Integrated modelling and generation of a reconfigurable network-on-chip

Doris Ching* and Patrick Schaumont

University of California at Los Angeles,
Los Angeles, 90024 CA, USA
E-mail: dorisc@ee.ucla.edu E-mail: schaum@vt.edu
*Corresponding author

Ingrid Verbauwhede

University of California at Los Angeles and K.U.Leuven, Belgium,
Los Angeles, 90024 CA, USA
E-mail: ingrid@ee.ucla.edu

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 specialised interconnection network. This paper presents an integrated modelling, simulation and implementation tool. The network architecture can be co-simulated with embedded-software to obtain cycle-accurate performance metrics. This allows an energy and performance tuning of the NOC. Next it can be converted into VHDL for implementation. We discuss our approach by designing a flexible network-on-chip and present implementation results. The performance of our automatically generated network is comparable with a reference design directly developed in HDL.

Keywords: design methodology; modelling; system analysis and design; multiprocessor interconnection.

Reference to this paper should be made as follows: Ching, D., Schaumont, P. and Verbauwhede, I. (2005) 'Integrated modelling and generation of a reconfigurable network-on-chip', *Int. J. Embedded Systems*, Vol. 1, Nos. 3/4, pp.218–227.

Biographical notes: Doris Ching is an Engineer in the Digital Design department of Raytheon Corporation under the Space and Airborne Systems sub-division. She holds a MSEE Degree in the field of Embedded Computing System from UCLA in 2004 and she received her BSEE from UCLA in 2002. Her research centres on VLSI design methodologies for real-time embedded systems that encompass system architecture design and reconfigurable interconnect.

Patrick Schaumont is an Assistant Professor at Virginia Tech's ECE Department. He received the PhD Degree in Electrical Engineering from UCLA in 2004, and the MS Degree in Computer Science from Ghent University, Belgium in 1990. Before joining UCLA in 2001, he was a Researcher at IMEC from 1992. His research interests include design methods and architectures for energy-efficient embedded systems. His research projects alternate between developing tools and developing demonstrator designs.

Ingrid Verbauwhede received the Electrical Engineering degree in 1984 and the PhD Degree in applied sciences from the K.U.Leuven, in Leuven, Belgium in 1991. She was a Lecturer and Visiting Research Engineer at UC Berkeley from 1992 to 1994. From 1994 to 1998 she was a Principal Engineer first with TCSI and then with Atmel in Berkeley, CA. She joined UCLA in 1998 as an Associate Professor and the K.U.Leuven in 2003. Her interests include circuits, processor architectures and design methodologies for real-time, embedded systems in application domains such as cryptography, digital signal processing, wireless and high speed communications.

1 Introduction

A typical low power embedded System-On-Chip (SoC) contains multiple processors, dedicated hardware processing units and peripherals. Such a distributed architecture is required for reasons of performance and energy-efficiency. But these goals can only be reached if the SOC also includes an efficient system-level communication. As technology advances with ever increasing processor speed, global wires spanning across significant portions of the chip will dominate the propagation delay (Charles, 1990), and therefore, become the performance bottleneck for SoC design.

The traditional method of interconnecting on-chip components is typically via a share medium, such as the IBM CoreConnect bus architecture, the ARM AMBA bus architecture, high performance, and other advanced peripheral bus systems (Ryu et al., 2001). While this approach is sufficient for today's SoC that typically consists of a central processing core plus surrounding co-processor blocks and peripherals, a future SoC will likely require more processing power, and demand more routing resources from the communication layer to support this parallel processing. Several research efforts have demonstrated that an on-chip packet interconnection network is a better candidate for handling on chip communication (Benini and Micheli, 2002). 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 (Marescaux et al., 2002), researchers implemented such a reconfigurable interconnection network on FPGA for improved hardware-software multitasking. In the result section, we will make a 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 (Dally, 1992a) and Smart Memory (Mai, 2000). However, very few researches have been done on modelling the on-chip communication architecture, and integrating the communication network with processor units running the embedded software in a single simulation environment. Traditionally, the architecture of the SOC is developed independently from the applications or application domain running on it. 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. For instance, streaming video applications should use a different NOC than database search applications. Also, making changes to the communication protocol at late stages of the design cycle is a costly and a 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, co-simulation of the architecture with multiple instruction set simulators, and implementation.

2 Related work

There have been very few researches on modelling methodologies that fill the design gap from high-level evaluation of communication network architectures with processing units down to implementation. A research group at Princeton University (Zhu and Malik, 2002) has proposed a hierarchical modelling framework for an on-chip communication architecture using two independent modelling environments, the Liberty Simulation Environment (LSE) and Ptolemy II. LSE is a fast simulation and modelling environment with a dedicated machine description (http://liberty.princeton.edu/Software/LSE/); it can be used to construct compiled code simulator and physical hardware blocks as logical functional modules that communicate through ports. PtolemyII is an object-oriented modelling framework written in JAVA (http://ptolemy.eecs.berkeley.edu). Its description language can model complex hierarchical interconnection of parameterised executable modules. 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 modelled in LSE or PtolemyII would have to be manually translated into a hardware description before it could be synthesised.

In the area of network design exploration, most of the prior research work focuses on creating an optimal mapping of a system's communication needs to a target template communication topology that consists of an arbitrary interconnection of shared bus and point-to-point connection (Lahiri et al., 2000, 2001). Such an approach may not be scalable for the next generation complex systems, where the communication needs can span across multiple processing nodes. Yet it does illustrate the increasing need for an application-driven design space exploration methodology for network on chip design.

We present an integrated approach to co-simulation and implementation of a reconfigurable interconnection network for system-on-chip. Our environment, called GEZEL, captures the architecture of the network at a high abstraction level and enables co-simulation 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 a 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.

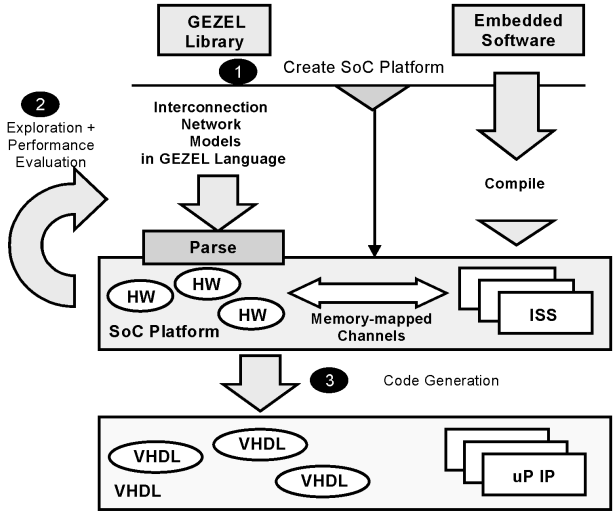
In the following section, a brief overview of the GEZEL environment will be given. Section 4 describes our interconnection network design in detail. Section 5 presents the system evaluation and verification approach. Section 6 describes the code generation model for hardware synthesis. Section 7 discusses 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) (http://www.ee.ucla.edu/~schaum/GEZEL), 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 initialises, it can read in one or more hardware models described in this language.

Figure 1 GEZEL system exploration and code generation process



The embedded software part of the system runs on the ISS, and interfaces with the 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 modelled 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 modelled in GEZEL code can be converted into synthesisable VHDL code and this process is labelled as Step 3 in the figure. We will now describe the features of our interconnection network and how it is modelled using FSMD semantics.

4 Reconfigurable network

4.1 Network exploration

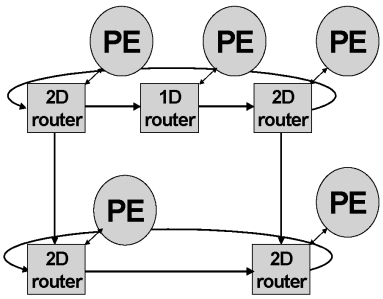
The purpose of an interconnection network is to provide a communication layer between various components of a SoC. In traditional design flow, the development of the embedded application and the communication layer are separated. The communication scheme is predominantly based on point-to-point or shared bus architectures. While this is suitable for today's embedded applications that mostly consist 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. Examples are the 2D mesh topology mapping proposed by Hu and Marculescu (2003), the SPIN micro-network that uses a fat-tree topology (Adriahantenaina et al., 2003), and the octagon network topology (Karim et al., 2001). In these interconnection schemes, system modules connect to each other through the network in a fixed architecture. However, it is possible to further optimise the system by taking the specific task distribution pattern of the application into account during design exploration. Future SoC will likely consist of different heterogeneous components, such as application specific IP blocks, specialised signal processing units, DSP processors, and general-purpose micro-processors. The communication needs between these heterogeneous components are likely to be very different. Some processing nodes might have a wide range of communication needs, while other nodes could exhibit a very local communication pattern. While it is possible to map such a system onto a regular NoC platform, it is likely that some links in the network will be under-utilised and other links will become a performance bottleneck caused by local congestion. These types of systems with predictable communication needs are expected to gain significant performance improvement from using an application specific on chip network.

Our design environment can provide the flexibility to support different system configurations. Given an application, a communication-optimal network topology and mapping can be derived that balances the throughput, gives a shorter transmission latency and provides better resource utilisation. This will also result in a system with improved power consumption, by reducing the congestion probability. Our proposed methodology allows the 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

In general, the network complexity is characterised 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.

Figure 2 Example of a network topology

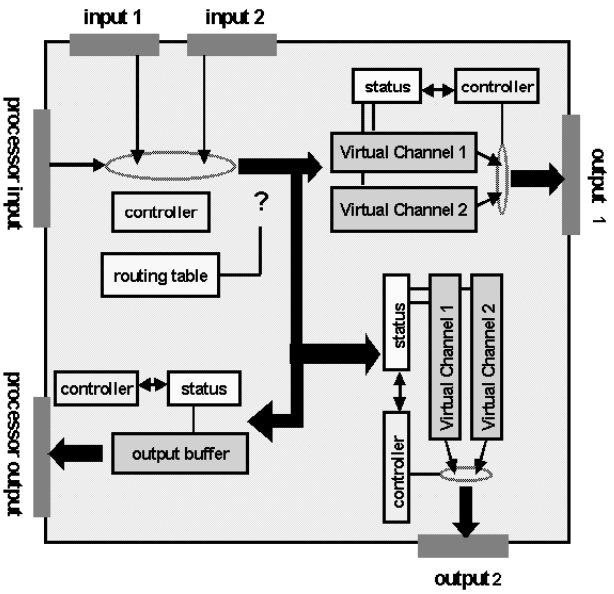


These routers are addressable for communication among processors. The network uses a deterministic routing algorithm in the form of a lookup table inside each router for routing to the neighbouring node. Although flow control is not supported, a deterministic routing approach significantly reduces hardware 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-router library blocks to build a dedicated network. Furthermore, they can reconfigure internal buffer sizes for each router, and in this way, trade area for speed. These two features allow the 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.

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 the synchronised communication between routers and the network interaction with processors. 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 a 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



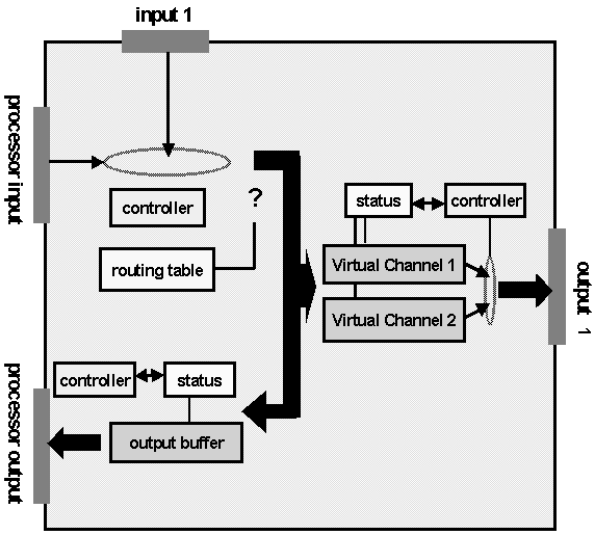
4.4 Router architecture

As illustrated in Figure 3, the router contains three concurrent controllers:

- an input controller
- a router output controller
- a processor output controller.

The input controller handles simultaneous input requests from neighbouring 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 neighbouring routers. The two virtual channels can avoid deadlocks in a two dimensional torus network topology (Dally, 1992b). 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, and its architecture is shown in Figure 4. A routing table is used to determine the routing path of each packet.

Figure 4 1D router's architecture



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 is modelled 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.

As an illustration of the compactness of the language, a GEZEL description of the input controller is shown in Figure 5. The FSM controls the execution sequence of datapath instructions (called sfg). The instructions include idling, handling input from neighbouring routers, granting input admission, acknowledging requests, and reading input data. The condition for state transition depends on the signals coming from the datapath, in this example it would depends on the various input request signals and priority setting. Likewise, the output controller is modelled with an FSMD description. A router is built by instantiating components, which include the FSMD model of virtual channel, input controller and output controller. The entire interconnection network is created by interconnecting multiple routers. GEZEL is an abstracted machine description language with structural hierarchy that has a simpler syntax than traditional hardware description language such as Verilog or VHDL (http://www.ee.ucla.edu /~schaum/GEZEL). At every clock cycle, the exact operation of the datapath is defined. This way, a hardware model given in GEZEL description can be simulated at a RT (register transfer) level.

Figure 5 FSMD model of a 1D router's input controller

```
dp inputcontroller(in inreq1:ns(1); out inack1_out:ns(1); in data: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 rea
  sig ready2 : ns(1); //input2 rea
  . . .
  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_input
 initial s0;
 state s1,s2,s3,s4,s5,s6,s7;
 @s0 if(inreq2 & ~inreq1) then (chk  k2first) -> 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;
 . . . }
```

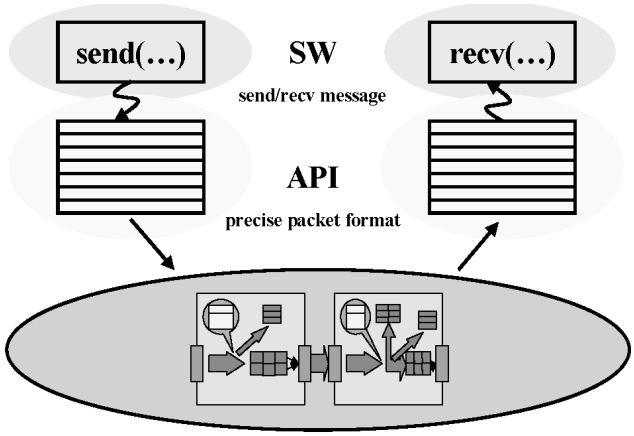
sfg contains operators, that describe the datapath

state machine controls which sfg to execute

4.6 Network dependent API

The proposed NOC multiprocessor platform is a parallel architecture based on message passing. Each of the processing elements communicates to one another through message passing via the network. The processing element is connected to a router through six interfacing signals. A request signal, acknowledgment signal, and data bus are defined for both the input and output port for a synchronised communication in and out of the network. The processing element connected to the NoC needs to use the same network protocol of a simple two-way handshake for each message transmission. A library of API calls written in C is provided with the tool. The API is an abstraction layer handling precise packet format and handshaking sequences between software and hardware. Using this network dependent API library, the parallel software running above the NoC can remain unchanged during network design exploration. From the view of the software, the NoC is a black box that interconnects multiple processing elements. The sender communicates to other nodes by composing a message, stamps it with the destination address and dispatches it to the router. The receiver will listen to the receiving router port for incoming packets. This concept is illustrated in Figure 6. The sender specifies a variable length message with the destination information. The API then splits the message into packets and forwards it to the router. At the receiving end, the API merges successive packets into message and returns the message to the upper software layer. This way, any changes to the network topology and configurable parameters will require at most, a small modification to the routing address specified for each processing node inside the API.

Figure 6 The hardware software interface



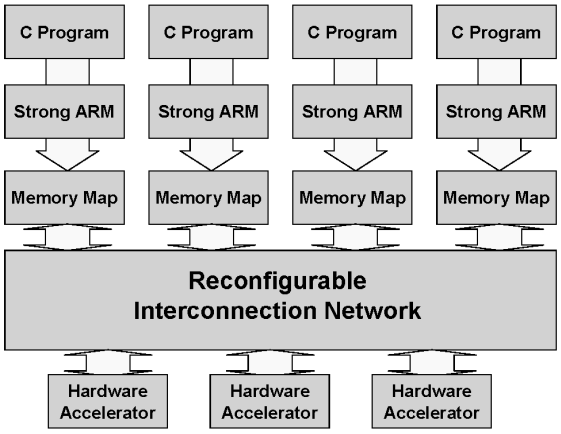
5 System verification

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 7). 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 system uses a memory-mapped interface between ISS and hardware, where each of the six interfacing signals are memory mapped location shared between hardware and software, and all components in the system run in lock-step. If, besides the network, dedicated hardware processing units are needed, they are modelled as part of the GEZEL description.

Figure 7 System simulation platform



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 cosimulation platform, cycle true measurements can be made. This gives us an actual cycle count for a sequence of packet transmissions among parallel-embedded programs. It can also verify whether a given network configuration meet the application's communication needs in term bandwidth, latency, and wait time. The accuracy of the instructions-simulators we used is better then 3% (Qin et al., 2003), while the network is modelled exactly.

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 3.2 GHz Pentium 4 PC, with 512 MB RAM. The 1×4 1D torus network connects four processors in a ring. The 2×2 2D torus network connects four processors in a 2 by 2 array. Simulation numbers are taken between the communication of two neighbouring 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.

Table 1 One packet simulation result

Simulation of 1 (32-bits) packet				
	Processor A to B		*Processor A to D*	
	Cycles	*Sim. time*	*Cycles*	*Sim. time*
Input handshake	91	–	91	–
Output handshake	11	–	11	–
Cycles/hop	3	–	3	–
1×4-1D torus (without init)	105	2 sec (total)	111	2 sec (total)
2×2-2D torus (without init)	105	6 sec (total)	105	6 sec (total)

The input handshake synchronises 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 1,000 packets transmission. The average round trip time (RTT) for sending a packet from source to destination (1 hop) is estimated to be 17 cycles. This number includes the handshake process that is needed to synchronise between hardware and software. As mentioned, the cycles required for the handshake process can vary and it depends on the state of the ARM processor. Therefore, our RRT number is obtained from an average of 1,000 transmissions. A simulation of 1,000 packets transmission in a 1D network approximately takes 46,226 cycles and 19 seconds to simulate.

Table 2 Simulation of 1,000 packets

Simulation of 1000 (32-bits) packets				
	Processor A to B		*Processor A to D*	
	Cycles	*Sim. time*	*Cycles*	*Sim. time*
Average RTT (1×4 – 1D network)	17	–	17	–
1,000 packets (1×4 – 1D network)	46,226	19 sec	46,231	21 sec

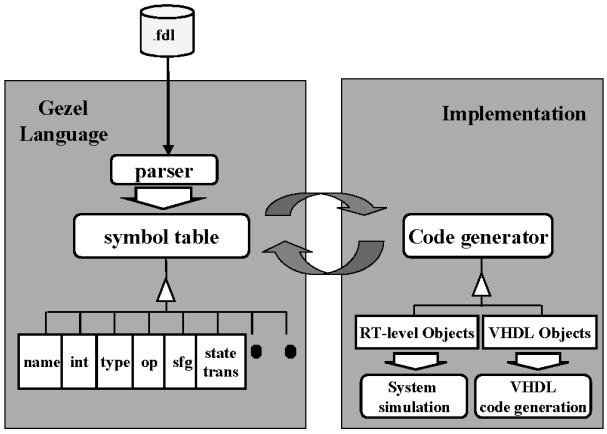
6 Code-generator model

After performance evaluation and architecture exploration, the selected network architecture can be converted from GEZEL into synthesisable VHDL. This section describes how the conversion process is done.

6.1 GEZEL model with object hierarchy

Upon parsing, an intermediate representation (IR) of the interconnection network described in GEZEL is created in the form of a symbol table. The IR will interface with the code generator for simulation or implementation. Figure 8 shows the structure of GEZEL.

Figure 8 GEZEL structure



The IR interfaces with the code generator by means of the *Visitor* design pattern (Gamma et al., 1994). The Visitor pattern defines an abstract code generation interface on the IR, and allows multiple concrete code generators to be used. In our case, a code generator can either create a new set of RT-level objects for simulation (Section 5) or else a set of VHDL objects for code generation (Section 6.2).

6.2 VHDL code-generation model

After performance evaluation and architecture exploration, the selected network architecture can be converted from GEZEL into synthesisable 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 9 presents this object hierarchy, which captures the complete hardware model for each component in the design. This includes the input output port assignments, signal assignments, each type of operators, the state machines, look up tables and control tables for each datapath in the system. This object structure retains the structural hierarchy of the original hardware model written in GEZEL. Each GEZEL model consists of a single system object that is linked to one or more datapath objects, and each of these datapath will be linked to a controller object that controls the execution of the datapath through state machines, sequencer machines, or hardwired signals. With this object 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 through a VHDL wrapper. 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. A correct VHDL translation from GEZEL is achievable because each individual construct in GEZEL can be represented with standard VHDL. Table 3 shows the correspondence from GEZEL modelling elements to VHDL representation. As illustrated, each building block has its corresponding representation in VHDL. Because every GEZEL description is constructed out of a number of these building blocks, translation to VHDL is always possible. The code-generator and network source file is available for download on the GEZEL homepage.

Table 3 GEZEL to VHDL translation table

GEZEL building blocks		*VHDL translation*
Datapath	Port	Entity declaration
	dp hierarchy	Component instantiation
	Register	Clocked signal update
	Signal	Combinational wire
	Lookup table	Rom
	sfg	Control signal enumeration
	Operator	std_logic_vector arithmetic library
Controller	State declaration	Type state declaration
	Conditional state transition	If, else, end if; logic
	State transition	Clocked state update
	sfg execution	Combinational control signal update
System	dp reuse	Component instantiation

6.3 VHDL simulation

The network model in VHDL can be simulated using standard EDA tools. In the event of a big multiprocessor design, it is costly in time to run a co-simulation with software at the VHDL level. To cope

with this issue, GEZEL provides a stimuli directive that can capture the transitions of input and output signals between the hardware software interfaces and it stores those into a file. This file allows the designer to replay the input-output activities between the processors and the hardware interconnect at the VHDL level. This concept is illustrated in Figure 10.

The GEZEL simulation generates the stimuli files, and these stimuli files are used to generate the VHDL testbench for the hardware-interconnect. This hardware simulation can verify the functionality of the NoC together with the system at the signal and gate level without doing a complete co-simulation with a VHDL model of the ARM processors, which would run at a very slow speed. In Figure 11, a VHDL simulation of our four processors system is shown using ModelSim.

Figure 9 Code generator object structure

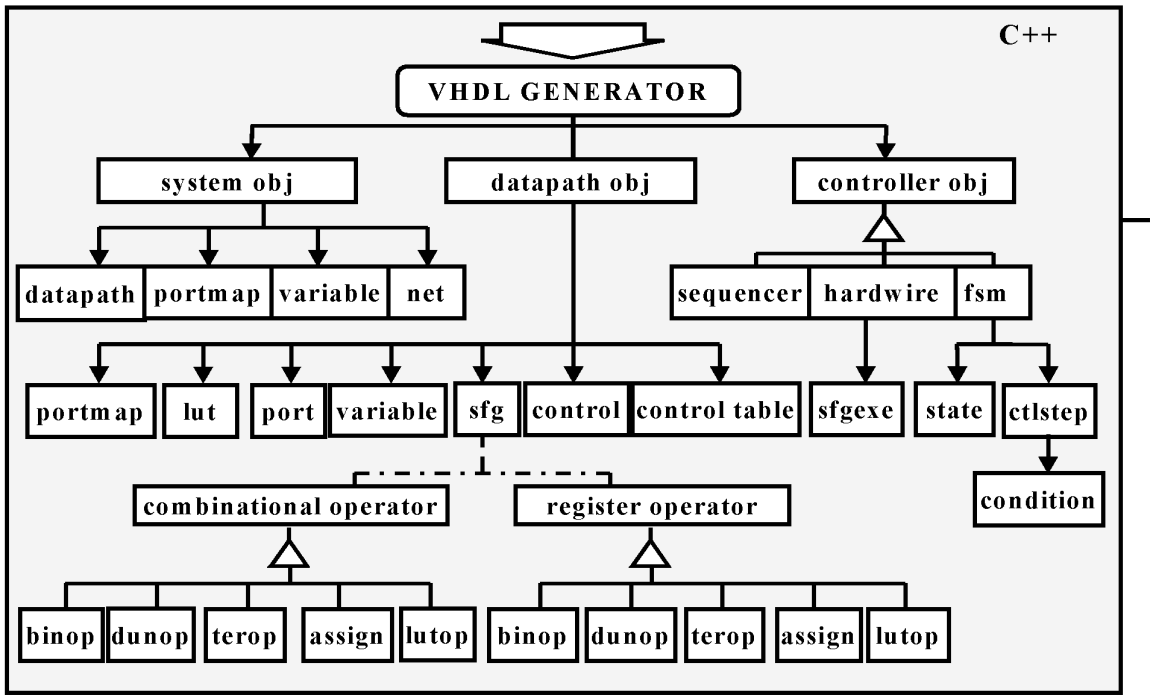


Figure 10 Stimuli testbench

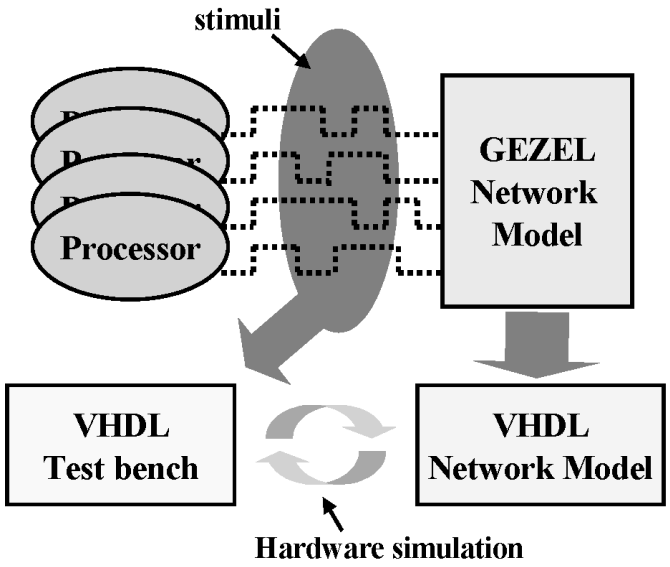
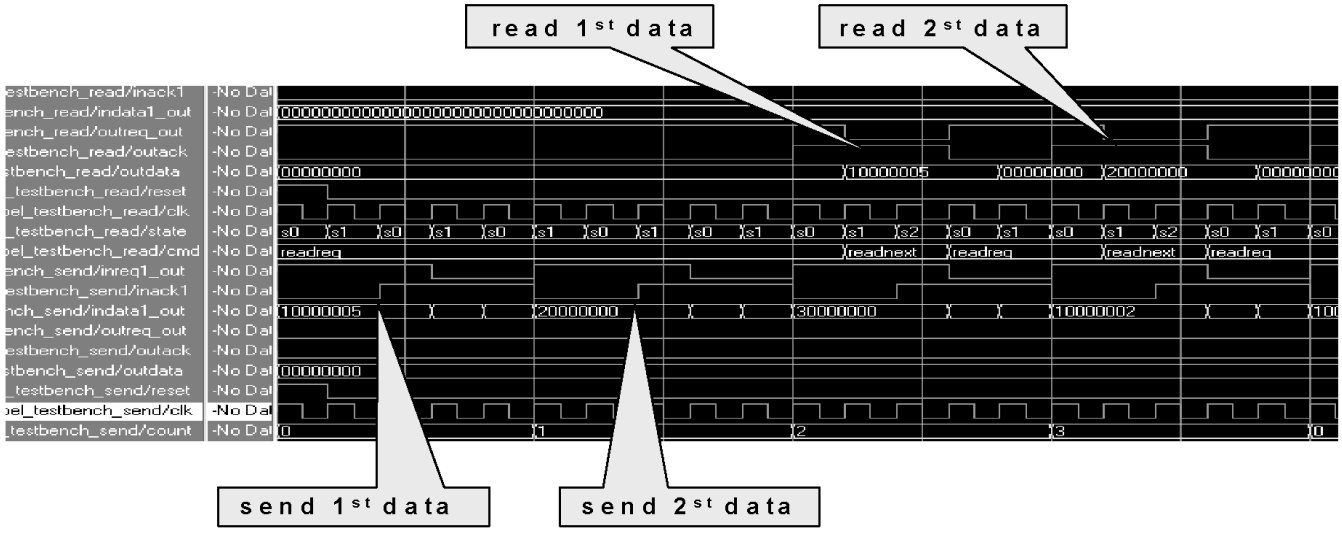


Figure 11 Hardware simulation



7 Results

As with any automated design tool, it is important to analyse the efficiency of the resulting design. The synthesis results of the prototype reconfigurable interconnection network from the GEZEL description will be presented in this section. We will also compare our automatically generated prototype network with a reference design that is directly developed in HDL using traditional network design methodology.

The synthesis software is Xilinx ISE. Table 4 shows the synthesis results of a 1D and 2D router supporting internal buffering up to two packets, which has the same parameters as the reference design (Marescaux et al., 2002). 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. This will result in a slightly higher cost in the area that is used for buffering.

Table 4 Router's synthesis result

	GEZEL		*Marescaux et al. (2002)*	
	Slices	*Speed*	*Slices*	*Speed*
1D-Router	253	–	223	n/a
2D-Router	674	–	n/a	n/a
1 × 2 1D torus	531	104 MHz	n/a	n/a
1 × 4 1D torus	1,061	104 MHz	2,385	n/a
2 × 2 2D torus	2,733	85 MHz	3,227	n/a

As shown, the resulting area used for each router is comparable to the reference 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. A one-dimensional torus network of four processors costs roughly the area of four routers. The clock speed supported by the 1D network goes up to 100 MHz and it is determined by the longest propagation delay between two communicating routers. The 2D network has a lower clock speed of 85 MHz, 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 × 100 MHz/3 × 30/32) = 1 G-bit/s among routers. However, due to the limitation of the software speed where the actual RTT between two communicating processors is about 17 cycles (details in Section 5), the transmission bandwidth of a packet is estimated to be (32-bit × 100 MHz/17 × 30/32) = 177 M-bit/s between ARM cores. A maximum speed comparison with the reference design cannot be made because this number was not available from the publication.

8 Conclusion

We propose an integrated modelling 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 synthesisable VHDL. This hardware model can be simulated at the gate level using standard EDA tools. The GEZEL stimuli feature captures the I/O activities coming from the processor interfaces. Once the design has met the given system constraint and cost budget, the network model is also synthesisable for implementation. We have demonstrated that the synthesised 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. This network on chip design technology allows system designer to build a prototype network implementation in a significantly reduced design time. 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.

Acknowledgement

This work was supported in part by SRC under [2003-HJ-1116], and NSF under [CCR-0310527].

References

Adriahantenaina, A., Charlery, H., Greiner, A., Mortiez, L. and Zeferino, C.A. (2003) 'SPIN: a scalable, packet switched, on-chip micro-network', *Design Automation and Test in Europe*, Embedded Software Forum, March, pp.70–73.

Benini, L. and Micheli, G. (2002) 'Networks on chips: a new SoC paradigm', *IEEE Computer*, Vol. 35, No. 1, pp.70–78.

Charles, S. (1990) 'Let's route packets instead of wires', *Proc. 6th MIT Conf. 1990, Advance Research in VLSI*, pp.133–138.

Dally, W. (1992a) 'The J-machine network', *Proc. International Conf on Computer Design*, IEEE VLSI in Computer and Processor, October, pp.420–423.

Dally, W. (1992b) 'Virtual-channel flow control', *IEEE Transaction on Parallel and Distributive System*, Vol. 3, No. 2, March, pp.194–205.

Gamma, E., Helm, R., Johnson, R. and Vlissides, J. (1994) *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison Wesley Professional Computing Series, ISBN: 0201633612, pp.1–395.

Hu, J. and Marculescu, R. (2003) 'Exploiting the routing flexibility for energy/performance aware mapping of regular NoC architectures', *Proceedings of Design Automation and Test in Europe*, March, pp.10688–10693.

Karim, F., Nguyen, A., Dey, S. and Rao, R. (2001) 'On-chip communication architecture for OC-768 network processors', *Proceedings of 38th Design Automation Conference*, June, pp.678–683.

Lahiri, K. *et al*. (2000) 'Efficient exploration of the SoC communication architecture design space', *Proc. of the Intl Conf on Computer-Aided Design*, November, pp.424–430.

Lahiri, K. *et al.* (2001) 'System performance analysis for designing on-chip communication architectures', *IEEE Trans. On Computer-Aided Design of Integrated Circuits and System*, June, No. 6, Vol. 20, pp.768–783.

Mai, K. (2000) 'Smart memories: a modular reconfigurable architecture', *Proc ISCA*, June, pp.161–171.

Marescaux, T., Bartic, A., Verkest, D., Vernalde, S. and Lauwereins, R. (2002) *Interconnection Networks Enable Fine-Grain Dynamic Multi-Tasking on FPGAs*, FPL, September, pp.795–805.

Qin, W. *et al.* (2003) SimIT-ARM Homepage, http://simit-arm.sourceforge.net/,

Ryu, K., Shin, E. and Mooney, V. (2001) 'A comparison of five different multiprocessor SoC bus architectures', *Proceedings of the EUROMICRO Symposium on Digital Systems Design*, September, pp.202–211.

Zhu, X. and Malik, S. (2002) 'A hierarchical modeling framework for on-chip communication architectures', *Proceedings of International Conf on Computer Aided Design*, November, pp.663–671.

Websites

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

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

PtolemyII Homepage, http://ptolemy.eecs.berkeley.edu.