Integrated Modeling and Generation of a Reconfigurable Network-On-Chip

Doris Ching
dorisc@ee.ucla.edu

Patrick Schaumont
schaum@ee.ucla.edu

Ingrid Verbauwhede
ingrid@ee.ucla.edu

Electrical Engineering Department, UCLA

Abstract

While a communication network is a critical component for an efficient system-on-chip multiprocessor, there are few approaches available to help with system-level architectural exploration of such a specialized interconnection network. This paper presents an integrated modeling, simulation and implementation tool. A high level description of a network-on-chip can be simulated and converted into VHDL. The system simulation supports multiple instruction-set simulators, and obtains cycle-accurate performance metrics. This way, an optimal network configuration can be determined easily. We discuss our approach by designing a flexible network-on-chip and present implementation results after mapping into FPGA. The performance of our automatically generated network is comparable with a reference design directly developed in HDL.

1. Introduction

Modern System-On-Chip (SoC) contain multiple processors, dedicated hardware processing units and peripherals. Such a distributed architecture is required for reasons of performance and energy-efficiency, but it also introduces the requirement of an efficient system-level communication. As technology advances with ever increasing processor speed, global wires spanning across significant portion of chip size will dominate the propagation delay [3], which becomes a performance bottleneck for SoC design.

In recent years, significant research has demonstrated that an on-chip packet interconnection network is a better candidate for handling on chip communication [2]. System modules communicate to one another by sending packets across the network. This approach has the advantages of both performance and modularity. In another example [11], researchers implemented such a reconfigurable interconnection network on FPGA for improved hardware-software multitasking. In the result section, we will make performance comparison between our automatically generated prototype network and their design.

The system level components of a SoC include, besides the on-chip network, also embedded cores and embedded software. Some on-chip communication networks that target general-purpose multiprocessors are the J-Machine [4] and Smart Memory [10]. However, very few research has been done on modeling the on-chip communication architecture and integrating the communication network with processor units in a single simulation environment. Architectural exploration of a network should be done in the early stages of the design, using system-level simulation. This exploration is required because the communication requirements of a SoC are often determined by the application. Also, making changes to the communication protocol at late stages of the design cycle is a costly and complicated matter. We are therefore interested in combining the tasks of network design and embedded software development. This includes concise capturing of the interconnection network architecture together with the embedded software, cosimulation of the architecture with multiple instruction set simulators, and implementation.

2. Related Work

There has been very few research on modeling methodologies that fill the design gap from high level evaluation of communication network architecture with processing units down to implementation. A research group at Princeton University [14] has proposed a hierarchical modeling framework for an on-chip communication architecture using the Liberty Simulation Environment (LSE) and PtolemyII. LSE is a fast simulation and modeling environment with a dedicated machine description [9]; while PtolemyII is an object-oriented modeling framework written in JAVA [12]. Although both design environments can model system simulation between communication network architecture and processing units at a higher abstraction level, neither of them provides a code-generation interface to VHDL. A network design modeled in LSE or PtolemyII would have to be manually translated into a hardware description before it could be synthesized.

We present an integrated approach to cosimulation and implementation of a reconfigurable interconnection network for system-on-chip. Our environment, called GEZEL, captures the architecture of the network at high abstraction level and enables cosimulation with instruction-set simulators. The network description can be readily translated into VHDL for synthesis. The proposed design flow significantly reduces the time spent going from high level design of a multi-processor network to system verification and implementation. Our interconnection network is scalable and it can easily be changed to handle different routing strategies and network topologies.

In the following section, a brief overview of the GEZEL environment will be given. Section 4 describes our interconnection network design in detail. Section 5 presents the system evaluation and verification approach. Section 6 describes the code generation model for hardware synthesis. Section 7 discusses the implementation results of the interconnection network onto FPGA and compares the proposed network with related work. Finally, we draw conclusions and discuss future work.

3. Tool Overview

Figure 1 demonstrates the main characteristics of the GEZEL environment. GEZEL is a C++ library that can be linked to a system simulation with one or more instruction-set simulators (ISS) [6], illustrated in step 1. To describe hardware, GEZEL uses a dedicated language. This language uses finite-state-machine datapath (FSMD) semantics, which allows designer to capture datapath and control operations of hardware models independently. The FSMD models are cycle-true. When the GEZEL library initializes, it can read in one or more hardware models described in this language.

The embedded software part of the system runs on the ISS, and interfaces with hardware using a memory-mapped interface. To model a SoC environment, multiple ISS are used, one for each embedded processor core. They are connected together with the interconnection network and hardware accelerator units and modeled in the GEZEL description language. The system simulation runs with cycle accuracy and returns various metrics of performance as shown in step 2 of figure 1. This way, system exploration can be done interactively. After this, the hardware description modeled in GEZEL code can be converted into synthesizable VHDL code and this process is labeled as step 3 in the figure. We will now describe the features of our interconnection network and how it is modeled using FSMD semantics.

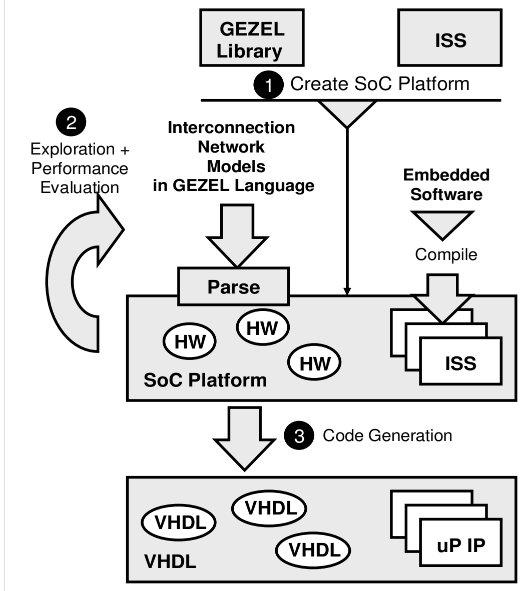


Figure 1: GEZEL System Exploration and Code Generation Process

4. Reconfigurable Network

4.1 Network Exploration

The interconnection network provides a fixed communication layer between various components of a SoC. In traditional design flow, the development of embedded application and communication layer are separated. The communication scheme is predominantly based on point-to-point or shared bus architectures. While this is suitable for today's embedded application that mostly comprise of a single processor core with memory modules and peripherals, the next generation SoC will demand more computation power and processing units, memory modules and on-chip traffic. With the increased complexity of the application, the interconnect design should be flexible to adapt to the needs of the system. Prior on-chip network design is typically arranged in a pre-defined form. The 2D mesh topology mapping for example proposed by Hu [7], the SPIN micro-network that uses a fat-tree topology [1], or the octagon network topology [8]. In these interconnection schemes, system modules connect to each other through the network in a fixed architecture. However, it is possible to further optimize the system by taking into account the specific task distribution pattern of the application during design exploration. Future SoC will likely consist of application specific node processor, video/image processing unit and general purpose micro processor coexisting on the same chip. Our design environment can provide the flexibility to support different system configuration. Given an application, a communication optimal network topology can be derived that can balance the throughput, give a shorter transmission latency and provide better resource utilization. This will also result in a system with improved power consumption, by reducing the congestion probability. Our proposed methodology allows designer to investigate on a variety of architecture during design space exploration, and finally determine an optimal topology and network configuration for a given system.

4.2 Network Structure

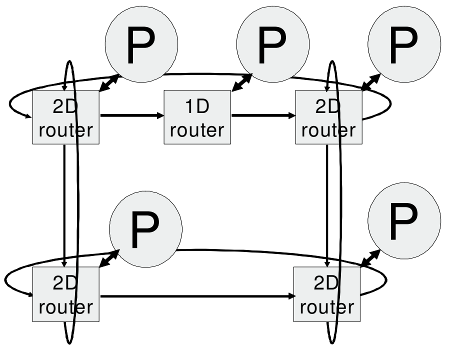


Figure 2: Example of a Network Topology

In general, the network complexity is characterized by two parameters: the routing algorithm and the network topology. Both parameters can be configured with our models. The tool can model any one or two-dimensional array of processor cores running embedded software. Each processor core is connected to a dedicated router for communication into and out of the network as illustrated in Figure 2.

These routers are addressable for communication among processors. The network uses a deterministic routing algorithm in the form of a lookup table inside each router for routing to the neighboring node. Although flow control is not supported, a deterministic routing approach significantly reduces hardware complexity and overhead. Reconfiguration of the network topology and placement of processing units only requires a modification to the routing table. Designers can arbitrarily instantiate multiple 1D- or 2D-routers library block to build a dedicated network. Furthermore, they can reconfigure internal buffer size of each router, and in this way, trade area for speed. These two features allow creation of a network topology that is matched to the traffic patterns of a special purpose SoC. It also allows for more efficient use of routing resources, which is an important design factor in embedded system design.

4.3 Router Interfaces and Packet Format

The 2D router shown in Figure 3 has data flowing in two directions. Each router has three input interfaces and three output interfaces dealing with synchronized communication between routers and the network interaction with processors. The communication reliability is guaranteed through a two-way handshake for each packet transmission. Each router performs wormhole routing with a packet size of 32 bits. This number is chosen to match a 32-bit embedded microprocessor. The transmission does not make any assumption on maximum message size or on the message data type as long as the proper packet format is abided. The first 2 bits of each packet contain control information indicating a header packet, a tail packet or a normal packet. The header packet will contain additional information on destination port. Furthermore, the transmission sequence is pipelined to obtain a transfer rate of three cycles per packet among routers.

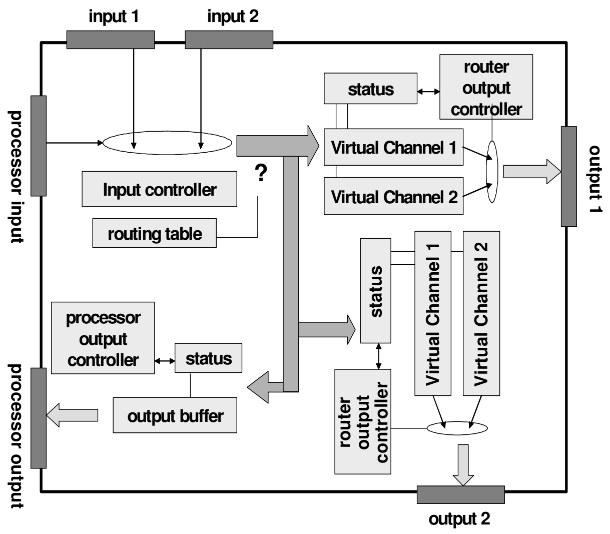


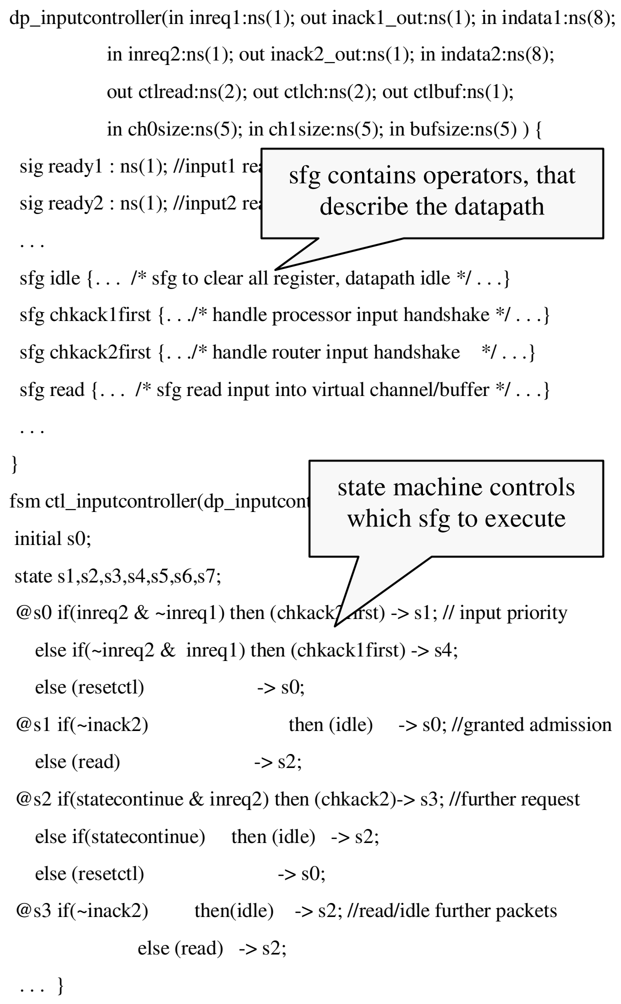
Figure 3: 2D Router's architecture

4.4 Router Architecture

As illustrated in Figure 3, the router contains three concurrent controllers: an input controller, a router output controller and a processor output controller. The input controller handles simultaneous input requests from neighboring routers and the processors. Priority is given to router inputs because the processor interfaces are driven by software, which is typically slower. A round-robin scheme is employed to arbitrate requests of equal priority. The router output controller and two virtual channels handle communication to neighboring routers. The two virtual channels can avoid deadlocks in a two dimensional torus network topology [5]. Finally, the processor output controller interfaces with the processor core to receive packets from the network. Because the communication between network and processor is handled in a blocking-send and receive manner, an additional output buffer is added between the routing channel and the processor output to relieve possible congestion caused by the blocking. A 1-D router has a similar structure but with a reduced interface and reduced number of virtual channels. A routing table is used to determine the subsequent routing path of each packet.

4.5 FSMD Model of the Network

The interconnection network is described in GEZEL in a FSMD model. The input controller, router output controller and processor output controller are modeled in a finite state machine with input requests triggering state transitions. Control signals generated from the finite state machine direct the operation of the datapath.


sfg contains operators, that describe the datapath
state machine controls which sfg to execute

```
dp_inputcontroller(in inreq1:ns(1); out inack1_out:ns(1); in indata1:ns(8);
             in inreq2:ns(1); out inack2_out:ns(1); in indata2:ns(8);
             out ctlread:ns(2); out ctlch:ns(2); out ctlbuf:ns(1);
             in ch0size:ns(5); in ch1size:ns(5); in bufsize:ns(5) ) {
 sig ready1 : ns(1); //input1 re
 sig ready2 : ns(1); //input2 re
 . . .
 sfg idle {. . .  /* sfg to clear all register, datapath idle */ . . .}
 sfg chkack1first {. . ./* handle processor input handshake */ . . .}
 sfg chkack2first {. . ./* handle router input handshake    */ . . .}
 sfg read {. . .  /* sfg read input into virtual channel/buffer */ . . .}
 . . .
}
fsm ctl_inputcontroller(dp_inputcon
 initial s0;
 state s1,s2,s3,s4,s5,s6,s7;
 @s0 if(inreq2 & ~inreq1) then (chkack2first) -> s1; // input priority
   else if(~inreq2 &  inreq1) then (chkack1first) -> s4;
   else (resetctl)                -> s0;
 @s1 if(~inack2)                    then (idle)     -> s0; //granted admission
   else (read)                    -> s2;
 @s2 if(statecontinue & inreq2) then (chkack2)-> s3; //further request
   else if(statecontinue)     then (idle)   -> s2;
   else (resetctl)                   -> s0;
 @s3 if(~inack2)        then(idle)    -> s2; //read/idle further packets
                 else (read)   -> s2;
 . . . }
```

Figure 4: FSMD model of a 1D Router's input controller

As an illustration of the compactness of the language, a GEZEL description of the input controller is shown in Figure 4. The FSM controls the execution sequence of sfg, which includes idling, handling input from neighboring routers, granting input admission, acknowledging requests, and reading input data. These concurrent operations (sfg) are specified in the datapath. Likewise, the output controller is modeled with an FSMD description. Because GEZEL supports hierarchical datapaths, a router is built by instantiating the components which include the FSMD model of virtual channel, input controller and output controller and making the connection. This way, the entire interconnection network is created by interconnecting multiple routers. GEZEL is an abstracted machine description language that has a simpler syntax than a traditional hardware description language such as Verilog or VHDL [6].

5. System Verification

5.1 Simulation Platform

The performance of the interconnection network is verified through a cycle true simulation that combines embedded software and simulation of the reconfigurable network (see figure 5). Embedded software written in C is cross-compiled into executables to be simulated on an ARM instruction-set simulator (ISS). They communicate with other C programs by sending packets using a set of API calls into the network. The API is an abstraction layer handling precise packet format and handshaking sequences between software and hardware. The system uses a memory-mapped interface between ISS and hardware, and all components in the system run in lock-step. If, besides the network, dedicated hardware processing units are needed, they are modeled as part of the GEZEL description.

The system simulation returns some important performance parameters of the communication scheme, allowing better design choice to be made in early stages of the design phase. The execution of a single C instruction often takes multiple hardware cycles due to the complexity of the processor architecture (cache misses, pipelining etc). Therefore, it is difficult to predict the performance of the handshaking sequence between the processor and network communication interface. With our co-simulation platform, cycle true measurements can be made. This gives us actual cycle count for a sequence of packet transmissions among parallel-embedded programs. The accuracy of the instructions-simulators we used is better then 3% [13], while the network is modeled exactly.

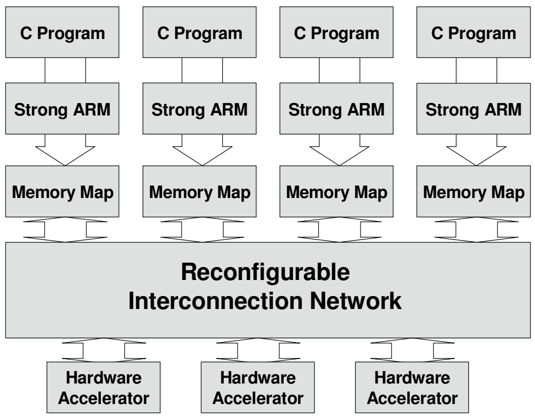


Figure 5: System Simulation Platform

5.2 Simulation Results

Table 1 presents various performance numbers of our 1D/2D torus network of four parallel processors. The evaluation platform is a DELL 3.2GHz Pentium 4 PC, with 512 MB RAM. The 1x4 1D torus network connects four processors in a ring. The 2x2 2D torus network connects four processors in a 2 by 2 array. Simulation numbers are taken between the communication of two neighboring processors (processor A to B), and of two processors that are two hops apart (processor A to D). The unit of transfer is a single 32 bits packet.

The input handshake synchronizes the input interface between embedded software and hardware, and takes 14 cycles in steady-state. From cold-start of the ARM software (with clean caches), these cycle counts are slightly higher (91 cycles for input and 11 cycles for output).

The cycle-per-hop performance number of 1D torus and 2D torus are the same. However, a 2D torus network gives a higher transmission bandwidth for the cost of increased area and reduced speed (as will be discussed in section 7). In the case where processor A sends a packet to processor D, a 2D topology gave a shorter routing path and faster transmission time. Most of the simulation time is spent on simulating software execution. Simulation of one packet's transmission from one processor to the next takes roughly 2 second for a 1D network.

Table 2 shows the simulation results of 1000 packets transmission. The average round trip time (RTT) for sending a packet from source to destination (1hop) is estimated to be 17 cycles. This number includes the handshake process that is needed to synchronize between hardware and software. As mentioned, the cycle require for the handshake process can varies and it depends on the state of the ARM processor. Therefore our RRT number is obtained from an average of 1000 transmissions. A simulation of 1000 packets transmission in a 1D network approximately takes 46226 cycles and 19 seconds to simulate.

SIMULATION OF 1 (32-BITS) PACKET				
	Processor A to B		Processor A to D	
	cycles	simulation time	Cycles	simulation time
Input handshake	91	--	91	--
output handshake	11	--	11	--
cycle per hop	3	--	3	--
1x4-1D torus (without init)	105	2 sec (total)	111	2 sec (total)
2x2-2D torus (without init)	105	6 sec (total)	105	6 sec (total)

Table 1: 1 Packet Simulation Result

SIMULATION OF 1000 (32-BITS) PACKETS				
	Processor A to B		Processor A to D	
	cycles	simulation time	Cycles	simulation time
average RTT (1x4 – 1D network)	17	--	17	--
1000 packets (1x4 – 1D network)	46226	19 sec	46231	21 sec

Table 2: 1000 Packets Simulation Result

6. Code-Generator Model

After performance evaluation and architecture exploration, the selected network architecture can be converted from GEZEL into synthesizable VHDL. This feature closes the design path for hardware implementation. The architectural exploration process involves design changes only in the GEZEL description. After design exploration, the target architecture will be implemented through the VHDL code-generator.

VHDL code-generation of the network model is implemented in three steps. Upon parsing, an intermediate representation (IR) of the interconnection network described in GEZEL is created in the form of a symbol table. In the first step, an internal object hierarchy is build through the symbol table interface. Figure 7 presents this object hierarchy, which captures the hardware model for each component in the design. With this representation, the code generator can reconstruct the hardware model in VHDL syntax. In the next stage, the code-generator goes through each datapath, controller and system to construct inter-block control signals and the system interconnect. In the final stage, the objects in the structure are mapped into corresponding VHDL syntax. As an example, a datapath object will consist of numerous signals and registers which will be mapped into a VHDL clocked process with signal update. Likewise, each of the GEZEL operators will be mapped into VHDL arithmetic operators from the IEEE standard library. Following this approach, a .vhd file is generated for each unique datapath and system. The code-generator and network source file is available for download on GEZEL homepage.

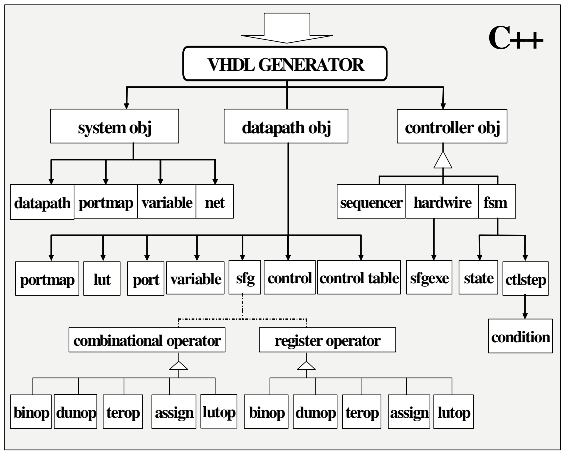


Figure 7: Code Generator object structure

7. Results

Synthesis results of the prototype reconfigurable interconnection network from the GEZEL description will be presented in this section. The synthesis software is Xilinx ISE. Table 3 shows the synthesis results of a 1D and 2D router supporting internal buffering up to 2 packets, which has the same parameter as the reference design [11]. However, it is important to notice that the reference design uses a 16-bit data bus while our design uses a 32-bit data bus.

	GEZEL		Reference [9]	
	slices	speed	slices	speed
1D-Router	253	-	223	n/a
2D-Router	674	-	n/a	n/a
1x2 1D torus	531	104MHz	n/a	n/a
1x4 1D torus	1061	104MHz	2385	n/a
2x2 2D torus	2733	85MHz	3227	n/a

Table 3: Router's Synthesis Result

As shown, the resulting area used for each router is comparable to their design. In a network of four routers, our area used is actually smaller. The area of our network scales proportionally with the number of routers in the network. The clock speed supported by the 1D network goes up to 100MHz and it is determined by the longest propagation delay between two communicating routers. The 2D network has a lower clock speed of 85MHz but it achieves a higher transmission bandwidth (illustrated in section 5). If the 1D network is clocked at 100MHz, the maximum transmission bandwidth goes up to (32-bit ∗ 100MHz/3 ∗ 30/32) = 1G-bit/s among routers. However, due to the limitation of the software speed (details in section 5), the transmission bandwidth of a packet is estimated to be (32-bit ∗ 100MHz/17 ∗ 30/32) = 177M-bit/s between ARM cores. A maximum speed comparison with the reference design cannot be made because this number was not available from publication.

8. Conclusion

We proposed an integrated modeling framework to generate an efficient reconfigurable network on chip. With our development tool, a designer can easily make architectural reconfiguration on the interconnection network targeting their specific application. Design changes can be verified through a cycle true system simulation, and a hardware model is readily available in synthesizable VHDL. The synthesized result is comparable to a network implemented with a traditional hardware design flow. With the GEZEL design environment, a close connection between system simulation and platform implementation can be made. We are currently developing a methodology to explore and select different on-chip reconfigurable network architectures. We are also developing a demonstrator that uses this technology.

9. Acknowledgements

The research work described in this paper was made possible by the supported from SRC [2003-HJ-1116] and NSF [CCR-0310527].

10. References

[1] A. Adriahantenaina, H. Charlery, A. Greiner, L. Mortiez, C. A. Zeferino, "SPIN: A Scalable, Packet Switched, On-Chip Micro-Network", Proceedings of the Design Automation and Test in Europe, March 2003

[2] L. Benini and G. Micheli, "Networks on Chips: A New SoC Paradigm," IEEE Computer 35(1) 2002, pp. 70-78.

[3] S. Charles,"Let's Route Packets Instead of Wires," Proc. 6th MIT Conf. 1990, Advance Research in VLSI, pp. 133-138.

[4] W. Dally, "The J-Machine Network," Proc.International Conf on Computer Design. IEEE VLSI in Computer & Processor, Oct 1992, pp 420-423

[5] W. Dally," Virtual-Channel Flow Control," IEEE Transaction on Parallel and Distributive System, vol 3, no 2, March 1992.

[6] GEZEL Homepage, http://www.ee.ucla.edu/~schaum/GEZEL

[7] J. Hu, R. Marculescu, "Exploiting the Routing Flexibility for Energy/Performance Aware Mapping of Regular NoC Architectures", Proceedings of Design Automation and Test in Europe, March 2003

[8] F. Karim, A. Nguyen, S. Dey, R. Rao, "On-chip Communication Architecture for OC-768 Network Processors", Proceddings of 38th Design Automation Conference, June 2001

[9] Liberty Simulation Environment Homepage, http://liberty.princeton.edu/Software/LSE/

[10] K. Mai," Smart Memories: A Modular Reconfigurable Architecture," Proc ISCA, June 2000, pp. 161-71

[11] T. Marescaux, A. Bartic, D. Verkest, S. Vernalde, R. Lauwereins, "Interconnection Networks Enable Fine-Grain Dynamic Multi-Tasking on FPGAs," FPL, Sep, 2002

[12] PtolemyII Homepage, http://ptolemy.eecs.berkeley.edu

[13] W.Qin, et al. SimIT-ARM Homepage, http://www.ee.princeton.edu/~wqin/armsim.htm

[14] X. Zhu, S. Malik, "A Hierarchical Modeling Framework for On-Chip Communication Architectures", Proceedings of International Conf on Computer Aided Design, Nov 2002.