarly, routing functions, port selection functions, and resource allocation functions can be chosen at run-time. For a comprehensive list of run-time configuration options, please refer to Appendix B.

C. Novel features

Ocin_tsim contains a number of novel features, several of which are highlighted below.

- **Per-node dynamic frequency modeling:** Ocin_tsim models per-node frequency in a manner analogous to sim clock in Verilog. Each clock period used in simulator is specified as a multiple of root clock time period where root clock is the time step at which simulation is advanced. Thus, a node with a clock period multiplier of two will be evaluated every two simulation steps. Signals crossing clock domains pay a synchronization latency of 2x the destination clock period.
  It is important to note here that root frequency sets the relative updating interval of software execution cycle and hence determines the simulation time. As a result, for optimum simulation time, root clock should normally be set to Highest Common Factor (HCF) of all the clock periods in the network.
  In addition to its applications in DVFS, dynamic frequency scaling (DFS) has been applied to globally asynchronous, locally synchronous (GALS) designs containing mesochronous clock boundaries. Our simulator does not impose any restrictions on the use of dynamic frequency tuning and can be used for DVFS as well as DFS designs.
- **Power modeling:** Another novel features of our simulator is support for power modeling at architectural level in a DVFS setup. Each node in Ocin_tsim can compute per-node power consumption based on actual, traffic-induced activity. Our simulator can be used with any off-the-shelf or custom power model to generate power estimates with desired accuracy.
- **Visualization:** Another novel feature of Ocin_tsim is visualization support. By processing the data gathered by various monitor modules, the simulator is able to generate graphic representations of network buffer occupancy, link utilization, and other network statistics at the end of the simulation or during the course of one, producing a time-varying series of images. When combined with a network power model, this tool can elucidate interesting trends into per-node differences in power consumption, thermals, and traffic congestion. For a complete list of supported visualization modes, please refer to Appendix B.

IV. Simulator Internals

This section presents the internals of our simulator. We describe its functionality, internal source structure and how

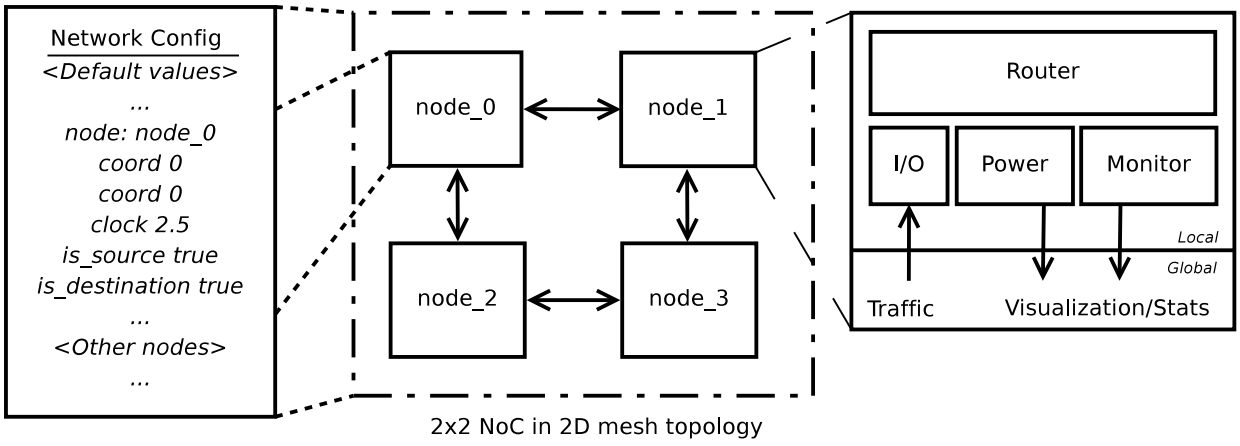

Fig. 2. Structural and syntactical organization of a node withing Ocin_tsim simulation environment for a simple 2x2 2D mesh connected NoC.

it maps to he microarchitecture of an NoC router. Various command line arguments and configuration fields used in our simulator are presented in Appendix A and Appendix B.

A. Topology and Router Microarchitecture

Figure 3 shows a simple five-node NoC connected in a star topology as implemented in our simulator. Each router connects a local resource to the rest of the network via a node interface. Packets are routed over network links and stored in Virtual Channel (VC) buffers. Virtual Channels and switch bandwidth are assigned by allocation modules. Data injection (from resource into the router) and ejection (from router into the resource) is handled by gen and ej modules. Each router instance also has a monitor module for statistics and visualization.

B. Simulator code descriptions

As shown in Figure 3, code for Ocin_tsim is completely modular; each file maps to a component C++ class. Top level files integrate the microarchitectural components of a router into a single router object instance. Multiple instances of this router object are replicated and connected together to simulate the behavior of an NoC. The following list describes the functionality of the code files in *src/* directory.

- *ocin/tsim_ocin.cpp*: top level file. Performs top level argument parsing, binds simulation agent to an instance of ocin_top.
- *ocin/ocin_top.**: top level integration files, perform parameter instantiation from config files, creates and initializes tiles and wires structure and invoke all required modules such as visualisation or ejector (Figure. 3).
- *ocin/ocin_router.**: router class. Includes routines for assigning numbered VCs and logical ports to each router, also include the evaluate subroutine (Figure. 3(a)).
- *ocin/ocin_channel.**: include routines to transmit flits and calculate corresponding credit/cost (Figure. 3(b)).
- *ocin/ocin_cost_msg.**: define the cost_msg struct to be used later on.
- *ocin/ocin_defs.**: top level define files containing various shared macros and custom data types.
- *ocin/ocin_helper.**: contains various utility functions used elsewhere in simulator.
- *ocin/ej_modules/**: default, non-blocking top-level ejector module (Figure. 3(d)).
- *ocin/gen_modules/**: various traffic generation modes including random, complement, transpose, reverse, self-similar, file trace and hotspot (Figure. 3(e)).
- *ocin/io_modules/**: router I/O component modules (Figure. 3(f)), input units (Figure. 3(m)) and output units (Figure. 3(n)).
- *ocin/monitors/**: monitor modules. Maintain a copies of various network parameters such as flits traversed through a node, injection/acceptance ratio, stalls encountered, request/grants made and etc (Figure. 3(g)).
- *ocin/rt_modules/**: routing functions such as adaptive, XY dimension-order and o1turn (Figure. 3(h)).
- *ocin/sel_modules/**: selection function. Arbitration modules (Figure. 3(i)).
- *ocin/vc_modules/**: virtual channel allocation modules (Figure. 3(j)).
- *ocin/xbar_modules/**: crossbar allocation including routines for simple, 2-level and speculative allocation (Figure. 3(k)).
- *ocin/vis_modules/**: visualisation modules. Includes routines for dumping interfacing with png/jpeg/GD libraries (Figure. 3(l)).
- *tsim/**: the simulation engine and modular agent tsim_module. Defines the simulation interface (Figure. 3(c)).

V. Sample Usage

This section describes installation procedure and presents typical use cases for NoC based design. The software is made available under MIT license as described in Appendix C.

A. Required libraries

The following libraries are required to successfully build Ocin_tsim:

- GD Graphics library, required by visualization module can be found at http://www.boutell.com/gd/.

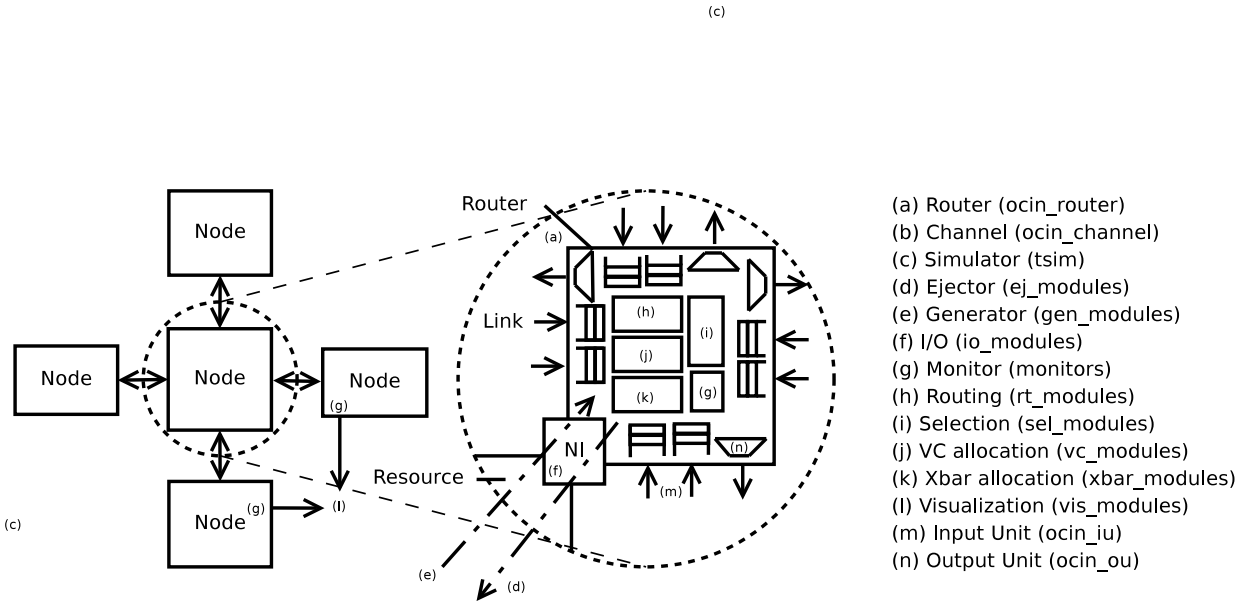

Fig. 3. Mapping between simulator structure and router microarchitecture.

– PNG Library, required by visualization module in drawing to png files can be found at http://www.libpng.org/pub/png/.
– JPEG Library, required by visualization module in drawing to jpeg files can be found at http://www.ijg.org/.
– Orion library, if you wish to perform power estimations in your simulation. We have tested our simulator to work with Orion version 2.0. Orion 2.0 library can be downloaded from http://www.princeton.edu/∼peh/orion.html. You may also use the version of Orion 2.0 library made available at http://www.ece.tamu.edu/∼ocintsim/ which has been modified to support programmatic control of voltage and frequency.

B. Installation

Ocin_tsim may be obtained online at http://www.ece.tamu.edu/∼ocintsim/ . The following commands may be used to extract and build the simulator:

tar -zxvf ocin_tsim.tar.gz
cd tmax
make ocin

C. Timing simulation

A simple Ocin_tsim use case is timing simulation for a small 2x2 NoC design. Ocin_tsim comes with a preconfigured 2x2 NoC network out-of-the-box. For this particular configuration, we have chosen a 10% injection bandwidth, 1 hop per cycle, 128 baseline channel width and run all the nodes at same clock frequency. Each node has 8 virtual channels and 4 element deep queues. Per-node configuration for this NoC is specified in the included simple_test_net.cfg file.

cd simple_test
../bin/tsim_ocin -config simple_test.cfg -n 1000

D. Timing simulation with power

To demonstrate the use of Ocin_tsim for DVFS modeling, we will use the same 4-node NoC from the example above. Node config file should be modified to assign different startup voltage/frequency, if so desired. By altering the voltage/frequency level of a node based upon a required parameter (say, the incoming traffic), end-user can implement a policy for DVFS level selection. It is important to note here that in the absence of a power model, any DVFS level may be provided and the simulator would still successfully finish the simulation. However, for the simulation to have any physical relevance, it is important to stick to the granularity and/or valid DVFS levels/range to switch between, as defined in your power model.

Next we provide the simulator a 'hook' to a power model. This hook is a simple API call for the power computing function (in power model) made for each flit activity. For this exercise we use a modified version of the Orion 2.0 library made available at http://www.ece.tamu.edu/∼ocintsim/. This library is derived from Orion 2.0 and modified to support programmatic control of voltage and frequency. By including the power calculating APIs in the simulator, any simulation now will perform both timing measurements as well as power estimation.

VI. Summary

An accurate estimation of power and timing of interconnection networks in early phases of the design process can prove to be effective in deeper exploration of NoC design space. Existing NoC simulators are inadequate for power-performance simulation of future technologies employing per-node DVFS with the goal of minimizing power consumption. In this work, we introduced a simulator that combines power and performance models, thereby enabling rapid and accurate evaluation of future NoC architectures. We have also presented

an extensible framework for integrating power models from external sources into our timing simulator.

Extensions to Ocin_tsim are both underway and planned. These are covered briefly in the Future Work section. At present, this tool provides researchers with a fast and efficient simulation infrastructure for screening architectural propositions before deploying them for FPGA testing or production. We hope that you find these tools useful, and encourage you to contact us with suggestions on improving the release, documentation and the tool itself.

VII. Future Work

Work is underway on using our simulator for performing a limit study on DVFS switching times. We believe that such a study could provide insights into emphasis required on approaches utilizing faster DVFS switching for performance benefits at the cost of reduced regulator efficiency.

An extension planned for future is to provide an API layer for integration of power models. Such an API layer would provide a standard interface enabling researchers to import their custom power models into Ocin_tsim framework.

Another future extension is using a 3D visualisation engine to provide real-time 3D rendering of various network parameters using time as the third dimension. Such a visualisation technique could be more intuitive for some users in faster debugging of performance degradation issues rooted in real-time bottlenecks. Current visualisation strategy is computationally prohibitive to be performed in real-time.

VIII. Acknowledgements

Ocin_tsim was written by Subodh Prabhu between Feb 2009 and Nov 2009. He continues to add improvements and updates. It is derived from a simulator written by Professor Paul V. Gratz and Boris Grot as research assistants at the University of Texas at Austin Computer Sciences Department, under the supervision of Prof. Stephen W. Keckler. Professor Jiang Hu provided valuable guidance in the inclusion of the power modeling feature. Professor Li-Shiuan Peh from Princeton University graciously provided Orion 2.0 power models for integration and testing. Bin Li from Princeton University provided support for Orion power model. The first release was assembled, debugged, and documented by Subodh Prabhu.

References

[1] L. Benini and G. D. Micheli, "Networks On Chips: A New SoC Paradigm," *Computer*, vol. 35, no. 1, pp. 70–78, Jan 2002.

[2] W. J. Dally and B. Towles, "Route Packets, Not Wires: On-Chip Interconnection Networks," in *International Conference on Design Automation (DAC)*, 2001, pp. 684–689.

[3] A. Hemani, A. Jantsch, S. Kumar, A. Postula, J. berg, M. Millberg, and D. Lindqvist, "Network on chip: An architecture for billion transistor era," in *IEEE NorChip Conference*, Nov 2000.

[4] S. Vangal, J. Howard, G. Ruhl, S. Dighe, H. Wilson, J. Tschanz, D. Finan, P. Iyer, A. Singh, T. Jacob, S. Jain, S. Venkataraman, Y. Hoskote, and N. Borkar, "An 80-Tile 1.28 TFLOPS Network-on-Chip in 65nm CMOS," in *IEEE International Solid-State Circuits Conference*, February 2007.

[5] Y. Hoskote, S. Vangal, A. Singh, N. Borkar, and S. Borkar, "A 5-GHz Mesh Interconnect for a Teraflops Processor," *IEEE Micro*, vol. 27, no. 5, pp. 51–61, Spetember-October 2007.

[6] P. Macken, M. Degrauwe, M. V. Paemel, and H. Oguey, "A voltage reduction technique for digital systems," in *IEEE International Solid-State Circuits Conference*, February 1990, pp. 238–239.

[7] C. Lai, J. H. Lin, and Y. F. Wang, "DVFS SoC Architecture and Implementation," *SoC Technology Journal*, vol. 3, pp. 84–91, 2006.

[8] D. E. Lackey, P. S. Zuchowski, T. R. Bednar, D. W. Stout, S. W. Gould, and J. M. Cohn, "Managing power and performance for System-on-Chip designs using Voltage Islands," in *IEEE/ACM international conference on Computer-aided design*, November 2002, pp. 195–202.

[9] J. Tschanz, S. Narendra, Y. Yibin, B. Bloechel, S. Borkar, and V. De, "Dynamic-sleep transistor and body bias for active leakage power control of microprocessors," in *IEEE International Solid-State Circuits Conference*, vol. 1, 2003, pp. 102–481.

[10] Q. Wu, M. Pedram, and X. Wu, "Clock-gating and its application to low power design of sequential circuits," in *IEEE Custom Integrated Circuits Conference*, May 1997, pp. 479–482.

[11] W. Kim, M. Gupta, G. Y. Wei, and D. Brooks, "System level analysis of fast, per-core DVFS using on-chip switching regulators," in *International symposium on high-performance computer architecture*, February 2008, pp. 123–134.

[12] U. Ogras, J. Hu, and R. Marculescu, "Key research problems in NoC design: a holistic perspective," in *3rd IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, February 2005, pp. 69–74.

[13] Y. Sun, S. Kumar, and A. Jantsch, "Simulation and evaluation of a network on chip architecture using Ns-2," in *IEEE NorChip Conference*, 2002.

[14] D. Whelihan and H. Schmit, "Nocsim: A NoC Simulator," January 2006, documentation for Nocsim NoC simulator.

[15] R. Thid, M. Millberg, and A. Jantsch, "Evaluating NoC communication backbones with simulation," in *IEEE NorChip Conference*, 2003, pp. 27–30.

[16] J. Xi and P. Zhong, "A Transaction-Level NoC Simulation Platform with Architecture-Level Dynamic and Leakage Energy Models," in *16th ACM Great Lakes Symposium on VLSI (GLSVLSI)*, 2006, pp. 341–344.

[17] D. Siguenza-Tortosa and J. Nurmi, "VHDL-based simulation environment for Proteo NoC," in *Seventh IEEE International High-Level Design Validation and Test Workshop*, October 2002, pp. 1–6.

[18] R. Marculescu, "Networks-on-chip: the quest for on-chip fault-tolerant communication," in *IEEE Computer*

Society Annual Symposium on VLSI, February 2003, pp. 8–12.

[19] A. Kahng, B. Li, L. Peh, and K. Samadi, "ORION 2.0: A Fast and Accurate NoC Power and Area Model for Early-Stage Design Space Exploration," in *Design Automation and Test in Europe (DATE)*, April 2009.

[20] T. Boutell, "GD Lib, a graphics library for fast creation of GIF images," [Online] Available: http://www.boutell.com/gd/.

[21] P. Gratz, B. Grot, and S. W. Keckler, "Regional congestion awareness for load balance in networks-on-chip," in *IEEE 14th International Symposium on High Performance Computer Architecture (HPCA)*, February 2008, pp. 203–214.

APPENDIX

A. Command line arguments

The following command-line arguments are available in the simulator.

-h, --help: displays usage information

-v, --version: displays version information

-n cycles: maximum number of simulation cycles (default = 10000000)

-config filename: input configuration file

-out filename: prefix for output file (and path)

-debug, -d: displays verbose debug info in the log

-debug_start cycle: verbose debug info display start at given cycle

-debug_stop cycle: verbose debug info display stops at given cycle

-parms, -p : creates simulator parameters file

-noparms: does not create simulator parameters file

-stats, -s: creates simulator statistics file

-nostats: does not create simulator statistics file

-logfile: enables simulator logging to file

-nologfile: disables simulator logging to file

-logout: enables simulator logging to stdout

-nologout: disables simulator logging to stdout

-logerr: enables simulator logging to stderr

-nologerr: disables simulator logging to stderr

--< name >: initializes simulator parameter $< name >$ with a value of 1

--< name >=< value >: initializes simulator parameter $< name >=< value >$

B. Configuration fields

Configuration parameters are broadly divided into global and local (node based) categories and are stored in two separate files. While global settings affect network-wide parameters, local settings determine per-node configurations.

1) Global configuration: Global configuration file specifies NoC-wide parameters such as packet generation mode, wire latency, switch for visualization module, percent injection bandwidth or number of warmup cycles. Valid values are presented in parentheses, with default parameters de-italicized.

General: used in controlling generic parameters of NoC and global parameters of routers.

- *inj_vc_count*: injector VC count (1)
- *ej_vc_count*: ejector VC count (1)
- *percent_inj_bw*: percent injection bandwidth (20)
- *hops_per_cycle*: number of hops per cycle (0)
- *wire_delay*: models per-hop link latency (0)
- *router_pipeline_lat*: models any additional router pipeline latency (0)
- *random_req_size*: request size in random mode. When enabled, all packets are sized randomly. (0)
- *bimodal_req_size*: request size in bimodal mode. When enabled, all packets are sized to one of two different possible lengths. (0)
- *bimodal_size1*: packet size 1 in bimodal mode (64)
- *bimodal_size2*: packet size 2 in bimodal mode (512)
- *selfsim_inj*: self similar injection switch. When enabled, random numbers used in packet injection rate calculations are read from the "selfsim_file" trace file rather being calculated via the psuedo random number generator. This allows the off-line generation of self-similar random numbers to drive the packet injection.
- *aggressive_vc_alloc*: switch for aggressive VC allocation (can be used only with deterministic routing functions) (0)
- *adaptive_1avail*: switch for ensuring winning out port has at least one VC available (0)
- *baseline_channel_width*: baseline channel width (128)

Regional Congestion Awareness (RCA): is an efficient method for port selection based on aggregated congestion information communicated by neighboring nodes [21]. Following configuration fields control this behavior.

- *extra_rca_delay*: models Regional Congestion Awareness (RCA) delay (0)
- *low_bw_rca*: switch for modeling serialized updates over multi-cycles for bandwidth limited RCA (0)
- *low_bw_rca_latency*: models serialization latency for low_bw_rca field (0)
- *0delay_cost_msg_update*: switch for updating cost message with zero delay (0)
- *same_cycle_local_cost*: switch for computing local congestion cost in same cycle. If local congestion is a function of grant signals which come late in cycle, then congestion cost should be computed in next cycle. On the other hand, if local congestion depends upon VC/crossbar requests, then local cost can be computed in same cycle. (0)
- *cost_multiplier_local*: weightage for local cost. Local costs at each node are scaled by this factor before computing total cost. (1)
- *cost_multiplier_remote*: weightage for remote cost. Remote costs arrived at by aggregating costs upstream are scaled by this factor before computing total cost. (1)
- *use_max_quadrant_cost*: switch for using maximum congestion as representative for a quadrant, by default performs average (0)
- *cost_precision*: cost precision (0)
- *dim_ave*: perform diminishing average with existing

value while updating history FIFO (0)

Visualization and Statistics: configures visualization module and also defines interval as well as verbosity with which statistics are reported.

- *vis_on*: visualisation switch (0)
- *vis_start*: time instant to start visualisation (MAX_VAL)
- *vis_stop*: time instant to stop visualisation (MAX_VAL)
- *vis_fifo_type*: buffer parameter selected for visualisation (*free_buff* | vc_alloc)
- *vis_link_type*: link parameter selected for visualisation (*xbar_gnts* | $used\_buff$ | vc_used | $xbar\_reqs$ | $xbar\_demand$ | xb_buff | $link\_util$ | pkt_delay)
- *stats_interval*: statistics print interval (1000)
- *node_stats*: switch to display per-node statistics (0)
- *node_bw_stats*: display switch for node by node offered bandwidth statistics (0)
- *chkpt_interval*: checkpoint interval (1)
- *incr_stats*: switch for printing incremental statistics (0)

Simulation and Trace File: contains various configuration fields for controlling simulator behavior as well as trace file handling.

- *flit_max*: maximum flit count (1)
- *midpoint_file*: fast forward to the middle of generator file (0)
- *seed*: unique seed (random)
- *netcfg_file*: location of node config file (")
- *tracefile_name*: trace file location (")
- *incr_chkpt*: switch for check pointing cost variables (0)
- *cost_reg_history*: limit on history depth for maintaining cost (1)
- *warmup_cycles*: number of warm up cycles not included in simulation log (0)
- *max_packets*: maximum number of packets allowed in simulation (0)
- *pkt_throttle*: throttle on packet size to avoid simulator hangups (5000)
- *selfsim_trace1*: self similar trace file 1 (")
- *selfsim_trace2*: self similar trace file 2 (")

2) Local configuration: Local configuration file specifies node-specific parameters such as source/destination info, mode of routing, connected ports, traffic pattern generation and selection as well as startup clock frequency/voltage level. Node-specific parameters follow a line starting with *node:* declaration and are applicable only for the node number specified. Following is a list of various local configuration fields supported. Valid values have been presented in parentheses with default value de-italicized.

- *node:* node_name declaring a new node with specified name, remaining lines till next node declaration or EOF to be treated in this node's context (")
- *coord*: node coordinates (x and y coordinates specified on two separate lines in that order)
- *is_source*: source switch (*true* | $false$)
- *is_destination*: destination switch (*true* | $false$)
- *src_type*: source traffic pattern (*rand* | $bitcomp$ | $transpose$ | $selfsim$ | $bitrev$ | $hotspot$ | $file$)
- *dst_type*: destination traffic pattern (*noblock*)
- *router_type*: type of router (*basic*)
- *xbar_type*: type of crossbar design (*fullcon*)
- *vc_alloc*: VC allocator type (*2level*)
- *xb_alloc*: crossbar allocator type (*2level* | $spec\_2level$ | bec_2level | bec_power_2level | bec_spec_2level)
- *clock*: clock frequency divider, divides the root clock frequency by specified value to arrive at clock frequency for each node (1)
- *volt*: voltage level (default Vdd of power model used)
- *rt_algo_type*: routing algorithm (*xydor* | $adaptive\_xy$ | $o1turn\_bec$ | $xydor\_bec$)
- *rt_sel_type*: routing selection function for choosing output port (bec | $o1turn$ | $last\_match$ | *first_match* | $first\_avail$ | no_turn | $random$ | $stat$ | $local$)
- *rt_cost_fn*: routing cost function in computing link cost (*local* | $free\_vc\_nohist$ | $buff\_nohist$ | $buff\_hist$)
- *rt_cost_reg*: routing cost function in computing cost of an output port (*free_vc_nohist* | $buff\_nohist$ | xb_demand)
- *rt_cost_mgr*: routing cost manager for aggregating and propagating congestion info ($none$ | *local* | $1D$ | $1domni$ | $fanin$ | $quadrant$)
- *vc_count*: VC count per port (4)
- *vc_classes*: number of priority classes in a VC (1)
- *que_depth*: FIFO queue depth per VC (2)
- *port_count*: number of ports (2)
- *port_dest*: named destination port (")

C. License and Copyright Notice