Fast Scalable FPGA-Based Network-on-Chip Simulation Models

Michael K. Papamichael
Computer Science Department
Carnegie Mellon University
Pittsburgh, PA, USA
Email: papamix@cs.cmu.edu

***Abstract*—This paper presents a set of two FPGA-based Network-on-Chip (NoC) simulation engines that composed the winning design of the 2011 MEMOCODE Design Contest in the absolute performance class. Both simulation engines were developed in Bluespec System Verilog (BSV) and were implemented on a Xilinx ML605 FPGA development board. For smaller networks and simpler router configurations a direct-mapped approach was employed, where the network to be simulated was directly implemented on the FPGA. For larger networks, where a direct-mapped approach is not feasible due to FPGA resource limitations, a virtualized time-multiplexed approach was used. Compared to the provided software reference implementation, our direct-mapped approach achieves three orders of magnitude speedup, while our virtualized time-multiplexed approach achieves one to two orders of magnitude speedup, depending on the network and router configuration.**

***Keywords*-Network; Network-on-Chip; Simulation; Time-multiplexing; Virtualization; FPGA;**

I. Introduction

The objective of the 2011 MEMOCODE Hardware/Software Codesign Contest was to build the fastest simulator for a class of simple Networks-on-Chip (NoCs) that precisely replicates the cycle-by-cycle behavior of a given software reference simulator. Our FPGA-based submission won the Absolute Performance category providing up to three orders of magnitude speedup over the software reference design on a Xilinx ML605 FPGA development board.

The contest reference design supported a large number of design parameters, which led to a very large design space consisting of different router configurations, network topologies and traffic patterns. To effectively cover this vast design space and at the same time stay within the resource limitations of our FPGA development platform, we implemented two network simulation designs: i) a high-performance direct-mapped design that laid out the entire simulated target network on the FPGA and ii) a virtualized time-multiplexed design used to efficiently simulate larger network configurations that would not fit using the direct-mapped approach.

This paper describes our contest submission and is organized as follows. Section II describes the problem in more detail and Section III outlines the design principles we adhered to when developing our contest submission. Section IV provides a high-level overview of our NoC simulator, while Section V discusses the architecture and implementation of the two developed NoC simulation engines. Finally, we present implementation results in Section VI and conclude with a discussion on related and future work in Section VII.

II. Problem Description

This year's MEMOCODE Design Contest called for implementing a fast parameterized network simulator that models a wide range of simple NoCs. Each NoC instance consists of a collection of routers that can be arbitrarily connected through bidirectional links that carry data and credits for flow control. Data links have a single-cycle latency, while flow-control links may either have single-cycle or multi-cycle latency. Routers need to support multiple virtual channels (VCs) and employ a fixed-priority allocator for packet scheduling that is invoked every cycle. Packets can be single- or multi-flit and are generated by traffic sources that are attached to each router. Multi-flit packets are allowed to lock resources as they traverse the network, which requires extra book-keeping at each router and complicates the allocator implementation.

The simulator takes two inputs: i) a NoC configuration that specifies the parameters of the simulated target network and ii) a traffic pattern. The main parameters specified in the NoC configuration file are:

- Network topology (list of links between routers)
- Number of routers (up to 256)
- Number of input and output ports per router (up to 16)
- Number of virtual channels per router (up to 8)
- Credit delay cycles (up to 16)

Each traffic pattern input specifies the following information for each router:

- Routing information (output port for each destination)
- Number of packets to send (up to 1024)
- Individual packet information (number of flits and VC)

The goal of the contest was to build the fastest NoC simulator that precisely replicates the behavior of a provided reference software implementation. During validation contestants were given 24 hours to produce NoC simulator instances for a set of five NoC configuration inputs. A set of traffic patterns was afterwards used to measure the performance and verify the correctness of each design against the

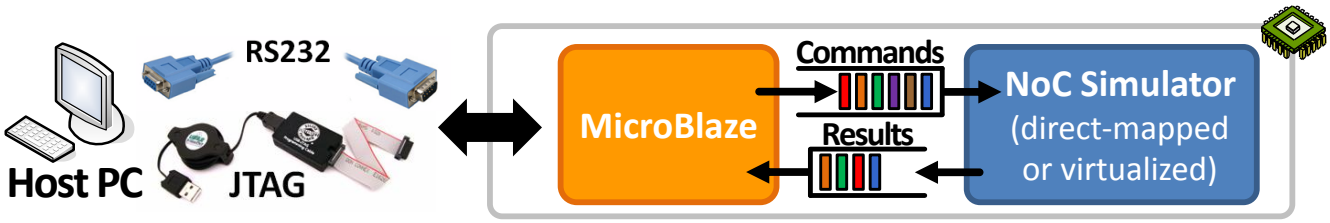

Figure 1. High-level block diagram of platform consisting of Host PC connected to a Xilinx ML605 Development Board.

reference simulator. For the full contest description please refer to the official contest document [1].

III. Design Principles

Given the strict contest deadline and the short implementation window we adopted a set of design principles to spend the available time as efficiently as possible.

Correctness First. Instead of directly implementing a highly optimized design we split our design time into two distinct phases, a correctness and an optimization phase. During the first phase we only focused on producing a correct design and ignored performance or FPGA resource utilization issues. Correctness was ensured through the use of python scripts that compared the output of an instrumented version of the reference simulator against simulation output for individual modules, as well as the entire system. Only once the entire system was fully validated, did we start optimizing each component. During each step of the optimization phase we diligently reran our validation tests.

Parameterization and Modularity. Instead of incrementally adding support for the various network and router parameters, which would potentially require revisiting each module multiple times, we directly implemented parameterized versions of all the modules in the simulator utilizing Bluespec's [2] powerful parameterization mechanisms. In addition to the parameterization, we tried to carefully develop standard interfaces for each module early in the design phase to promote modularity and allow for easier localized optimizations within each module.

Harnessing the power of Bluespec. Overall the use of Bluespec System Verilog greatly accelerated both the design and verification time. Bluespec's static elaboration mechanisms allowed us to quickly design parameterized modules and define clean module interfaces. Even when substantial interface changes were required later in the design cycle, we were able to rely on Bluespec's powerful type checking system to quickly identify all of the affected code regions that required modification and eliminate potential bugs.

IV. Design Overview

Figure 1 shows a high-level block diagram of the platform we used, which consists of a host PC that has JTAG and serial (RS232) connections to a Xilinx ML605 development board [3]. The FPGA on the ML605 hosts the NoC simulation engine and a MicroBlaze processor. Both the direct-mapped and virtualized implementations of the NoC simulator expose a common FIFO-based interface for accepting initialization commands from and streaming out simulation results to the MicroBlaze. Since the MicroBlaze and NoC simulator might run at different clock frequencies, the FIFOs between them are asynchronous to allow crossing between the two different clock domains.

Running Simulations. To setup the FPGA for a given NoC configuration, the Bluespec compiler is invoked to generate the Verilog code for the set of parameters specified in the NoC configuration. The produced Verilog code is then fed to the Xilinx XST synthesis tool and the resulting netlist is then connected to the MicroBlaze processor as a peripheral on the PLB bus [4]. Once the FPGA is configured, scripts are used to convert each traffic pattern to MicroBlaze code that will initialize the traffic tables for each router in the network along with other simulation parameters. Since the traffic tables are allowed to contain hundreds of thousands of entries, the initialization data can grow very large and is thus stored in off-chip DRAM.

Once the MicroBlaze has initialized all of the traffic and routing tables through the Commands FIFO, a final command is sent that triggers the traffic sources and starts the simulation. The MicroBlaze then starts polling the Results FIFO until the NoC simulator detects that the simulation has terminated – either because the traffic is done or because the maximum number of cycles has elapsed – at which point the number of simulated cycles along with other statistics are enqueued in the Results FIFO and then printed by the MicroBlaze through the serial port.

V. Architecture and Implementation

In order to efficiently cover the vast design space of different possible NoCs and stay within the resource limits of the ML605 FPGA platform, our contest submission consists of two separate NoC simulation engines: i) A high-performance direct-mapped simulation engine that supports up to moderately sized networks (~100 routers) with medium-complexity routers (e.g. 5 ports w/ 4VCs) and ii) a highly scalable virtualized simulation engine that can handle the entire design space – including the largest network/router

configurations – by time-multiplexing all target routers in the simulated NoC using a single virtualized router.

In the direct-mapped approach, the actual simulated target network is implemented on the FPGA. In other words, a fully functional prototype of the target network, including all individual routers, links, traffic sources, etc, is laid out in its entirety on the FPGA. The benefit of such an approach is that all routers in the network are simulated in parallel, thus yielding very high speedup compared to the reference software simulator. Moreover in our specific implementation there is a one-to-one mapping between host and target[1] cycles, i.e. each cycle on the FPGA corresponds to a cycle in the simulated network. In hindsight, we discovered that an additional benefit of having a simulator implementation that closely mimics the actual network is that the simulator can be easily converted to a general-purpose network with minimal changes to the source code.

Even though a direct-mapped approach can provide very high performance, it is not able to fit all possible NoC configurations on the ML605. To fill this gap and support the remaining NoC instances that would not fit on the direct-mapped engine, we also developed a virtualized time-multiplexed NoC simulation engine, that uses the FPGA resources in a very efficient manner. In this approach, instead of simulating all routers at the same time, each router is simulated in successive FPGA cycles. Even on a moderately-sized FPGA board, like the the ML605, such a design can easily scale to support even the largest allowed network configurations, consisting of hundreds of high-radix routers with many VCs. However, compared to a direct-mapped approach, a time-multiplexed implementation has to deal with additional complications, which are discussed later in this section.

A. Direct-Mapped Implementation.

In a direct-mapped implementation the network is build as a collection of router instances that are connected according to each NoC configuration. Figure 2 shows the architectural block diagram of a single such router. Each router module receives flits through a set of input ports and sends flits through a set of output ports. The first input port of each router is connected to a traffic source that injects packets according to a traffic pattern table that is populated during initialization. Similarly, the first output port is connected to a traffic sink that drains packets once they have reached their destination. The remaining input and output ports of each router are either used to create links with other routers in the network or may remain unconnected. For each flit link connecting two neighboring routers there is a corresponding credit link going in the opposite direction for flow-control.

[1]Throughout the rest of this paper the term host will be used to refer to the system on which the network simulator is executed and the term target will be used to refer to the network that is being simulated.

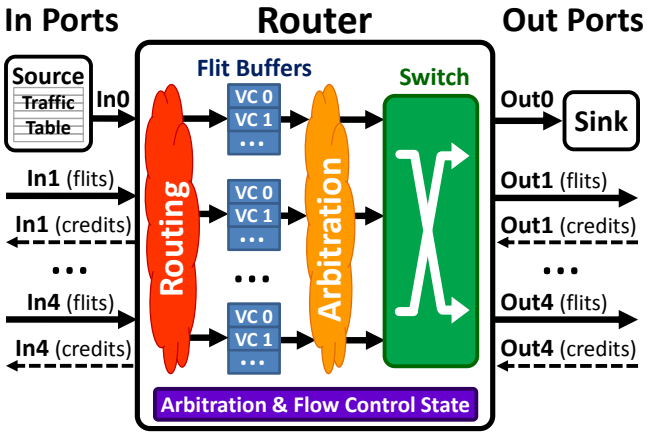

Figure 2. Architectural block diagram for direct-mapped router.

During each clock cycle a router receives and stores new flits from its input ports and forwards previously received flits through its output ports. Each incoming flit is first processed by routing logic to determine the proper output port it needs to be forwarded to and is then stored in single-entry flit buffers according to the virtual channel that it belongs to. To determine which flits will be scheduled to depart from the router, arbitration logic considers flit buffer occupancy and credit availability to decide which flits will traverse the switch and be forwarded through the output ports. In addition to scheduling flits, the arbitration logic is also responsible for respecting VC and port priorities, as well as preventing flits from different multi-flit packets from being interleaved on the same virtual channel.

Termination Conditions. Additional logic monitors the system to detect when the simulation has finished, which can either happen because the maximum number of allowed cycles has been reached or because all of the packets and credits in the network have been delivered. The first case simply requires logic that compares the current simulation cycle against the maximum number of allowed cycles that was set during initialization. To deal with the second case the simulator each cycle constantly monitors all flit and credit links – including all intermediate links in the presence of additional credit delay – for activity. If all links are found to be idle, the simulation is terminated and the results are printed.

B. Virtualized Implementation.

To minimize FPGA resource usage, our virtualized implementation of the NoC simulator performs time-multiplexing using a single virtualized router that simulates a different target router during each FPGA cycle. Figure 3 shows a block diagram of the implemented virtualized simulator, at the heart of which is the Virtualized Router module. In terms of functionality this module is very similar to the direct-mapped router, discussed earlier, with the exception that it

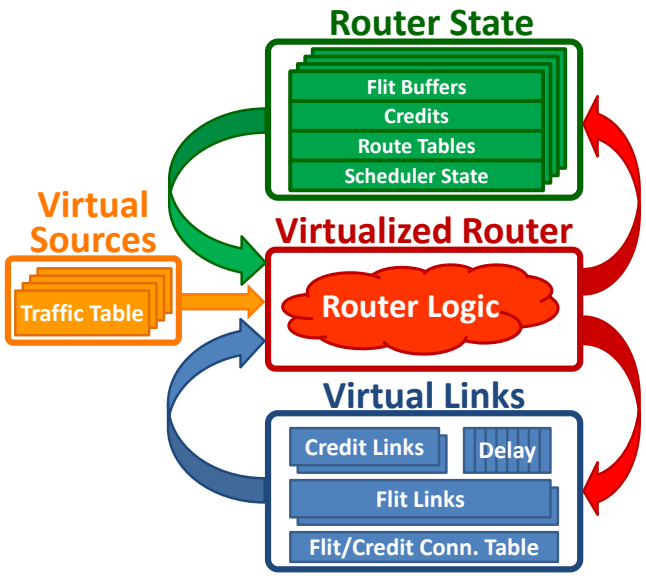

Figure 3. Architectural block diagram for time-multiplexed simulation engine.

only consists of combinational logic; it does not store any state, but is only used to transform state. Each cycle, it receives the current state of a router in the system along with the incoming flits and credits destined to this router. Based on this information it generates the new router state and sends any outgoing flits and credits. Once all routers in the simulated system have been processed by the Virtualized Router, the simulation proceeds to the next target cycle.

The remaining modules of the virtualized simulator are used to maintain the state of the various routers in the network and to facilitate flit and credit transfers. In particular, the Router State module stores the state for all of the routers in the simulated system, such as flit buffers, credits, route tables and other scheduling information. Since in the virtualized implementation only a single router is active at any given cycle, the Router State only requires using a single-port memory; in contrast, the direct-mapped implementation requires using multiple memories, because all routers need to access their state every cycle. In addition to requiring fewer memory ports, due to the discretization of on-chip FPGA memory, the virtualized implementation also makes more efficient use of on-chip memory, because it consolidates and stores all of the routers' state in a single monolithic memory.

The Virtual Sources and Virtual Links modules are responsible for injecting packets into the network and moving flits or credits between routers. Since only a single traffic source is active at any given cycle, the Virtual Sources module employs the same time-multiplexing techniques as the Virtualized Router module, thus making more efficient use of FPGA resources. The Virtual Links module manages a set of buffers that store flits and credits until it is time to deliver them to one of the simulated routers. Connection tables, that are specific to each NoC configuration and are initialized at the beginning of each simulation, hold information about the NoC topology and determine the proper set of buffers to be used by the currently simulated router.

Time-multiplexing and Network Simulation. As has been shown in previous work [5], [6], time-multiplexing in the context of network simulation requires special care to retain proper ordering of events and careful state management to ensure that all routers in the network have a consistent view of the system. For instance, within a single target cycle, a router might send traffic to routers that were simulated in previous host cycles, but also send traffic to routers that will be simulated in subsequent host cycles. To avoid ordering violations and properly simulate the concurrent exchange of flits between all routers, the simulator needs to isolate events that belong to different target clock cycles.

To deal with such issues our virtualized implementation employs double-buffering for all flit and credit links in the network. During each target clock cycle, incoming flits and credits are read from one set buffers and outgoing flits and credits are stored in a different set of buffers. In the next target clock cycle these two buffers are swapped to allow the network to make progress, but also ensure that events from different target clock cycles are isolated from each other. If the simulated NoC specifies extra credit delay, an additional set of delay buffers is introduced before the existing double buffers that handle credits.

Termination Conditions. As was the case with the direct-mapped implementation, additional logic is required to detect when the simulation is finished. However, the time-multiplexed design requires slighly different logic, because only a subset of the flit and credit links are exposed each cycle. Thus instead of monitoring all links each cycle, the virtualized design keeps track of the number of idle target cycles. If the number of idle target cycles exceeds the maximum link delay in the network, then the simulation has finished. For networks that specify additional credit delay, the number of idle cycles, after which the simulation is considered to be finished, needs to be adjusted to ensure that no traffic is still active at any of the intermediate credit links.

Critical Path. The router implementation in both the direct-mapped and the virtualized approach are single-cycle designs, both to mimic the behavior of the reference simulator and to also minimize FPGA resource usage. This leads to a long critical path, which is dominated by the arbitration logic that takes care of assigning output ports to different inputs and VCs, while at the same time respecting the scheduling rules that apply to multi-flit packets.

The fixed-priority arbiter employed in the reference design considers all VCs, inputs and outputs in succession, which inevitably creates a very long combinational chain when implemented in hardware, that grows proportionally to the number of inputs, outputs and VCs. Since all the

networks specified in the contest need to support single-flit packets, the arbitration logic cannot be pipelined, because, in the worst case, a new arbitration decision needs to be made every cycle. The effect of this long combinational path is reflected in the synthesis results presented in the next section.

VI. Results

To first get a sense of how the two presented NoC simulator implementations scale in terms of FPGA resource usage and clock frequency, we present FPGA synthesis results for both the direct-mapped and virtualized simulator implementations. We then show more detailed results for the five specific networks that were used in the contest validation. TODO: Finally we present with a brief case study that looks at one of these five networks in more depth.

Direct-mapped Implementation Results. As mentioned earlier, the direct-mapped implementation of our NoC simulator is a collection of interconnected router modules. Table I shows FPGA resource usage and clock frequency synthesis results for different router configurations targetting a Xilinx Virtex-6 LX760T FPGA. All reported results are for a single router within a 256-node network, the largest network allowed in the contest. As expected, increasing the number of router ports and VCs leads to higher LUT counts and negatively impacts clock frequency.

	LUTs / Clock Frequency (in MHz)		
Router Config.	**2 VCs**	**4 VCs**	**8 VCs**
4 in/out ports	785 / 152	1393 / 101	2848 / 59
8 in/out ports	3243 / 81	6134 / 54	12754 / 33
12 in/out ports	7717 / 62	11596 / 36	19198 / 20
16 in/out ports	11655 / 45	28294 / 30	33689 / 14

Table I
SYNTHESIS RESULTS FOR SINGLE ROUTER IN DIRECT-MAPPED DESIGN.

Virtualized Implementation Results. Table II shows synthesis results for different router configurations using our virtualized implementation of the simulator to model a 256-node network. Since the virtualized implementation relies only on a single instance of a time-multiplexed router to model the entire network, the presented results are for the entire network and not for an individual router, as was the case for the direct-mapped implementation results. To get a better feeling of the scalability differences between the two designs, note that the cost of four routers in the direct-mapped implementation is comparable to the cost of the entire 256-node network in the virtualized design. In fact, the largest allowed contest router and network configuration only occupies 13% of a Xilinx LX760T FPGA.

Results For Contest Networks. To validate and compare the performance of different contest submissions, a set of five network and router configurations were provided by the contest organizer, which are listed in Table III. Note that all network configurations lie on the "edge" of the design space as they all max-out at least one of the configuration parameters. Moreover three of the five given networks are very large, consisting of more than 250 routers.

	LUTs / Clock Frequency (in MHz)		
Router Config.	**2 VCs**	**4 VCs**	**8 VCs**
4 in/out ports	3050 / 66	4117 / 56	6346 / 34
8 in/out ports	7912 / 35	11833 / 28	28859 / 17
12 in/out ports	13653 / 30	28461 / 16	48081 / 10
16 in/out ports	30399 / 17	52288 / 12	101500 / 7

Table II
SYNTHESIS RESULTS FOR ENTIRE NETWORK IN VIRTUALIZED DESIGN.

Network Name	**Routers**	**Ports/Router**	**VCs**	**Credit Delay**
butterfly	112	3	8	1
highradix	16	16	8	15
mesh	253	5	4	3
torus	252	7	5	2
hypercube	256	9	1	1

Table III
CONFIGURATION OF CONTEST NETWORKS.

Table IV shows actual implementation results for the five contest network configurations running on the ML605 board that is built around the Xilinx Virtex-6 LX240T FPGA. To give a sense of how our simulator would perform on a larger FPGA, we include synthesis results for a larger Xilinx Virtex-6 LX760T FPGA. For each network and FPGA we also indicate the chosen simulation engine – direct-mapped (DM) or virtualized (V) – and report the average speedup compared to the reference software simulator running on an Intel Xeon X3460 processor at 2.8GHz.

Due to the large size of the contest networks and the limited resources on the LX240T FPGA, we were only able to map one of the five networks (butterfly) to our fast direct-mapped engine, in which case we achieved three orders of magnitude speedup over the software reference design. For the remaining networks, which were implemented using the virtualized approach, speedup values range from 5x to 30x, leading to an overall average speedup of 470x. However, when using the larger LX760T FPGA, three out of the five contest networks can be implemented using the high performance direct-mapped approach, leading to significant speedup improvements; overall average speedup in that case is $\sim$1570x.

Deterministic Performance. A nice property of the two developed NoC simulation engines is that performance is fully deterministic for any given network and router configuration. In a system with N routers running at $Freq$ frequency the performance of the simulator, measured in

simulated target cycles per second, is equal to $Freq$ for a direct-mapped implementation and $Freq/N$ in the case of a virtualized implementation.

	Xilinx LX240T			Xilinx LX760T		
Network	**DM/V**	**%LUTs**	**Speedup**	**DM/V**	**%LUTs**	**Speedup**
butterfly	DM	86%	1511	DM	27%	2330
highradix	V	63%	5	DM	93%	421
mesh	V	3%	28	DM	96%	4281
torus	V	8%	786[2]	V	2%	789[2]
hypercube	V	8%	21	V	2%	33

Table IV
IMPLEMENTATION RESULTS FOR CONTEST NETWORKS.

VII. DISCUSSION

Multiple Virtualized Routers. Even though our virtualized simulation engine can scale to very large network and router configurations, this scalability comes at the cost of lower performance compared to the direct-mapped approach. To bridge this gap, one idea for future work is to use multiple virtualized routers that run concurrently. To maintain proper event ordering in such a setting, the system needs to ensure that only independent (i.e. not neighboring) sets of routers are simulated at the same time. This issue has been studied in previous work [5] and a straightforward way to resolve it would be through a separate preprocessing step that identifies independent sets of routers in the network and then generates a fixed valid simulation schedule.

An Alternative Approach to FPGA-based NoC simulation. Another interesting approach to FPGA-friendly NoC simulation is FIST [7], a simulation technique previously explored by our group that abstractly models each router as a set of load-delay curves, which are obtained through training using a software-based cycle-accurate NoC simulator. In addition to high simulation speed and scalability, an important benefit of such an approach is reduced implementation complexity. In contrast to the two NoC simulation approaches presented in this paper, FIST does not require implementing the actual router in hardware; instead it relies on the presence of a software-based model that will be used for training purposes.

Automatic Network Generation. Given that the direct-mapped design is already fully parameterized and essentially builds a working prototype of the target network on the FPGA, an interesting extension to this work would be building a flexible configurable NoC generator. Such a tool could prove useful to FPGA designers that need an FPGA-friendly NoC that is custom-built to meet the specific needs of their application. In fact, a heavily modified version of the direct-mapped NoC code base is currently used as the interconnect within the CoRAM project [8].

[2]During validation, one of the supplied traffic patterns deadlocked the torus network. In such cases, even though the network is stuck, the software reference simulator needlessly continues simulation until it reaches the maximum number of simulation cycles. Our implementation, however, can detect a deadlock in the network, in which case it immediately terminates the simulation and prints the results. Since deadlock occured early in the particular simulation our implementation was able to achieve a very high speedup for that traffic pattern. This also explains why the torus network achieves a speedup that is comparable to the direct-mapped engine, even though it is using the virtualized engine.

VIII. ACKNOWLEDGMENTS

We thank Prof. James C. Hoe, Eric Chung, Gabe Weisz and the rest of the members of the Computer Architecture Lab at Carnegie Mellon (CALCM) for the helpful discussions and comments. We thank Xilinx for their FPGA and tool donations. We thank Bluespec for their tool donations and support.

REFERENCES

[1] D. Chiou, "MEMOCODE 2011 Hardware/Software CoDesign Contest", https://ramp.ece.utexas.edu/redmine/attachments/25/MEMOCODE2011_DesignContest.pdf

[2] Bluespec Inc, http://www.bluespec.com

[3] Xilinx, "ML605 Hardware User Guide", http://www.xilinx.com/support/documentation/boards_and_kits/ug534.pdf

[4] Xilinx, "LogiCORE IP Processor Local Bus (PLB) v4.6", http://www.xilinx.com/support/documentation/ip_documentation/plb_v46.pdf

[5] M. Pellauer, M. Adler, M. Kinsy, A. Parashar, and J. Emer, "HAsim: FPGA-Based High-Detail Multicore Simulation Using Time-Division Multiplexing", HPCA, 2011

[6] P. Wolkotte, P. Holzenspies, and G. Smit, "Fast, Accurate and Detailed NoC Simulations", NOCS, 2007

[7] M. K. Papamichael, J. C. Hoe, and O. Mutlu, "FIST: A Fast, Lightweight, FPGA-Friendly Packet Latency Estimator for NoC Modeling in Full-System Simulations", NOCS, 2011

[8] E. Chung, J. C. Hoe, and K. Mai, "CoRAM: An In-Fabric Memory Abstraction for FPGA-based Computing", FPGA, 2011