Modeling Operation and Microarchitecture Concurrency for Communication Architectures with Application to Retargetable Simulation

Xinping Zhu, Wei Qin and Sharad Malik

Department of Electrical Engineering, Princeton University, Princeton, NJ 08544, USA

{xzhu, wqin, sharad}@princeton.edu

ABSTRACT

In multiprocessor based SoCs, optimizing the communication architecture is often as important as, if not more than, optimizing the computation architecture. While there are mature platforms and techniques for the modeling and evaluation of computation architectures, the same is not true for the communication architectures. A major challenge in modeling the communication architecture is managing the concurrency at multiple levels: at the operation level, multiple communication operations may be active at any time; at the microarchitecture level, several microarchitectural components may be operating in parallel. Further, it is important to be able to clearly specify how the operation level concurrency maps to the microarchitectural level concurrency. This paper presents a modeling methodology and a retargetable simulation framework which fill this gap. This framework seeks to facilitate the design space exploration of the communication sub-system through a rigorous modeling approach based on a formal concurrency model, the Operation State Machine (OSM). We first introduce the basic notions and concepts of OSM and show by example how this model can be used to represent the inherent concurrency in the architecture and microarchitecture of processors. Then we demonstrate the applicability of OSM in modeling on-chip communication architectures (OCAs) by walking though a router based packet switching network example and a bus example. Due to the fact that the OSM model is naturally suited to handle the operation and microarchitecture level concurrencies of OCAs as well, our OSM-based modeling methodology enables the entire system including both the computation and communication architectures to be modeled in a single OSM framework. This allows us to develop a tool set that can synthesize cycle-accurate system simulators for multi-PE SoC prototypes. To demonstrate the flexibility of this methodology, we choose two distinct system configurations with different types of OCA: a 4x4 mesh network of 16 PEs, and a cluster of 4 PEs connected by a bus. We show that by simulation, critical system information such as timing and communication patterns can be obtained and evaluated. Consequently, system-level design choices regarding the communication architecture can be made with high confidence in early stages of design. In addition to improving design quality, this methodology also results in significantly shortened design-time.

Categories and Subject Descriptors

I.6.5 [**Simulation and Modeling**]: Model Development; J.6 [**Computer-Aided Engineering**]: Computer-aided Design (CAD)

General Terms

Measurement, Performance, Design, Experimentation, Languages, Verification

Keywords

retargetable simulation, on-chip communication architecture, bus, packet-switching network, multiprocessor system, simulator synthesis, design exploration

1. INTRODUCTION

Modern high performance System-on-Chip (SoC) designs generally consist of two integrated parts, the Processing Elements (PEs) and the On-chip Communication Architecture (OCA). The PEs process the computational tasks while the OCA performs the necessary communication between these PEs. It is critical to optimize the structure of the OCA so that it can meet the ever increasing demand for high bandwidth, low-latency on-chip communication.

To evaluate architectural designs, usually designers begin with a cycle-accurate microarchitecture simulator to model and simulate sequential application programs. Based on simulation results, designers then refine the design iteratively. However, this current uni-processor centric modeling methodology does not adequately meet the new design challenges posed by multi-PE SoCs running concurrent application programs. Such platforms include network processors and multimedia processors. The mounting complexity of multi-PE SoCs calls for a higher level of abstraction in the modeling approach. In this paper, we propose to model such SoCs as composed of two equally important coarse-grained types of components: the PEs and the On-Chip Communication Architecture (OCA) connecting these PEs. During the SoC design process, there exist a number of architectural choices for both types of components, e.g. the types of PEs, the interconnection topology and the flow control protocols. Making these choices needs to be guided by simulation. One of the central issues is to ensure the trustworthiness of the architectural models and the simulation results. As commonly observed, both the PEs and the OCA are highly concurrent hardware. In order to accurately model them, we need a formal and flexible concurrent model to capture their inherent parallel nature.

Currently the common practice to integrate the PE models and the OCA model is to use a wrapper to mediate between them [7, 10, 17]. Such approaches can complicate the design and negatively affect the simulation efficiency by adding another level of indirection [5]. In this paper we advocate a modeling style in which there is no intrinsic difference between PEs and OCA elements. Thus, this approach eliminates the need for an interfacing wrapper. Designers
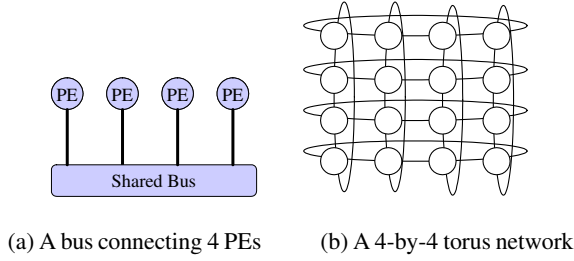
(a) A bus connecting 4 PEs   (b) A 4-by-4 torus network

Figure 1: Two interconnection network examples

only need to be familiar with one style of concurrency model to faithfully model and simulate the entire SoC.

Based on the above rationale, our system modeling environment focuses on the following two aspects of the prototyping and design exploration process: first, how to provide rigorous and flexible cycle-accurate performance models for different types of on-chip communication architectures ; second, and often more importantly, how to integrate the OCA models with existing timing PE models to provide a unified system-level simulation framework for multiprocessor based integrated circuits.

Thus, the contributions of this paper are twofold. First, we propose a novel modeling methodology of on-chip communication architecture based on a formal concurrency model, the Operation State Machine (OSM) [11]. Previously this model has been applied successfully in modeling microarchitectures of processors. Our work shows that the same principle can be also applied to model the concurrency inherent in the OCA, thus can help generate an accurate timing simulator. Second, we implement a unified modeling and simulation framework which integrates both the models for PEs and the OCA. By applying the same modeling style for both, this work enables designers to explore design options in a unified fashion, thus enhancing design productivity and shortening design turn-around time.

The remainder of this paper is organized as follows. In Section 2, we introduce some concepts and terminology of parallel systems and the OCA that we use in this paper. Then, in Section 3 we introduce the basic concepts of the OSM model and how it is used in modeling processing elements. This is followed by Section 4, where we focus on the modeling details of two on-chip communication architecture examples. To translate these modeling details into formal machine readable representation, in Section 5, we present our hierarchy of description languages. These specifications are in turn synthesized into a retargetable simulator for the full multiprocessor system. In Section 6, we present case studies demonstrating the validity of our modeling approach in comparison with previous studies. Section 7 then discusses prior related work and Section 8 concludes the paper.

2. SYSTEM ARCHITECTURE OF THE PE AND THE OCA

This section provides a top-down view of several interrelated aspects of the PEs, the OCAs and the interactions between them.

Two types of OCAs are commonly used in multi-PE SoCs - *viz.* buses and packet switched networks. A bus is usually composed of interfaces, arbiters and a shared backplane. The PEs are connected to the shared backplane through master and slave interfaces. The access to the backplane is arbitrated through the arbiters. Figure 1(a) shows a bus topology connecting 4 nodes. In comparison, a packet switched interconnection network consists of routers and links which are connected by a specific network topology. Figure 1(b) shows a 4-by-4 network with a torus topology.

The multiprocessor platform we consider here is a parallel architecture based on message passing, -*viz.* the PEs communicate with each other through passing *messages* via the OCA. Examples of this type of architecture include the RAW [12] and Nos-
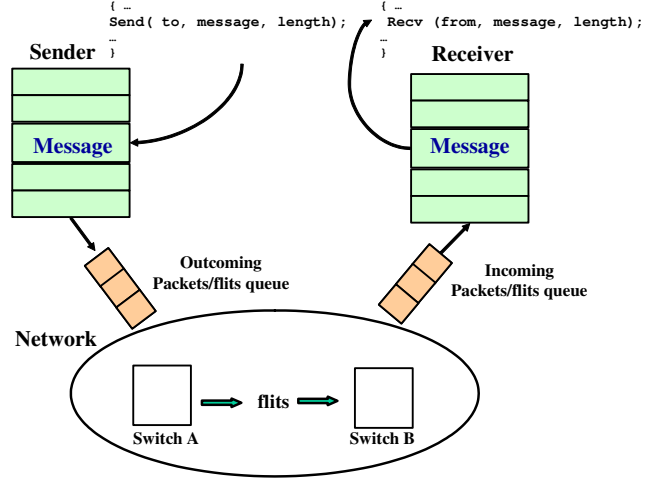

Figure 2: An example of detailed OCA actions during a matching *send* and *recv* pair

trum architectures [9]. To give a high-level overview of how the OCA interfaces with the PEs, we trace the path of a message sent from PE A to B. Figure 2 illustrates this process when it is implemented in a packet-switched network with a message-passing application model. The process starts with user programs communicating with each other using pre-defined message-passing library routines. Then, these library routines are compiled into PE instructions such as *send* and *receive*. Each instruction has a one-to-one correspondence with one OCA functional primitive defined in [16], e.g. *OCA_send, OCA_receive*. They are in turn understood by the OCA interface and translated by the OCA modules into specific OCA operation sequences. Each instruction has its own syntax and semantics. The whole transaction involves concurrent actions from various components of the OCA, such as switches, channels, send and receive interfaces and their associated queues. Furthermore, at any cycle, there are multiple transactions going on accessing different hardware resources to accomplish their actions.

The above description points to the requirements imposed on the ideal modeling environment. First, it should capture the inherent concurrency. Second, the model should enable designers to quantitatively analyze the system performance through simulation - *viz.* it should be able to model each functional unit and each OCA operation illustrated by Figure 2 in a trustworthy manner.

3. MODELING METHODOLOGY

3.1   OSM basics

The OSM model was initially proposed to model processors [11]. It is based on the observation that the execution progress of an operation, i.e. instruction, can be represented by a finite state machine. The states of the state machine represents the execution status of the operation, such as "decoding" or "in reservation station". The edges of the state machine represents the possible transition paths of execution.

The execution of operations requires both structural and data resources. The OSM model adopts the abstract notion of *tokens* to model these resources. The resource management policies in processors are represented by *token managers*. It controls a set of tokens of the same type and reacts to the token requests from the finite state machines through a common token transaction interface. These token requests are associated with the edges of the state machines and serve as predicates controlling their state transition.

Figure 3 gives an overall picture of the OSM model. It contains two layers, the operation layer and the hardware layer. In the operation layer, multiple state machines execute concurrently, each representing a machine operation in the pipeline of the processor. The
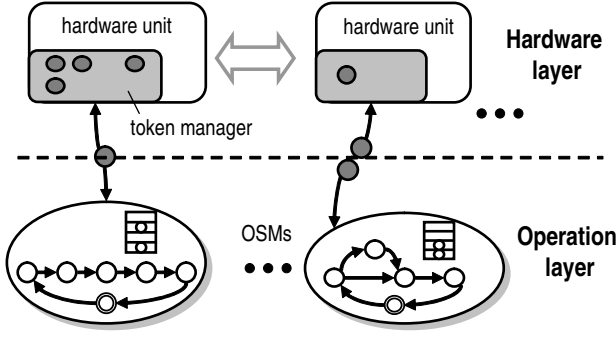
Figure 3: The layered representations of OSM based modeling

state machines compete for execution resources by sending their token transaction requests to the token managers in the hardware layer. The hardware units in the hardware layer communicate with each other under the discrete-event (DE) computation model. A hardware unit may contain a token manager as an abstraction of its control policy. The token managers respond to the requests from the state machines and collectively control their execution statuses. The key feature of the OSM model is that it captures the concurrency at both the hardware and the operation level as well as the interaction between them. The properties of the state machines are fully exposed by the state diagrams and the token transactions. Therefore, it is possible to analyze the state machines to extract these properties and verify them formally. This aspect of OSMs is beyond the scope of this paper.

The Mescal Architecture Description Language (MADL) is a description language for describing models using OSM. The language describes the state machines and their interaction with the token managers. A cycle-accurate simulation framework has been implemented based on MADL. The framework has been used to generate very fast simulators for several types of processors [11]. In this framework, we used a clock driven kernel for the hardware layer, which is a simplified case of DE for fully synchronous systems. In a more general implementation, we can model the hardware layer using SystemC.

3.2 Why OSMs?

Communication architectures and computation architectures are usually two distinct domains. In today's computation architectures, pipelines are implemented to execute streams of instructions to minimize computation latency. The microarchitectural components here are ALUs, registers, caches, etc. On the other hand, highly complex communication architectures contain efficient datapaths so that the intended data can be delivered to the destination as fast as possible. Here the commonly used microarchitectural components are communication interfaces, switches/routers, buffers, arbiters, etc. Communication architectures display high degrees of concurrency as is the case with computation architectures, i.e. processors. For example, most modern buses utilize pipelined data transactions. Also, a router in a packet-switching network usually has a multi-cycle pipelined operation including buffering, arbitration and data transfer stages. Thus, the data delivery mechanism in the OCA is analogous to the pipelined operation execution in the processor. Therefore, we can model the communication processes in a way similar to the computation processes. We can identify the comparable operation level of the OCA as a concurrent layer involving data delivery. At the same time, the OSM model preserves the necessary microarchitecture information in its hardware level. The two levels of concurrency, the operation level and the microarchitecture level, are tightly coupled through a common resource transaction interface. This allows us to completely capture the cycle-accurate behavior of the OCA.

In addition, a unified modeling style enhances the productivity for system-level design. Designers only need to be familiar with one style of modeling to model both domains, the PE and the OCA. The unified approach avoids the otherwise necessary interface be-

tween these two parts and increases the potential of simulation speed improvement.

3.3 Decoupled Control Path and Data Path

Modeling using OSMs provides us a natural way to decouple the data path and the control path of the OCA. The state transitions of the state machines represent the progress of data transfers inside the OCA, while token (representing resources) transactions specify the corresponding control conditions for these state transitions. The token managers implement the control policies. These two parts, namely, the state transitions and the token transactions, fully specify the function and timing of OCA actions. The separation between the control path and the data path enables the designer to cleanly distinguish these two aspects. There are two advantages from this separation. The design exploration of control and data paths can be done separately and in the final design they can be integrated together seamlessly. This results in the shortening of the design turn-around time and improves design reuse. Second, it simplifies the tasks of debugging and verification.

4. DETAILS OF MODELING OCAS WITH THE OSM

4.1 Operations and Resources in OCAs

To model the microarchitecture of OCAs with OSMs, first we need to choose the granularity of the OCA operations to model. There exist a number of choices. We use a packet-switching network as an illustrating example. One obvious choice *operation* is an operation as the complete multi-hop packet/flit transmission process. If we consider each source and destination pair of the transmission, we need at least $n(n-1)$ OSMs for all the packet transmission operations on an n-node network. An alternative candidate is to use one operation for each hop during the transaction. In this way, we need only n OSMs for n nodes. In order to generate the most succinct and efficient MADL description possible in our synthesis flow, we employ the "single-hop operation" scheme even though both are functionally equivalent.

The OSM resources are clearly identified as the hardware structural resources inside the OCA, such as buffer entries, input/output ports, arbitrated data paths, etc. The resources are represented by the token managers, each of which corresponds to a structural component in the OCA. These token managers accept or reject token transaction requests to implement their resource control polices. In implementation, various types of token managers are organized into a C++ template class library to enable reuse across different types of OCAs. Since the token transaction interface is standardized in the OSM model, designers can achieve a high level of reusability.

4.2 An OSM Model for a Pipelined Router

The flexibility of the OSM model enables us to model the same microarchitecture in a number of different ways. One of the contributions of our research is to explore these different modeling options and to find the most suitable approach for accurate and efficient simulation. As illustrated in Figure 4(d), the structure of a typical pipelined network router is composed of ports, buffers, schedulers and crossbars. When a flit, i.e. the smallest unit of a packet, arrives at the input port of the router, it is in turn stored in the corresponding buffer if available. Then the buffer requests the *scheduler* for a data path inside the crossbar so that the flit can be routed to the desired output port. After the scheduler accepts the request, the flit is sent to the output port in the next cycle.

In the above description, there exist two types of key resources, the buffer entries and the crossbar data paths. The crossbar data paths are controlled by the scheduler. In the OSM terminology, these two types of resources are modeled as tokens regulated by token managers. Figure 4(e) shows an OSM that we used to model a hop of the flit within a 5-port pipelined switch with a 5x5 crossbar as shown in Figure 4(d). The OSM has 3 states in addition to the initial state, representing its 3-stage pipelined execution. After the flit arrives at the input port of the switch, the OSM is activated and progresses from State I to State B upon allocating a token from the buffer token manager TM_{buffer}. In the next cycle, the OSM tries
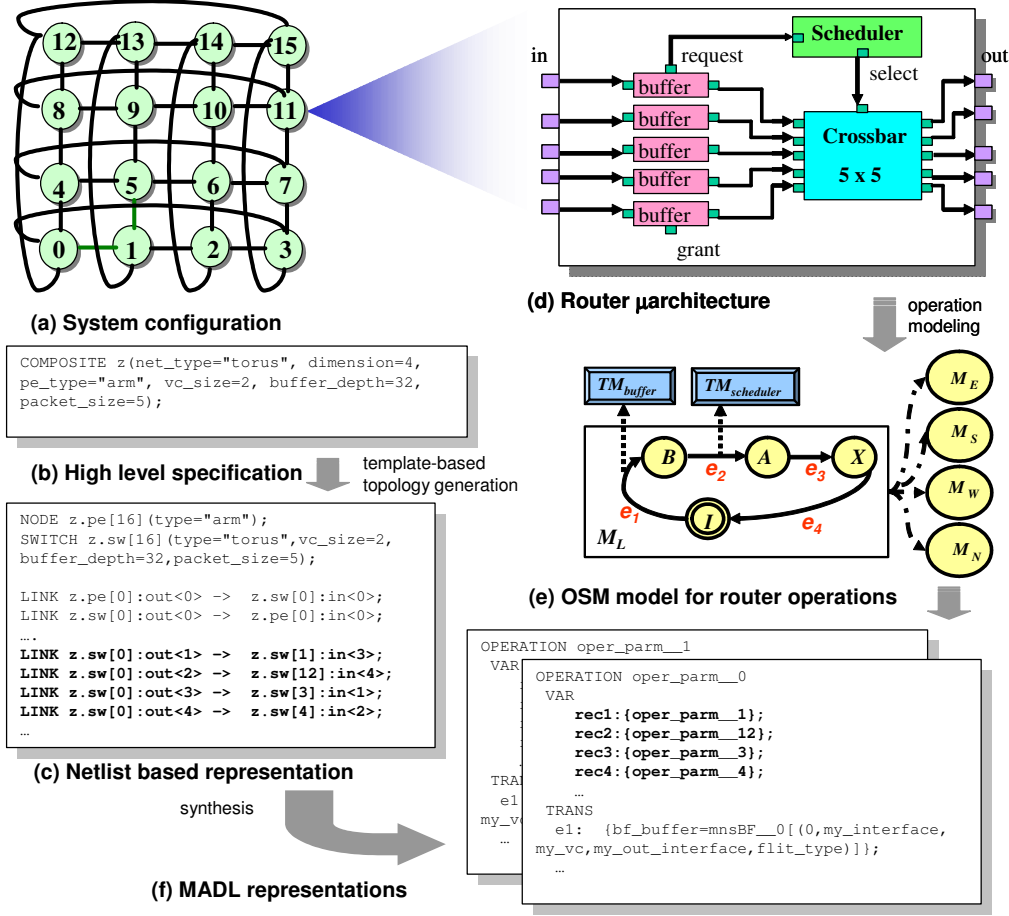
Figure 4: Illustrated example of modeling a 4x4 torus packet-switching network

to allocate a crossbar path by sending an allocation request to the token manager $TM_{scheduler}$ and advance to State A. Note that multiple such OSMs may be running at the same cycle in the switch. Therefore $TM_{scheduler}$ may receive more than one request. Upon receiving the requests, the scheduler decides which request(s) to grant according to its scheduling policy. The successful requesting OSM(s) will receive the token(s) and proceed to State A. Then after one cycle delay in the crossbar (through State X), the flit will be routed to either the local data path or a neighboring node. The local OSM thus finishes its task of modeling one hop of the flit and retires itself by returning to State I. In the former case, the flit ends its life cycle by delivering its tagged data to the PE data path. In the latter case, an OSM of the identical structure will be activated in the neighboring node. Then, in the neighboring node, the above process replicates with a different context. The process continues until the flit reaches its final destination node.

In all, whenever a flit arrives at a switch, one corresponding OSM is activated and competes with other executing flit-OSMs for resources - i.e. *tokens*. If the flit needs to be routed to another node, the current OSM retires itself after its execution finishes and another OSM representing the next hop is activated. Take for example the life time of a flit which traverses from $NODE_0$ to $NODE_5$. Suppose that its route is as: $NODE_0 \rightarrow NODE_1 \rightarrow NODE_5$. So at first an OSM M_0 is generated at $NODE_0$. After a few cycles, when the flit arrives at the output port of $NODE_0$, M_0 retires itself and activates M_1 at node N_1. The process repeats along the route until M_5 retires itself, denoting that the flit has been received by the destination node - i.e. $NODE_5$.

These token transactions are in turn recorded in the MADL description highlighted in Figure 4(f). Notice the paragraph with bold fonts corresponds to the netlist representation (also in bold fonts) in Figure 4(f).

4.3 An OSM Model for a Pipelined Bus Architecture

There exist many similarities between modern on-chip bus architectures and packet-switching networks. Both employ highly concurrent, pipelined hardware to achieve high data throughput and low transmission latency.

In bus architectures, there are two basic types of bus operations, *read* and *write*. It is straightforward to model them with corresponding OSMs. The modeling task thus becomes how to represent these hardware actions as OSM token transactions.

As illustrated in Figure 5, a multi-beat bus write transaction can be modeled as one multi-branch OSM *operation*. At the start of the transaction, the OSM changes its state from the initial state I to State B, when the master interface stores data contingent on the buffer availability. Next the master interface requests an arbitration token from the token manager $TM_{arbiter}$, which usually implements a round-robin arbitration policy for accessing the bus backplane. Upon the successful grant of the bus backplane, the transaction goes into the data transferring stage, represented by the states T_i. Since there exists the possibility of multiple beat transactions, we use the multi-branch state diagram as shown in Figure 5 to model the transaction. The different branch edges from State A, denoted by e_1, e_2, e_3, are used to model quad-beat, dual-beat and single beat transactions, respectively.

4.4 Summary and Discussions

Just as we extract the *operation concurrency* in the parallel execution of microprocessor pipelines, we identify the *operation concurrency* in the above OCA examples as the following.

- For the packet switching network case, we treat the parallel packet/flit processing at each node as the *operation concur-*
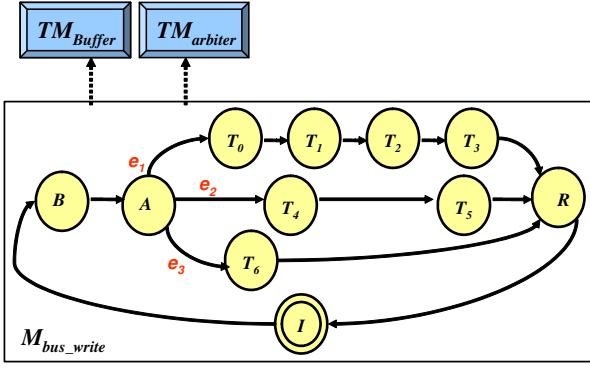
Figure 5: The OSM model of a bus write operation

Figure 6: The language hierarchy for OCA specifications

rency.

- For the bus case, concurrent bus read and write operations are taken as the *operation concurrency*.

These two particular choices are just two of the many possibilities in identifying the *operation concurrency* in applying the OSM principle to model OCAs. We note that the execution of each *operation* is sequential, but there exists plenty of concurrency in the collective execution of these operations running in parallel.

Although our current OCA models utilize one global clock domain, it is straightforward to extend the work to cover multiple clock domains. In such a case, the state machines are divided into multiple groups according to their local clock domains. A group is activated for the transition of its states at the local clock edge. Further, in the OSM model, the scheduling order of the state machines may affect the execution outcome. In our implementation, the state machines are ordered according to their ages, or the durations since they become alive. With this scheme, an older state machine has higher priority to access resources than a younger state machine by default. However, if a hardware component implements a different policy, the corresponding token manager can be customized so that it "hand-picks" its preferred state machine. This way, the approach is flexible enough to model any desired control policy.

5. HIERARCHICAL MACHINE DESCRIPTION LANGUAGES

One of the key observations in system-level design flow is that designers need various levels of flexibility in specifying the architecture and microarchitecture of SoCs. To better support this requirement, we propose a hierarchy of description languages for the OCA specification.

At the top level, system-level designers only need to specify the most basic system parameters when a rough estimate is required of what the performance of the intended SoC would be for a specific application. For example, the designer may be evaluating the following design candidate, explained in plain English:*"A multiprocessor with 9 ARM processors connected by a 2D torus network with virtual channel routing"*. The first-level description language accomplishes exactly this task - *viz.* it provides a convenient way of describing template based system-level characteristics, such as topology, network type, PE type, etc.

At the middle level, a net-list based specification is provided to enable more detail-oriented OCA designers to tune the system topological interconnections between various components, including PEs and OCA modules. For example, if one design calls for the elimination of one of the wrap-around links in a torus network, designers can find this particular link in the mid-level description generated from a torus template of the level above, and then textually delete it from the specification.

At the lowest level, OCA designers have the possibility to fine tune the microarchitecture of the OCA components, such as the type and parameters for pipelined switches, the time lag between
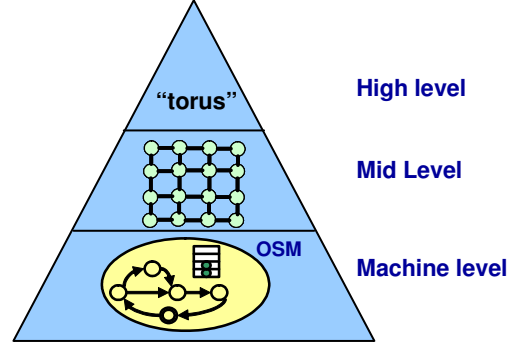
the request and the grant for an bus arbiter, etc. This level of detail can be expressed in the OSM representation of the architecture as illustrated in Section 4.

The relationship between these 3 levels of specifications is illustrated in Figure 6. Seen from top to bottom, the complexity increases and the abstraction level decreases. In order to integrate these 3 levels of specification, we have developed a toolsuite based on the synthesis flow from the top level down to the microarchitectural level. After the lowest level description is generated, the simulator synthesizer reads the description, which is a valid OSM model representation, and automatically generates a customized system simulator.

6. CASE STUDIES

To evaluate the efficacy of our OSM based OCA modeling approach, we conducted several case studies. In these case studies, our automatically generated system simulators are validated against the hand-crafted simulators from previous studies for design explorations of OCAs [13]. For the virtual channel based routing example, we validate against the experiments detailed in [13]. For the bus example, we validate against the experiments detailed in [17]. For each type of OCA we pick one or several representative system configurations for comparison. Although we have an integrated PE model based on the OSM to simulate multiprocessor embedded applications, in our case studies, we used a statistical data traffic generator which is widely used to evaluate interconnection networks [13]. In the experiment setup, the packet switching network uses credit-based flow control and fixed source-routing.

The synthesized simulators are compiled by g++ 3.3.2 with compilation flags "-O3 -finline-limit=3000 -fomit-frame-pointer". The simulations are run on a P4 2.8GHz machine with Linux. In order to measure the performance of the synthesized simulator, the global system cycle counts are used to calculate the simulation speed. The following is a brief description of each system configuration listed in Table 1.

- *TORUS_16_VC_2*: A 16 nodes multiprocessor connected by a 4-by-4 torus network. The network routers are 5x5 crossbar based virtual channel (VC) routers [3] with 2 VCs per channel. There are 32 buffer entries for each input buffer inside the routers. Each packet is composed of 5 flits. Each router has a 3-cycle pipeline for buffering, arbitration and crossbar transferring, respectively.

- *TORUS_16_VC_4*: A torus network connecting 16 nodes with 4 VCs per channel. The rest of the parameters are the same as *TORUS_16_VC_2*.

- *BUS_4*: A simple pipelined bus architecture connecting 4 nodes. The bus arbiter is a round-robin arbitration unit with a minimum one-cycle delay. A normal single beat write operation with no outstanding requests takes 3 cycles to complete.

Table 1 reports the experiment results in *simulation speed, length of machine level configuration (the lowest level)* and *maximum and average number of concurrent OSMs during the simulation* for each

Table 1: Simulation Results

Configuration	Simulation speed (cycles/sec)	Description length (lines)	# OSMs running (max/avg.)
TORUS_16_VC_2	6.58K	1418	776/23
TORUS_16_VC_4	4.10K	1418	1131/30
BUS_4	425.6K	292	36/2

configuration. The first two metrics represent the simulation and description efficiency and the last metric measures the level of concurrency inherent in our simulation models. We attribute our relative speed advantage to the rather high level "transaction-centric" modeling style, similar to the common approach of "Transaction Level Modeling" even though we provide a cycle-accurate communication channel model. Also our results confirm that there exists abundant operational concurrency in the OCA, especially for packet-switching based OCAs. Finally we notice that the length of configuration *TORUS_16_VC_4* is the same as *TORUS_16_VC_2* since there is only a simple change of parameter value.

7. RELATED WORK

There is a growing number of research publications dealing with the general topic of network-on-chip and custom on-chip communication architectures. To our knowledge, there has been no similar work where a full "model synthesis+simulation" flow for multi-PE processor design exploration is publicly released. [7] discusses the development of a SystemC-based cycle accurate simulator for a cluster of ARM processors. OCCN [2] is an open-source research framework for OCA modeling based on SystemC. Even though both of them and similar work in [15] target the design exploration of OCAs, currently they only examine the on-chip bus and its variants as OCA candidates. [6] introduces a tool to generate a SystemC-based multiprocessor SoC representation for simulation from a library of parameterized OCA components. Currently [6] does not include a PE model for application driven simulation and it focuses on packet-switching network as OCA candidates. Sharing similar goals in designing heterogeneous embedded systems, [8] also presented a multiprocessor SoC design flow. In comparison, we focus more on the communication architecture modeling while employing a wrapper-less approach.

Performance models for interconnection networks can be generally classified into three categories, analytical models, simulation models and empirical models. Our model falls into the second category. There exists extensive research dealing with building accurate simulation model for interconnection networks, e.g. Timed Petri Net (TPN) Models [14]. Compared to TPN, OSM is more domain specific, i.e. it targets the modeling of clock-driven concurrent hardware components. This makes the modeling process more straightforward and convenient. In addition, the Discrete Event (DE) model is still popular to model interconnection networks [4]. In comparison our approach has the advantage of proposing a description from which formal inter-component relationships can be extracted, analyzed and verified. Further this presents a unique potential for further validation and verification effort for the OCA models together with PE models. Finally, as an abstract model for system-level specifications, OSM enjoys significant simulation speed advantage over DE based models such as SystemC-based models. One drawback is that OSM does not directly support hardware synthesis due to its high level of abstraction.

8. CONCLUSIONS

This paper presents a modeling technique which captures both the operation level and microarchitecture level concurrency for on-chip communication architectures. We introduce the Operation State Machine (OSM) as our concurrency model for highly parallel communication hardware. Using the OSM model, the two components of multi-PE SoCs, namely, the PEs and the OCA, can be treated in an undifferentiated fashion. This avoids the overhead in-

curred by the otherwise unnecessary interface between them. Thus, the OSM model serves as a formal foundation for the efficient modeling of multi-PE based processing platforms. Our case studies show that the OSM models can faithfully represent the functional and timing behaviors of the OCA microarchitecture. Moreover, this achieves a good balance between model complexity and simulation efficiency. In order to enhance the design productivity, we have developed a complete synthesis and simulation toolsuite to automatically generate a machine specific simulator for multi-PE SoCs. The tool has been publicly released for peer review [1]. Compared to other existing methods, our approach can capture the inherent concurrency in the OCA more explicitly and conveniently. It also provides a unified cycle-accurate modeling framework where the models for both the PEs and the OCA are integrated seamlessly.

9. REFERENCES

[1] Released Simulator. http://www.princeton.edu/~ mescal/software.html.

[2] M. Coppola, S. Curaba, M. D. Grammatikakis, and R. Locatelli. OCCN: a NoC modeling framework for design exploration. *Journal of Systems Architecture*, 50(2-3):129–163, 2004.

[3] W. J. Dally. Virtual-channel flow control. *IEEE Transactions on Parallel and Distributed Systems*, 3(2), 1992.

[4] K. Fall and K. Varadhan. ns notes and documentation. *The VINT Project, UC Berkeley, LBL, USC/ISI, Xerox PARC*, 2000.

[5] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Publishing Company, New York, NY, 1995.

[6] A. Jalabert, S. Murali, L. Benini, and G. De Micheli. xpipescompiler: A tool for instantiating application specific networks on chip. In *Proceedings of 2004 Design Automation and Test in Europe Conference (DATE 04)*, 2004.

[7] M. Loghi, F. Angiolini, D. Bertozzi, L. Benini, and R. Zafalon. Analyzing on-chip communication in a MPSoC environment. In *Proceedings of 2004 Design Automation and Test in Europe Conference (DATE 04)*, 2004.

[8] D. Lyonnard, S. Yoo, A. Baghdadi, and A. A. Jerraya. Automatic generation of application-specific architectures for heterogeneous multiprocessor system-on-chip. In *Proc. Design Automation Conference*, 2001.

[9] M. Millberg, E. Nilsson, R. Thid, and A. Jantsch. Guaranteed bandwidth using looped containers in temporally disjoint networks within the nostrum network on chip. In *Proceedings of 2004 Design Automation and Test in Europe Conference (DATE 04)*, 2004.

[10] P. G. Paulin, C. Pilkinton, and E. Bensoudane. StepNP: A system-level exploration platform for network processors. *IEEE Design & Test Computers*, 19(6), 2002.

[11] W. Qin, S. Rajagopalan, and S. Malik. A formal concurrency model based architecture description language for synthesis of software development tools. In *Proceedings of ACM SIGPLAN/SIGBED 2004 Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES'04)*, 2004.

[12] M. B. Taylor, J. Kim, J. Miller, D. Wentzlaff, F. Ghodrat, B. Greenwald, H. Hoffman, P. Johnson, J.-W. Lee, W. Lee, A. Ma, A. Saraf, M. Seneski, N. Shnidman, V. Strumpen, M. Frank, S. Amarasinghe, and A. Agarwal. The Raw Microprocessor: A computational fabric for software circuits and general-purpose programs. *IEEE Micro*, 22(2), 2002.

[13] H.-S. Wang, X. Zhu, L.-S. Peh, and S. Malik. Orion: A power-performance simulator for interconnection networks. In *Proceedings of the 35th International Symposium on Microarchitecture (MICRO)*, 2002.

[14] J. Wang. *Petri Nets: Properties, Analysis and Applications*. Kluwer Academic Publishers, Boston, MA, 1998.

[15] A. Wieferink, T. Kogel, R. Leupers, G. Ascheid, H. Meyr, G. Braun, and A. Nohl. A system level processor/communication co-exploration methodology for multi-processor system-on-chip platforms. In *Proceedings of 2004 Design Automation and Test in Europe Conference (DATE 04)*, 2004.

[16] X. Zhu and S. Malik. A hierarchical modeling framework for on-chip communication architectures. In *Proc. International Conference on Computer-Aided Design*, 2002.

[17] X. Zhu and S. Malik. Using a communication architecture specification in an application-driven retargetable prototyping platform for multiprocessing. In *Proceedings of 2004 Design Automation and Test in Europe Conference (DATE 04)*, 2004.