A Modular Simulation Framework for Architectural Exploration of On-Chip Interconnection Networks

Tim Kogel, Malte Doerper, Andreas Wieferink,
Rainer Leupers, Gerd Ascheid, Heinrich Meyr
Integrated Signal Processing Systems
Aachen University of Technology, Germany
http://www.iss.rwth-aachen.de
Tim.Kogel@iss.rwth-aachen.de

Serge Goossens
CoWare, Inc.
Interleuvenlaan 15A
B-3001 Leuven, Belgium
http://www.CoWare.com
Serge.Goossens@CoWare.com

ABSTRACT

Ever increasing complexity and heterogeneity of SoC platforms require diversified on-chip communication schemes beyond the currently omnipresent shared bus architectures. To prevent time consuming design changes late in the design flow, we propose the early exploration of the on-chip communication architecture to meet performance and cost requirements. Based on SystemC 2.0.1 we have defined a modular exploration framework, which is able to capture the effect on performance for different on-chip networks like dedicated point-to-point, shared bus, and crossbar topologies. Monitoring of performance parameters like utilization, latency and throughput drives the mapping of the inter-module traffic to an efficient communication architecture. The effectiveness of our approach is demonstrated by the exemplary design of a high performance Network Processing Unit (NPU), which is compared against a commercial NPU device.

Categories and Subject Descriptors

B.8.2 [**Performance And Reliability**]: Performance Analysis and Design Aids; C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design—*Network topology*; I.6.7 [**Simulation And Modelling**]: Simulation Support Systems—*Environments*

General Terms

Performance, Design, Measurement

Keywords

Network-on-Chip, Architecture Exploration, SystemC, Simulation

1. INTRODUCTION

IP reuse has been long identified as a key technology to cope with ever increasing System-on-Chip (SoC) complexity along with ever shrinking time-to-market constraints. However, the problem of dealing with the interconnection of tens or even hundreds of IP blocks has only recently entered into the focus of researchers.

Current design practice for on-chip communication is predominantly based on shared bus architectures, which are available from core vendors [1, 2] or industry partnerships [3]. Due to the easy programming model, high flexibility and IP availability, this concept is clearly advantageous for today's small and medium scale embedded systems, where a small number of blocks exchange moderate amount of data. However, shared busses are not eligible for high throughput requirements and chip-wide interconnect, because of the limited scalability with respect to performance, number of connected blocks and range of coverage as well as limited support for QoS requirements.

Consequently, researchers in academia and industry have conceived alternative topologies to cope with the limitations of shared bus architectures. These efforts have recently been subsumed under the *Networks on Chip* (NoC) design paradigm. NoC advocates to replace current ad-hoc wiring of IP blocks with a disciplined approach, where full-fledged on-chip networks provide communication services according to the ISO/OSI reference model [4, 5]. By that the manifold problems in on-chip communication like signal integrity issues, link reliability, or Quality of Service (QoS) are separately resolved on the respective OSI layer. Depending on the intended application domain, recent proposals for NoC architectures differ significantly in terms of cost, performance, QoS and ISO/OSI compliance [6], [7], [8].

This kind of on-chip networks will probably not replace but complement bus based and point-to-point communication schemes, such that predominantly self-contained IP islands using local communication are connected via the global NoC. Hence, one of the most challenging tasks in future System-on-Chip design projects is to *conceive an on-chip communication infrastructure for heterogeneous platform architectures* in adherence to the specified flexibility, performance and cost requirements.

In this context we propose a modular NoC exploration framework, which addresses the system level design of the SoC communication architecture. Data exchange is modeled on the high abstraction level of packet transfers, which hides low-level implementation details but captures the effect on performance introduced by the network type. Thus the designer can partition the system communication into global network(s) and local interconnect and parameterize the selected communication architecture.

The unique contribution and major benefit of our approach is the ability to *capture the impact of on-chip communication on the system performance in a unified way*. By providing a library of different network modules with configurable topologies and arbitration protocols, our approach enables the rapid exploration and coherent comparison of architectural alternatives. Additionally, the high abstraction level of packet based communication yields *highest achievable simulation speed* together with *excellent modeling efficiency*.

The accuracy of the simulation results is of course affected by the coarse granularity of packet based communication. The proposed exploration framework is therefore not adequate to measure with full cycle accuracy the communication in small-scale embedded systems employing just a single bus. Especially particularities

CODES+ISSS'03, October 1–3, 2003,Newport Beach, California, USA.

of certain bus architectures, like e.g. explicit insertion of busy or idle cycles of the AMBA high performance bus, cannot be captured in our generalized communication models. Instead, we address the early exploration and profiling driven partitioning of large-scale systems with complex and heterogeneous communication schemes.

The following section discusses related work in the area of exploration of system level communication architectures. After that the projected design flow followed by the technical details of our NoC exploration framework are presented. Section 5 contains our experiences from an industry cooperation, where the framework has been deployed in the design of a NPU platform. Finally we conclude our approach and give an outlook on future research topics.

2. RELATED WORK

System Level Design is considered the appropriate way to deal with the ever increasing complexity and heterogeneity of SoC architectures [9]. Various frameworks are proposed addressing design space exploration and the partitioning/mapping of an abstract functional model to the target architecture [10, 11, 12]. Complementary to this general approach, our focus is on the conceptualization and performance analysis of the on-chip communication architecture.

Recent work on system level exploration of the communication architecture can be subdivided into static analysis [13, 14] and simulation based techniques [15, 16]. Lahiri et al. [17] have proposed a hybrid procedure combining simulation with analytical post-processing to achieve higher accuracy of the performance estimation.

However, current approaches do not cope with the requirements introduced by the system level design of full-fledged on-chip networks. In order to apply analytical estimation, enhanced algorithms are necessary to analyze the performance of complex network topologies with sophisticated arbitration mechanisms. Equally, current simulation environments fall short to provide efficient support for exploration of on-chip networks.

On the other hand, traditional networking design environments like OPNET [18] are not suited for SoC design, since only the abstract communication structure can be captured without any support for chip-level architecture modeling.

The SystemC Open Core Protocol (SOCP) communication channel in the StepNP simulation platform [19] addresses the exploration of the communication infrastructure based on the OCP semantics. In a more general approach, we further abstract from architecture specific communication primitives to establish a unified framework for the investigation of heterogeneous on-chip networks.

3. SYSTEM LEVEL DESIGN FLOW

The NoC exploration framework is implemented on top of the SystemC library [20], which provides the underlying event-driven simulation engine and is considered as the emerging EDA standard for System Level Design. In this section we briefly introduce the basic concepts SystemC 2.0.1 based System Level Design and highlight the relation to our work. Please refer to Groetker et al. [21] for a thorough representation of this topic area.

As illustrated in figure 1, SystemC 2.0.1 follows the *Interface Method Call (IMC)* principle to achieve high modularity in communication modeling. Processes are wrapped into modules and access communication services through ports. The available methods are declared in the interface specification and implemented by the channel. Graphically a port is represented as a square containing two diametrical arrows and an interface is represented as a circle containing a u-turn arrow. A channel implementing an interface shows a rhombic shape.

The IMC scheme has been conceived to realize a *Transaction-Level Modeling (TLM)* style, where communication is abstracted from the low-level implementation details of the Register Transfer Level. The resulting improvement in terms of simulation speed and modeling efficiency enables the system architect to create an executable specification of the complete platform architecture.

As depicted in table 1, the TLM paradigm can be further sub-

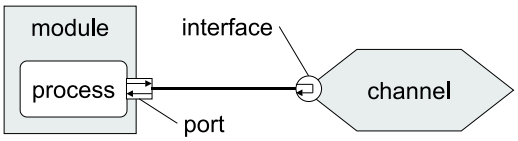

Figure 1: Interface Method Call Principle

divided by applying abstraction w.r.t. data and timing accuracy. By that the manifold design problems during the definition of the system level architecture can be resolved in the appropriate design step. Note that not all steps are necessarily performed for all platform designs: e.g. cycle- and bit-accurate TLM is usually sufficient for the design of small and medium scale embedded systems.

Timing Accuracy	Data Accuracy	Addressed Design Problems
untimed-functional	Abstract Data Type	functional specification
timed-functional	Abstract Data Type	architecture exploration
cycle true	bit true	finalize architecture, SW development

Table 1: TLM Abstraction Levels

This paper addresses the conceptualization of large scale heterogenous systems, which need a customized communication infrastructure to meet performance and cost requirements. Hence, the proposed NoC exploration environment is situated on the untimed- and timed functional level. In analogy with traditional network design environments, we advocate to design the on-chip network on the packet level, i.e. the considered data granularity are sets of functionally associated data, which are combined to Abstract Data Types (ADTs).

Quantitative numbers on the simulation speed are highly dependent on the complexity of the considered application and are thus not well comparable. However, experiments show, that a medium scale embedded system modeled on the cycle- and bit-accurate transaction level runs at about 200k cycles per second on a 2.4 GHz Linux PC, which is about two orders of magnitude faster than the corresponding RTL model without loss of accuracy [22]. In our framework an *additional* order of magnitude is achieved by abstracting the data representation to the ADT level. Migrating from the untimed to the timed functional level is realized by adding execution delays in terms of timing annotations, which has no significant impact on the simulation speed.

4. NOC EXPLORATION FRAMEWORK

This section introduces the proposed simulation framework for system level exploration of the communication architecture. We first present our unified approach to the modeling of on-chip communication and the associated NoC architecture exploration methodology. Afterwards we elaborate on the internal structure of the framework and the algorithms, which capture the performance characteristics of the distinctive on-chip networks.

4.1 Unified Modeling of On-Chip Communication

In general two different kinds of modules can participate in a communication event or *transaction*: While master modules actively initiate transactions, the slave modules can only react passively. Typical masters are processors, DMA controllers or autonomous ASIC blocks, whereas typical slaves are memories or co-processors. Of course peer-to-peer communication between two masters is also possible.

This general master-slave communication scheme is reflected in

the overall organization of the NoC evaluation framework depicted in figure 2.

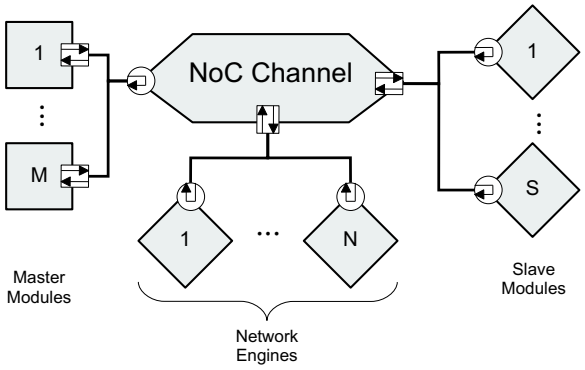

Figure 2: NoC Channel Overview

On-chip communication services are offered through a generalized master interface. In concordance with the *Interface Based Design* principle [23] [24], master modules initially not compliant to this interface can be attached by means of adapters.

The processing of communication is handled by the NoC Channel, which constitutes the central module of the simulation framework. The attached *network engines* are responsible for modeling the traffic characteristics and the impact on performance related to the the on-chip network.

4.2 NoC Exploration Methodology

Our framework enables the systematic design space exploration of most complex on-chip communication networks. During the simulation, the evaluation modules connected to the network engines collect statistical information like resource utilization, latency, and throughput. Based on these data, the system architect designs the communication infrastructure according to the following successive refinement steps:

- During the *initial throughput measurement*, the overall on-chip traffic is functionally captured by means of an unconstrained point-to-point network engine. The resulting communication profile identifies interacting partners and rough throughput requirements.
- The *coarse network partitioning* is dedicated to the identification of the optimum mix of network types. The system architect maps the point-to-point communication to an appropriate set of network types by configuring the NoC Channel with the corresponding set of network-specific engines.
- By iterative *parameter calibration*, the selected communication architecture is fine-tuned to the traffic requirements. Parameters refer to e.g. the bandwidth of a bus system or the queue-length of a crossbar architecture.

Of course the network partitioning has superior impact on the final quality of results. Here our unified approach enables a rapid exploration of totally different network architectures by simply exchanging the network engines. By that the system architect can optimize the communication in an iterative exploration cycle. In case of very complex applications, the simulation driven approach can be complemented with statistical post-processing techniques like regression analysis to reduce the overall design space.

4.3 The NoC Channel

The network independent master interface provides a symmetric request-response communication scheme, that follows the generic TLM channel proposal [25]. The interface implementation inside the NoC Channel translates master requests into an internal Transaction Data Structure (TDS), which contains all relevant information related to the transaction, like e.g. source- and destination address, the actual data packet, and an engine identifier. As illustrated in figure 3, the TDS is then forwarded to the selected network engine.

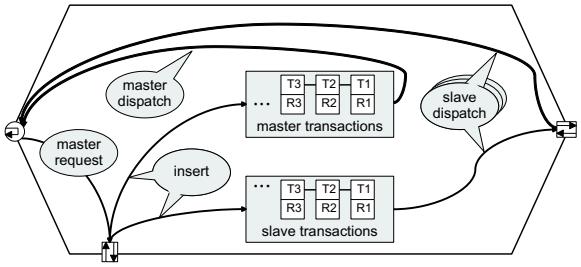

Figure 3: Internal NoC Channel Structure

After the transactions are processed by the network engine, the respective TDS is passed back to the NoC Channel with an attached tag. This tag denotes the calculated arrival time of the transaction. According to this arrival tags T_i, three types of processes inside the NoC Channel deliver the transactions R_i to the destination module. The processes work on two separated list data structures for master and slave transactions, where transactions are sorted according to the arrival tag.

The *insert* process is sensitive to the routed transactions from the network engines and inserts them into the appropriate list structure according to the forward tag and the type of the destination module.

The two types of *dispatch* processes are sensitive to the arrival tags of the head-of-line transaction stored in the list. When simulation time reaches this arrival tag, transactions are dequeued and processed according to their destination module and transaction type. For illustration purposes this processes are presented in a condensed C/C++-like pseudo-syntax.

```
master_dispatch() {
  tds = dequeue_transaction();
  notify_master(tds.id);
}

slave_dispatch() {
 tds = dequeue_transaction();
 if (tds.type == WRITE) {
  slaves[tds.dest]->write(tds.addr, tds.data);
 }
 else {
  slaves[tds.dest]->read(tds.addr, data);
  tds.data = data;
  notify_master(tds.id);
 }
}
```

In case the destination module is one of the attached master modules, this master is notified and subsequently can read the data via the response channel of the master interface. If otherwise the destination is a slave module, the respective slave method is called: write transactions are simply mapped to the `write()` method of the addressed slave, instead for read transactions the slave `read()` method returns the data via a call-by-reference parameter. Thus the TDS is updated and the master is notified on the completion of the read transaction to read the data method via the master response channel.

Since the slave methods are blocking, one dispatch process per attached slave is instantiated. Thus we are able to account for the (potentially significant) execution delays introduced by the attached slaves on this high level of abstraction.

4.4 Network Engine Algorithms

In the following we will elaborate on the algorithms for selected network engines, i.e. basically the calculation of the arrival tag. The selection is by far not complete, but the modular structure enables easy integration of further network topologies and routing algorithms.

The mechanism to notify the NoC Channel of a finished transaction is common for all network engines listed below and implemented by the `notify()` method. Upon calling the latter, the

`insert()` process in the NoC Channel is activated to get the TDS together with the corresponding tag from the network engine.

The *point-to-point engine* models transactions between source and destination module over exclusive link connections. This engine is configured by a bandwidth-matrix $B = (M \times (M + S))$, where M and S refer to the number of attached masters and slaves respectively and b_{ij} denotes the available bandwidth of the connection between producer i and consumer j. A second tag-matrix $T = (M \times (M + S))$ maintains per connection the last arrival tag.

```
add_request(tds) {
 if (b[tds.id][tds.dest] > 0) {
  last_tag = maximum(now(), t[tds.id][tds.dest]);
  arrival_tag = last_tag + tds.size / b[tds.id][tds.dest];
  t[tds.id,tds.dest] = arrival_tag ;
  notify(tds,arrival_tag);
 }
 else { error("no connection"); }
}
```

The point-to-point engine processes incoming requests immediately, whereas the following engines model shared communication resources. For this purpose the requests are stored temporarily until they are handed back to the NoC Channel by an additional `arbitrate()` process.

The *bus engine* models data exchange over a shared bus medium. Internally this engine maintains a list, where pending requests are queued until the bus is granted by the arbitration process. This engine can be specialized towards priority or TDMA like bus arbitration schemes by different implementations of the `select()` algorithm, which determines the grant from the list of pending requests. The priority based bus arbiter simply selects the request with highest priority, whereas the TDMA arbiter is based on a static allocation table. The bus engine is configured by the `bandwidth` parameter to determine the duration of the transaction.

```
add_request(tds) {
 if (list.is_empty()) { wake_up_arbiter() }
 list.enqueue(tds);
}

arbitrate() {
 do {
   tds = select(list);
   arrival_tag = tds.size / bus_bandwidth;
   notify(tds, arrival_tag);
   wait( arrival_tag );
   if( list.is_empty() ) { wait(); }
 } while( true );
}
```

A refined version of the bus engine provides additional configurability with respect to the calculation of the arrival tag. Accounting for detailed timing characteristics of the bus architecture like number of pipeline stages, the re-arbitration latency or consideration of split-transactions significantly improves the simulation accuracy. During our refinement flow we have compared the accuracy for different versions of the bus engine against a fully cycle accurate reference bus model. Whereas the coarse calculation based on the bandwidth is up to 50% inaccurate, configuration with the correct bus specific parameter settings improves the accuracy of the bus engine towards a negligible 3% deviation.

The *crossbar engine* models on-chip network architectures, that perform more than one transaction at the same time. Since crossbar networks differ significantly, it is not possible to capture all possible incarnations with a single engine. The engine elaborated below models Virtual Output Queued (VOQ) [26] architectures for equal-size data packets with a non-blocking, buffer-less $((M + S) \times P)$ crossbar matrix, where P denotes the number of supported priorities. According to the VOQ principle, incoming packets are stored separately per output before they pass through the crossbar matrix. This technique prevents from head-of-line blocking in case of asymmetric traffic or partially blocked outputs and hence improves overall throughput and fairness.

In analogy to the bus engine, the crossbar engine can be specialized to static, weighted and un-weighted arbitration algorithms by different implementations of the `select()` method. The engine maintains an internal VOQ data structure, which queues requests per destination module and per priority. For every arbitration interval, the `select()` algorithm generates a grant matrix G from the head-of-line snapshot S of the VOQ queues.

```
add_request(tds) {
 if (queue[tds.dest][tds.prio].is_empty()) {
  s[tds.dest][tds.prio] = tds.prio;
 }
 queue[tds.dest][tds.prio].append(tds);
}

arbitrate() {
 do {
  g = select(s);
  for (i=0; i<M+S; i++) for (j=0; j<P; j++)
   if (g[i][j] == true) {
    tds =  queue[i][j].dequeue();
    notify(tds,arbitration_interval);
    if (queue[i][j].is_empty()) s[i][j] = -1;
    else s[i][j] = queue[i][j].first().prio;
   }
  wait(arbitration_interval);
 } while(true);
}
```

The implementation of network engines is of course not restricted to the algorithmic modeling style outline above, instead the designer has the full expressiveness of C++ and SystemC to extend the engine library. In particular complex network topologies can be composed from elementary engines by recursively instantiating the NoC Channel inside *hierarchical engines*. For example

- an engine modeling a homogenous multi-hob topology contains an NoC Channel along with master modules representing the network nodes, which are connected using the point-to-point engine.
- an engine modeling a complex multi-layered bus network contains an NoC Channel along with master modules representing the bridges, which are connected using several bus engines.

5. NPU CASE STUDY

In this section we present the results of a design project, where the proposed NoC exploration methodology has been applied to the design of Network Processing Unit (NPU) for IP forwarding. After a brief introduction of the application, we illustrate the benefits of a) a custom optimized communication architecture and b) the proposed exploration framework by comparing our application specific NPU against the commercial IXP2400 NPU from Intel [27].

5.1 IP Forwarding with QoS Support

IP forwarding is a central part of the IP network infrastructure. This challenging application domain combines sophisticated functionality for QoS support with highest performance requirements: at OC-48 wire speed (corresponds to 2.5Gbit/s) the timing budget in the NPU for the processing of a minimum size 48Byte packet is as short as 147ns.

IP Forwarding with QoS Support according to the DiffServ proposal [28] requires the following functional blocks: The *Parser* performs sanity checks on the incoming packet and provides the following units with a unique packet descriptor (PD), which holds all the relevant header information of the respective IP packet. The *Route Lookup Unit* (RLU) performs forwarding of IP Packets based on the longest match table search algorithm. The forwarding decision is based upon destination IP address and the routing table. The *Classifier* classifies incoming packets into Classes of Service (CoS), so the packets are processed according to their negotiated Quality of Service parameters by the following blocks. The *Meter* measures the IP packet rate and drops packets exceeding the negotiated traffic characteristics to protect the succeeding queuer unit from greedy traffic streams. The *WRED* unit drops additional packets employing

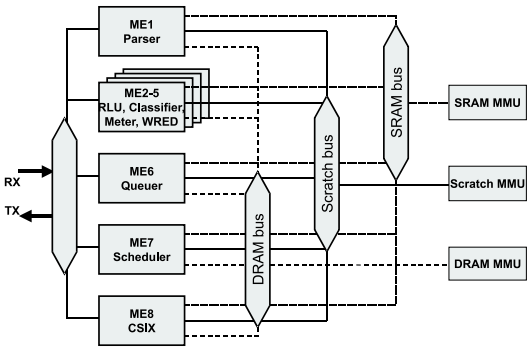

Figure 4: a) Intel IXP2400 Mapping

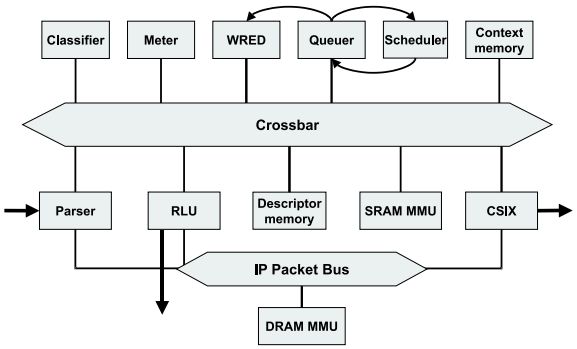

Figure 4: b) NPU Final Architecture View

the weighted RED algorithm [29] to avoid throughput degradation in a TCP friendly manner due to congestion. The *Queuer* stores IP packets according to their CoS until they can be forwarded to the CSIX unit. The *Scheduler* decides on the basis of the priority and the actual fill status of the packet queues in the queuer unit. Finally the *CSIX* unit segments IP packets into fixed size packets in compliance with the standardized CSIX bus protocol [30] to interface the switch fabric. The *Memory Management Unit (MMU)* performs the memory allocation and deallocation of the IP- and PD-memory.

5.2 Exploration of the Communication Architecture

Figure 4 displays the mapping of the IP forwarding application onto two different platforms: 4 a) shows the realization on the IXP2400 reference NPU and 4 b) depicts the final version of the custom optimized communication architecture. To achieve a fair comparison between both NPU platforms, the processing and communication requirements of all functional units as well as the IXP-2400 mapping are taken from the original Intel documentation [31].

According to the general design flow described in section 3, first an untimed functional SystemC model is created from the functional specification document [28, 31]. This functional model is already based on the NoC Channel, but employs only the point-to-point network engine. This model is well suited to validate completeness and functional correctness, but does not yet impose any assumptions on the architectural realization.

This initial setup of the NoC exploration framework was created and validated within 2 months by a team of two experienced engineers. The functional model comprises 6,800 Lines of SystemC code and simulates about 20,000 IP packets per second on a 1.0GHz Pentium III having 256KB L2 cache running Linux 2.4.18.

line	engine descriptor	engine config.	utilization	trn. delay in cycles	IP latency in cycles
1	p2p	unconstr.	0%	0	910
2	p2p	32bit	8%	22	1526
3	ixp2400	32bit_bus	83%	143	4910
4	custom	mix	68%	73	3470
5	sram_p2p	32bit	4%	17	408
6	sram_dual_bus	32bit	78%	125	3000
7	sram_dual_bus	64bit	39%	107	2568
8	sram_tripel_bus	32bit	67%	65	1488
9	sram_crossbar	32bit	39%	65	1560

Table 3: NoC Exploration Results

After annotating the estimated execution delay to each of the functional blocks in conformity with [31], the SystemC model is prepared for the exploration of the communication architecture. An excerpt of the NoC exploration results is printed in table 3. Besides the engine-descriptor and -configuration we list the resource utilization and the mean delay per transaction of the respective NoC setup. The right column contains the overall packet latency introduced by the IP processing chain. Since the timing annotations of the functional units are constant during all our experiments, the measured packet latency clearly unveils the impact of the selected NoC configuration on the system level performance.

According to the exploration methodology described in section 4.2, the point-to-point (p2p) engine is employed to generate an initial traffic profile. The unconstrained p2p engine performs all communication with zero overhead, so the IP latency in line 1 reflects only the timing annotations of the functional units. The 32 bit wide p2p engine in line 2 serves as a 'lower-bound' reference configuration and not as a realistic design option, because - among other unrealistic assumptions - this would implicate a packet descriptor memory with 6 read and 6 write ports.

Besides an insignificant amount of local communication between the buffer and its neighboring units, the statistics generated by the p2p engine admits the classification of the on-chip traffic into two domains: (a) Parser, RLU and CSIX unit regularly access the IP memory (DRAM) to store and forward the arriving IP Packets and (b) all master blocks heavily access the SRAM and scratch memory, where the packet descriptor, routing information, traffic parameters and per flow statistics are stored.

Line 3 in table 3 shows the performance of the engine configuration, which corresponds to figure 4 a) and emulates the mapping to the IXP2400 NPU as proposed in [31]. The huge communication overhead of 4000 cycles is caused by the expensive access of external memories via shared busses. To keep up OC-48 wire-speed processing, the IXP2400 hides the memory latency by exploiting the potential for packet-level parallelism: the application is executed on a functional pipeline formed by 8 micro-engines, each supporting up to 8 parallel processing threads.

The second section in table 3 is dedicated to the exploration of a customized communication architecture for the access to the SRAM and scratch memories, which together account for 3000 cycles in the IXP2400 architecture.

The complete p2p mesh in line 5 is way too expensive and inefficient, while line 6 corresponds to the suboptimal dual bus configuration of the IXP2400 NPU. Duplicating the bandwidth effectively halves the bus utilization, however the sequential access to the 32 bit memories still mark the actual bottleneck. A closer analysis of the simulation results shows, that the scratch memory is best exclusively allocated for the packet descriptor and that the the context information used by the Classifier and Queuer units can be efficiently stored in an additional on-chip context memory. As shown in line 8, the transaction delay is significantly improved as soon as the context memory is accessed through an additional third bus.

Replacing the three separate busses with a dedicated crossbar network introduces a minor performance penalty due to increased arbitration complexity. However this option was selected as the final communication network, because a switched NoC yields several architectural advantages like e.g. improved scalability, simplified block interface and potential for physical optimization.

Based on the proposed NoC exploration framework, the specifi-

cation of the communication architecture has been performed and quantitatively verified within 2 weeks by a team of two experienced designers. In comparison with the reference IXP2400 Intel NPU, the resulting custom NPU configuration in line 4 gains an overall 30% performance improvement along with major architectural optimizations.

After the architecture exploration is finished, the NoC channel is replaced with cycle-accurate communication models, while the functional units are reused by means of adapters bridging the different levels of abstraction. The comparison of packet-based and cycle accurate communication revealed, that the simulation accuracy depends on the type of the network engine: The point-to-point engine as well as the crossbar engine produce 100% correct results, since the "packet nature" of the communication events is preserved at the cycle accurate level. The accuracy of the bus engine is already discussed in section 4.4.

6. CONCLUSION

In the near future, SoC platforms will integrate tens or even hundreds of IP blocks. The resulting demand of on-chip data exchange can only be accomplished by a sophisticated communication infrastructure, which employs local shared bus systems as well as chip-wide network architectures.

This paper presents a unified approach to the conceptualization of heterogeneous on-chip networks. The proposed Network-on-Chip Exploration Framework combines concepts from the traditional networking design environments with recent System Level Design methodologies to enable a rapid exploration of architectural alternatives.

The presented approach has been successfully applied to the system architecture design of an application specific Network Processing Unit, which performs IP forwarding with Quality of Service support. The high simulation speed, modeling efficiency and support in performance monitoring enabled the specification and quantitative evaluation of a sophisticated communication architecture within two weeks by a team of two experienced designers. Comparison aginst the commercial state-of-the-art IXP2400 NPU from Intel clearly demonstrates the value of a) customized communication architectures and b) the proposed exploration framework to rapidly explore and benchmark competing design options. For the considered case-study, the loss of accuracy caused by the packet-based communication paradigm proved to be negligible.

Our future work will focus on the modeling of further network engines and the additional deployment of analytical analysis- and optimization-techniques.

7. REFERENCES

[1] D. Flynn. Amba: enabling reusable on-chip designs. *IEEE Micro*, 17(4):20–27, July-Aug 1997.

[2] IBM CoreConnect. *http://www.chips.ibm.com/products/powerpc/cores*.

[3] Open Core Protocol International Partnership (OCP-IP). *OCP datasheet, http://www.ocpip.org*.

[4] L. Benini, G. De Micheli. Networks on Chips: A New SoC Paradigm. *IEEE Computer*, pages 70–78, January 2002.

[5] Marco Sgroi, M. Sheets, A. Mihal, Kurt Keutzer, Sharad Malik, Jan M. Rabaey, and Alberto L. Sangiovanni-Vincentelli. Addressing the system-on-a-chip interconnect woes through communication-based design. In *Design Automation Conference*, pages 667–672, 2001.

[6] F. Karim, A. Nguyen, S. Dey, R. Rao. On-Chip Communication Architecture for OC-768 Network Processors. In *Proceedings of the Design Automation Conference (DAC)*, 2001.

[7] P. Guerrier, A. Greiner. A Generic Architecture for On-Chip Packet-Switched Interconnections. In *"Proc. Int. Conf. on Design, Automation and Test in Europe(DATE)"*, 2000.

[8] E. Rijpkema, K.G.W. Goossens, A. Radulescu, J. Dielissen, J. van Meerbergen, P. Wielage, and E. Waterlander. Trade Offs in the Design of a Router with Both Guaranteed and Best-Effort Services for Networks on Chip. In *"Proc. Int. Conf. on Design, Automation and Test in Europe(DATE)"*, 2003.

[9] K. Keutzer, S. Malik, A.R. Newton, J.M. Rabaey, A. Sangiovanni-Vincentelli. System-level design: Orthogonalization of concerns and platform-based design. *IEEE Transactions on Computer-Aided Desig of Integrated Circuits and Systems*, 19(12):1523–1543, December 2000.

[10] F. Balarin, Y. Watanabe, H. Hsieh, L. Lavagno, C. Passerone, A. Sangiovanni-Vincentelli. Metropolis: An integrated electronic system design environment. *IEEE Computer*, 36(4):45–52, April 2003.

[11] D. Gajski, J. Zhu, R. Dömer, A.Gerstlauer, S. Zhao. *SpecC: Specification Language and Methodology*. Kluwer Academic Publishers, 2000.

[12] Vladimir D. Zivkovic, Ed Deprettere, Erwin de Kock, Pieter van der Wolf. Fast and Accurate Multiprocessor Architecture Exploration with Symbolic Programs. In *"Proc. Int. Conf. on Design, Automation and Test in Europe(DATE)"*, 2003.

[13] M. Gasteiner and M. Glessner. Bus-based communication synthesis on system level. *AM Trans. Design Automation Electronic Systems*, pages 1–11, January 1999.

[14] Peter Voigt Knudsen and Jan Madsen. Integrating communication protocol selection with partitioning in hardware/software codesign. In *Proc. Int. Symp. on System Synthesis*, 1998.

[15] K. Hines, G. Borriello. Dynamic communication models in embedded system co-simulation. In *Proceedings of the Design Automation Conference (DAC)*, 1997.

[16] A. Baghdadi, D. Lyonnard, N.-E. Zergainoh, A. Jerraya. An Efficient Architecture Model for Systematic Design of Application-Specific Multiprocessor SoC. In *"Proc. Int. Conf. on Design, Automation and Test in Europe(DATE)"*, 2001.

[17] K. Lahiri, A. Raghunathan S. Dey. Performance analysis of systems with multi-channel communication architectures. In *"Proc. Int. Conf. VLSI Design"*, pages 530–537, 2000.

[18] OPNET. *http://www.opnet.com*.

[19] Pierre G. Paulin, Chuck Pilkington, and Essaid Bensoudane. StepNP: A System-Level Exploration Platform for Network Processors. *IEEE Design & Test of Computers*, 19(6):17–26, Dev-Dec 2002.

[20] SystemC initiative. *http://www.systemc.org*.

[21] T. Grötker, S. Liao, G. Martin, S. Swan. *System Design with SystemC*. Kluwer Academic Publishers, 2002.

[22] O. Ogawa, K. Shinohara, Y. Watanabe, H. Niizuma, T. Sasaki, Y. Takai, S. Bayon de Noyer and P. Chauvet. A Practical Approach for Bus Architecture Optimization at Transaction Level. In *"Proc. Designers' Forum, Int. Conf. on Design, Automation and Test in Europe(DATE)"*, 2003.

[23] J.A. Rowson and A. Sangiovanni-Vincentelli. Interface-Based Design. In *Proceedings of the Design Automation Conference (DAC)*, 1997.

[24] W. Cesario, A. Baghdadi, L. Gauthier, D. Lyonnard, G. Nicolescu, Y. Paviot, S. Yoo, A. Jerraya, M. Diaz-Nava. Component-Based Design Approach for Multicore SoCs. In *Proceedings of the Design Automation Conference (DAC)*, 2002.

[25] Norman Weyrich, Anssi Haverinen. *A SystemC Generic Transaction Level Communication Channel, http://www.systemc.org*, 2003.

[26] Y. Tamir, G. Frazier. High performance multi-queue buffers for VLSI communication Switches. In *Proc. Of 15th Ann. Symp. On Comp. Arch.*, 1988.

[27] Intel Network Processors. *http://developer.intel.com/design/network/products/npfamily/*.

[28] An Architecture for Differentiated Services. *http://www.ietf.org/rfc/rfc2475.txt*.

[29] S. Floyd, and V. Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on Networking*, 1(4):379–413, August 1993.

[30] The Network Processor Forum. *founded by CSIX/CPIX members in 2001 http://www.npforum.org*.

[31] S. Lakshmanamurthy, K.-Y. Liu, Y. Pun, L. Huston, U. Naik. Network Processor Performance Analysis Methodology. *Intel Technology Journal*, August 2002.