System-Level Modeling Environment: MLDesigner

Ankur Agarwal [1], Cyril-Daniel Iskander [2], Ravi Shankar [1], Georgiana Hamza-Lup [1]
ankur@cse.fau.edu, cyril_iskander@hotmail.com, ravi@cse.fau.edu, ghamzal@fau.edu
[1]Dept. of Computer Science and Engineering, Florida Atlantic University, Boca Raton, FL, 33431
[2]Hi-Tek Multisystems, Québec, QC, Canada

***Abstract* – System modeling has the potential to enhance system design productivity by providing a platform for system performance evaluations. This model must be designed at an abstract level, hiding system details. However, it must represent any subsystem or its components at any level of specification details as needed. In order to model such a system, we will need to combine various models-of-computation (MOC). MOCs provide a framework to model various algorithms and activities, while accounting for and exploiting concurrency and synchronization aspects. Along with supporting various MOCs, a modeling environment should also support a well developed library. In this paper, we have explored and compared various system modeling environments. MLDesigner is one such modeling environment that supports a well developed library and integrates various MOCs. We discuss the process of system modeling with MLDesigner. We further present an abstract model of a Network-on-Chip (NoC) in MLDesigner and show latency results for various customizable parameters for this model.**

***Keywords* – Models of Computation, system modeling, design methodology, MLDesigner.**

I. INTRODUCTION

A system model developed at an abstract level (in the system modeling phase) can save the re-design time and design re-spins due to unsatisfied requirements and specifications in the design/development phase [1]. With new platforms, system modeling becomes more important as we support more applications. We want to analyze system-level parameters such as cache, number of processors, bus or network architecture, system power consumption, system resources, among others, to determine the performance of the system in terms of each one of these parameters. We must properly design the system before we can analyze the system performance parameters and Quality-of-Service (QoS) metrics.

With a proper system-level design tool, we should be able to model a system at an abstract level, hiding internal system details. At the same time, we should be able to specify the sub-system or components at any level of complexity. This allows us to hide the system details, making the behavior the important aspect of the system. Such a system-level design tool has to address all these characteristics. It should also be able to simulate a system for different system-level performance parameters and QoS metrics. We should be able to control the parameters of the system components from the system itself. As an example, let the resolution in a digital camera be a system-level concept. This resolution can be in turn a local concept translated into the number of pixels, bitstream size, decoding capability of the decoder, among others. So we should be able to control these component parameters by selecting the value of the resolution. If we change the resolution of a system, all other components are properly modified. For designing such a system we integrate different MOCs onto the system architecture. A MOC is a mathematical formalism that captures and allows us to reason about a computational system or concept independent of its implementation details. Different MOCs have evolved to represent the reasoning in different areas of focus. MOCs provide a framework to model various algorithms and activities, while accounting for and exploiting concurrency and synchronization aspects. Thus, several MOCs are needed for modeling an integrated system. In such a system each component or the subsystem of the system should be able to use any allowed MOC and more importantly should retain its behavior after integration of the system. Along with supporting various MOCs, a modeling environment should also support a well developed library.

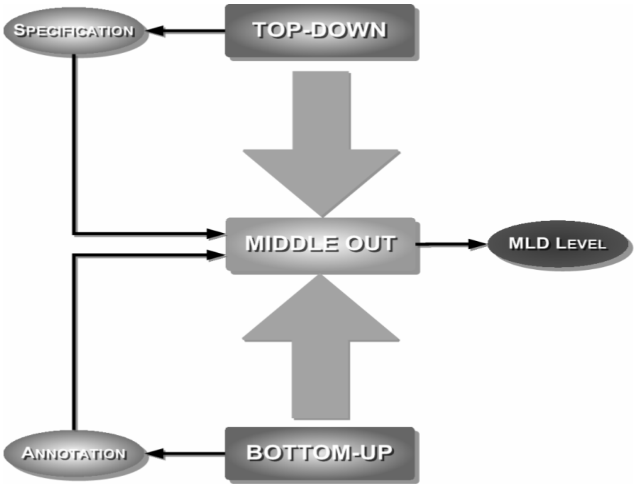

Fig 1: Middle-out design philosophy.

A modeling environment should further be able to integrate top-down and bottom-up design philosophies to provide a middle-out design methodology for modeling systems (Fig. 1). Usually, we use a top-down or bottom-up design methodology to model a system. In a top-down design methodology we analyze the system specifications and partition a system into its subsystems based on system specifications [2]. Similarly, a subsystem is then partitioned into components based on subsystem specifications. We start modeling a system and add subsystems to it whenever needed. Similarly, while modeling a subsystem, components are discovered and added as per the requirements. On the other hand, in a bottom-up approach, we know exactly the details of system functionality and each of its subsystems and components. We model components, and then combine these components to make a subsystem and finally, we combine subsystems to make a system. In practice, we need both design methodologies. We can annotate performance parameters from a bottom-up design methodology and represent these parameters in a system model. These parameters will help us carry out system performance evaluations. At the same time, a top-down methodology allows us to model a system in an abstract manner, hiding internal system details. Thus, it provides an advantage in terms of time-to-market. This combination of top-down and bottom-up is known as middle-out philosophy.

This defines the objectives of this paper. We would like to explore a modeling environment which can model a system at an abstract level by integrating various models of computation. We compare various system modeling environments such as POLIS [3], COSYMA [4], RSIM [5], Ptolemy [6], and MLDesigner [7]. MLDesigner is a system-level design tool that supports all of these features. MLDesigner also has a well-supported library and integrates various models of computation. Since with MLDesigner we can design a system at an abstract level, we can do performance evaluation for the different system parameters and QoS metrics. Consequently, product development time and cost can be significantly reduced. MLDesigner further uses the middle-out design philosophy (Fig. 1), and hence has the potential to represent the system at an abstract level and allow performance evaluations at the same time. In this system-level design tool, we would like to model a Network-on-Chip (NoC) architecture as a case study, where system performance is analyzed based on different configuration aspects of this architecture.

II. SYSTEM MODELING ENVIRONMENTS

In this section, we discuss and compare various system modeling environments.

2.1. POLIS

POLIS is a software program for hardware/software co-design of embedded systems, developed at the University of California, Berkeley [3]. It provides a framework for specification, automatic synthesis, and validation of control-based embedded systems. POLIS uses a Co-design Finite State Machine (CFSM). Compared to the classical FSM model which uses synchronous communication, the CFSM model has a finite unbounded reaction time. This CFSM model of computation can be described as Globally Asynchronous, Locally Synchronous (GALS). The POLIS design flow includes the following tasks: high-level language translation (in which specifications written in high-level languages are translated into CFSM's), formal verification, system co-simulation, design partitioning, hardware synthesis, software synthesis, and the interfacing of implementation domains.

2.2. COSYMA

COSYMA (COSYnthesis for eMbedded micro Architectures) is a platform for exploration of the co-synthesis process. It is an academic tool developed at the Technical University of Braunschweig, Germany [4], [8]. It is an experimental system for HW/SW co-design of small embedded real-time systems. The major aim of COSYMA is the automated HW/SW partitioning process. The input description consists of a function description in C^x, which is a minimum extension of the C programming language.

2.3. RSIM

The Rice Simulator for Instruction Level Parallelism (ILP) Multiprocessors, (RSIM), is an academic tool developed at the Rice University. The purpose of RSIM is intended to model architectures which involve shared memory systems built from processors which exploit ILP. [5]. RSIM is implemented in a modular fashion, and is written using C++ and C for extensibility and portability.

2.4 Ptolemy

Ptolemy is a software framework developed by the Department of Electrical Engineering and Computer Sciences of the University of California. Its application is towards modeling, simulation, and design of concurrent, real-time, embedded systems with focus on assembly concurrent components [6].

The core of Ptolemy is a software infrastructure called the Ptolemy *kernel*, which is made up by of a set of C++ classes and functions definitions used by the rest of the Ptolemy. This kernel doesn't implement any particular model of computation: a model of computation is defined instead by a *domain*, which itself is defined by creating new C++ classes derived from the base classes in the Ptolemy kernel. The goal of a domain is to define the semantics of the model, and not to define how the computations are performed. The latter can be performed using different languages, such as C++, C, MATLAB, and VHDL.

Table1. Characteristics of the system-level design tools.

Tool	**POLIS** UC, Berkeley	**COSYMA** Braunschweig University (Germany)	**RSIM** Rice University	**Ptolemy** UC, Berkeley	**MLDesigner** MLD Tech., California
System specification	ESTEREL, Graphical FSM, VHDL, Verilog	C^x (extension of C)	C/C++	SpecCharts, C/C++, Java	C/C++
Application type	Control intensive reactive real systems	Complex systems	Multiprocessor systems	Control and communication systems	Complex systems
Design approach	Formal verification Co-simulation HW/SW synthesis	Migration of SW to HW using simulation and profiling	Performance simulation of ILP in multiprocessor systems	HW/SW partitioning refinement, implementation	HW/SW partitioning refinement, implementation, co-simulation, HW/SW synthesis
Target architecture	Micro-controller architectures	CPU+ASIC with a bus and a memory	Multiprocessor systems with ILP	Multiprocessor systems with ASICs	Multiprocessor systems with ASICs
MOCs supported	FSM, CFSM, DE	CSP, SDF, DE	DE	Most MOCs	Most MOCs
Component library support	No component library	No component library	No component library	Extensive for multiple MOCs	Extensive for multiple MOCs and multiple applications
Memory requirements	Not defined	Not defined	High memory needs, difficulty to free memory (must be implemented in program)	High memory requirements,	Lower memory requirements as compared to PtolemyC
Simulation speed	Fast	Fast	Moderate	Slow	Faster than PtolemyC

2.4 MLDesigner

MLDesigner is a system-level modeling environment. [7]. It allows one to model a system at an abstract level. It supports modeling in different domains such as Discrete Events (DE), Synchronous Data Flow (SDF), Finite State Machine (FSM), Dynamic Data Flow (DDF), and Synchronous Reactive (SR) among others. We can further combine multiple domains to represent a system model. Hence, we can represent a system model with any level of details in any part of it. One can do performance analysis on a developed system model. As this performance analysis is done at an abstract level, therefore it is optimized in running the simulation faster as compared to other modeling environments, which are C, C++ or SystemC-based. MLDesigner also allows modeling of applications related to different domains such as wireless applications, computer architecture, network design, embedded system design, and distributed system processing, among others. System models are implemented in MLDesigner through a graphical editor or Ptolemy Tool command language, PTcl. The functionality of the modules may be specified by a hierarchical block diagram, a finite state machine, a module defined in C/C++ language, or by a PTcl module definition. A primitive is the lowest level model in the MLDesigner hierarchy. A primitive can also be seen as a component, a building block of MLDesigner models. Its functionality is defined using the Ptolemy language containing C++ code fragments. The model part of a primitive defines its external interface, whereas the primitive source code written in Ptolemy defines the behavior of the primitive. FSM primitives are a special type of a primitive model. In contrast to normal primitives, the functional model of an FSM primitive is described by the FSM model. A module is a model made up of primitives or other modules. A module can be a simple structure with one or multiple levels of a hierarchical structure. In other words, a module in MLDesigner is a collection of primitives and/or other modules interconnected together as a sub-system. Like primitives, modules can have input/output ports as well as arguments for interfacing. A system is the top-level model and consists of a number of primitive or module instances. A system model does not have any input/output ports and cannot be instantiated in other models. One can define design parameters in the system, module or a primitive. These parameters can then be exported at a higher level (to the system) and can be controlled by system parameters. Thus, it also provides a concept of global optimization of a system model. Defined parameters are used to parameterize the system model. Each block in this architectural hierarchy communicates using Particles. A Particle is a package that contains data. There are eleven key data types defined to be used in MLDesigner. There are four numeric scalar types – complex, fixed-point, double-precision floating-point, and integer. Each of the scalar numeric types has an equivalent matrix type. There are strings and file references, and also a user-defined type, called *Message*. A detailed tutorial for MLDesigner system modeling is given in [10].

III. COMPARISION OF SYSTEM-LEVEL MODELING ENVIRONMENTS

In this section, important characteristics of the studied system-level design tools are compared against each other. Table 1 shows the primary characteristics for all studied system-level design tools.

The modeling methodology is very similar among all tools. POLIS is the only one that diverges from a C/C++ based modeling methodology using Verilog, VHDL and ESTEREL. Verilog and VHDL require a very detailed model by which only hardware can be specified, and the software aspects cannot be specified using these modeling languages. For this reason, POLIS uses ESTEREL to do the software specification. Furthermore, in order to develop a modeled system in VHDL and Verilog, we should have completed the design phase, and thus, it would defeat the core purpose of system modeling – performance evaluation at the design phase. Hence this system-level modeling environment may not be suitable for modeling embedded systems for abstract system-level (at a very high level) performance analysis and evaluation.

RSIM models architectures involving shared memory systems which are built from processor micro-architecture that aggressively exploit ILP. The performance analysis and evaluation in this system-level modeling environment is limited to ILP for multiprocessor architecture. Instruction caches are not simulated and are assumed to be perfect. RSIM does not support virtual memory. A processor's accesses to its private data space are considered to be cache hits in all multiprocessor simulations and in uniprocessor simulations configured for this purpose. Therefore, while resource contention and instruction scheduling for private accesses are simulated, the actual cache behavior is not. Therefore, RSIM does not provide performance evaluations for other key system level parameters, such as cache performance, HW/SW profiling and partitioning. Hence, this system-level modeling environment is limited in its scope of applications.

COSYMA is not deficient in these two previous classifications. It uses C^x for system specification and provides ways of performance evaluation through migration of SW to HW using simulation and profiling. But on the other hand, COSYMA only supports bus-based architectures and therefore it does not have support for future architectures, such as packet-based architectures, or event support for testing different intercommunication architectures in today's available technology. Since we would like to provide a case study of the system-level modeling tool using a NoC architecture, COSYMA does not fit as a suitable tool for our work.

MLDesigner and Ptolemy are robust system-level modeling environments suitable for embedded and other applications. Both tools do not have in any way any deficiency on the previous classifications. However, Ptolemy is an academic tool and it is still in its development phase. MLDesigner is a commercial modeling environment with more integrated design libraries, it integrates other environments, has optimized memory requirements as compared to Ptolemy and most importantly, it has a strong customer support. Furthermore, MLDesigner is used for various applications, from wireless applications to embedded system design. Hence, it is not limited to a particular target application or domain. For all these reasons, MLDesigner is the system-level design tool used to develop this work.

IV. NOC MODELING IN MLDESIGNER

We designed a 4x4 mesh-based NoC. We modeled all the key NoC building blocks (we call them "classes") in MLDesigner. These classes were identified from a high level concurrency model specification. These classes were: *Producer* (P), *Consumer* (C), *InputBuffer* (IB), *Scheduler* (S), *OutputBuffer* (OB) and *Router* (R). In our NoC model, the *Producer* class defines a resource and a resource network interface (RNI); the *InputBuffer* class corresponds to an input buffer, a buffer scheduler, virtual channels (VC) and a virtual channel allocator; the *Scheduler* class corresponds to a switch allocator; and the *Router* class corresponds to a router. These components were modeled with customizable parameters such as buffer size (varied from 1 to 10 for both input and output buffers), scheduling criteria for schedulers and buffers, (round robin, priority-based, priority-based round robin, first come first serve), data packet priorities (High, Mid, and Low priorities), routing algorithms (static routing algorithms like X-first or Y-first, shortest path), and packet injection ratio (0.1 to 1.0 at intervals of 0.1). This model is discussed in detail in [9], [11].

In this section, we provide some simulation results for data latency for different scheduling algorithms, buffer sizes and injection rates. The minimum latency for a single hop will be 6 clock cycles due to synchronization signals (concurrency cost) among different components: the 1st clock cycle is for storing the data flit into the input buffer; the 2nd clock cycle is for requesting the data output to the scheduler; the 3rd clock cycle is for receiving the grant signal from the scheduler; the 4th clock cycle is for forwarding the data to the router; the 5th clock cycle is for confirming the output path availability; the 6th clock cycle is for forwarding the data packet to the output buffer.

To simulate the NoC, we injected 10,000 flits into the network. High priority flits are used for transmitting control signals such as Memory Read, Memory Write, Input/Output Read, Input/Output Write, and interrupts among others, while Mid priority flits and Low priority flits are used for transmitting real-time data and non-real time block transfers, respectively. For the network simulation, control signals were assumed to account for 10 % of the total data traffic flow, with real-time data and non-real time data accounting for 20 % and 70 %, respectively. Therefore, we injected a total of 1,000 High priority flits, 2,000 Mid priority flits and 7,000 Low priority flits to test our NoC model.

Figs. 2, 3, and 4 show High, Mid, and Low priority data latencies against traffic injection rates, with first come first serve (FCFS) scheduling (at the buffers and the scheduler) and dimension-order routing. These plots have been drawn for different buffer sizes, and there are thus three parameters: buffer size, injection rate, and latency. It is seen that data latency results for all three priorities are almost the same. It is because the FCFS scheduling algorithm does not prioritize the data packets. Thus, packets with all three priorities suffer almost the same latency. As the buffer size increases from 1 to 10, the data latency also increases from 14 to 107. With a larger buffer size, a data packet has to wait longer in the buffer. Thus, larger buffer sizes lead to higher data latencies.

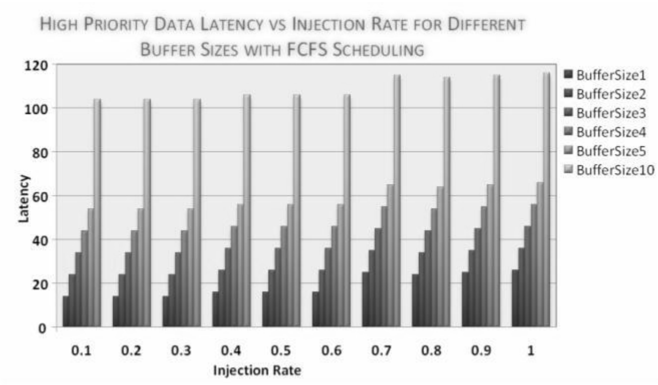

Fig. 2: High priority data latency vs. injection rate for FCFS scheduling.

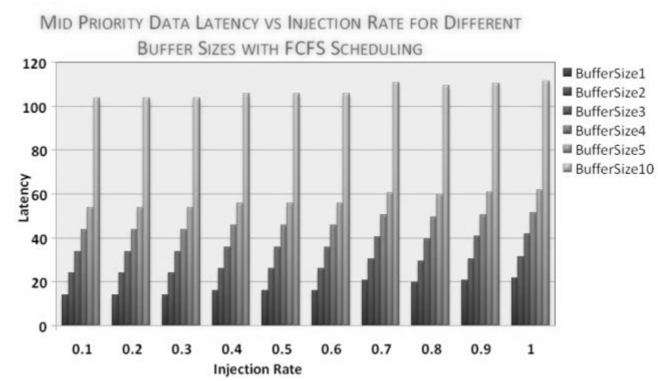

Fig. 3: Mid priority data latency vs. injection rate for FCFS scheduling.

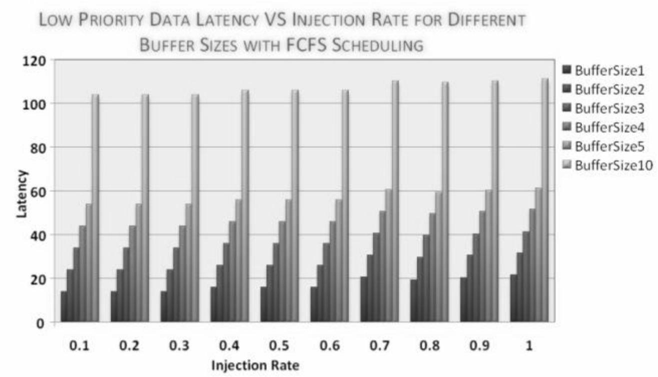

Fig. 4: Low priority data latency vs. injection rate for FCFS scheduling.

Figs. 5, 6, and 7 show High, Mid, and Low priority data latencies against the traffic injection rate for priority-based round robin (PBRR) scheduling for the input and output buffers and round-robin (RR) scheduling for the scheduler. Due to PBRR scheduling, High priority data packets must secure the lowest latency while Low priority packets have the highest latency. The data packet latencies vary from 8 to 30 clock cycles for High priority, from 15 to 36 clock cycles for Mid priority, and from 15 to 140 clock cycles for Low priority. The main advantage of the PBRR algorithm is its capability to serve all the five input buffers connected with the scheduler at the same time. Each input buffer receives an equal scheduler response time.

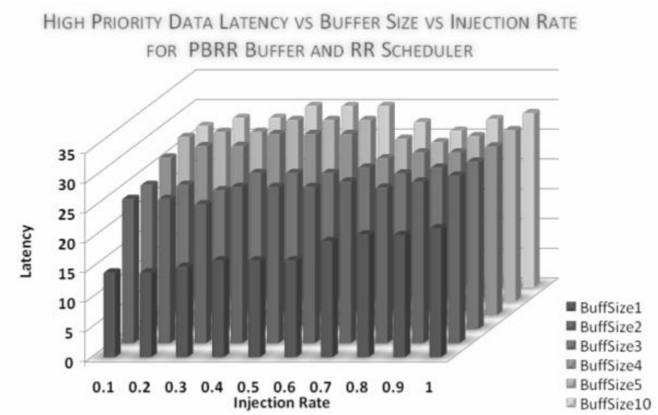

Fig. 5: High priority data latency vs. injection rate for PBRR scheduling.

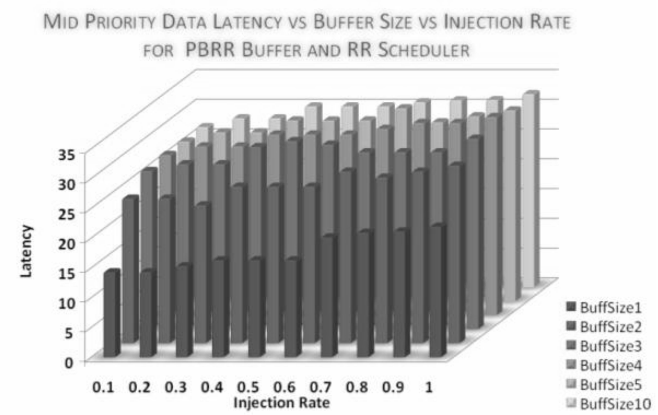

Fig. 6: Mid priority data latency vs. injection rate for PBRR scheduling.

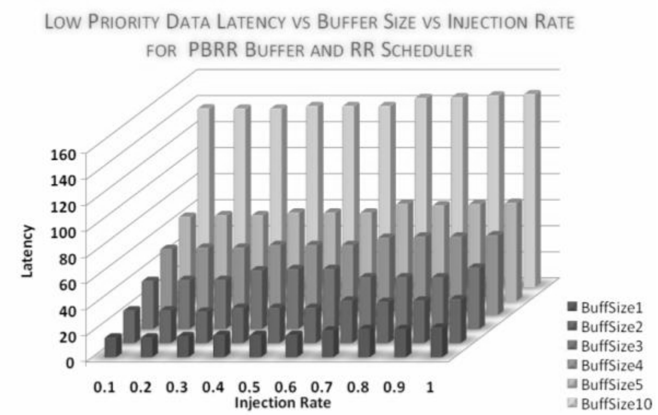

Fig. 7: Low priority data latency vs. injection rate for PBRR scheduling.

The latency results for PBRR scheduling are very similar to those for RR scheduling: indeed, the PBRR scheduling algorithm rotates and serves each input buffer in a RR fashion. However, it provides better results when more than three input buffers are producing data at the same time. In such a scenario, RR scheduling will not be able to deliver the High priority data in time. However, PBRR will work effectively under such a condition.

Figs. 8, 9, and 10 show High, Mid, and Low priority data latencies against the traffic injection rate for priority-based (PB) scheduling for both the input/output buffers and the scheduler. Similar to the PBRR case, High and Mid priority data have the lowest latencies, which vary little with buffer size, while Low priority data are subject to much higher latencies, especially for larger buffer sizes.

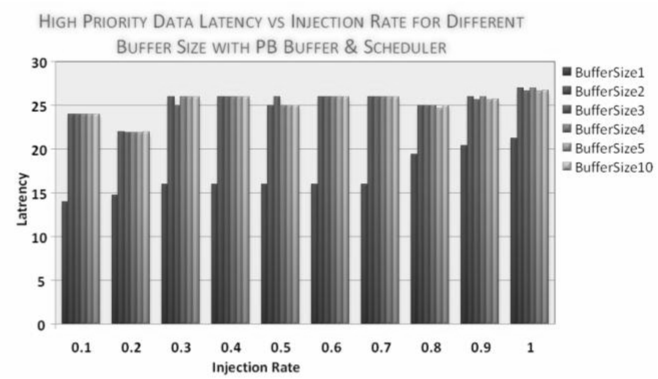

Fig. 8: High priority data latency vs. injection rate for PB scheduling.

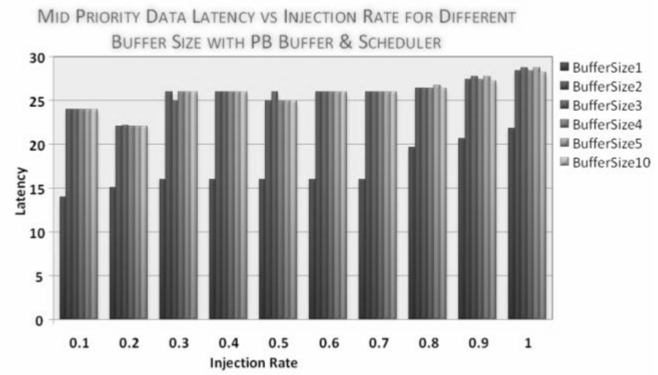

Fig. 9: Mid priority data latency vs. injection rate for PB scheduling.

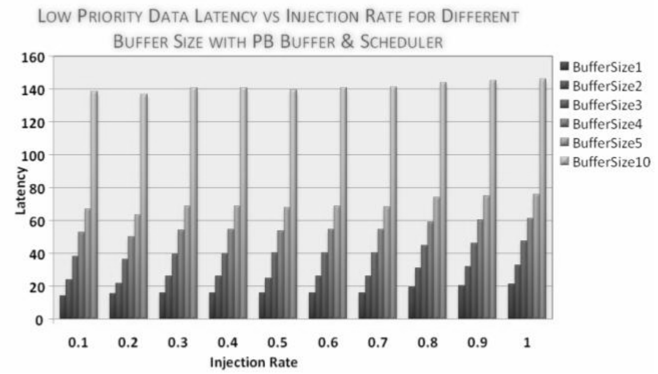

Fig. 10: Low priority data latency vs. injection rate for PB scheduling.

We further simulated a NoC architecture with the maximum load condition. We refer to the maximum load condition when all the sixteen nodes in a 4×4 NoC are producing and consuming data packets. We have simulated such a network load for calculating High priority, Mid priority, and Low priority data latencies with a buffer size of 5 and PBRR scheduling. Results are shown in Fig. 11. Even in the case of the maximum load condition, which is rare, the High priority data latency is about 55 clock cycles, the Mid priority data latency is about 63 clock cycles, and the Low priority data latency is in the range of 200-250 clock cycles. Therefore, it is recommended that an architect must over-dimension the NoC architecture if there is a possibility of all the NoC nodes consuming and producing data packets simultaneously. Such over-dimensioning of the network will reduce the data latency for embedded systems.

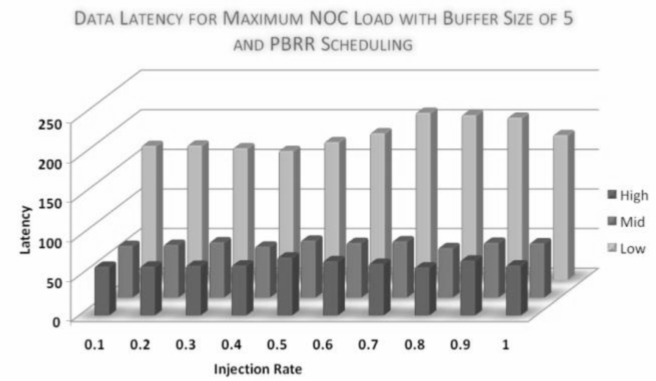

Fig. 11: Data latencies for maximum load condition.

V. CONCLUSION

We have compared and analyzed various system-level modeling and simulation environments, and found that MLDesigner supports multiple MOCs that allow one to model the system at an abstract level. This helps with both fast model building and simulation. MLDesigner provides a well developed set of libraries for different domains. These libraries are very helpful for modeling a system, since building blocks of the model found in the library set do not need to be designed. This makes MLDesigner an excellent environment for performance evaluations. We developed a NoC model in MLDesigner. The developed model is abstract enough to facilitate quick analysis, and is composed of customizable components. From the simulation of the model, these different component parameters give a set of performance characteristics of the model. The performance characteristics are extracted and assembled together resulting in the performance evaluation of the model. As an example of the model's use, we chose three different traffic patterns and three priority levels to show that the communication backbone can be designed to optimally meet different latency requirements even when the traffic patterns are markedly different.

REFERENCES

[1] R. Shankar, H. Kalva, A. Agarwal, and A. Jain, "Annotation methods and application abstraction", *IEEE International Conference on Portable Devices*, Orlando, FL, April 2007, pp. 1-5.

[2] A. Agarwal, G. Hamza-Lup, and R. Shankar, "An integrated methodology for QoS driven component design and component selection", *1st Annual IEEE Systems Conference*, Hawaii, April 2007, pp. 1-7.

[3] F. Balarin, M. Chiodo, D. Engels, P. Giusto, W. Gosti, H. Hsieh, A. Jurecska, M. Lajolo, L. Lavagno, C. Passerone, R. Passerone, C. Sansoè, M. Sgroi, E. Sentovich, K. Suzuki, B. Tabbara, R. von Hanxleden, S. Yee, and A.L. Sangiovanni-Vincentelli, “POLIS – A design environment for control-dominated embedded systems”, User’s Manual, 1999.

[4] D. Herrmann, J. Henkel, and R. Ernst, “An approach to the adaptation of estimated cost parameters in the COSYMA system”, *Proceedings of the Third International Workshop on Hardware/Software Codesign*, pp. 100-107, 1994.

[5] C. Hughes, V. Pai, P. Ranganathan, and S. Adve, “RSIM: Simulating shared-memory multiprocessors with ILP processors”, *IEEE Computer*, vol. 35, no. 2, pp. 40-49, Feb. 2002.

[6] J.T. Buck and S. Ha, “Ptolemy 0.7 Kernel Manual”, University of California, Berkeley, 1997.

[7] MLDesign Technologies, Inc, “MLDesigner Documentation”, Palo Alto, California, 2006.

[8] A. Osterling, T. Benner, R. Ernst, D. Herrmann, T. Scholz, and W. Ye, *Hardware/Software Co-Design: Principles and Practice*, chapter “The COSYMA System”, Kluwer Academic Publishers, 1997.

[9] A. Agarwal, *QoS Driven Communication Backbone for NoC-Based Embedded Systems*, Ph.D. Dissertation, Florida Atlantic University, Dec. 2006.

[10] F. Kovalski, *System Level Modeling and Simulation*, M.S. Thesis, Florida Atlantic University, Dec. 2006.

[11] A. Agarwal, C.-D. Iskander, H. Kalva, and R. Shankar, “System-level modeling of a NoC-based H.264 decoder”, *Proc. IEEE Systems Conf.*, Montréal, April 2008.