Fast, Accurate and Detailed NoC Simulations

Pascal T. Wolkotte and Philip K.F. Hölzenspies and Gerard J.M. Smit
University of Twente, Department of EEMCS
P.O. Box 217, 7500 AE Enschede, The Netherlands
P.T.Wolkotte@utwente.nl*

Abstract

Network-on-Chip (NoC) architectures have a wide variety of parameters that can be adapted to the designer's requirements. Fast exploration of this parameter space is only possible at a high-level and several methods have been proposed. Cycle and bit accurate simulation is necessary when the actual router's RTL description needs to be evaluated and verified. However, extensive simulation of the NoC architecture with cycle and bit accuracy is prohibitively time consuming. In this paper we describe a simulation method to simulate large parallel homogeneous and heterogeneous network-on-chips on a single FPGA. The method is especially suitable for parallel systems where lengthy cycle and bit accurate simulations are required. As a case study, we use a NoC that was modelled and simulated in SystemC. We simulate the same NoC on the described FPGA simulator. This enables us to observe the NoC behavior under a large variety of traffic patterns. Compared with the SystemC simulation we achieved a speed-up of 80-300, without compromising the cycle and bit level accuracy.

1 Introduction

In the Smart chipS for Smart Surroundings (4S) project [1] we propose a heterogeneous Multi-Processor System-on-Chip (MPSoC) architecture. The SoC architecture contains a heterogeneous set of processing tiles interconnected by a Network-on-Chip (NoC). Future systems will consist of several tens or hundreds of tiles [3].

The development of such a heterogenous multi-tiled platform introduces problems related to hardware/software codesign. The MPSoC architect wants to study all kinds of trade-offs, e.g. operation bit-widths, memory sizes, and performance parameters/bottlenecks, e.g. latency and throughput. Common practice is to do extensive simulations of the MPSoC architecture before the system can be realized in silicon. In general the approach of simulating such large MPSoC designs is to either use (non-cycle accurate) high level modelling or accept long simulation times with cycle accurate simulations. For systems consisting of several tens or hundreds of tiles, cycle-true simulation leads to excessive simulation times.

Despite these excessive simulation times, cycle and bit accurate tests are required. These tests enable the designer to verify the design before manufacturing. Furthermore, it is possible to make accurate performance trade-offs and determine the consequences in area and power consumption of the actual design. Using the same concrete implementation (i.e. the same HDL source code) for simulation as well as synthesis we minimize the risk of errors in the design flow.

Future on-chip communication networks, the routers and their interfaces to the tiles, will have to support an increasing amount of services [16] compared to traditional busses. The communication infrastructure has to be able to provide guarantees, for example, guaranteed throughput, bounded latency and/or low jitter. These network services are required, because of the increasing number of applications that require real-time support. To understand those requirements and their influences on the NoC design we need to study the application-generated traffic on the on-chip network in detail. For the performance of the NoC, we cannot just analyze a single router and determine a local optimal schedule, because this might cause buffering problems in the neighboring routers. Testing the network under random traffic load and evaluating its routing or arbitration mechanism will not be sufficient to understand the extra levels of services required by real-time applications.

In this paper, we present a method to study the design choices of the network at the highest level of detail, without sacrificing simulation speed. We will demonstrate this by analyzing the latency and throughput behavior of our NoC [13] under different scenarios and configurations.

1.1 Related Work

There are several methods to analyze large heterogeneous systems. High level formal analysis methods can be applied [17], where an application of the system is characterized by high level parameters (e.g. latency of a task). Data dependencies and interactions of processes can only be analyzed if their characteristics can be described with

*This research is conducted within the Smart Chips for Smart Surroundings project (IST-001908) supported by the Sixth Framework Programme of the European Community.

Proceedings of the First International Symposium on Networks-on-Chip (NOCS'07)
0-7695-2773-6/07 $20.00 © 2007 IEEE

the high level model. However, this is only applicable for a restricted number of cases. For example, at design time the latency and execution time of a task have to be known (manifest).

Another method is system level simulation, such as SystemC [2] at different levels of abstraction. It can be used to describe systems from functional level to RTL level. The level of abstraction determines the speed of simulations. An example of SystemC simulation for NoC is the On-Chip Communication Network (OCCN) project [5], which defined a universal Application Programming Interface (API) for specification, modelling, simulation, and design exploration of NoCs. Another framework is presented by Kogel [14]. In the design flows of Æthereal [11] and xpipesCompiler [12], SystemC simulation is used for performance validation. The level of detail in the SystemC simulation tremendously influences the speed of simulation. Transaction Level Modelling (TLM), abstract data types and timed simulations showed almost a 3 orders of magnitude speedup compared to Register Transfer Level (RTL) modelling [14]. However, optimizations to increase the simulation performance sacrifice the level of detail in the simulated results.

For our packet switched NoC, we would like to monitor the cycle and bit accurate behavior under different traffic loads. A bit and cycle accurate simulation is required, because adaptations to the logic behavior of the router and the network as a whole are foreseen. Furthermore, we need to do extensive cycle true simulations, with the synthesized sources, before a chip can be realized. We developed a SystemC description of the router [13], but the simulation frequency was disappointing. Seriously testing a single scenario on one specific network configuration already took a full day. Therefore, an FPGA based simulator was considered. For very large multiprocessor systems, an FPGA based emulation platform makes accurate and fast system simulation possible, as proposed in the RAMP project [4]. This approach requires multiple FPGA platforms as for example provided by the Zebu-XL system emulator [8]. We would like to adopt the RAMP method and develop a simulator that requires a single FPGA.

Several FPGA based implementation to validate NoCs are described. Marescaux [15] describes their implementation of interconnection networks on an FPGA. Genko [10] proposed a NoC emulation framework implemented on a Virtex-II FPGAs. The emulation platform combines traffic generators, network interfaces, routers and traffic receptors. The platform is controlled by the FPGA's PowerPC and can work at 50 MHz. Genko's approach is bounded by the maximum available slices. For example, a 6 router network required 79% of a Virtex-II Pro VP20. The approach in this paper can simulate far more routers in a single FPGA. By sequential simulation we relaxed the hardware requirements and increase the size of the simulated NoC. We demonstrate the power of the simulator with a series of test scenarios using our Network-on-Chip.

The rest of the paper is organized as follows. In section 2, we describe the NoC that we would like to analyze. In section 3, we describe three methods we evaluated to simulate this network. In section 4, we describe the method that is used to simulate a parallel system sequentially. In section 5, we describe the implementation of this method onto the FPGA platform. In section 7, we show the speed improvements that are achieved by the FPGA method. In section 8, we discuss the simulator and conclude the paper in section 9.

2 Network-on-Chip

For the NoC, we have defined two networks (packet-switched [13] and circuit-switched [18]) that can both handle guaranteed throughput (GT) traffic and best-effort (BE) traffic simultaneously. The guaranteed throughput traffic is defined as data streams that have a guaranteed throughput and a bounded latency. The best-effort traffic is defined as traffic where neither throughput nor latency is guaranteed. In this paper we focus on the packet-switched network. However, the approach can also be used for the circuit-switched network and other designs.

2.1 Packet-Switched Network-on-Chip

The packet-switched router described by Kavaldjiev [13] implements wormhole routing with virtual channel (VC) flow control. The advantage of wormhole routing is the packet-size independent buffer-size. The virtual channels are used to decrease the chance of blocking and enable the routing of GT traffic. Each packet in the network consists of H header flits to setup a route, an arbitrary number of data flits that contain the packet's information and one tail flit that will free the router's resources for other packets. The header describes the route, which is predetermined via source-routing, and one header flit is consumed at each hop of the route.

The router has five input and five output ports and four virtual channels per port. The flits (atomic unit) of a packet are labelled with their virtual channel number and they are buffered in four flit deep queues at the input ports. Per port, four queues are available — one queue per virtual channel.

The outputs of the queues are not multiplexed per port, but directly connected to the crossbar. This is used to ease the arbitration compared to a standard wormhole router with virtual channels [6]. The crossbar is asymmetric and has 20 inputs, one input for every queue, and five outputs that are directly connected to the router's output ports.

The access to the crossbar is arbitrated by 5 round-robin arbiters - one arbiter per crossbar output. This arbitration is sufficient, since a conflict can only arise when more than one queue contains flits destined for the same output port. Due to the predictable round-robin arbitration the router is able to handle GT traffic, if one single data stream is assigned per VC. Multiple BE packets can be assigned to the same output VC.

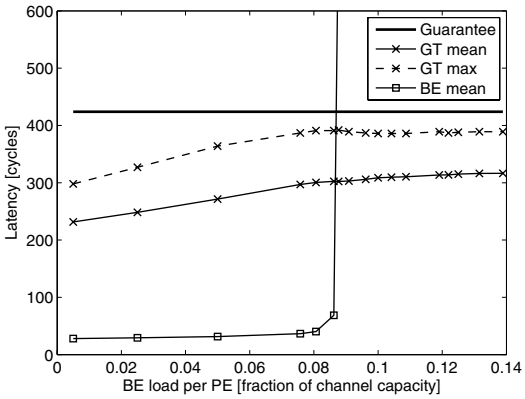

Figure 1. Message delay of the GT and BE traffic vs. BE load for 6-by-6 network (queue size 2 flits)

For BE packets, the packets competing for the same output VC are tagged by the sender with an unique header identifier (ID) per header flit. The routers are connected to a free-running global counter whose value is distributed to all inputs. When an output VC is freed the next packet that claims it is the one whose header ID equals the current counter value. The uniqueness of the BE ID guarantees conflict-free arbitration and crossbar access, but does not guarantee bandwidth or latency. Since, at any time, the counter contains a arbitrary value, fairness is provided.

Although the output round-robin arbitration is deterministic and maximum latency per hop can be determined, the specific latency and timing of packets largely depends on the global behavior of the network. Figure 1 shows the result of the latency simulation for a 6-by-6 network that has been performed in SystemC. A similar simulation is also performed on the new FPGA based simulator, where a faster and more extensive exploration of the results is possible. The details of this detailed simulation are described in section 6.2.

The graph shows how the latencies of the GT and BE messages depend on the offered BE load. For the GT traffic, the mean and the maximal latency of packets are given. When the offered BE load is low, the latency of the GT packets is smaller than the guaranteed (or allowed) latency. The reason is that the GT traffic utilizes the bandwidth unused by the BE traffic. Note: the latency of the GT packets is higher than the latency of the BE traffic because in this experiment the GT packets are larger (256 bytes against 10 bytes for BE packets). With the increase of the BE load, the average and maximum latency of the GT traffic increases, but the maximum GT latency never exceeds the guaranteed latency.

3 Simulation of Network-on-Chip

To simulate a NoC (a network of routers) we examined three options using:

1. VHDL
2. SystemC
3. An FPGA

For all options we modelled the NoC cycle and bit accurate. We started with a VHDL description, that could also be used to obtain synthesis results. A VHDL description was necessary, because, besides latency analysis, we are also interested in the area and power consumption of the NoC design. Router implementation with with very good throughput and latency, might result in a very high power consumption. For example, we found that buffers require a relatively large amount of area and energy. So we would like to redo the simulation of Figure 1 with different buffer sizes and investigate what the effect of buffer size on performance and energy consumption is.

For the latency and throughput analysis, we were hampered by the 10 cycles per second simulation speed of the VHDL simulation. Therefore, a SystemC model seemed a good approach with, as literature suggests, orders of magnitude speed improvements [14]. The SystemC simulations gave the first insights into the behavior of the network as is shown by Figure 1. However, to generate all the information that was required for this single graph we needed 29 hours of simulation time on a single Pentium 4.

The attempt to simulate a NoC in an FPGA was inspired by the fact that an FPGA has a lot of internal storage, which enables updating a large number of registers in a single cycle. In our lab we have a platform available with a multiprocessor SoC as is described in section 5.1. This SoC has two ARM processors that can communicate via a memory interface with a single Virtex-II FPGA. This platform was used in our experiments. The method to simulate the network in an FPGA is described in the next section. In section 5 we describe the details of the FPGA simulator.

4 NoC Simulation in an FPGA

There are several ways to simulate a Network-on-Chip in an FPGA. The first idea was to instantiate the whole network in the FPGA including simple traffic generators, but initial synthesis tests showed a limitation of approximately 24 routers in a Virtex-II 8000. These results were obtained with a reduced data-path of 6-bit and without the network interfaces, traffic generators and simulation controllers. The two major bottlenecks were the number of CLBs and available number of tri-states in the FPGA.

Therefore, a sequential simulator of the network was considered. The sequential simulator has another trade-off between hardware resource requirements and simulation speed. The details of this sequential method are described in [19], but a coarse description is given in this section.

4.1 Sequential simulation

The basic idea of the sequential simulator is as follows. In a synchronous parallel system, each block has some func-

tionality (its combinatorial circuitry) and a number of registers that store the block's internal state. In a normal single *system cycle*, all the blocks will update their state concurrently, based on the current state and its inputs. In our sequential simulator we will sequentially evaluate each block in one *function cycle*, which requires more overall clock cycles but less hardware. A system cycle is partitioned in N function cycles, where N is the number of parallel functions (blocks) that are sequentially simulated. All blocks that have identical functionality can reuse the combinatorial circuits. The block's state is stored in a large memory to maintain the state of the complete parallel system. When the state of all blocks is modified we switch to a new system cycle.

The method described above can be used to simulate any parallel system on any sequential processor. The frequency of the processor, the simulated functionality and the amount of registers that have to be updated determine the speed of the simulation. If we use a 32-bit processor like the Pentium 4 we can update at most 32 bits per function cycle in a bit accurate simulation. Furthermore, the evaluation of a function cycle will require multiple clock cycles. Although a Pentium 4 is a very fast processor, the simulation will not be very fast. Speed improvements can be achieved by only scheduling the registers that require evaluation in the system cycle because their inputs have changed (event driven).

The number of bits that can be updated in a function cycle is much larger in an FPGA. It can read and write a large number of bits in the available internal RAM. For example the Xilinx Virtex-II 8000 FPGA has 192 dual-port block-rams of 512 positions each 36 bits wide per RAM. This makes it theoretically possible to access 6912 registers in parallel. Furthermore, the FPGA has a large amount of logic, which can evaluate the functionality of multiple blocks in parallel or a single large block. This also makes it possible to do both the evaluation and update of the registers in parallel. A function cycle can be evaluated in one FPGA clock cycle.

4.2 Sequential simulation of a NoC

In a synchronous, homogeneous NoC, each router has an identical functionality. We partition each router into a single large block and sequentially evaluate each router in one function cycle. The partitioning of a whole router per block is possible due to the amount of registers and functionality per router. The amount of function cycles per system cycle equals the amount of routers in the network.

Our NoC routers are described in VHDL, which can immediately be used to synthesize to FPGA or silicon technology. It is very important that we can use (almost) unmodified VHDL sources, as this will minimize the risk of errors between the synthesized hardware and the simulator. For the sequential simulator it is required to apply a small modification of the sources, i.e. we have to re-map all the registers of the router to a large memory.

The current values of the links are stored in a memory as well. Any network consisting of a homogeneous set of N routers can be simulated independent of its interconnection topology. The topology is determined by the generated read and write address pattern for the link's memory.

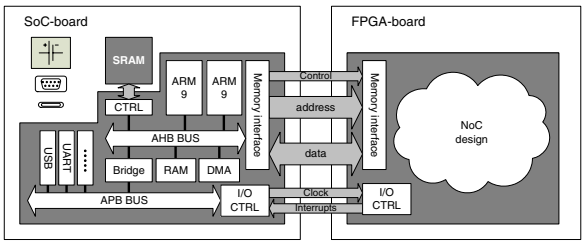

Figure 2. Schematic view of the hardware

5 Implementation

The implementation of the simulator requires a hardware platform, an FPGA design and software. First, we need a hardware platform with a (large) FPGA that is able to simulate the network. In section 5.1 we describe the platform we had available. Second, we need an FPGA design that is able to simulate the router according to the described method of section 4. In section 5.2 we describe the architecture and functionality of our FPGA design. Third, the NoC design needs stimuli and analysis software to evaluate the router's performance. This is described in section 5.3.

5.1 Hardware Platform

Figure 2 depicts the general block diagram of the available platform with the most important components. It consists of a SoC board and an FPGA board. The SoC board contains a dual-core ARM general purpose processor. The two ARM processors have a frequency of 86 MHz and 192 MHz. The SoC is connected with 1 MB of on-board SRAM memory and lots of peripherals and connectors. One of the connectors connects the FPGA board with the SoC board. This connector contains a memory interface with a 32-bit wide data-bus and a 17-bit wide address-bus. Via this memory interface we can control the logic of the FPGA and exchange blocks of data between the RAM of the SoC and the memory instantiated in the FPGA. The FPGA board itself does not have a separate off-FPGA memory. The FPGA board contains a Virtex-II 8000 FPGA.

Any other platform that has a large FPGA and on-board SRAM memory available will be suitable for the simulations. The on-board SRAM is required by the general purpose processor that controls the simulation. This control is both generation of stimuli vectors as well as analysis of the results. This general purpose processor can either be implemented on the FPGA or, as in our case, a separate processor. With the dual-core ARM chip we can partition the control software among the two processors.

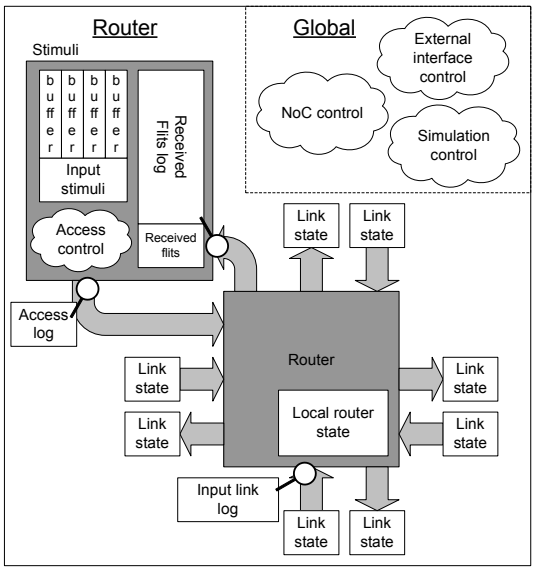

Figure 3. Schematic view of the FPGA design

State	Size
Input queues	1440 bits
Router control and arbitration	292 bits
Links	200 bits
Stimuli interfaces	180 bits
Total	2112 bits

Table 1. Required number of bits per router

5.2 FPGA implementation

Figure 3 depicts the major blocks of the FPGA design. The design can be partitioned in two major parts.

1. The router part, that describes the logic of a single router and its stimuli interface.
2. The global part, which controls the FPGA and the Network-on-Chip that is simulated.

For the router we separated the combinatorial logic from the registers in the original router design. The inputs and output signals of all registers are concatenated into two memory words: *old* and *new*. The old word is the current state of the router and is read from the memory at the start of a function cycle. The new word is the result of the evaluation and has to be written into the memory at the end of a function cycle. The address of the memory corresponds to the router that is evaluated. Table 1 summarizes the width of the memory word. The number of routers in the network determine the depth of the memory.

In the current implementation reading the values from memory takes 1 cycle. Evaluation of the combinatorial logic and writing the result in memory takes another cycle. In total a function cycle equals 2 FPGA cycles. For a design with N routers one system cycle takes $2N$ FPGA cycles.

The stimuli for the design are generated by software in the ARM9. We have chosen to generate the stimuli in software, because it is easier to define new tests and analyze the results in software. The disadvantage is the large amount of data that has to be copied from the ARM9 to the FPGA and vice versa. The stimuli are buffered per virtual channel (VC) in cyclic buffers in the FPGA. The output values of the network are stored per router, and not per VC, because a tile will receive at most 1 flit per system cycle. The data in the buffers have a timestamp and can be read or written by the ARM9. The timestamps make it possible to store only valid data, which requires less storage space and less time to copy data. Two extra cyclic buffers make it possible 1) to log the traffic of a specific link and 2) to log the access delay a flit notices before it enters the network. These two buffers cannot influence the traffic in the NoC.

The cyclic buffers make it possible to run the simulation independently from the copying of data. Of course, we have to prevent buffer under- and overrun, because it will influence the traffic in the NoC. The ARM will request the FPGA to simulate a specific number of system cycles. The FPGA will signal the ARM when it is ready with the requested number of system cycles. During the simulation in the FPGA, the ARM performs other tasks as described in the next section.

5.3 Software

The simulation is completely controlled in software by one or two ARM processors. The choice for one or two processors depends on the estimated simulation length and desired simulation speed. The software is partitioned in processes that communicate via cyclic buffers. All the processes can run in parallel and do not have dependencies.

Figure 4 depicts the organization of these processes and what part of the hardware is involved. The top processes require only the ARM processor and the NoC simulation itself only requires the FPGA. The two processes that interchange data between the boards require both ARM and FPGA. Each process that requires an ARM can be mapped on either of the two ARM processors.

Because each process uses its own cyclic buffers, it only needs to be fired when data and free memory are available. The processes that only require the FPGA or ARM run in parallel, which tremendously reduces the simulation time. We also have two ARMs available on the SoC-board that make it possible to run the generate and analysis process in parallel. Running the two processes in parallel requires more effort and time from the simulator user. Therefore, the parallelism with two ARMs is only beneficial if long simulations are performed. The simplest partitioning is running the generating and loading of data on one ARM, and retrieving and analyzing on the other ARM. This requires the least amount of inter processor communication and gives a good speed-up.

The simulation is performed in steps. We start with generating a routing information table. After all routes are de-

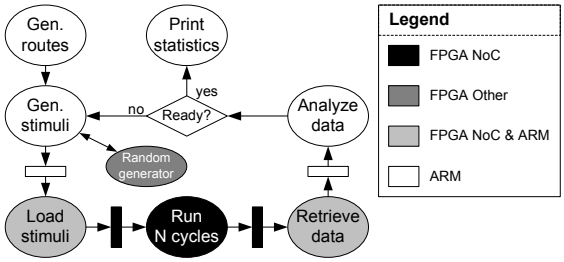

Figure 4. Software processes of the simulation

termined, a loop is started that has five phases. 1) We start by generating the traffic for each node in a stimuli table. Any data pattern can be generated as the generation is done in software. The generation process uses a random number generator on the FPGA. Reading a 32-bit random number from the FPGA is noticeably faster compared to the standard *rand()* function in C. The generated stimuli table contains stimuli for at least x system cycles. 2) The generated stimuli have to be written into the input buffers of the FPGA. All input buffers are maximally filled unless no data is available. 3) After filling the buffers we start the simulation in the FPGA and evaluate x system cycles. This sequence of x simulated system cycles is called a *simulation period*. To prevent buffer underrun, the simulation period is fixed to the size of the VC stimuli buffers in the FPGA. The simulation in the FPGA needs to be started by software, but can run autonomously. 4) After a single simulation period, we have to empty the output buffers. We retrieve the data from the output buffers that we think are interesting for the analysis. For the buffers that are not interesting we can update the read-pointer, which empties the buffer. 5) After the data is retrieved from the FPGA it is analyzed and the desired statistics are stored. When the simulation is not finished we go to step 1 and generate extra traffic in the stimuli table. This makes it possible to simulate an arbitrary number of simulation periods which is not limited by the software or hardware. However, due to back-pressure in the network, not all generated data might have been written into the FPGA. To prevent the loss of this data and the potentially resulting undefined state of the stimuli, all unconsumed data will eventually be written into the FPGA. If the network is overloaded with traffic and it does not accept data on virtual channels for a longer time, this is reported to the user and simulation is stopped.

6 Simulation examples

In this section we describe simulation examples that are performed on the NoC simulator. These examples are used both to analyze the behavior of the network and to profile the simulator. The analysis of the NoC is included in this section and the profile results are described in section 7.

6.1 Best effort traffic

Most of the NoC architectures are simulated with random traffic uniformly distributed over time and tiles. The average latency per packet is calculated and plotted versus the packet injection rate. In this example we apply different traffic patterns to a various set of NoC configurations. As a reference we use the virtual channel wormhole routing router as described in section 2 and a two dimensional 6x6 mesh topology.

Before the start of the simulation we randomly select pairs of communicating tiles. Per communicating pair of tiles, we determine the route through the mesh-topology via XY-routing. XY-routing is applied, because it is deadlock-free. For a 6x6 network we could map 493 randomly selected pairs onto the network. On average, a tile communicates with 14 other tiles, which is close to the maximum of 16 (4 VCs and 4 IDs per VC). All the routes that use a specific link are uniformly distributed over the four virtual channels and four IDs per virtual channel.

Each communicating pair will transport a packet of D data flits at random moments in time. The data flits are preceded by H header flits that contain the routing information and followed by single tail flit that will free the router's resources for other packets. The number of header flits is equal to the number of hops of the packet's route. Per hop, a header flit is consumed, which reduces the average number of flits/cycle that arrive at a specific tile. Therefore, the latency figures in this section are plotted versus the *injection* and *extraction* rate of the tiles. The injection rate is defined as the average number of flits per system cycle a tile injects into the network. The extraction rate is defined as the average number of flits per system cycle a tile receives from the network. The latter represents the amount of useful data transported by the network. The latency per packet is measured and grouped depending on the length of its route. The minimum latency per packet is equal to $2H + D + 1$ (the number of hops + the length of the packet).

Figure 5 depicts the average and maximum latency of a packet versus the injection and extraction rate when all packets contain 5 data flits. The difference of the average latency depending on the injection or extraction rate is caused by the header consumption of the network. Furthermore, it is visible that the average latency is influenced by the length of the route. The average latency is given for a selection of route lengths. However, the maximum latency of a single packet is not directly related to the length of the route, but mainly dependent on the packet injection rate.

6.1.1 Improvements

The majority of the latency seems to be caused by the access mechanism to the crossbar for BE traffic at the setup of the route through the network. The access mechanism prevents two or more packets to be granted to access the crossbar and select the same output virtual channel. The packets from different input VCs have an unique ID if they

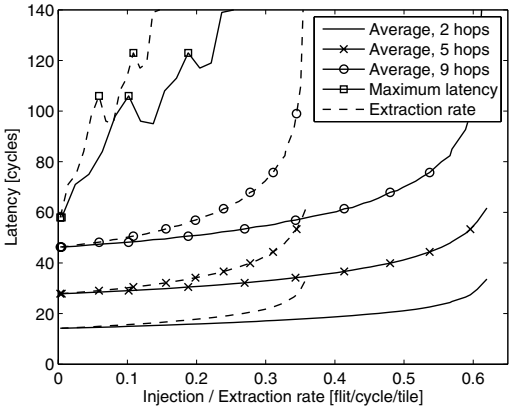

Figure 5. Average and maximum latency for 5 flits BE packets

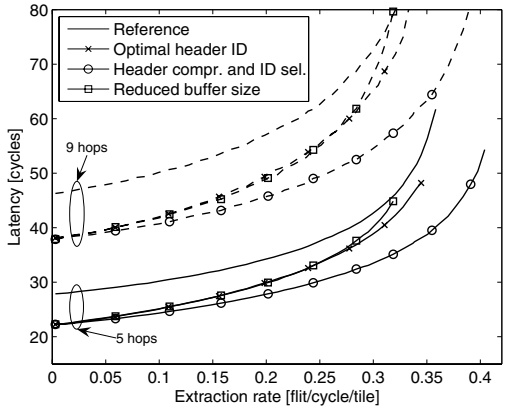

Figure 6. Improved latency for 5 and 9 hops

compete for the same output virtual channel. At most one packet per VC is granted if its header ID matches the global counter. In the reference simulation we chose a random ID for each header. Although this is fair, it is not fast. As the ID counter is global for the whole NoC, we can predict what the counter value is when the next header of the packet requests crossbar access at the next router, based on the minimum header delay. In the current design this delay is 3 cycles. Figure 6 depicts the latency if the header ID is optimized to this 3 cycle delay. Especially for low traffic loads the latency is reduced by approximately 20%. The network almost saturates at the same extraction rate.

Furthermore, for these small packets of 5 data flits we noticed a relatively large overhead of the header flits as the average route in a 6x6 network is 4.7 hops. The header/data ratio is almost one. Each header uses only 6 of the available 16 bits, which makes it possible to combine two header flits into one and reduce the average number of header flits to 2.4. The router requires only one extra multiplexer per VC to select the correct bits when it receives its first header of a packet. We expected the network to saturate at higher loads

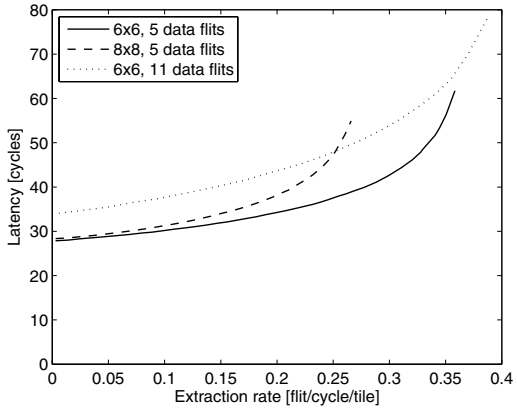

Figure 7. Average and maximum latency for 5 flits BE packets for different topologies

and have a lower latency, which is depicted in Figure 6. In this case we combined optimized header ID selection with header compression. Solely header compression did not show a major improvement. Header compression is especially beneficial for higher loads and small packets. The latency at low loads is mainly effected by the ID selection.

A third test was to reduce the queue size (2 flits) of the NoC and see its effect on the performance. Both header compression and ID optimization were included as well. At low traffic loads the buffer size marginally influences the latency. Due to the smaller buffers, the network saturates at lower loads.

6.1.2 Other scenarios

For a second scenario we varied the network size and number of data flits in the packet. We simulated various mesh topology sizes and packet sizes. In Figure 7, the latency depending on the extraction rate is depicted for both a 6x6 network and 8x8 network. For the 6x6 network we display the average latency for the scenarios with 5 or 11 data flits per BE packet. From this figure, it is clear that a larger network will saturate at a lower extraction rate (lower throughput per tile). Similar behavior is observed by Duato [7, chapter 9]. Larger packets have a higher average latency, but no noticeable change of the saturation rate.

6.2 Jitter analysis

As described in section 2 our virtual channel wormhole router can support both best-effort and guaranteed throughput traffic. Kavaldjiev [13] performed latency measurements to show that a guaranteed throughput stream will never violate its given deadline, independent of other traffic in the NoC. Figure 1 shows an example where an increasing best-effort load will increase the average and maximum latency of the guaranteed streams, but the maximum latency never exceeds the guaranteed latency.

In the following test we performed a similar test, but we want to study the inter packet jitter of the various guaranteed throughput streams. We assume the simulated network has a clock frequency of 333 MHz and uses applications that are similar to HiperLAN/2 [9]. In this application all communication occurs in burst with periods of 4 μs. We mapped the following three types of streams onto the network:

1. Guaranteed throughput (GT) packets of 128 flits with an average inter-packet time of 1333 cycles. At 333 MHz this is equal to the 4 μs period.
2. GT packets of 256 flits with an average inter-packet time of 4000 cycles.
3. Best effort packets with 5 data flits. The inter-packet time is varied to increase the load in the network. All BE streams are mapped on one single virtual channel. The unique ID will prevent conflicts at the crossbar.

Both GT streams have minimum guarantee of 1/3 of the link bandwidth. The maximum latency of a single GT packet equals: $3 \cdot (H + D)$ cycles, where H is the length of the route and D the length of a single packet. The test has 29 streams of type 1, 18 streams of type 2 and 103 streams of type 3, all uniformly distributed over the NoC. In this test, we are interested in the inter-packet jitter of the GT packets. Of course, it is important that the packets will arrive at the destination in less than the guaranteed latency. An extra level of service is that the packets will also arrive at a very regular interval (low jitter).

In the test, we examine the latency of the two GT packet types and we make a histogram of the latencies that are grouped in the length of the route. We vary the best effort load over time and see how this influences the latency of the other streams. With this test we are able to show one of the major benefits of the fast, but accurate, simulator.

We need a very large number of packets to get enough accuracy of the variation in the packet latency. Suppose we simulate for 1.5 million cycles. This transports 1125 packets per stream of type 1 and only 375 packets of type 2. If we want to calculate an average latency, this is possible as we can group multiple streams of a type that have the same length, but for a histogram a lot of detail is still missing, as the latencies range from 130 to almost 400 clock cycles. Simulation of 1.5 million cycles in the SystemC simulator required roughly 1 hour and 45 minutes. In the FPGA simulator we do the same test in less then a minute. Therefore, we simulated for 15 million cycles in approximately 10 minutes and obtained detailed histogram results.

Figure 8 depicts the distribution function (the percentage of packets that notice a specific maximum latency) of the latency of all streams of type 1 under different best effort loads. The y-axis gives the percentage of the packets notice the latency on the x-axis or less. For different BE loads, the distribution function changes. From this figure, it is clear that under almost no BE load, 50% of the packets arrive with the minimum latency (length route + length packet) and use the maximum bandwidth of the links. Because there are multiple GT streams in the NoC, part of the

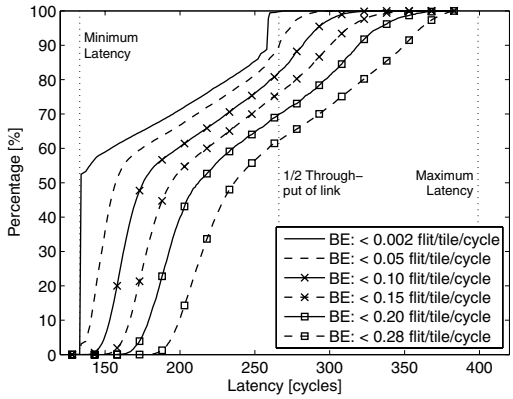

Figure 8. Latency distribution under different BE loads

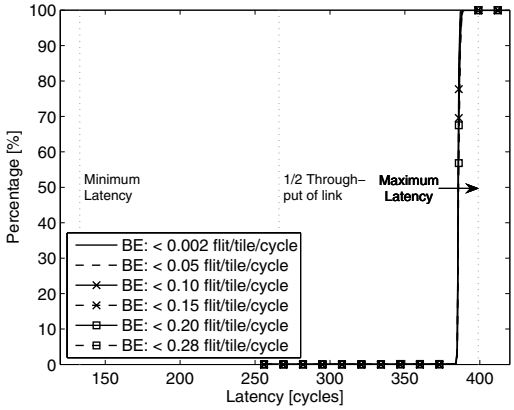

Figure 9. Latency distribution under different BE loads, where the GT streams are traffic shaped at the input

packets have to share bandwidth of the link. Depending on the phase alignment of the packets they influence each other. However, this can be at most a reduction to half the bandwidth under low BE load. A little more than 10% of the packets observe this reduced throughput. Under higher best effort loads the GT packets gets influenced more and more. This has a major impact on the latency distribution function. This change in the distribution function will be noticed by the receiver as jitter in the arrival time of packets.

If we now apply traffic shaping and release the packets with their given bandwidth of 1/3, we expect them to experience less jitter in the network. This is depicted in Figure 9. It is clear that the streams do not get influenced by the BE traffic, but experience a large latency as the packets enter the network at their guaranteed bandwidth.

Although it is not directly visible in the graphs, this level of detail is only possible with long simulation times. In the previous SystemC simulations, this would have taken weeks and with our simulator the results were obtained in less than

Block	CLB	RAM
Router	1762	61
Stimuli interface	540	62
Network	2103	16
Random number generator	2021	0
Global control	627	0
Total	7053 (15%)	139 (82%)

Table 2. FPGA resource usage

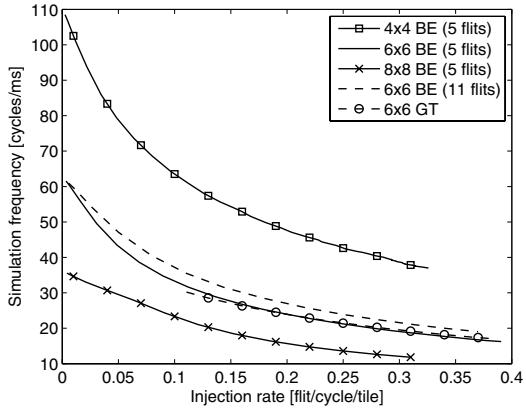

Figure 10. Simulation frequency versus injection rate

5 hours for 30 different levels of BE loads. Both simulation approaches can be speeded up by running multiple tests in parallel on multiple platforms.

7 Profile Results

In this section we describe the performance of the simulator. The simulator is realized for a Xilinx Virtex-II 8000 FPGA. Table 2 shows the resource usage of the simulator in the FPGA. From these results, it is clear that the limiting factor of the design is the number of RAM-blocks that are used. It would be possible to simulate the design in smaller FPGAs, but it would limit the maximum number of routers and/or the amount of state registers of the design.

The router design is synthesized for a frequency of 6.6 MHz, which gives a function cycle frequency of 3.3 MHz. This limits the maximum simulation frequency of the simulator to $(3.3 \cdot 10^6)/36 = 91.6$ kHz for a 6-by-6 network. No effort was made to increase this frequency, because it was sufficient for the current tests. The interface frequency is equal to the ARM frequency of 86 MHz, which makes it possible to copy data at a higher frequency.

The number of system cycles that can be simulated depends on the simulation settings. As a reference we use the simulation frequency of our SystemC simulator that was used to derive Figure 1. These and other SystemC simulations on a 6x6 mesh network had an average frequency of 0.215 cycles/msec independent of the applied network load.

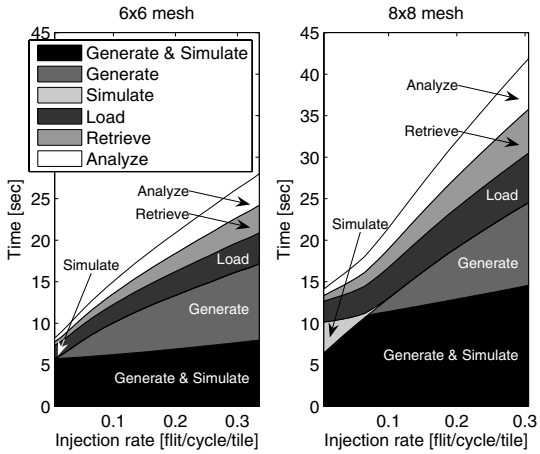

Figure 11. Simulation times for 0.5M cycles

Figure 10 depicts the simulation frequency (in cycles/msec) under different network loads for the series of performed tests as described in section 6. In all tests, we use the slowest of the two ARM processors at 86 MHz. From this figure we can conclude that the FPGA simulator is a factor of 80-300 faster compared to our SystemC simulation. Under low traffic loads, the frequency is close to the theoretical maximum that is limited by the FPGA. Under higher loads the frequency decreases due to extra time required by the software processes of Figure 4. Furthermore, it is visible that the frequency is approximately inversely proportional to the size of the network.

For different tests on the 6x6 topology with identical load, the minor differences are caused by the relation between injection rate and packet rate. For example, the analysis and generation time for the BE tests is almost independent of the packet size. Larger packets (11 flits) will cause a higher injection rate, which results in shorter simulation times at identical injection rates.

The time required per process of Figure 4, to simulate 1/2 million system cycles, is depicted in Figure 11 for the best-effort tests with 5 data flits in each packet for two network topologies. In these tests we run the generation and simulate process in parallel on ARM and FPGA. For most loads, the simulation process requires less time than the generation process, which completely hides the simulation time spent in the FPGA. Only in the 8x8 topology and under low loads can we achieve lower simulation times if we run more processes in parallel to the simulation process on the FPGA. Although the exact time varies between tests, in general we see that approximately 55% of the time is spend in generation of the stimuli. Loading, retrieving and analyzing data all require 15% of the time. The time for analysis increases if complex analysis techniques are included (e.g. determine variance of inter-flit arrival times). As most of the time is spent in software and not by the FPGA, there is no reason to increase the FPGAs function cycle frequency.

8 Discussion

8.1 Flexibility of the NoC simulator

The simulator on the FPGA is implemented as a homogeneous wormhole switching network with virtual channel flow control with a torus topology. The software on the ARM can change the network size from 1-by-2 to any 2 dimensional size with a maximum number of 256 routers. The maximum number of routers is limited by the amount of RAM that is required for the cyclic buffers and the router's state.

In the current simulator we have the same functionality for all the routers. It is possible to select a different router functionality depending on the position in the network. The limiting factor is the number of registers in the router. The topology of a network tested was either a torus or a mesh, which is determined by software. Other topologies are possible and only require a change in the addressing function of the link memories in the FPGA.

The same technique used for the NoC simulator can also be used for testing other parallel systems on an FPGA. In particular systolic algorithms with many equal parts with a small state space. If the source code for synthesis is available, it is relatively straightforward to modify the code for the sequential framework. Heterogeneous systems can be supported as well, as long as the required extra combinatorial logic fits in the FPGA. In the NoC case, less then 10% of the logic resources are used for combinatorial circuitry of the routers. The registers can be mapped in the same memory space.

9 Conclusion

In this paper we described the results obtained with our FPGA based network-on-chip simulation method. The method is especially suitable for parallel systems were lengthy cycle and bit accurate simulations are required. Those tend to demand a considerable amount of time using software only simulation on a desktop PC. Using an FPGA, we are able to speed up the simulation with a factor of 80-300 compared to a SystemC simulation without loss of accuracy and a small code difference with the original VHDL source code. Although an FPGA cannot handle high frequencies, it benefits from its large internal memory bandwidth and parallel execution of many combinatorial circuitries.

The simulator gave us detailed insights in the behavior of our wormhole network. Detailed latency histograms were possible, because millions of system cycles can be simulated in less than a minute. This enabled us to modify and verify the router's circuitry like, for example, header compression, or adapt the routing function for optimal header ID selection. Many other tests were performed as well, but were not included in the paper.

References

[1] 4S-project. http://www.smart-chips.com.

[2] Open SystemC Iniative OSCI, SystemC documentation. http://www.systemc.org, 2004.

[3] International Technology Roadmap for Semiconductors (ITRS'05). Technical report, Semiconductor Industry Association, http://public.itrs.net/, 2005.

[4] Arvind et al. RAMP: Research Accelerator for Multiple Processors - a community vision for a shared experimental parallel HW/SW platform. Technical report, MIT, 2005.

[5] M. Coppola et al. OCCN: A NoC modeling framework for design exploration. *Journal of Systems Architecture: the EUROMICRO Journal*, 50(2-3):129 – 163, 2004.

[6] W. J. Dally. Virtual-channel flow control. *IEEE Trans. Parallel Distrib. Syst.*, 3(2):194–205, 1992.

[7] J. Duato, S. Yalamanchili, and L. Ni. *Interconnection Networks, An Engineering Approach*. Morgan Kaufmann Publishers, San Fransisco, CA, USA, revised printing edition, 2003.

[8] Emulation and Verification Engineering. ZeBu-XL system emulator. http://www.eve-team.com, 2007.

[9] European Telecommunication Standard Institute (ETSI). *Broadband Radio Access Networks (BRAN); HIPERLAN Type 2*, ETSI TS 101 475 v1.2.2 (2001-02) edition, February 2001.

[10] N. Genko et al. A complete network-on-chip emulation framework. In *Proceedings of DATE'05*, pages 246–251, 2005.

[11] K. Goossens et al. A design flow for application-specific networks on chip with guaranteed performance to accelerate soc design and verification. In *Proceedings of DATE'05*, pages 1182–1187, Washington, DC, USA, 2005. IEEE Computer Society.

[12] A. Jalabert et al. xpipesCompiler: A tool for instantiating application specific networks on chip. In *Proceedings of DATE'04*, Paris, France, Februari 2004.

[13] N. K. Kavaldjiev. *A run-time reconfigurable Network-on-Chip for streaming DSP applications*. PhD thesis, University of Twente, Enschede, The Netherlands, January 2007.

[14] T. Kogel et al. A modular simulation framework for architectural exploration of on-chip interconnection networks. In *Proceedings of CODES+ISSS'03*, pages 7–12, New York, NY, USA, 2003. ACM Press.

[15] T. Marescaux et al. Networks on Chip as Hardware Components of an OS for Reconfigurable Systems. In *Field-Programmable Logic and Applications*, volume 2778/2003 of *Lecture Notes in Computer Science*, pages 595–605. Springer Berlin / Heidelberg, 2003.

[16] E. Rijpkema et al. Trade-offs in the design of a router with both guaranteed and best-effort services for networks on chip. *IEE Proceedings: Computers and Digital Techniques*, 150(5):294–302, sep 2003.

[17] M. H. Wiggers et al. Buffer capacities for multi-rate real-time systems with backpressure. In *Proceedings of CODES+ISSS'06*, pages 10–15, Seoul, Korea, October 2006. ACM Press.

[18] P. T. Wolkotte et al. An energy-efficient reconfigurable circuit-switched network-on-chip. In *Proceedings of IPDPS'05*, Denver, Colorado, USA, April 4-5 2005.

[19] P. T. Wolkotte, P. K. F. Hölzenspies, and G. J. M. Smit. Using an FPGA for fast bit accurate SoC simulation. In *Proceedings of IPDPS'07*, Long Beach, California, USA, March 26-27 2007.