Proceedings of the 17th IASTED International Conference
Parallel and Distributed Computing and Systems
November 14-16, 2005, Phoenix, AZ, USA

A LAYERED ARCHITECTURE FOR NOC DESIGN METHODOLOGY

A. Agarwal, R. Shankar,
aagarwa2@fau.edu, ravi@cse.fau.edu,
Dept of Computer Science and Engineering, Florida Atlantic University,
Boca Raton, FL-33431

ABSTRACT
Multiprocessor system on chip (MPSoC) platform is an innovative trend of System on Chip (SoC) that enhances system performance. Demanding quality of service parameters and performance metrics, especially in mobile applications, are leading to the exploration of even more innovative architectures for SoC. These will have to incorporate highly scalable, reusable, predictable, cost and energy efficient architectures. Network on Chip (NOC) is a key example of this trend. NOC separates computing and communication concerns in an elegant manner. We propose here a seven layered architecture for designing NOC-based systems. Such a platform can separate domain specific issues in separate layers, which will allow for more effective modeling of concurrency and synchronization issues, in an attempt to develop an optimized system. For such a layered architecture, models of computation (MOC) will provide a framework to model various algorithms and activities, while accounting for and exploiting concurrency and synchronization aspects. These MOCs may differ from one to another NOC region. Further, a combination of these MOCs may be needed to truly represent a given NOC region. We have analyzed various models of computation (MOC) suitable for NOC. MLDesigner provides a system level modeling platform which allows one to integrate such MOCs together. We present our efforts and experiences so far.

KEY WORDS
Network on Chip, Modeling and Simulation, Quality of Service, System Level Design

1. Introduction

The System Level Design era where creativity, innovative ideas, ingenuity and inspiration are expected to come to the fore, has arrived. System complexity, driven by both increasing transistor count and customer appetite for more savvy applications, has increased so dramatically that system design and integration can no longer be an after-thought. Moore's law predicts that a chip in 2010 will count more than four billion transistors operating in multi GHz range [1] [2]. This is mainly due to the exponential decrease in the transistor size enabling faster transistor switching times and more densely integrated circuits. Such computation power has posed some challenges which include the disparity in transistor and wire speed and increased power dissipation, leading to a decrease in the area of the chip which can be utilized with a single clock cycle [3][4]. Another dominant factor is the ability to design the system in an acceptable timeline known as time-to-market. Also, system level designers have to constantly look for ways to support a set of Quality of Service (QoS) parameters and performance metrics that have become more demanding, as expectations have soared.

Technology scaling has also had unwanted side effects which include cross coupling, noise, and transient errors [5]. This has resulted in reuse of design blocks, sometimes referred as components, which have been carefully designed by expert designers. However it can never be guaranteed that those sub-micron effects will not pop-up again when such reusable components are integrated into larger designs, such as a sub-system or a system. Thus it can be concluded that components or the sub-systems which perform as expected when isolated, might not perform in the same way after system integration [6]. This has led to a new domain of research work for system level integration and verification viz, the NOC architecture [2, 6, 7].

ITRS (International Technology Roadmap for Semiconductors) [8] provides a plot that depicts the change in engineering design cost over the time span of 1990 to 2005. One might expect this to have increased substantially, since the chips and systems got more complex during that period. However, the exact opposite was the case.

Many EDA (engineering design automation) innovations kept the engineering design cost fairly flat, that is, the engineering design productivity actually increased about 70 fold. This brought down the cost of product development which led to an affordable price for the customers, while providing more functionality. However, this trend may be at an end [8] unless innovative architectures and methodologies for system design are not developed. ITRS predicted in 2001 that the engineering design cost would increase to an unaffordable amount of one billion dollars by 2010 [8]. Much innovation has come since then. It is expected that the future systems will

have increasing roles of design automation, reusability and componentization, thus increasing the role of the EDA Industry. For such scenario, a system-level modeling environment should be developed that essentially supports the middle-out design philosophy to exploit reuse to the maximum in order to reduce the design effort [9] [10]. The high volume of reuse should cut down the overall system design cycle [11] [12]. In this paper we present one such unified framework for the architectural design and simulation for the NOC environment.

2. Background work

Integration of a system comprising of hundreds and thousands of cores with adequate memory and communication backbone on a single silicon die is feasible but highly complex [13]. Designing such systems on a chip (SOC) is a challenging process, and is currently approached with few principles of organization [14]. Researchers have contributed in various dimensions to this activity to make such an effort plausible. One such design discipline is platform based design [15] [16]. Essential elements of platform-based design are the functional behavior of each core and separate consideration of the interaction among the cores. This separation of concerns is essential to the success of a reuse strategy [15]. The future SOCs, in order to be economical, will have to aim at designing highly scalable and configurable systems so that it can be adapted to different workload needs, applications and products, while maintaining the generality of the application development methods and practices.

As emphasized in the International Technology Roadmap for Semiconductor (ITRS) 2001 document [8], it is very important, especially at system level, to separate the computation from communication aspects to enhance design productivity [15]. From the communication perspective, several researchers have suggested a 2-D mesh architecture for NOC [13] [17] that connects (computational) resources at various nodes via network elements. The network elements are switches, channels and resource-network interfaces (RNI). A (computational) resource in an NOC can be any general purpose processor core, memory, specified controller, FPGA, ASIC etc. This infrastructure will be ideal where QoS and performance parameters will be traded off based on the user requirements. New algorithms have been proposed in this domain to reduce power consumption while securing cost optimization [18]. It has been a well established fact that such NOC architectures will be based on packet switched networks. This has led to new and efficient principles for design of routers for the NOC architecture [19]. These routers will be responsible for routing the entire traffic across and have to be interfaced with switches and resources in the NOC architecture. The RNI should be highly scalable and re-usable in order to be integrated with resources with different types of interfaces and data requirements. Substantial research has been conducted to propose the right data formats needed for various layers in the protocol stack. A reusable switch is used for effectively routing the packet through the entire NOC; they buffer packets at both input and output [20].

Once the design of the basic NOC architecture became established, new techniques evolved to address advanced issues such as dynamic load balancing on to a node of the NOC architecture, the shortest/fastest path for the data flow through NOC, and energy efficient NOC architecture design. Most researchers have focused on the communication architecture of NOC; few have focused on exploiting the computing capability of NOC. Computer architects have come up with many innovative ideas over the past decade to enhance system performance; however not all of them are viable in NOC. One innovation that is useful for NOC is to divide the system into smaller, *locally decoupled synchronous regions* and then composing a few of them to yield a localized subsystem. These synchronous regions and subsystems would be easier to integrate into a global solution and verify. At the same time there will be an *asynchronous* way in which all the local synchronous regions will communicate at the system level. Thus these different synchronous regions need not have to be synchronized to a single global clock. This approach would reduce the requirement for chip-wide clock trees; the designers could focus on local synchronous regions only which would be far less complex than the complete system. Also, since one has the flexibility to reduce the clock speed of a given synchronous region (or node) independent of other such regions, the amount of power consumption in a system can be managed better and reduced.

3. A Novel Approach for the NOC Architecture

The key challenge for the system level design with integration of thousands of cores is the modeling of concurrency and synchronization issues. This requires representation of various activities and algorithms with appropriate Models of Computation (MOC). This defines our two goals for the system level designers: (1) To model the system functionality well in advance of building the actual computing system in order to provide a level of flexibility in system design. (2) To be able to manipulate an abstract set of design elements simultaneously to generate different sets of QoS parameters and performance metrics and fine tune the system model. For designing such a system we first define the concept [21] of a layered NOC architecture and then try to integrate different MOCs onto this layered architecture.

3.1 Layered architecture for NOC design

Layered architecture is an effective way of dividing and conquering a problem. We have seen it working in an effective way in the of open system interconnection (OSI) model. It provides a way to separate the concerns of each different domain thus providing an effective solution to a domain specific problem. It can also be argued that this approach will provide us with a scalable solution in addressing inter-domain specific issues. We define below the seven-layer NOC architecture:

3.1.1. Application: This will be the top functional level of the NOC system where different applications are represented in an abstract manner. This layer will model the application software. Since a NOC system should support a wide range of the applications, there is need for application libraries. These software applications should be portable in nature so that we can use them in different application domains. This will enhance reuse by system level designers. Thus these applications should be well designed to have proper interfaces with the system.

3.1.2 Algorithm: This will be the next lower level layer where the real-time control and multimedia algorithms, as well as the optimization algorithms, will run. These algorithms will be application and performance specific.

3.1.3 RTOS (Real-Time Operating System): The next lower layer will comprise of RTOS. This layer will be responsible for effective scheduling of the NOC resources to run application and algorithmic software. RTOS will schedule applications based on a priority scheme which in turn will be defined by specified QoS parameters and performance metrics. The other main functionality of this layer will be the management of concurrency and synchronization issues.

3.1.4 Architecture: Architectural exploration is done at this layer. It will comprise of architectural building blocks such as a processor, cache, bus, and memory, and parameters that capture their pertinent properties. Hardware and software are traded off depending upon performance metrics such as utilization, latency, and transactions. Different combinations of these building blocks are simulated to run certain target applications. Based on such simulation runs, a set of optimal architectures will be selected. However, do note that for complete and valid results, the lower layers will also have to be simulated.

3.1.5 Communication Protocol: Packet switching has been proven to be the right way to implement this layer. This will define the size of the packet, the queuing discipline, routing strategies, and the issues related to the traffic density. For a real time application this layer will forward the traffic on to a dedicated channel so that the timely response can be guaranteed in order to generate the required level of service. The algorithms related to dynamic load balancing can also be addressed as a part of this layer. This model will influence the protocol stack design.

3.1.6 Communication Backbone: This layer below the communication protocol layer will model the actual routers, switches, and resource network interface, along with buffers, busses, queues, and FIFO. This is the layer where actual transfer of data among the different resources takes place.

3.1.7 VLSI: This will be model the final system implementation at the transistor level. The VLSI related issues will be abstracted to this layer.

3.2 Model of Computation (MOC)

A MOC is a mathematical formalism that captures 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. A synchronous local region of an NOC might require one ore more such MOCs to co-exist and interact. Further, to reduce simulation time and to integrate the subsystems into an integrated system model, other MOCs may be needed. 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. Consider also the case in which a subsystem uses two or more local regions (or islands) of the NOC. These are connected by a switch in an NOC. For example consider a digital camera as a component or a subsystem of a much larger system, such as a wireless handheld device (system). It is possible that its design would not fit into one local region. Under such a scenario we should also be able to address the concurrency and synchronization issues because of shared resources (both local and global).

We propose to integrate appropriate models of computation to model our layered NOC architecture (See [21] for similar work for more general cases). We address below the various MOCs needed to model NOC:

3.2.1 NOC at System Level (Global Region): At the system (top) level, the NOC model, in order to be useful for what-if type analysis, should be at a high level of abstraction. Lower level (RTL code, VLSI, and Source code) issues, while appropriate for the design level, would slow down the trade-off analysis. Assume that a digital camera's processing is implemented on an NOC; then the system level issues to be addressed would be resolution, image size, power dissipation and cost, which we would refer to as parameters. We should be able to adjust one of these parameters to fine tune system performance and yield different combinations of cost-performance-QoS-power dissipation. One may wish to do this on a dynamic

basis for better power management of the system in the field. Thus we can argue that at system level we would need some a kind of manager to dynamically optimize the system level design parameters.
At the same time such a control (manager) should have some level of concurrency in time domain. This is due to the fact that control (manager) might have to optimize two or more different local regions at the same time. This requirement rules out a finite state machine (FSM) model to sit at the top as a manager. At the system level this manager should be able to observe and control only those signals which are changing their state or behavior rather than monitoring all the signals at every clock event. A discrete event simulation at the system level would suite such a requirement.

Another possible MOC to model global communication and management in NOC, with asynchronous mechanisms, is the process network (PN). This MOC was described by Kahn and McQueen [20]. Two important properties of the PN domain which make such computation plausible are that processes communicate asynchronously and that the memory used in the communication is unbounded. Any practical implementation of PN cannot support an infinite memory requirement; therefore we specify upper bound for memory requirements whenever possible. The PN domain has the capability to model a system as a network of processes that communicate with each other by passing messages through unidirectional first-in-first-out channels.

3.2.2 NOC at Subsystem Level (Local Region): The local region for NOC is again divided into two different domains as NOC at subsystem level and NOC at component level. At sub-system level we will have to address local issues (as relating to a particular subsystem) rather than the global issues. Such a sub-system will usually be some DSP sub-system, IP-core, field programmable gate arrays (FPGA), or application specific integrated circuit (ASIC). This subsystem in conjunction with the packet switching network would be required to consume some fixed amount of data (tokens) at the input and to produce some fixed amount of data (tokens) at output that will be routed over the network. Synchronous data flow (SDF) is one such MOC which has appropriate properties. At subsystem level, there is also some need for a smaller control element. This control element would be required to address the synchronization issue at that subsystem level only making FSM as another plausible solution to MOC at the subsystem level.

3.2.3 NOC at Component Level (Local Region): This is another essential part of the local region. These components together would constitute the higher subsystem level. At this level designer will not have to worry about addressing the system wide concurrency and synchronization issues and the design should be highly reusable to be able to be utilized in other products and scalable to be mapped into the higher domains, viz., the subsystems and the systems. This component level would comprise of software components, and a computation part which in turn could be represented by electrical components and computer architecture components. For electrical components to be specified in such domain, we can use mathematical equations to represent properties of the model giving continuous time (CT) MOC as a feasible solution. Hierarchical FSM can contribute to synchronization among the software elements and the hardware components at component level where each state can be programmable in itself. Such a state machine is often referred as a state chart, where each state has trigger, action and guard conditions. At the same time we may also utilize SDF MOC to model the architectural issues.

We need a platform to support and integrate different models of computation to exploit the computation and capability of the future generation systems. Such a platform should be able to model software and hardware along with communication and scheduler components, to account for resource sharing and consequent limitations. Such a platform would not distinguish between hardware and software components but rather integrate both into the system in almost the same way.

We discuss such system below.

4. Mapping Various Models of Computation into a Single System

Electronic Design Automation (EDA) tools are able to play a vital role in optimizing the entire product development life cycle for the system level designs. Powerful EDA tools and interface software ensure more efficient use of resources leading to high quality designs. EDA software product development teams constantly strive to increase the intelligence and functionality of the EDA tools by incorporating more and more sophisticated algorithms in the tool to aid the system designer.

We have analyzed one such design automation tool “MLDesigner” [22], which is a system level solution that integrates multiple level designs. This provides a platform on which we can implement our layered architecture for NOC. We will be able to integrate different MOCs at the same time allowing us to separate the concerns and limit them to a particular layer rather than addressing all the issues at the same time. This will allow us to build each layer separately and to integrate them subsequently to obtain a fully modeled system.

We have included two examples to demonstrate the effectiveness and the feasibility of the proposed design methodology [23]. The first example demonstrates the idea of a layered approach for the design, while the

second example will bring forth the concept of integration of different MOCs in a single system [23].

4.1 Modeling a Multi-processor Architecture using MLDesigner: A Layered Approach for the Architecture Design [23]

The multi-processor computing system under study consists of four CPUs, a shared bus, shared memory, and various controllers. Each CPU has its own cache memory. MLDesigner is used to model the computer architecture as shown in Figure 1. This system design has been partitioned into layers of software and hardware. The software layer is the application layer which is separate from and above the hardware layer. A concept of the Operating system layer has been introduced via a model of a Dispatcher block at the hardware layer. For the communication backbone the basic bus based model has been used. This model represents another layer. (For the NOC architecture this layer will be described with two different layers: the communication protocol and the communication backbone layers.) This design platform provides an interactive environment for simulation-based analysis and design of a broad range of complex embedded systems including NOC.

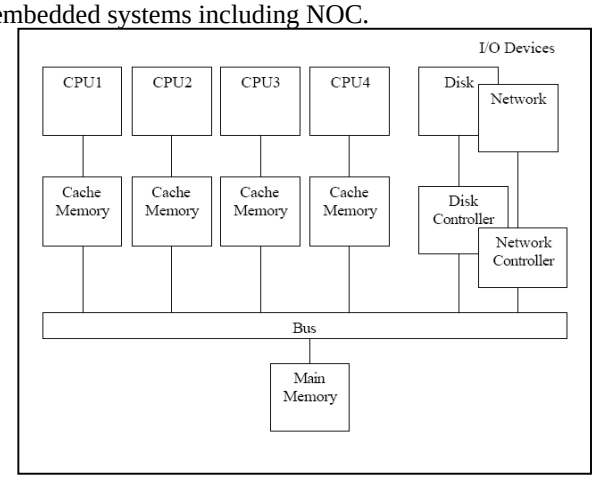

Figure 1: Multi Processor Computer Architecture

The model uses sophisticated Discrete Event (DE) modeling and abstraction techniques such as data structures, quantity resources, and shared memory.

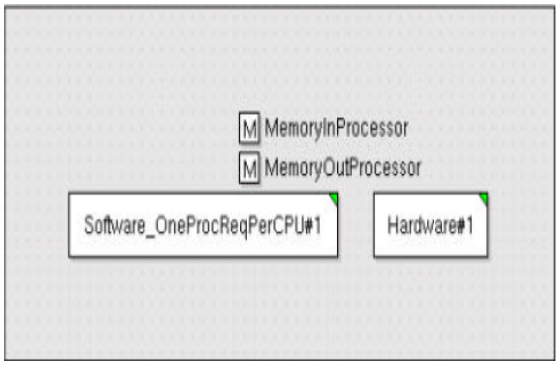

Figure 2: System (Hardware-Software) Model

The simulation model abstracts computer cycles as costs and offers a flexible approach for studying the interaction of hardware and software. At the system level, the model consists of hardware and software blocks that are virtually connected via the shared memory as shown in Figure 2.

The hardware module has the following blocks – Dispatcher, CPU, cache, bus, memory, etc. as shown in Figure 3. The dispatcher works as a scheduler: it receives an instruction from the software model, assigns it to the first available idle CPU. If none of the CPUs is idle then the dispatch queues that instruction for future execution. The CPU block decides if the instruction requires access of the cache to satisfy the request. If the requested data is not available in the cache, it is obtained from the shared memory using the shared bus.

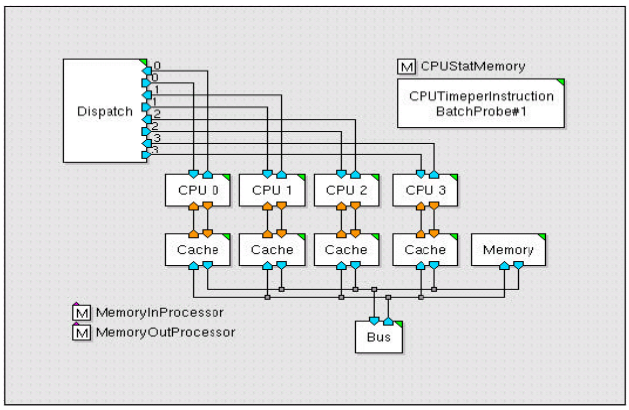

Figure 3: Hardware Module

The parameters for the simulation are shown as shown in Figure 4.

Processor Speed Per Time-unit: 1000000
Mean Memory Accesses per Instruction: 1.4
Cache Hit Rate: 0.89
Cache Hit Time: 5.0-9
Memory Access Time: 4.0E-8
Number of Bus Cycles: 2
Bus Cycle Time: 5.0E-9
Number of Instructions: 50000; 100000; 150000
Number of Active Processors: 1; 2; 3; 4
Global Seed: 1234567890
Run Length: 1

Figure 4: Parameters for Simulation

The software model generates one processor request per processor and drives the hardware model. This simulation iterates on two parameters: Number of Instructions and Number of Active Processors. In this simulation, the first set of results show what happens to the mean time per instruction when the number of CPUs is increased.

The second set of results looks at bus activity with four active processors.

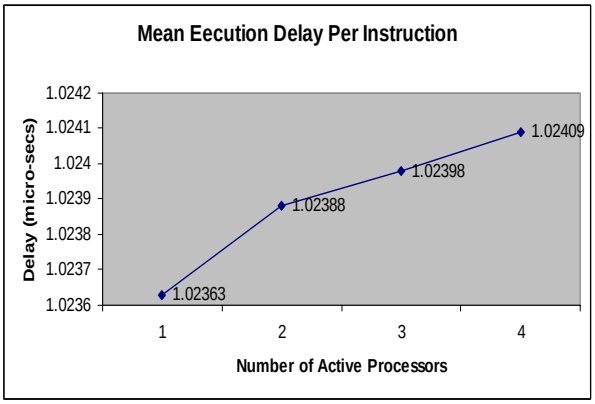

Figure 5: Single Instruction Delay versus Number of Processors (for 50,000 Instructions)

Figures 5, 6, and 7 show the mean execution delay per instruction with increase in number of active CPUs and with 50000, 100000, and 150000 instructions per processor request. The simulation results show that, the average time for the execution of an instruction increases with the increase of active processors in the system. As more CPUs become active in the system, more cache misses occur and each CPU makes more demands on the bus to access main memory.

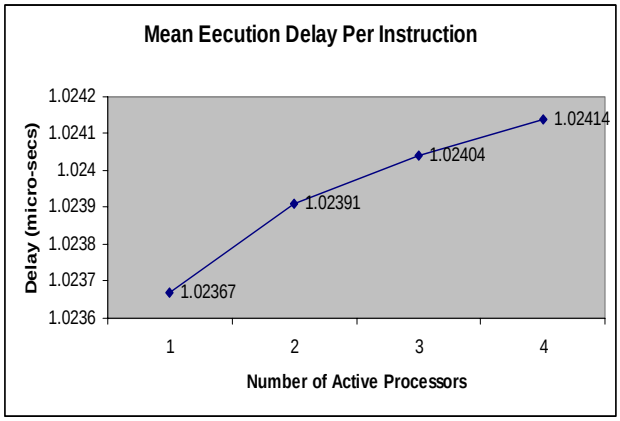

Figure 6: Single Instruction Delay versus Number of Processors (for 100,000 Instructions)

This causes the requests to get queued inside the bus and results in bus contention. This will reduce the effective speed up below the theoretical limit of n, where n is the number if processors. Figure 8 shows the bus activity in the system to provide more detail on bus contention. In the plot the processor IDs are represented from 0 to 3. The plot shows the bus activity due to requests from 4 different processors. The designs of both hardware and software components are modularized and parameterized so the model can be quickly modified to address a variety of design issues and use cases. Modification could include adding additional components or adding additional detail to components.

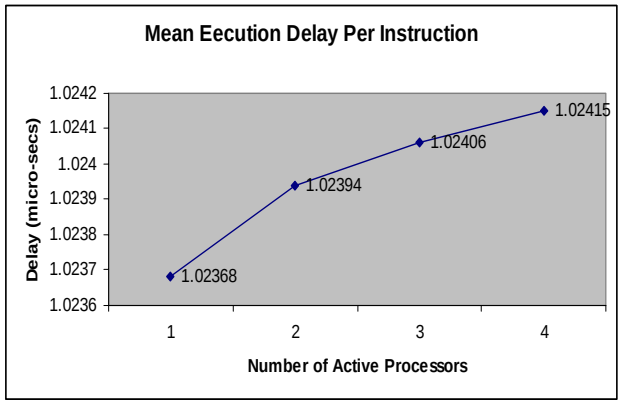

Figure 7: Single Instruction Delay versus Number of Processors (for 150,000 Instructions)

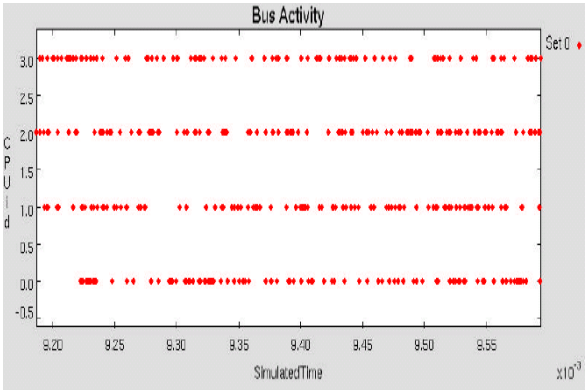

Figure 8: Bus activity for 50,000 Number of Instructions and 4 CPUs

4.2 Water-pump Controller Model using MLDesigner [23]

This system model demonstrates the integration of various MOCs together. In particular this example integrates MOCs for Finite State Machine (FSM), Synchronous Data Flow (SDF), and Discrete Event (DE).

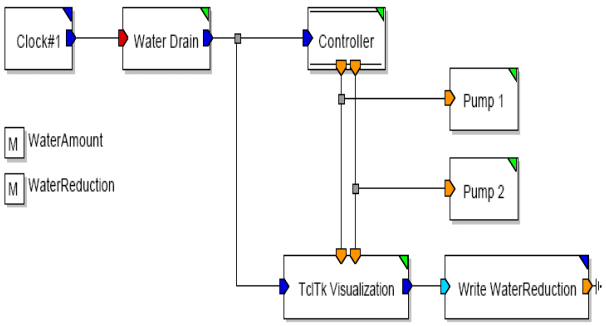

Figure 9: High Level System Model

The model also shows the use of dynamic control and animation components. The simulated system consists of a water tank equipped with a drain to remove water from the tank, two pumps to refill the tank, and a controller that monitors the level in the tank and controls the operation of the two pumps. The high level model is shown in Figure 9. The system model consists of a clock, a water drain, the controller, two identical pump modules, the visualization components, and a block to update the tank water level reduction.

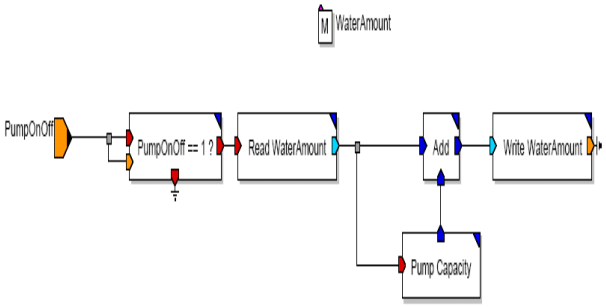

Figure 10: Water Pump Module

The water drain block models the operation of the tank drain and associated tank levels. Shared memories (blocks Read WaterReduction and Write WaterReduction) track the water level and the water level reduction. Two identical pumps are used to refill the tank. A water pump model is shown in Figure 10. The controllers are different in functions (Figure-11).

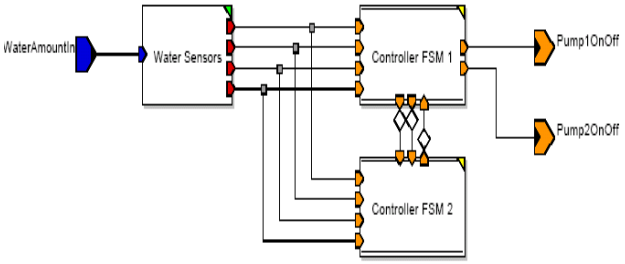

Figure 11: Controller Module

FSM-1 handles the two water pumps. It consists of 3 states (s0, s1, and s2) as shown in Figure 12. FSM-2 controls their equal distribution. FSM-2 consists of 2 states (s0 and s1). When the model starts, the drain control and tank animation (Figures 13) appear and the tank fills. The drain is initially closed. When the drain is opened, it triggers the sensors and the water flows out.

The controller detects the change in level and starts the pumps when necessary. When the tank reaches a set level, the controller shuts off the pumps. While the drain is open the model cycles constantly. Water flows out, the tank level drops, sensors detect the lowered level, and controller gets the information from the sensors and switches on the pumps to raise the level to a preset level. The pumps stop, the water level drops, and the cycle starts again.

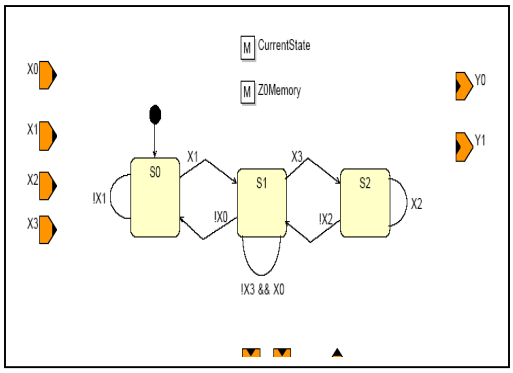

Figure 12: Controller Module FSM-1

This simulation demonstrates the feasibility of integrating various MOCs.

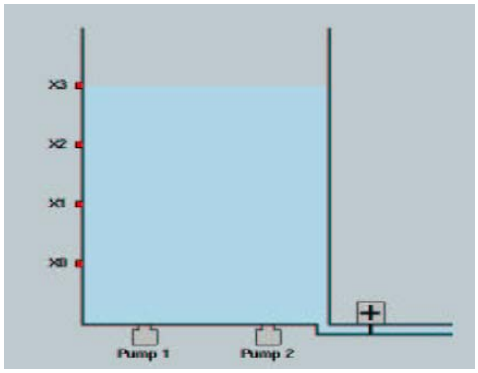

Figure 13: Full Tank

5. Conclusion

System complexity is increasing due both to continued miniaturization and increase in demanding applications. NOC (Network on Chip) elegantly separates the concerns of computing and communication, and is expected to be ideally suited to address this increased system complexity. We propose a layered architecture so the design complexity can be well managed and design issues can be addressed at different levels in the layered architecture. A system for modeling of such systems, using our layered approach, will require support for components and interaction among these components, using various MOCs (models of computation). We show the use of MLDesigner for this. The simulation results show that MLDesigner can be one such effective environment to model our layered NOC architecture. Such a model can be used to trade off different system parameters to yield appropriate quality of service and performance metrics.

Acknowledgements

We would like to acknowledge the help of a PhD student Abu Asaduzzaman and Mr. Colin Mick, MLDesigner, for providing the models.

References

[1] L. Benini and G. De Micheli. Networks on chip: a new SOC paradigm, *IEEE Computer*, 35(1), January 2002, 70-78.

[2] Xu, Jiang, W. Wolf, J. Hankel, S. Charkdhar, A Methodology for design, modeling and analysis for networks-on-Chip, *IEEE International Symposium on Circuits and Systems*, May 2005, 1778-1781

[3] Hemani, Axel Jantsch, Shashi Kumar Adam Postula, Johnny Öberg, Mikael Millberg, Dan Lindqvist, Network on Chip: an architecture for billion transistor era, *Proc. of IEEE NorChip Conference*, November 2000, 8-12.

[4] Paul Wielage, Kees Goossens. Network on silicon: blessing or nightmare? In *Euromicro Symposium on Digital System Design, Dortmund*, Genmany, September 2003. Keynote Speech.

[5] Tejasvi Das, Clyde Washburn, P. R. Mukund, Steve Howard, Ken Paradis, Jung-Geau Jang, Jan Kolnik, Effects of technology and dimensional scaling on input loss prediction of RF MOSFETs, *International Conference on VLSI Design held jointly with 4th International Conference on Embedded Systems Design*, 2005, 295-300

[6] Alexandre M. Amory, Érika Cota, Marcelo Lubaszewski, Fernando G. Moraes, Reducing test time with processor reuse in network-on-chip based systems**,** *Proceedings of the 17th ACM symposium on Integrated circuits and system design*, 2004, 111 - 116

[7] Cota, E.; Carro, L.; Wagner F., Lubaszewski, M, Power-aware NoC reuse on the testing of core-based systems, Test Conference, 2003. *Proceedings of International Test Conference* 2003, Sept. 30-Oct. 2, 2003, 612 - 621

[8] Semiconductor Industry Association, The international Technology Roadmap for Semiconductors (ITRS) 2001. http://public.itrs.net/Files/2001ITRS/Home.htm

[9] Edward A. Lee, Yuhing Xiong, System level types for component-based design, *Workshop on Embedded Software*, California, October 2001

[10] Y. Xiong and E. A. Lee, "An extensible type system for component-based design", *6th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, Berlin, Germany, April 2000.

[11] R. A. Bergamaschi, S. Bhattacharya, R. Wagner, C. Fellenz, M. Muhlada, F. White, W. R. Lee, and J.-M. Daveau. Automating the design of SOCs using cores. *IEEE Design and Test of Computers*, 18(5), September 2001, 32-44

[12] Cota, E.; Kreutz, M.; Zeferino, C.A.; Carro, L.; Lubaszewski, M.; Susin, A., The impact of NoC reuse on the testing of core-based systems**,** *21st Proceedings of VLSI Test Symposium*, 2003, April 2003, 128 - 133

[13] A. Jantsch and H. Tenhunen. *Networks on Chip* (Kluwer Academic Publisher, 2003)

[14] S. Kumar, A. Jantsch, J-P. Soininen, M. Forsell, M. Millberg, J. Oberg, K. Tiensyrja, and A. Hemani. A Network on Chip Architecture and Design Methodology. *In IEEE Computer Society Annual symposium on VLSI*, April 2002, 117-124

[15] K. Keutzer, S. Malik, A. Richard Newton, Jan M. Rabaey, A. Sangiovanni-Vincentelli, System level design: orthogolanlzation of concerns and platform based design, *IEEE Transaction on CAD of Integrated Circuits and Systems*, 19(12): 2000, 1523-1543

[16] A. Ferrari, A. Sangiovanni Vincentelli, System Design: traditional concepts and new paradigms. *In International Conference on Computer Design*, 1999, 2-12

[17] L. Benini, D. Bertozzi, Network on chip architecture and design methods, *IEEE proceeding Computation Digital Technology*, vol. 152, No. 2, March 2005, 261-271

[18] P. Pande, C. Grecu, M. Jones, A. Ivanov, R. Saleh, Performance evaluation and design tradeoffs for network on chip interconnect architecture, *IEEE Transaction on Computers*, vol. 54, Issue 8, August 2005, 1025-1040

[19] D. Rostilav, V. Vishnyakov, E. Friedman, R. Ginosar, An asynchronous router for multiple service levels networks on chip, *11th IEEE international symposium on asynchronous circuits and systems*, March 2005, 44-53

[20] Yi Ran Sun, S. Kumar, A. Jain, Simulation and evaluation for network on chip architecture using NS-2, 20th IEEE *International Conference preceding for NorChip* vol. 5, May 2003

[21] T. Grotker, S. Liao, G. Martin, S. Swan, *System Design with SystemC*, (Springer Publication, May 2002)

[22] Modelling environment for software and hardware, MESH, www.ece.cmu.edu/~mesh

[23] MLDesigner, http://www.mldesigner.com