An Evaluation of the Scalable GigaNetIC Architecture for Access Networks

Jörg-Christian Niemann[1], Mario Porrmann[1], Christian Sauer[2], Ulrich Rückert[1]
[1]Heinz Nixdorf Institute, University of Paderborn, Germany
[2]Infineon Technologies, Corporate Research, Munich, Germany
[1]{niemann, porrmann, rueckert}@hni.upb.de, [2]{christian.sauer}@infineon.com

Abstract

We present an architecture for network processing nodes based on a massively parallel processor structure. Due to its regularity, our architecture can be easily scaled to accommodate a range of packet processing applications with disparate performance and throughput requirements at high reliability. Furthermore, the composition from predefined building blocks guarantees fast design cycles and eases system verification. For particular resource efficiency in terms of power consumption, computational performance, and area requirements, specialized hardware accelerators can be embedded into the tailored processor cluster, which have been optimized for a particular target application. We demonstrate our approach using a real-world network access scenario that implements a full Internet protocol based digital subscriber line access multiplexer (IP-DSLAM) on our architecture. For this scenario, we achieve substantial increases of performance with only a slight area increase of less than 0.3 %. At the same time, the processors are strongly relieved and are thus available for the remaining tasks.

1 Introduction

Especially network applications with their ever growing performance requirements are predestined for parallel processing systems [1]. By processing uncorrelated flows of packets concurrently, these systems are able to achieve the required throughput, i.e., wire speed. Evolving markets and changing protocols demand flexible architectures that at the same time are reusable among different application domains with disparate performance requirements. Our scalable approach based on HW (Intellectual Property) modules enables us to design appropriate architectures with respect to computational performance, area requirements, and power consumption.

The backbone of our GigaNetIC architecture [2] (cf. Fig. 1) is the GigaNoC, a hierarchical hybrid network-on-chip (NoC). Local on-chip busses cluster small numbers of processing elements (PEs). Switch boxes (SBs) connect these clusters at the higher level. Their interconnection can form arbitrary topologies, such as meshes, tori, or butterfly networks. The parallel and redundant structure of our switch box approach offers a high potential for fault tolerance: in case of a failure of one or more components, other parts can take over the respective operations. This can cause a significant increase of production yield. Furthermore, the topology can be efficiently integrated and provides high scalability due to its modularity and regular interconnect scheme. New components, e.g., hardware accelerators and I/O interfaces, can be easily attached to any switch box. These IP blocks transparently handle and terminate our on-chip communication protocol. The initial silicon of our architecture will consist of 32 processing elements and eight switch boxes. Each PE is connected to the on-chip bus of a SB via a multiprocessor cache (cf. Fig. 1). We have chosen a mesh topology, due to efficient hardware integration.

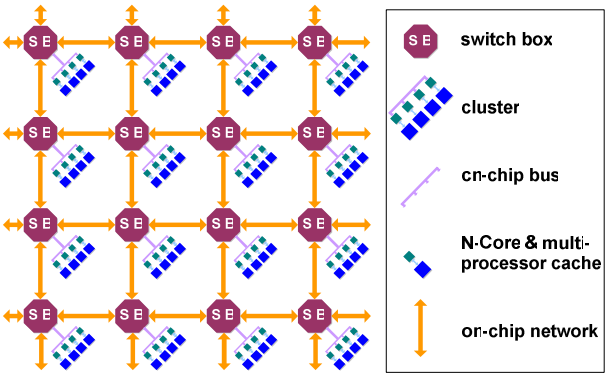

Fig. 1: Scalable parallel GigaNetIC SoC architecture based on hierarchical IP blocks

One particular application of our scalable processor architecture is the network access domain, i.e., the aggregation of customers' first/last mile connections to the Internet via a service provider's network. Core systems for this task are DSLAMs. We use this application for verification and demonstration of our system and analyze compute-intensive tasks for system optimizations.

Concerning the system evaluation and verification environment we follow a top-down approach. To test the software in an early design phase, and to optimize different parameters of the architecture accompanied with a much faster simulation speed, we have developed SiMPLE, a simulation model of the architecture in SystemC. In this design flow, global system parameters are specified

at a high level of abstraction. Coming closer to the hardware implementation, these parameters are iteratively refined. For the verification of the basic hardware blocks at the register-transfer level (RTL), we use a VHDL simulation environment. At this detailed level, simulation is very slow. However, since the complete environment is synthesizable, it can be mapped to a hardware emulation system. By using our rapid prototyping system RAPTOR2000, we have analyzed an exemplary system of our network processor with basic functionalities of our application scenario as a proof of concept.

The paper is organized as follows. In the following section, we discuss main concepts and building blocks of our architecture in more detail. Section 3 provides an overview of the IP-DSLAM application and the used configuration. The evaluation and verification setup in Section 4 is used to characterize our scalable architecture. Section 5 provides results from synthesis and performance evaluation and is the main contribution of this work. The paper is concluded in Section 6. We discuss related work throughout the paper.

2 Architectural Building Blocks

A central goal of our approach [3] is a parameterizable network processor architecture in respect to the number of the clusters, the processors instantiated per cluster, the provided hardware accelerators, and the available bandwidth of the on-chip communication channels. By this, a high reusability of our architecture can be guaranteed. Further advantages of such a uniform system architecture lie in the simplified testability and verification of the circuit and in a more homogeneous programming model. The programmer is relieved from handling the on-chip communication protocol explicitly since routing, memory management, and I/O are transparently controlled by the SBs. At the cluster level the programming model is based on a proprietary parallelizing ANSI C compiler. A preceding tool partitions and schedules the tasks for the individual processor clusters.

2.1 Processing Element

At the processor level, we use our 32 bit RISC processor N-Core [4][5], as processing element of the system. The core is a softmacro and can be adapted to the needs of the respective area of application. Instructions have a fixed width of 16 bit, providing a high code density, which is of special importance for embedded systems with limited memory resources. The instruction set can be extended through additional operations, because of 11% free opcode space. Therefore, it is possible to optimize the architecture for specific application domains, e.g. the networking area. In this field we have already implemented several new instructions for header modification and for encryption algorithms used in IPsec protocols [6][5]. These instruction set extensions can be seen as the smallest HW blocks used in our architecture. Additionally, the N-Core provides a coprocessor interface for hardware accelerators, facilitating further acceleration by adding larger HW blocks. The processor core has been verified in silicon successfully [4].

2.2 Switch Box

At the cluster level, switch boxes act as high speed routing nodes, which combine the individual processor clusters with each other. The on-chip communication is based on a packet switched network-on-chip [7][8]. Data is transmitted by means of packet fragments – Flits (Flow Control Digits) – that represent the atomic on-chip data transmission unit [9]. A locally connected processor cluster consists of four processors that are connected to the SB by an AMBA or Wishbone on-chip bus. The number of communication ports is variable and depends on the anticipated on-chip network topology.

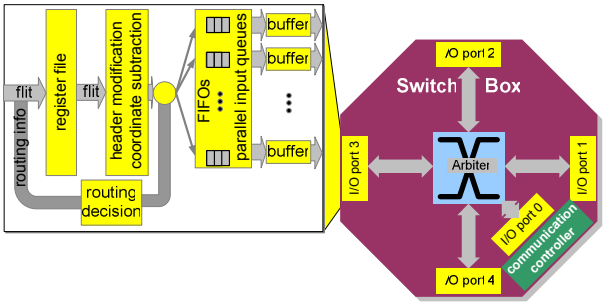

Fig. 2: Switch Box HW block

Despite its simplicity, this architecture allows parallel operation and a pipelining of the processor fields. Additionally, it guarantees almost equal link lengths and thus an identical and short propagation delay of the signals between SBs. Furthermore, the parallel structure of the SB concept allows a high fault tolerance: if the software detects a malfunctioning processor unit, others can take over the pending tasks.

Each SB is also connected to a multiprocessor array by an extra input/output port (port 0 in Fig. 2), which interfaces the communication controller (CC). The CC transfers data to and from the local clusters. The SB consists of two main parts. The first part combines the I/O port and the crossbar to form the communication structure that ensures that the data packets are smoothly transmitted and reach the correct switch box output port on the basis of the routing strategy. The second part of the SB comprises the control structures that serve as an interface between the processor field and the on-chip network. This task is performed by the communication controller, which is located between the port 0 and the bus of the local cluster.

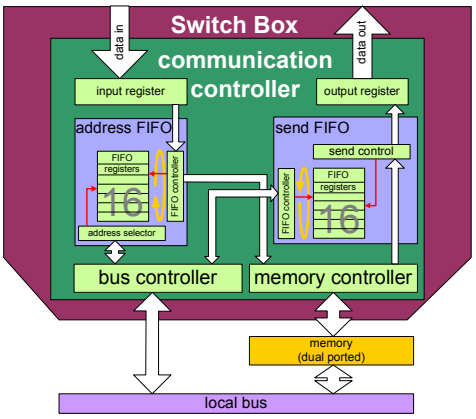

Fig. 3: Communication Controller

The communication controller performs the following tasks: It receives the flits, reorders them if necessary, and forwards them to the connected HW blocks, e.g., processors or hardware accelerators. Another important function is the initialization of the PE code memories at system startup.

2.3 Hardware Accelerators and Other HW Blocks

Beside the processor cores and the local memories, further HW blocks can be integrated using several ways. The processor supports the connection of hardware accelerators via a coprocessor interface. If those units are supposed to be available to a number of processors, they can be coupled via the local bus at cluster level. Coupled closely to the processor field, these HW blocks are integrated through additional master/slave interfaces of the local bus system and addressed via memory-mapped I/O ports. Beside hardware accelerators, additional modules, such as UARTs for debugging purposes, can be integrated.

At the SoC level, more independent units can be connected to any SB and are addressable via the on-chip network. These can be loosely coupled hardware accelerators enqueued in the data path, such as encryption modules. These can also be units that realize outwards connections such as memory controllers for external memory or Ethernet controllers that take over the connection to external networks. To connect a unit to the GigaNoC, a CC is connected to the respective component and a SB port. The CC performs the conversion of the data into the flit protocol and the termination of the protocol, respectively. Due to this connection mode, the units are universally suited and enable a slight adaption of the system to new application areas.

3 Network Application Scenario(s)

The deployment in network nodes is a major focus of our architectural platform. Due to its scalability, a wide range of network applications in access, edge, and core networks with different processing and performance requirements can be covered. In the following, we concentrate on access networks and briefly introduce our IP-DSLAM application scenario that describes a full IP based DSL access multiplexer system. This scenario is used for the subsequent performance evaluation.

3.1 IP-DSLAM Application

DSLAMs connect individual customers with the broadband service provider's network. Today's DSLAMs can become quite large systems by aggregating several thousands of subscriber lines. They are built modularly out of different line cards aggregating xDSL lines, a backplane connecting line and uplink cards, and one or two uplink cards providing access to the service provider network (ISP), see Fig. 4. A *maximal* system configuration may contain up to 64 line cards, where each card aggregates up to 96 DSL lines. Usually, the number of cards and ports per card vary with DSL version and protocol mix. In [10], typical configurations with 32 line cards and up to 72 ports per card depending on the DSL protocol version are reported.

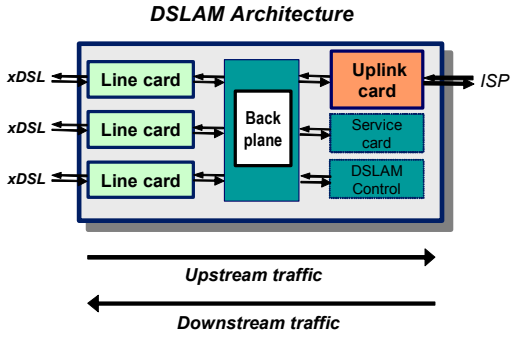

Fig. 4: DSLAM system and components

Common processing steps experienced by each packet on its way through the IP-DSLAM system are: Layer-2 protocol termination (Ethernet/AAL5), IP header check, IP address validation, 5-tuple classification to determine the destination port and traffic class, policing and scheduling for QoS, forwarding, and layer-2 encapsulation. Usually, upstream and downstream packet flows require additional direction-dependent processing steps, e.g., multicast duplication in downstream direction. In [10] these steps are explained in more detail.

3.2 Scenario Setup

For the subsequent performance evaluation, we assume a maximal setup with 96 ports per line card (either

SHDSL or VDSL), 64 line cards, and one uplink card. Workload for the system is generated using IPv4 packets. At the link layer, Ethernet and/or ATM are used. In order to derive peak performance and throughput requirements for our architecture, we look at the worst-case for this scenario, i.e., all DSL lines utilize their full bandwidth, are active at the same time, and process only minimum size packets. Of course, typical real world scenarios may require fewer resources due to more favorable packet length distributions and other statistical traffic properties that can be exploited. However, we are interested in the upper bound of the imposed system load.

3.3 Benchmarking Methodology

The performance analysis for our scalable architecture follows an application driven approach. Based on a complete and performance indicative IP-DSLAM reference implementation, we identify system bottlenecks and iteratively explore architectural alternatives. At the system-level, weight and interaction of tasks need to be known for performance indicativeness and evaluation of different partitioning and mapping decisions. For this reason, we extended our initial approach, described in [11], to a full IP-DSLAM reference application [10] that is platform independent. Similar to the standard cores in [10], we map this reference to the N-Core using ANSI C as intermediate description. The software implementation of tasks is changed manually when specialized hardware is incorporated. In case of instruction set modifications, the C compiler is retargeted [6][5].

4 Evaluation & Verification Environment

Before presenting performance and synthesis results in the next section, we describe the various evaluation and verification environments that have been used to obtain our figures. As our system architecture is implemented fully on various abstract levels we have several verification and simulation options. At Register Transfer Level (RTL), we use Modelsim from Mentor Graphics for the verification of the hardware building blocks, that is, for simulation and also for code coverage analysis. Unfortunately, simulation of the full system at this detailed level is very slow. However, since the environment is synthesizable, we deploy our hardware emulation system RAPTOR2000. To enable software development for the whole system at an early design phase, we also provide a SystemC simulation model of the complete architecture. This way, we are able to test and optimize the system together with the C programmed application.

4.1 SystemC-based Simulation

To test the software in an early design phase and to optimize different parameters of the architecture accompanied with a much faster simulation speed, we have developed the simulation environment SiMPLE (SystemC integrated Multiprocessor Level Environment) of the architecture in SystemC. SiMPLE contains a cycle-accurate simulation model of the N-Core, parameterizable caches and SBs. We follow a strict top-down design flow: global system parameters are specified on a high level of abstraction; coming closer to the hardware implementation, these parameters are iteratively refined.

Although this cycle-accurate model is on a high abstraction level, the simulation for a system consisting of 32 PEs, 32 caches, 8 SBs and 8 local busses is about 2kHz on a 3 GHz Pentium-4 PC. To speed up the simulation, we mapped main parts of the system to an FPGA-based rapid prototyping system. In addition, this emulation enables insights into the behavior of the hardware in more detail.

4.2 FPGA-based Rapid Prototyping

The RAPTOR2000 system [12] consists of a PCI card and a set of daughter boards that carry, e.g., FPGAs or Ethernet PHYs. By using this rapid prototyping system, detailed hardware analyses can be done in short time. The design under consideration can be embedded and tested in a real-world system environment, e.g., attached to an Ethernet module, and can be tested in the whole system environment. When using our RAPTOR2000 rapid prototyping system, we achieve a speed-up of at least 10,000 compared to the VHDL simulation, and a speed-up of about 6,250 compared to the SystemC simulation, on a 3 GHz Pentium-4.

Using two Virtex II 4000 FPGAs, fully utilized with a setup of: 1 SB, 32 kB packet buffer, 2 N-Cores with 32kB program memory each, a timer, a PIC, a wishbone bus, 2 Ethernet interfaces and a UART for debugging, we are able to test the application software with real data traffic. With this implementation running at 12.5MHz we carry out an exact analysis of the bus and NoC usage.

5 Realization and Performance Analysis

In this section, we evaluate the potential of our scalable architecture. We start with a *basic system*, containing 32 PEs and the necessary communication infrastructure. We estimate area and evaluate its performance mapping the most compute-intensive tasks of the IP-DSLAM scenario in software onto the PEs. Then we extend the system with hardware accelerators and instruction set modifications and compare the results of this *enhanced system* to the original system. This will demonstrate how flexibility is traded off with performance and area.

As a prerequisite, we analyzed the scenario and identified the most compute-intensive functions in software. Three functions have been selected for the performance

analysis of the system: IP header check, CRC8, and CRC32. The IP header check is carried out each time an IP packet arrives. Among the functions that have to be accomplished are: checking the version of the IP protocol, checking the time-to-live field, and checking or recalculating the IPv4 checksum. The CRC8 (cyclic redundancy check) is necessary, e.g., to protect the header of ATM cells. A CRC32 checksum is important for both ATM and AAL5 segmentation Ethernet framing.

5.1 Basic System – Area Estimation

The standard cell synthesis delivers first results for the future ASIC implementation of a 32 PE system. The area and clock frequency estimations for the main components of the network processor are based on synthesis results for two industrial standard cell technologies in 130nm and 90nm. The entire size of this system is about 50 mm² and 43.7 mm², respectively (cf. Table 1).

Table 1: Synthesis results for the main building blocks of the SoC architecture

SoC main components	Area [mm²]		Frequency [MHz]	
	130nm	90nm	130nm	90nm
32 N-Cores	32 x 0.16	32 x 0.12	205	285
8 switch-boxes [with 5 ports]	8 x 1.129	8 x 0.53	560	650
32 local RAMs, (32 KB) + 8 local packet buffers (2 x 16 KB)	32 x 0.875 + 8 x 2 x 0.466	32 x 0.875 + 8 x 2 x 0.466	400	450
8 local on-chip busses	8 x 0.05	8 x 0.02	211	290
total	**50.01**	**43.7**	**205**	**285**

Unless otherwise noted, all values are given for typical operating conditions.

5.2 Basic System – Performance Estimation

The basic system consists of N-Core processors, local bus components of the processor clusters, switch boxes, and basic I/O interfaces (cf. Table 1). The basic system performs all tasks solely in software. The necessary system size is determined in consideration of the application scenario. Table 2 shows the required function calls per second for the most compute-intensive tasks of the DSLAM scenario.

Table 2: Function calls per second

function calls [millions per second]	line card		uplink card	
	SHDSL	VDSL	SHDSL	VDSL
IP header check	0.375	0.563	24.000	264.000
CRC8	0.906	5.660	57.962	362.264
CRC32	12	75	768	4800

For the IP header check, different variants have been implemented that primarily differ in respect to the calculation of the checksum. The fastest variant uses 32 bit additions and needs a total of 108 cycles. The fastest 16 bit variant, similar to the implementation mentioned in the RFC1071, needs 111 cycles. Using a line card with 96 DSL ports under full load (SHDSL: 2Mbit/s uplink / downlink), the worst case scenario would be that 0.38 million header checks per second and per line card were to be carried out in the uplink, based on the assumption that only IP packets with a minimal length of 64 bytes are transmitted. This means that one processor per line card would be needed and 20% of its capacity used (cf. Table 3). If one analyzes the uplink card with a full extension of the DSLAM (64 line cards), at this stage, the full capacity of 13 processors would be used exclusively for the IP header check, or, when using one N-Core, this would have to be operated with 2.6GHz. With VDSL (3/22Mbps uplink/downlink), 140 CPUs would be needed on the uplink card or, one N-Core would have to run with a clock rate of 28.5GHz to handle this load on its own.

The software implementation of the CRC8 needs 71 clock cycles per 32 bit (4 byte header) ATM cell on the N-Core. With SHDSL, the worst case outcome would be a data volume of 384 Mbits per line card. This equals about 900,000 ATM cells needing a checksum calculation. With this, 31% of the N-Core's capacity is used. If the ATM cells on the uplink card were to be checked also, this would mean that 21 N-Cores were needed only for the CRC8 checksum in the case of SHDSL and 126 CPUs in the case of VDSL. However, this does not take into consideration that there is always an additional parallelism overhead; as a consequence, more processors were to be employed.

Table 3: Required N-Cores for the given scenario

required CPUs for task:	130nm				90nm			
	line card		uplink card		line card		uplink card	
	SHDSL	VDSL	SHDSL	VDSL	SHDSL	VDSL	SHDSL	VDSL
IP header check	0.20	0.30	12.64	139.08	0.14	0.21	9.09	100.04
CRC8	0.31	1.96	20.07	125.47	0.23	1.77	14.44	90.25
CRC32	5.21	32.56	333.42	2083.90	3.75	23.42	239.83	1498.95

The CRC32 in the case of SHDSL, imposes a computational load that uses the capacity of the 6 or 33 N-Cores with VDSL on the line card (130nm technology). Here, the N-Core needs an average of 89 clock cycles per 32 bit. As far as the uplink card is concerned, due to the higher data volume, these numbers are even higher. Here too, the numbers argue for the use of a hardware accelerator. Table 3 summarizes the necessary processor numbers in consideration of the given tasks.

Table 4: Required size of the system

technology	system parameters	line card		uplink card	
		SHDSL	VDSL	SHDSL	VDSL
130nm	required CPUs	8	38	368	2350
	overall system size [mm²]	7.3	35.0	335.8	2144.7
90nm	required CPUs	6	28	265	1691
	overall system size [mm²]	3.6	15.5	146.8	934.4

Finally, we obtain the system parameters of the required architecture that is able to fulfill the mentioned functionalities (cf. Table 4). It is noticeable that systems for the uplink card can not be implemented without appropriate hardware accelerators. For the line card, the basic architecture is able to fulfill the requirements of SHDSL (90 nm and 130 nm) and for VDSL (90 nm). The next section describes enhancements of the basic system

to increase the performance for dedicated tasks. This decreases the costs by means of area and power consumption and optimizes the resource efficiency of the system.

5.3 Enhanced System

The enhanced system integrates hardware accelerators, which relieve the processors from the compute-intensive tasks. It also considers potential instruction set extensions. The residual processors can be used for control functions and tasks that are not suitable for hardware acceleration. Accelerators connected locally may operate at core frequency those coupled loosely to the system allow higher clock speed.

Before explaining the integration of coprocessors into our architecture, we investigate acceleration potential of the N-Core by application-specific instruction set extensions. In [5], we have presented a tool-chain that facilitates the required optimizations for the software (i.e., integration of new instructions into the compiler) as well as for the hardware (i.e., integration of new instructions into the hardware architecture of our N-Core). For network applications, our analyses have shown that a combination of the instructions LDW (load word) and XOR is rather promising. The implementation of the resulting super instruction XORLDW has no impact on the critical path of the processor and increases the processor area only by 0.03%. With this slight modification of the processor, a speed-up of 1.1 is achieved for CRC calculation. Table 5 shows the resource requirements for CRC8 with 32bit input data. Power consumption decreases only marginally because of our changes in the control structure of the processor. But the energy consumption is reduced by nearly 10% for both VLSI technologies.

Table 5: System parameters of the original and extended processor

technology	Core	total area		total power		total energy	
		absolute (µm²)	*relative increase*	*absolute (mw)*	*relative decrease*	*absolute (nWs)*	*relative decrease*
130 nm	N-Core	158012	-	10.084	-	2.81	-
	N-Core (XORLDW)	158489	0.301%	10.080	0.04%	2.53	9.89%
90 nm	N-Core	121280	-	9.009	-	2.37	-
	N-Core (XORLDW)	122314	0.845%	9.002	0.07%	2.13	9.93%

For the network applications that are analyzed here, a speed up of 10% in comparison to a pure software solution is not sufficient for CRC calculation. Therefore, more complex hardware accelerators are presented in the following. During the design space exploration a decision has to be made as to whether it is sufficient to use small accelerators with limited impact on performance or whether more complex accelerators have to be embedded that offer higher performance and relieve the processor but require additional resources.

The hardware accelerator that has been implemented for IP header check needs 8 clock cycles to perform the complete functionality. If the coprocessor is attached to our switch box via the communication controller, we can guarantee that the NoC is capable of writing the required data into the buffer of the accelerator with the needed performance. Thus, up to 228 millions header checks per second can be performed if the accelerator is working at a clock frequency of 1.82 GHz (90 nm). Even if the accelerator operates with the same (low) clock frequency as the PEs, 35.6 million header checks can be performed per second. If we compare these results to the 2.6 millions checks that are performed in one second with the software implementation on our N-Core, a speed-up between 13.7 and 87.7 can be achieved. The area is depending on the maximum clock frequency and ranges from 0.012 / 0.008 mm² (130nm / 90nm) to 0.0157 / 0.0146 mm² (130nm / 90nm) at maximum clock speed (cf. Table 6).

Table 6: Required area for the HW accelerators

technology	IP header check		CRC	
	max frequency [GHz]	required area [mm²]	max frequency [GHz]	required area [mm²]
130nm	1.69	0.0157	1.42	0.0115
	0.54	0.0120	0.96	0.0102
90nm	1.82	0.0146	1.41	0.0098
	0.49	0.0078	0.775	0.0066

For CRC calculation, a hardware accelerator has been developed, too. Due to its parallel implementation, it processes 32 bit input data in one clock cycle. The unit performs CRC8 as well as CRC32 checks and achieves a speed-up of up to 630 (90 nm) if the interfaces deliver the data with the required bandwidth. If the coprocessor is attached to the local bus, additional clock cycles for memory access have to be considered. Without congestion, four clock cycles are required for memory access. Table 6 gives an overview over the area requirements for different operating frequencies. The first clock frequency is the maximally achievable clock rate for the specific technology. Due to less strict timing constraints, the second clock frequency is lower, consequently, the area requirements are smaller.

Table 7: Needed coprocessors for the given scenario with local bus connection

needed units @205MHz	(optimal memory access)				(non-optimal memory access)			
	line card		uplink card		line card		uplink card	
	SHDSL	VDSL	SHDSL	VDSL	SHDSL	VDSL	SHDSL	VDSL
IP header check	0.015	0.022	0.937	10.3	0.102	0.154	6.6	72.1
CRC8	0.004	0.028	0.283	1.8	0.181	1.132	11.6	72.5
CRC32	0.059	0.366	3.746	23.4	0.527	3.293	33.7	210.7

Table 7 shows the number of coprocessors that are required if the three tasks are performed exclusively in hardware. For this example, the coprocessors are attached via the local on-chip bus. Therefore, they are clocked synchronously to the PEs with a much lower frequency (205MHz) than possible. We differentiate between optimal memory access (i.e., no congestion) and a conservative bus arbitration. In the latter case (non-optimal mem-

ory access), we suppose that bus accesses are uniformly distributed between all units (here 5). It appears that a single IP header checker and one CRC module are sufficient for the implementation of a line card in the two DSL variants. For the processing-intensive applications on the uplink card, the number of required coprocessors increases dramatically if they are attached to the local bus and operate with reduced clock frequency.

Table 8 points out the acceleration that could be achieved if the coprocessors are integrated into the datapath of the system, working with their maximum clock frequency. In this case, it could be necessary to enhance the NoC, e.g., by increasing the payload of the flit protocol, to achieve the required high on-chip bandwidth. These aspects remain to be analyzed further. For the line card, one hardware accelerator is sufficient for the IP header check, independent of the used VLSI technology. One CRC coprocessor can be utilized for both CRC8 and CRC32 since the unit is never utilized more than 6%.

Table 8: Needed coprocessors for the given scenario with high-speed on-chip connection

needed units @max. frequency	130 nm				90 nm			
	line card		uplink card		line card		uplink card	
	SHDSL	VDSL	SHDSL	VDSL	SHDSL	VDSL	SHDSL	VDSL
IP header check	0.002	0.003	0.114	1.250	0.002	0.002	0.105	1.160
CRC8	0.001	0.004	0.041	0.255	0.001	0.004	0.041	0.257
CRC32	0.008	0.053	0.541	3.380	0.009	0.053	0.545	3.404

For the implementation of uplink cards, coprocessors are inevitable but one IP header check unit and 4 CRC coprocessors deliver the required performance. Of course the embedded processors are also required in this scenario. The N-Cores are used for remaining tasks and control purposes as well as for future extensions of the system. This is especially important in respect to increasing the time in market. During the design space exploration, a system has to be defined that offers the required performance and flexibility. Performance is achieved by integrating an appropriate number of hardware accelerators while flexibility is achieved by integrating embedded processor cores.

5.4 Performance Comparison

Table 9 shows the area requirements of an exemplary network processor that meets the requirements of line cards and uplink cards. Compared to the basic system (without hardware extensions; cf. Table 1), the area has only increased by 0.23%. However, in contrast to the basic implementation, the extended system is able to cope with the SHDSL scenario and with the VDSL scenario for the line card without any restrictions when using the same clock frequency for all components (cf. Table 4, Table 7).

To assure a correct processing of the VDSL scenario on the uplink card under full load and with the usage of the coprocessors (running at the same clock speed as the other SoC components), the architecture should be extended to a 8x4 cluster. At each cluster, one IP header check module and eight CRC units are required. Furthermore, one had to guarantee that the flits at their destination always in time. This may force an enlargement of the NoC bandwidth. This architecture would require approx. 203 mm² / 178 mm², and the SoC would contain 256 PEs, 4 MB memory as well as 288 coprocessors.

Table 9: Exemplary system with 4x2 clusters for a given DSLAM scenario

SoC main components	Area [mm²]	
	130nm	90nm
32 N-Cores	5.12	3.84
8 switch-boxes [with 5 ports]	9.03	4.24
32 local RAMs, (32 KB) + 8 local packet buffers (2 x 16 KB)	35.46	35.46
8 local on-chip busses	0.40	0.16
2 IP header check coprocessors	0.03	0.03
8 CRC coprocessors (capable of CRC8, CRC32)	0.09	0.08
total	**50.13**	**43.80**

Table 10 compares the performance of the basic system and the enhanced system that includes the hardware accelerators for CRC and IP header check. The basic system is only suited for the SHDSL scenario whereas the enhanced system meets the requirements of the SHDSL as well as of the VDSL scenario. Executing, for example, the IP header check on a single N-Core in the basic system, leads to a performance of 1.9 million header checks per second while the two hardware accelerators in the enhanced system deliver a performance of more than 51 million function calls per second. This results in a speed-up of 27.

Table 10: Performance – basic system vs. enhanced system (line card SHDSL)

max. performance for task @205MHz	basic system		enhanced system		
	used N-Cores	possible fct. calls/ops per second	units	possible fct. calls/ops per second	speed-up
IP header check	1	1,90E+06	2	5,13E+07	27
CRC8	1	2,89E+06	4	8,20E+08	284
CRC32	6	1,38E+07	4	8,20E+08	59
remaining N-Cores	24	4,92E+09	32	6,56E+09	
					increase
area		50,01		50,13	100,24%

6 Conclusion

In this paper, we have presented a scalable, IP(Intellectual Property)-based network processor architecture that is adaptable to different application domains in respect to performance, power consumption and area requirements. We characterized this architecture by key figures for achievable performance and area requirements based on elementary tasks for an IP-DSLAM scenario. Currently, we are in the process of taping out the initial silicon for the prototype realization in 90nm.

Concluding this work, we found that with the aid of our powerful GigaNoC, application-specific architectures can

be designed in a resource-efficient way. For especially compute-intensive tasks, hardware accelerators can simply be integrated. The coupling takes place either at the local bus or, with the presented communication controller, on an arbitrary port of a switch box. The system is easily adaptable to dedicated application scenarios, can be optimized, and is scalable to other domains of network processing. By using coprocessors, we achieved substantial increases in performance by factors between 27 up to 284 with only a slight area increase of less than 0.3%. The embedded PEs could be strongly relieved and are thus available for additional tasks. We apply a library-based programming approach that can be maintained across various hardware and software implementations and thus facilitates a simple integration of the new hardware units.

Our simulation environment SiMPLE facilitates an early verification of the application software for the entire system. With this environment, we determine key architectural parameters for a set of networking applications and verify the functionality of our system early in the design phase. Furthermore, our rapid prototyping system RAPTOR2000 facilitates the fast low-level analysis of our architecture concept. We are able to test the hardware under real-world conditions. The results gained from SiMPLE can immediately flow into the hardware design and can then be verified on FPGA basis. Especially compute-intensive tasks can be recognized semi-automatically by our tool chain. These tasks are accelerated through instruction set extensions of the processors or through dedicated coprocessors. Consequently, we can increase the resource efficiency of the system.

Our future work comprises the integration of additional hardware accelerators such as CAMs for fast route look-ups and enhanced QoS functionality. We also analyze new instruction set extensions for additional functions of the IP-DSLAM scenario to achieve speed-ups on the control plane. Currently, our system uses a globally synchronous clock. In order to simplify the extension to larger systems, we will change to a locally synchronous (at the cluster level) and globally asynchronous architecture.

Acknowledgements

This work has been supported by the German Government (BMBF) grants 01M3062A (GigaNetIC) and 01AK065A (PlaNetS) and Infineon Technologies AG, especially the department CPR ST (Prof. Ramacher).

References

[1] Comer, D. Network Systems design Using Network Processors. Prentice Hall, 2003.

[2] J.-C. Niemann et al, A holistic methodology for network processor design, In Proc. of the Workshop on High-Speed Local Networks, 28th Conference on Local Computer Networks (LCN2003), p. 583-592, 2003, Germany.

[3] J.-C. Niemann, M. Porrmann, U.Rückert, A Scalable Parallel SoC Architecture for Network Processors, in Proc. IEEE Computer Society Annual Symposium on VLSI (ISVLSI), Tampa, FL., USA, 2005.

[4] D. Langen, J.-C. Niemann, M. Porrmann, H. Kalte, U. Rückert, Implementation of a RISC Processor Core for SoC Designs – FPGA Prototype vs. ASIC Implementation, In Proc. of the IEEE Workshop on Heterogeneous reconfigurable Systems-on-Chip, 2002.

[5] J.-C. Niemann et al., Network Application Driven Instruction Set Extensions for Embedded Processing Clusters, In Proc. PARELEC 2004, pp. 209-214, Dresden, Germany, 2004.

[6] U. Kastens, D. K. Le, A. Slowik, M. Thies, Feedback Driven Instruction-Set Extension, In Proc. of ACM SIGPLAN/SIGBED 2004 Conf. on Languages, Compilers, and Tools for Embedded Systems (LCTES'04), Washington, D.C., USA, June 2004.

[7] W.J. Dally and B. Towles, Route packets, not wires: On-chip interconnection networks, in Proceedings of DAC 2001, 2001, pp. 684–689.

[8] J. Duato, S. Yalamanchili, and L. M. Ni, Interconnection Networks: An Engineering Approach, Morgan Kaufmann Publishers, 2003.

[9] Zhonghai Lu and Axel Jantsch, Flit admission in on-chip wormhole-switched networks with virtual channels, In Proceedings of the International Symposium on System-on-Chip 2003, November 2004.

[10] C. Sauer, M. Gries, S. Sonntag, Modular Reference Implementation of an IP-DSLAM, 10th IEEE Symposium on Computers and Communications (ISCC'05), Cartagena, Spain, 2005.

[11] G. Hagen et al., Developing an IP-DSLAM Benchmark for Network Processors, Advanced Networking and Communications Hardware Workshop (ANCHOR), Munich, Germany, 2004.

[12] H. Kalte, M. Porrmann, U. Rückert, A Prototyping Platform for Dynamically Reconfigurable System on Chip Designs, in: Proc. of the IEEE Workshop Heterogeneous reconfigurable Systems on Chip (SoC). Hamburg, Germany, 2002.