OBJECT-ORIENTED SYSTEM-ON-NETWORK-ON-HIP TEMPLATE AND IMPLEMENTATION

[1]Anandaraj Shunmugam, [2]Ramkumar Ramaswamy
Lecturer II,
Department of Computer Science and Information Technology, DMI- St. John the Baptist University,
Mangochi, The Republic of Malawi

Abstract-Network-on-chip (NoC) technology enables a new system-on-chip paradigm, the system-on network-on-chip (SoNoC) paradigm. One of the challenges in designing application-specific networks is modeling the on-chip system behavior and determining on-chip traffic characteristics. A universal object message level model for SoNoC was defined and an object-oriented methodology was developed to implement this model in hardware and software. The model supports "object to core" synthesis and "function invoking to network" mapping. A case study of an H.263 system verifies the model and methodology. System prototypes are easily built and on-chip traffic can be observed using the SoNoC model to provide real benchmarks for on-chip network design.

Key words: network-on-chip; system-on-chip; system-on-network-on-chip

I.INTRODUCTION

As feature sizes shrink, system-on-chips (SoCs) integrate more and more cores, and intra-connections between cores become more complex. Peer-to-peer links utilized in application specific integrated circuits (ASICs) tightly couple different parts of a chip and are difficult to route on two-dimensional silicon surfaces, and bus-based structures cannot provide scalable communication when there are too many cores. Global wires cannot be owned privately or shared publicly, but should be owned or shared segment ally, which is what networks do. Network-on-chips (NoCs) enable SoCs to integrate more cores by providing scalable, flexible, and reliable intra-connections. In the system-on-chip environment, application-specific on-chip networks are possible with the systems providing information to analyze the given communication requirements, and are necessary with the chips limiting the resources that NoCs can utilize. One of the advantages of NoC-based multi-core SoCs is parallel execution. To develop parallelism and utilize NoC services, cores in the system on- network-on-chip (SoNoC) should be enhanced with new features. SoNoC cores with a single thread model and single message pool may not work well to send, receive, and reclaim messages correctly. A new thread model should be applied to SoNoC cores to avoid deadlocks in execution. The first step in designing a system is partitioning it and the first step in designing a network is determining the traffic pattern. Object-oriented methodology provides human-written specifications. Within a SoNoC, cores are regarded as objects with data and function members and can be synthesized from objects by using some basic principles; network characteristics are determined by the function calls between objects and can be generated from the call map. As Fig. 1 shows, SoNoCs are unified as an object message model with objects and networks synthesized from specifications.

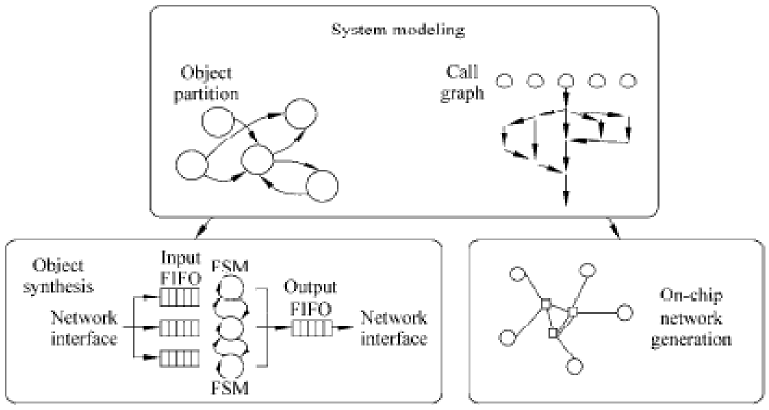

Fig. 1 Object-oriented design methodology for SoNoC: Begin with an object message model, then extend the object to module synthesis and function calls to network mapping

Synthesis-based on-chip network design approaches [1,2] have been studied in various ways, but one of the difficulties in NoC synthesis is the on-chip traffic pattern modeling. Because SoNoCs are much more application-specific, general synthesized stochastic
Traffic patterns are not suitable for NoC analyses. Several NoC simulation frameworks have been researched [3-5], but these works focused on network architectures and protocols, and the traffic was artificially synthesized. This paper presents how to build aReal SoNoC prototype to obtain real traffic patterns, which will facilitate on-chip topologies and protocol syntheses. For a SoNoC to succeed in heterogeneous multi-core applications, the system and network must be considered and integrated together. A unified component integration flow developed for multi-core applications in Ref. [6] is a bottom-up flow that provides wrappers for the hardware and software components. The mapping of object's function call into messages [7] was the inspiration for this paper. Such research has investigated system modeling using formal language such as UML and SysML [8,9]. These works focused on formally describing a system to define its behavior and architecture requirements. The object message level (OML) model presents a SoNoC hardware template enabling a consistent system module synthesis process.

II. OML MODEL FOR SONOC

The register transfer level (RTL) model provides a fundamental basis for logic synthesis in ASIC designs. In the RTL model, circuits are regarded as registers with combinational logic used to model signals starting from registers, traveling through logic gates and getting captured by registers at the next clock edge. In the object message level (OML) model, systems are regarded as heterogeneous cores with micro-networks having messages being transmitted through networks between cores. The OML model enables a unified SoNoC form, which can be used to develop object-tocore synthesis and function call-to-network mapping methodologies.

III. ABSTRACTION OF COMPUTATION AND STORAGE:

Objects

An object is a collection of public/private data and operations with an object-ID (OID) attached:

- Public function members: submit tasks or trigger state shifts.
- Private function members: provide a library for public functions or conduct initialization and self checking.
- Private data members: representing the implicit configuration and status or temporary data and intermediate results.
- Public data members and property access functions: representing explicit configuration and status. Special access functions are defined to access these properties by messages.

The only way to access an object is to invoke its member functions. Each public function has a function- ID (FID) attached to it, to identify different member functions.

IV. ABSTRACTION OF INVOKING FUNCTIONS: MESSAGES

When an object source invokes another object sink's member function with a group of parameters, the source calls the function: rtnValue = Sink.Function (Parameters). In SoNoC, this function invoking will be packed into a message: (OID=Sink, ID=Function, Parameters, OID=Source, tag). When the sink object receives the message and finishes the task, it will send back a return message to notify the source: (OID=Source, FID=0, tag, rtn Parameters). Each message is marked by a tag; therefore, the source can send and wait for many messages at the same time to achieve parallel execution. Therefore, all messages are formatted as (OID, FID, and Parameters) and are executed by a member function with an FID. If the FID=0 the message is defined as a return message and is handled by a special built-in "wait" function. A message with an FID≠0 is defined as an invoking message. Objects that send invoking messages and receive return messages are called active objects; objects that only receive invoking messages and send return messages are called passive objects.

V. SONOC HARDWARE THREAD MODEL

SoNoCs are multi-thread systems synchronized by messages and limited-thread systems because each object has limited thread resources. The sending of an invoking message forks a thread; receiving a return message joins a thread. SoNoCs support the simultaneous transmission of several messages with multi-core execution. Within a core, tasks can also be executed in parallel with the application and release of object resources synchronized by semaphores within objects.

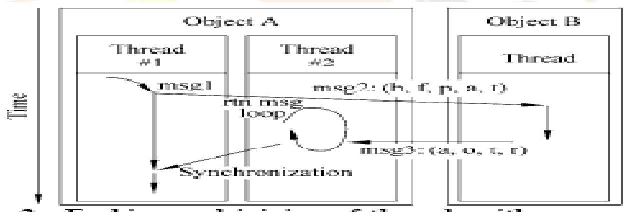

Fig. 2 Forking and joining of threads with messages. Object A executes msg1 in Thread #1, sends msg2 to B, and reclaims msg3 in Thread #2 to avoid deadlocks because Thread #1 cannot be re-entered.

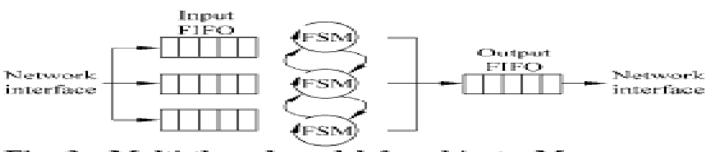

Fig. 3 Multi-thread model for objects. Messages are dispatched into input FIFOs, pumped and executed by FSMs, and new messages are pushed into output FIFOs. FSMs are synchronized by semaphores such as message tags.

Figure 2 shows a sequence of mesages.

Object A has received message msg1 and triggered function f1.During this execution object A called object B function f2 by sending message msg2 and is waiting for it to return. When B has finished f2, the return message msg3 is sent back to A, but A is trapped in the execution of msg1 waiting to receive a message back. Therefore, if A has only one thread, messages msg1 and msg3 depend on each other to continue. Active objects should have at least two threads: one for sending and waiting messages and one for reclaiming messages. The two threads are synchronized by message tags which are semaphores representing the three message states, sent, reclaimed, and null (used). Tags are attached to messages and follow the message transmission. In general, objects can have more than two threads, and this is true even for passive objects. Each thread extracts messages from the network interface, executes them, and sends other messages. Threads share resources that are synchronized by the semaphores.

VI. SONOC HARDWAREARCHITECTURE

SoNoCs are heterogeneous multi-core systems with application-specific network infrastructures. Unified hardware model design principles were developed for the synthesis of objects, network interface design, and network topology generation to develop SoNoC design methodology.

Implementation of objects: Object synthesis

Many forms are used to implement objects on silicon. These forms are generally categorized into hardware and software + processor.

Hardware implementation

A typical active object consists of storage and computational resources, several task finite state machines (FSMs), a reclaiming FSM, and message tags, as shown in Fig. 4. A hardware core has storage resources including registers, register files, and embedded RAM or ROM that are used to map data members, computational resources such as the arithmetic logic unit (ALU), multiplier, and divider, and communication resources such as a network interface and FIFO. A member function is mapped into a sub finite state machine (sub-FSM) with a "start" signal in and a "done" signal out. Each sub-FSM commits interfaces to the resources as they utilize them. Synchronization points within each member function provide time constraints for computations with high level synthesis techniques for the design. The network interface dispatches the messages into two FIFOs, the task FIFO for invoking messages and the return FIFO for return messages. The task thread is an FSM attached to the task FIFO, which monitors the FIFO, triggers the appropriate function sub-FSM according to the FID contained in the message, switches the resources to the function sub-FSM, and waits for termination of the sub-FSM. Function sub-FSMs share resources under the control of the task thread FSM. Message tags are semaphores with sent, reclaimed, null (used) states and attached buffers. When function sub-FSMs send messages, they apply a null tag first, attach it to the message, and switch the tag to the "sent" state after the message is sent. Then they wait for the message to be sent back, indicated by the tag state changed to "reclaimed". Reclaimed threads are FSMs watching the return message FIFO, which changes tag states to "reclaimed" according to the tag field of the received messages and saves return parameters to buffers addressed by the tags. When task FSMs have received valid return messages and extracted the return parameters, the tag is changed to the "null" state. In this way, message tags are semaphores that protect return messages and parameter buffers. Passive objects do not need reclaimed threads, so this discussion about active objects is also true for passive objects with some abridgments.

Implementation on a processor

Objects can also be mapped into software running on processors. Processors have larger storage, computational and semaphore resources, and the ability to simulate multi-thread environments in software. With unlimited thread resources, software can allocate multiple threads even for the same member function, which is impossible for hardware. Therefore, only one input FIFO will satisfy message dispatching requirements. Extensions to processors include an input FIFO, an output FIFO, and network interfaces. Since processors are mega-cells with tremendous resources within a chip, this implementation approach is not the emphasis of this study.

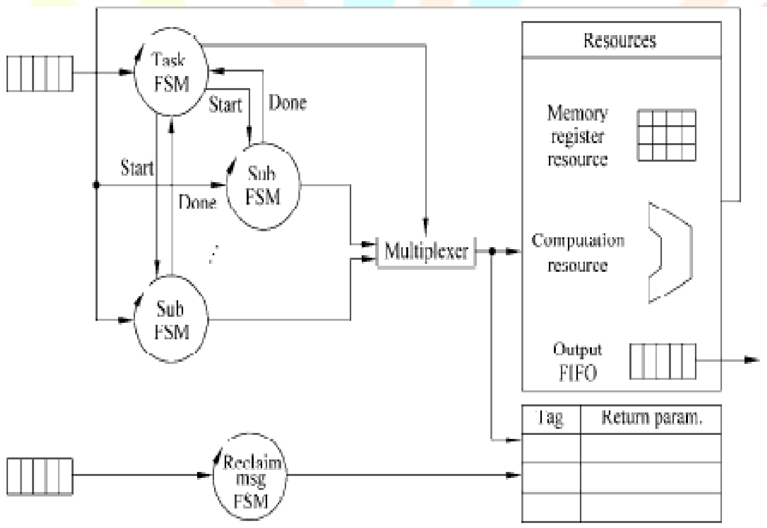

Fig. 4 Active object hardware architecture. Message tags synchronize the two FSMs. Storage and computation resources are shared by sub-FSMs in the task FSM.

Implementation of message delivering: NoC generation

Function calls between objects reflect relations and traffic constraints of objects. Design of a NoC according to these constraints must apply principles and templates to network interfaces, protocols, routing algorithms, and topologies. Ma and Sun proposed designing a NoC using an evolutionary method based on the information from the OML. Variable length messages, a source-based routing scheme, arbitrary topologies, and configurable switches can be used to evolve the NoC for specific applications.

Implementation of network interface

Network interfaces send messages in order, handle variable length messages, and translate object IDs into network addresses. A network interface includes several signals as shown in Fig. 5: req, ack, tail, and data. The "req" signal indicates that the data is valid, and the "tail" signal, used in wormhole routing, indicates the last flip being sent, so these two signals can be used to build and remove a virtual circuit and to transmit flip sequences of any length. The signal "data [BW−1:0]" is BW bits wide, and contains the routing information and/or data payloads. The signal" ack" is an acknowledge signal that when true, indicates that "data[ ]" is allowed to be updated. Figure 5 shows a possible sequential pattern when sending a message "12345". The object ID must be translated into a network addressable ID to apply routing algorithms. The source-based routing scheme takes the path from the source object to the sink as the sink's ID to the source. Therefore, one object has different network IDs for different objects. The two methods to resolve this translation are equipping a ROM addressed by an OID to get its network ID or taking the network ID as the OID at the very start

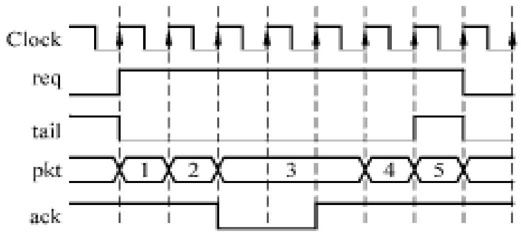

Fig. 5 Network interface signals. A sequence of 5 flips takes 7 clocks because "ack" suspends 2 clocks.

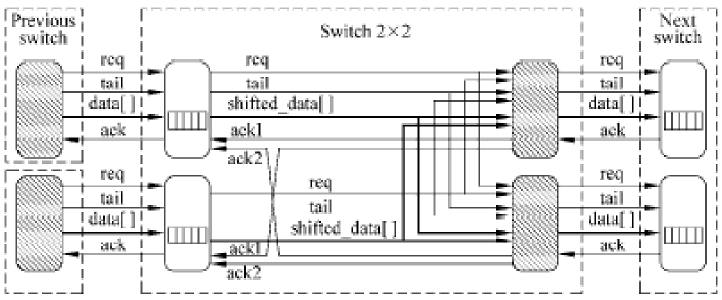

Fig. 6 Parameterized switch for an arbitrary topology for a 2×2 case. Flexible switches can be generated from this template.

NoC topology design

Topologies dominate the performance of NoC systems. Many specific applications demand arbitrary topology design methods. Arbitrary automatic network generation automation has been developed to quickly explore the design space[11]. The source-based routing scheme supports arbitrary topologies, simplifies switch structures by known routing results, and regulates switch templates. The wormhole scheme is employed to transmit arbitrary sequence lengths. Figure 6 shows a typical 2×2 switch generated from a parameterized template. Interfaces between switches are the same as described in Section 2.2.1. The template parameters can be graded into network, switch, and port (input port or output port) hierarchies. At the network level, the parameters include the number of input ports, thenumber of output ports, the number of switches, and the connection edges. At the switch level, parameters include the number of input ports, the number of output ports, and a virtual circuit mapping table. More detailed parameters are attached to ports. Simulation and logic synthesis experiments have shown the impacts of these parameters.

VII. SONOC SOFTWARE PROTOTYPE FRAME WORK

The OML model for SoNoC can be mapped into hardware and is also suitable for software prototypes. SystemC, a hardware description language based on C++ for system modeling, was used here to construct a SoNoC software framework.

Proposed modeling stages

There are four modeling stages or thread models for SoNoCs.

 For the no-thread model, systems are modeled as collections of objects. Functions called and parameters returned are processed on the program stack.

 For the single-thread model, only one message is transmitted on the network and only one module is active. Each function call transmits the active state from one module to another.

 For the unlimited thread model, messages are spread over the network and modules run in the unlimited multi-thread mode for the software simulation.

 For the limited thread model, networks work in the multi-message mode but modules run in the limited multi-thread mode, which coincides with the real SoC hardware model.

C++ program framework for SoNoC

Base classes are defined including network, module, functor, and message classes. A functor is a C++ structure recording a function's parameters and returned values, which is equipped for each member function. When other objects call this object's function, a message is created recording the source, sink, and functor, which is sent to the network and dispatched to the sink. The sink invokes functors to trigger appropriate tasks. Figure 7 shows the code to translate a function call into a message transmission. All objects are derived from the module class to provide message sending, waiting, and reclaiming abilities. The module also defines the interface to network; therefore, all message activities are managed and monitored by the network. The times are managed in the SystemC scheme; therefore, traffic patterns in the time domain can also be analyzed.

```
Message msg(
Sink, new Sink::Funtor(parameters),
*this, new WaitFunctor( ) );
post(msg);
// time elapsing.
WaitFunctor fs = waitmsg(msg);
Sink::Functor* ft = fs.callee;
rtn = ft->rtn.
```

Fig. 7 C++ pseudo-code for source sending and waiting for messages. Constructing, posting, waiting for,and using a message are formatted and recorded.

VIII. CASE STUDY: H.263 CODEC

An H.263 coder and decoder system was studied to verify this object-oriented methodology for SoNoC. This example shows that multi-core execution in SoC and traffic on NoC can be organized, managed, and optimized in both the time domain and the space domain.

System object message model

H.263 is a video communication standard. As Fig. 8 shows, the H.263 codec is composed of 17 objects. Since this is a system simulation, the codec includes virtual objects such as "Display" and "Camera" that are not integrated into the real chip, but with real traffic of virtual objects across the chip's boundary. The encoder and decoder integrate objects instanced from the same classes such as Display, PM, and ME, and they also share the object DctQuan, which illustrates object reuse methods. The arrows in Fig. 8 indicate the function call direction. Actually, all messages appear in pairs in this application although SoNoCs support nonreturn messages. Some messages include hundreds of parameters and some take none. Packing function calling into one message means that control (FID) and data (parameters) information are packed together, so long or short messages are of the same importance. Gray objects that only receive arrows are passive objects while others are designed as active objects. All activities are authorized from the control module—ENC or DEC. The Display class plays different roles in the encoder and decoder with the encoder authorizing PM to show pictures while the decoder authorizes Display.

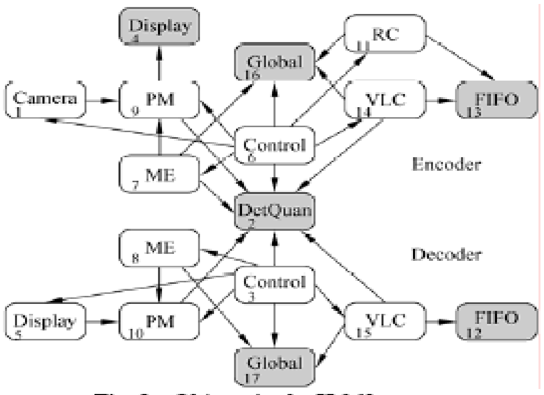

Fig. 8 Objects in the H.263 system

Design pattern analysis

The encoder and decoder run simultaneously. Inside the encoder, several tasks are also executed in parallel. For example, at the group of blocks (GOB) level, a new GOB is filled from Camera to PM, and the current GOB is encoded while the old GOB is shown at the same time. In this case study, when, where, and how messages enter the network is controlled by system execution so that the parallelism does not result in chaos. DctQuan is a passive object employed by both the encoder and decoder which is an example of a passive object having two threads. A semaphore named "busy" that resides in DctQuan synchronizes the two masters. The encoder or decoder must first apply for permission before utilizing DctQuan and release DctQuan after utilization. The DctQuan apply function is separated from other members and resides in a different thread. When DctQuan is occupied, subsequent apply messages will be processed by the thread. Thus, system multi-threading is based on object multithreading and synchronization of messages is based on semaphores within objects.

Simulation results

The simulation ran for 15 frames of the quarter common intermediate format (QCIF, Foreman) in 0.6 s of simulation time and then dumped traffic patterns over the space and time domains. Figure 9a shows the quantities of messages between objects, which is a symmetric matrix because messages appear in pairs. Figure 9b shows the bandwidth which varies since some objects send large numbers of small messages and some send small numbers of large messages. The network also recorded the bandwidth with time, as shown in Fig. 10. The task execution time estimates were referenced from the simulation on the very long instruction word (VLIW) processor[12]. In this example system, the number of peak parallel concurrent messages was about 5. Only concurrent messages compete for network resources, so the concurrent bandwidth not the overall bandwidth over the space domain influences network congestion. The design process and the simulation results show

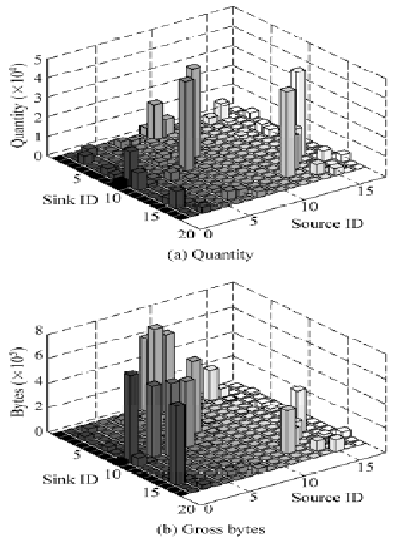

Fig. 9 Traffic over space domain for 15 frames. The object ID is marked in Fig. 8.

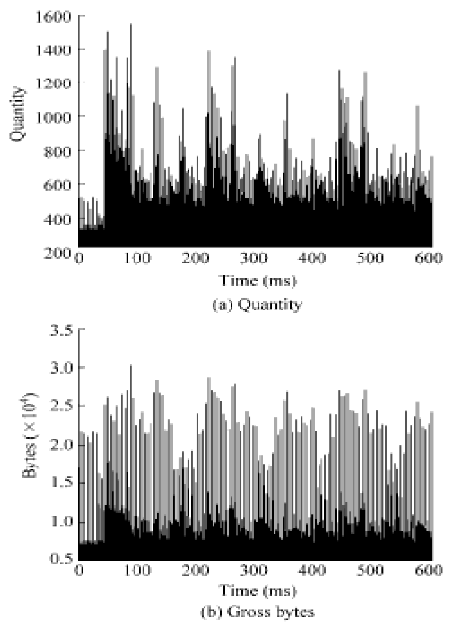

Fig. 10 Bandwidth valuations over time with 1 ms intervals

that traffic on a chip can be recognized and controlled. Analysis of the distribution over the space and time domains shows that the optimization space of on-chip networks for specific applications is quite large.

IX CONCLUSIONS AND FUTURE WORK

The NoC is triggering complex SoC that challenges current SoC design methodologies. NoC designs for SoCs must recognize applications, define slips between objects, and automatically synthesize optimized networks. This paper focuses on the first two issues using an OML model for SoNoC with an object-oriented SoNoC modeling and implementation method which was easily built to analyze real on-chip traffic. An example application using the H.263 codec was studied with this methodology, and the results show that traffic on a chip can be managed. Future work will be focused on network topology optimization for a given SoNoC OML prototype.

REFERENCES

[1] Bertozzi D, Jalabert A, Murali S, et al. NoC synthesis flow for customized domain specific multiprocessor systemson- chip. IEEE Transactions on Parallel and Distributed Systems, 2005, 16(2): 113-129.

[2] Goossens K, Dielissen J, Gangwal O, et al. A design flow for application-specific networks on chip with guaranteed performance to accelerate SoC design and verification. In: Proceedings of Design, Automation and Test in Europe. Los Alamitos, CA, USA: IEEE, 2005: 1182-1187.

[3] Coppola M, Curaba S, Grammatikakis M, et al. OCCN: A network-on-chip modeling and simulation framework. In: Proceedings of Design, Automation and Test in Europe. Los Alamitos, CA, USA: IEEE, 2004, 3: 174-179.

[4] Mahadevan S, Angiolini F, Storoaard M, et al. Network traffic generator model for fast network-on-chip simulation. In: Proceedings of Design, Automation and Test in Europe. Los Alamitos, CA, USA: IEEE, 2005, 2: 780-785.

[5] Wiklund D, Sathe S, Liu D. Network on chip simulations for benchmarking. In: Proceedings of System-on-Chip for Real-Time Applications. Los Alamitos, CA, USA: IEEE, 2004: 269-274.

[6] Dziri M A, Cesario W, Wagner F, et al. Unified component integration flow for multi-processor SoC design and validation. In: Proceedings of Design, Automation and Test in Europe. Los Alamitos, CA, USA: IEEE, 2004, 2: 1132-1137.

[7] Goudarzi M, Hessabi S, Mycroft A. Overhead-free polymorphism in network-on-chip implementation of objectoriented models. In: Proceedings of Design, Automation and Test in Europe. Los Alamitos, CA, USA: IEEE, 2004, 2: 1380-1381.

[8] Vanderperren Y, Dehaene W. UML 2 and SysML: An approach to deal with complexity in SoC/NoC design. In: Proceedings of Design, Automation and Test in Europe. Los Alamitos, CA, USA: IEEE, 2005, 2: 716-717.

[9] Sys ML. SysML specifications. http://www.sysml.org/ specs.htm, 2007-09-03.

[10] Ma Liwei, Sun Yihe. On-chip network evolution using NetC. In: Proceedings of VLSI Design, Automation and Test. Hsinchu, China: IEEE, 2005: 249-252.

[11] Ma Liwei, Sun Yihe. On-chip networks design automation with source routing switches. Tsinghua Science and Technology, 2007, 12(1): 77-85.

[12] Zhang Yanjun, He Hu, Sun Yihe. A new register file access architecture for software pipelining in VLIW processors. In: Proceedings of Asia and South Pacific Design Automation Conference. Shanghai, China: IEEE, 2005, 1: 627-630.