Mapping a Pipelined Data Path onto a Network-on-Chip

Stephan Kubisch, Claas Cornelius, Ronald Hecht, Dirk Timmermann
University of Rostock
Institute of Applied Microelectronics and Computer Engineering
18051 Rostock, Germany
Tel.: +49 381 498 7276
stephan.kubisch@uni-rostock.de, http://www.networks-on-chip.com

***Abstract*—During the last years, Networks-on-Chip (NoCs) have become a true alternative for the design of complex integrated Systems-on-Chip (SoC). Much effort has been spent for research on functionalities, mechanisms, and Quality-of-Service (QoS) features in NoCs. Hence, a broad and multi-faceted design space exists but leaves open, which mechanisms and design paradigms actually tip the scales for the chosen application domain. In this paper, we discuss the level of QoS needed in a specific NoC for a packet processing application. This is done in the light of preliminary investigations for the redesign of an existing packet processing system because that system's current architecture exhibits drawbacks regarding performance and further scalability. Therefore, we considered to take advantage of an NoC communication architecture. A simple NoC was developed, which knowingly omits sophisticated QoS mechanisms. Relying on the lessons, which have learned from the history and development of the Internet, we argue that a simple and plain NoC suffices for applications as the one discussed.**

Keywords—Network-on-Chip, System-on-Chip, Quality-of-Service, Application-specific, FPGA

I. Introduction

The need for a new approach to design large and complex integrated systems is mostly driven by the so called design-productivity-gap. Current design flows can not effectively exploit the potential of the tremendous number of transistors, processing elements, and functional blocks that is available on a single die today and even more in a few years. Therefore, point-to-point connected and bus-based designs will be constrained by the continuously increasing integration density and its related problems ranging from the architectural level down to the physical level. To cope with these issues, Networks-on-Chip (NoCs) have been proposed [1], [2], [3].

In the meantime, numerous publications have appeared presenting studies on NoCs, their features, mechanisms, and topologies. However, most research focuses on functionality and high performance, which results in fairly large, complex, and feature-rich systems providing various levels of QoS to handle the available physical bandwidth. Æthereal [4] and Nostrum [5] are probably the two most well-known examples for suchlike NoCs. Unfortunately, QoS comes at a price. A design's complexity increases as well as power consumption and silicon area, which are crucial parameters for both ASIC and FPGA designs. As most SoCs serve only a certain application domain (e.g., realtime, automotive…) instead of being general-purpose systems, the underlying NoC infrastructure needs to be tailored for that domain only. Hence, the required QoS level has to be considered in-depth to avoid unnecessary design overhead. For these application-specific NoCs, the workloads, traffic patterns, and constraints are known a-priori [6]. Stensgaard et al. [7], for example, present a case study for a DSP audio chip to prove the effectiveness of an NoC in comparison to the initial design with a crossbar. The application aims at a unidirectional data transfer rate in the range of kilobits per second ($\mathrm{kbit/s}$) whereas this paper describes the implementation of a packet processing system for gigabit Ethernet (GbE). This includes several bidirectional channels requiring one gigabit per second ($\mathrm{Gbit/s}$) for each channel. The current system is implemented as a conventional pipelined architecture and shows restrictions in terms of further scalability and performance. In this paper, the motivation for the use of a simple NoC-based architecture for this packet processing system is discussed.

The remainder of this paper is organized as follows: Section II addresses commonalities between the Internet and NoCs that encouraged our work. Section III specifies our motivation and related questions. We introduce the existing packet processing system MATMUNI in Section IV. Section V addresses our own NoC implementation. In Section VI, we discuss our observations in relation to the projected transformation of MATMUNI into an NoC-based design. Section VII concludes the paper.

II. Internet versus Network-on-Chip

As outlined in the previous section, research on NoCs is mainly driven by the limitations and problems of conventional SoC buses for future devices. In contrast to this low level approach, one may look at the Internet as an example for complex networks. The question arising from our point of view is, whether the lessons learned from the development of the Internet may be scaled down to NoCs. In other words: Is an NoC alike on-chip Internet?

Obviously, the Internet has many more clients and a complex topology organized in a multi-layered addressing scheme. Moreover, NoCs have just wires and routers whereas the Internet is built up of an overwhelming

number of different communication elements. But there is also another fundamental difference between on-chip and Internet communication. Whereas the Internet and its infrastructure is mostly not congested following the "big pipe" approach—the DE-CIX node is only utilized by half [8] for example—on-chip buses are often used to their full capacity [9]. As we will see later, this is an important discrepancy to keep in mind.

On the other hand, NoCs and the Internet share many commonalities. Both are networks designed in an ISO-OSI manner defining the same abstraction layers. Only the implementations differ. Computation and communication are separated, which is the most referenced argument for NoCs. Designs can be chosen to be communication or computation centric or both. An application's designer does not have to deal with wires. Instead, he relies on abstract communication channels also found in distributed computer programming [10]. Thus, even if Internet and NoCs are designed for different scales, long time experiences obtained in the Internet may fit for NoCs as well.

Coming back to the focus of this paper—Big Pipe or QoS for NoCs—interesting observations can be made. The Internet Protocol (IP) provides QoS in the form of a type-of-service field. But it is mostly ignored! In fact, everything is just best-effort. Another great example is ATM (asynchronous transfer mode). It is very efficient and has extensive QoS features on all levels. But this also makes it extremely complex and expensive. Evidently, it is vanishing from the market at the moment. Everyone does Ethernet without any QoS. A third example is the FastPath option for DSL connections. Thereby, QoS mechanisms such as error detection and correction are simply deactivated to substantially improve response time and thus end-to-end latency. It seems that everything aims at the old principle "Keep it simple and smart!" (KISS). The question is: Why does it work so well? And the simple answer is: We have enough bandwidth, and we *will* get more and more [11]. Applying this principle to NoCs means that we *must not* utilize it at its limits as it is done for SoC buses up to now. We have to leave enough headroom for critical situations. The expectation is that when NoCs are fully matured in future SoCs, they will serve us with enough bandwidth because of their *simplicity* and their *perfect implementation* tailored for a certain technology or platform.

As nothing is either black or white, we do have and need some QoS in the Internet and do so in NoCs as well, e.g., for real-time and multi-media applications. Because best-effort is no good business model, customers mostly subscribe for some bandwidth. Guaranteeing this bandwidth is one reason for QoS. Traffic shaping is another to avoid bursts often followed by congestion. But even on these issues, the Internet teaches us how to deal with it. The current trend in Ethernet-based access systems is to move complexity out of the core network to the customers premise equipment. This decentralizes the complexity and makes QoS much easier. Applying this idea to NoCs means that every IP core's network interface or even the IP core itself limits the bandwidth and does some traffic shaping. This contract between the IP cores and the network allows the literal NoC infrastructure to be designed much simpler and thus serving us again with more bandwidth.

III. Motivation and Questions

The motivation is to exploit NoCs for the redesign of the packet processing system. Since an FPGA will serve as target device, a small hardware footprint of the NoC is necessary. We want to keep it simple and small.

Our considerations are driven by experiences derived from our research on NoCs as well as Internet communication and their similarities as explained in Section II. Associated questions that need to be considered are specified below and will be discussed in Section VI:

Huge design space Which mechanisms and features for NoCs—especially functions providing QoS—have to be implemented to meet the demands of the packet processing system and, at the same time, meet the footprint constraints of the chosen FPGA? Which features can be omitted?

Feasibility Is an NoC an adequate target architecture for the redesign of the packet processing system?

Exploitation Will the packet processing system exploit the potentials of an NoC or is this—especially on an FPGA—an oversized approach?

The true nature of QoS What is the true nature of QoS features? What are they really? Is there a risk for an over-specification of NoCs?

Different platforms Are there differences between NoCs in FPGAs and in ASICs?

IV. MATMUNI – The Packet Processing System

Packet processing in the domain of (Inter)networking is a demanding task. The requirements on performance and flexibility of packet processing equipment as well as on security and availability rise permanently. Currently, important driving forces in the Internet are new technologies [11], the growing number of Internet users, and oversubscription of transmission lines. Hence, only hardware solutions provide sufficient performance for packet classification, manipulation, and forwarding. Due to their flexibility, FPGAs are widely used as target platform.

Access Networks (ANs) aggregate and connect thousands of subscribes and customers to the core networks of the Internet services providers. Basically, a large number of individual data streams are multiplexed on high bandwidth media suchlike fiber optics or 10-GbE. Therefore, different processing steps and mechanisms are necessary to differentiate, route, and switch individual data streams, to authorize, authenticate, and account subscribers, to provide QoS in terms of security, availability, and guaranteed bandwidth, and to be transparent for communication end-points.

MATMUNI is such a packet processing solution. It is implemented on a Xilinx platform FPGA [12], [13], [14]. MATMUNI's functional submodules offer mechanisms for Medium access controller Address Translation (MAT), Traffic Management (TM), and a Multi Protocol Label Switching User-to-Network Interface (MPLS-UNI).

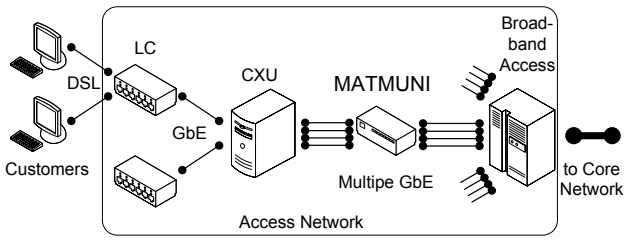

Fig. 1. Typical structure of an Access Network and location of MATMUNI

A. General Function

As sketched in Figure 1, MATMUNI is located in the AN behind the line cards (LC) and the central switching unit (CXU). The CXU aggregates several thousand digital subscriber lines (DSLs). On independent GbE channels, traffic of multiple $\mathrm{Gbit/s}$ must be handled. MATMUNI covers packet processing tasks that prepare and preprocessing traffic for the core network.

MAT MAT targets scalability and security issues by flexible n:1 translation of data link layer addresses—Ethernet addresses of the medium access controller (MAC) in our scenario. Untrustworthy customer MAC addresses are translated into distinct, trustworthy MAC addresses of the provider.

TM The TM functionality meters traffic on a per customer base. Each frame is marked either green, or yellow, or red according to the customer's current bandwidth utilization. With policing mechanisms, the subscribed bandwidth is ensured as long as possible for each customer.

MPLS-UNI The MPLS-UNI encapsulates complete frames and prepends MPLS labels. Forwarding decisions of core routers solely depend on these labels' information. Usually, a full-blown label edge router (LER) is required. But in the case of MATMUNI, only a subset of an LER's functionality is necessary. Therefore, an adapted, compact MPLS module was realized in hardware.

For detailed information on MATMUNI's functionality, the interested reader is referred to [12], [13], [14].

B. System Architecture

As sketched in Figure 2, MATMUNI is currently implemented as pipelined data path architecture. Two main data paths exist: the uplink from the customers to the core network and the downlink vice versa. At first, a frame entering the system through the FPGAs internal MACs passes through a synchronization buffer for reasons of clock domain crossing. The frame is further forwarded to a key parser (KP) and stored in a frame buffer (FB). The KP extracts a key from the frame headers and generates a lookup request in the memory. The memory arbiter administrates the memory containing rules for each key. When the memory lookup was successful, the frame byte-serially passes through the functional modules. The functional modules modify the frame according to the rules, which are assigned to the appropriate functional module. The last module forwards the frame to the egress MAC via another synchronization buffer. The basic data flow is the same in up- and downlink. Up to 4 parallel main data paths are supported in both directions. Therefore, the functional submodules are instantiated twice or fourfold. Figure 2 depicts the internal structure of the MATMUNI system. Additionally, the MATMUNI system contains an interface to a CPU for management and configuration. A special protocol also allows for in-band configuration of the MATMUNI system. Therefore, a protocol multiplexer (PM) is inserted in the uplink path. Memory contents—keys and their according rules—and internal system parameters can be modified at runtime.

C. Performance and Bottlenecks

MATMUNI's current architecture was designed to handle GbE. A frequency of at least $125\,\mathrm{MHz}$ is required for non-blocking performance. Although single submodules reach frequencies of more than $125\,\mathrm{MHz}$, the whole system barely fulfills this requirement due to the complex wiring for module interconnection. Particularly, the complexity of the memory arbiter as the central entity for memory management increases when instantiating extra data paths. As often, a shared memory is the bottleneck. While MATMUNI supports up to 8 independent data paths (4x uplink, 4x downlink), the system does not scale any further using this architecture. Beyond it, non-blocking performance cannot be ensured. Especially in the worst case (although is it unlikely), in which large bursts of minimal Ethernet frames arrive, a large number of frames will be discarded. Increasing the buffer size would not solve the underlying problem. MATMUNI's architecture is tailored for a specific scenario but lacks further scalability and flexibility. Future performance requirements are not likely to be fulfilled with the current conventional approach. Bus-based architectures do also not meet these demands. A central bus is just another shared medium similar to the existing memory arbiter submodule. For these reasons, NoC-based architectures appear most feasible for a redesign.

D. Dependency Analysis

MATMUNI's internal communication and control dependencies are analyzed using the formal representation of Communication Task Graphs (CTG) and Application Characterization Graphs (APCG) as defined in [9]. A CTG is defined as $G' = G'(T, D)$ with each $t_i \in T$ being a computational module or task and each $d_{i,j} \in D$ being a data or control dependency between task t_i and t_j annotated with its communication volume. An APCG is defined as $G = G(C, A)$ with each $c_i \in C$ being an IP core within the NoC and $a_{i,j} \in A$ being the communication process from IP core c_i to c_j annotated with application- and NoC-specific constraints.

Figure 3 shows the CTG of the MATMUNI system. Only a single data path is shown since the CTG is the same in both up- and downlink. Figure 4 shows the APCG of the MATMUNI system for a single data path. Closely coupled tasks in the CTG are combined to one IP core to reduce the number cores and thus the size of the NoC. The

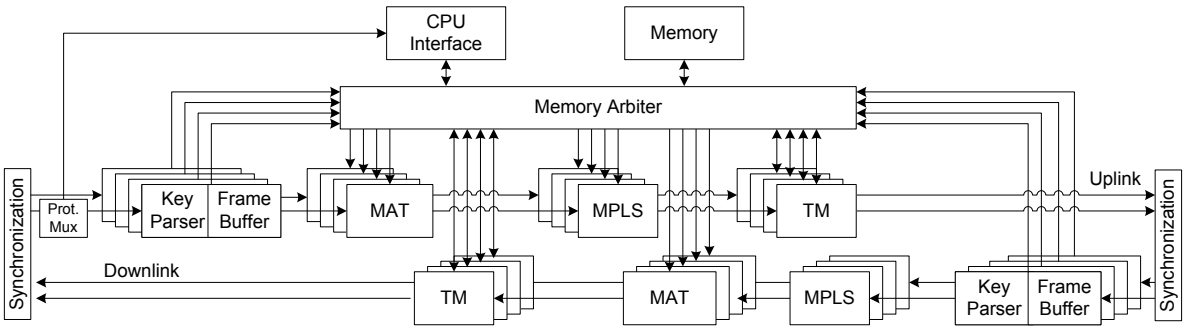

Fig. 2. Internal structure of the MATMUNI system with serially ordered functional modules

PM is directly connected with the external I/O interface. The KP joins the FB. Furthermore, all memory-related tasks are combined in the memory module. An $a_{i,j}$ marked with a thick line has a bandwidth constraint of at least $1\ \mathrm{Gbit/s}$, which represents the main data path. An $a_{i,j}$ drawn with a dotted line is a control path with a bandwidth constraint of up to $350\ \mathrm{Mbit/s}$ in the worst case (bursts of minimal frames). A thin line indicates an $a_{i,j}$ with a bandwidth constraint of less than $200\ \mathrm{Mbit/s}$.

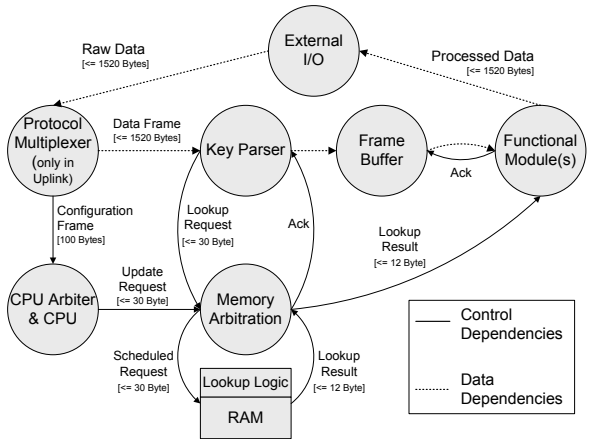

Fig. 3. CTG of the MATMUNI system

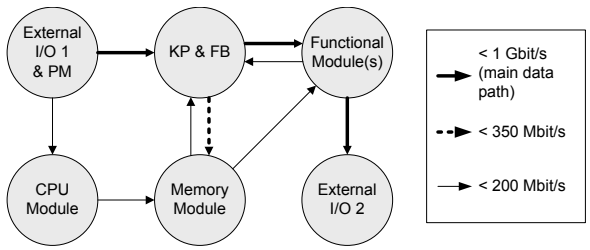

Fig. 4. APCG of the MATMUNI system

V. The Network-on-Chip

An NoC consists of routers and links. Unlike bus architectures, NoCs allow for concurrent transmission of data between modular IP blocks. By means of separated computation and communication, designers benefit from a divide-and-conquer approach and high reusability.

Figure 5 shows the basic structure of a single NoC router (R) and the connected IP core—a so-called resource. Each router is connected to a resource and up to 4 neighboring routers. Thereby, a 2D mesh topology is realized as shown in Figure 6. A resource can be of different nature: off-chip I/O, memory, or processing units. The resource-network-interface (RNI) provides bridging functionality from the NoC interface to a resource's specific interface. The RNI itself is divided into a resource independent network interface (RINI) and a resource dependent network interface (RDNI). The RINI connects to the NoC router and is identical for each resource. The RDNI connects to the resource and is specifically tailored for that resource. It belongs to the reconfigurable, functional segments of the NoC-based system. A router consists of five ports; one for each routing direction (N, S, W, E) and one port for local access to the RNI (L). Each port contains a buffer and the basic routing logic. The arbitration and switching functionality connects input and output ports.

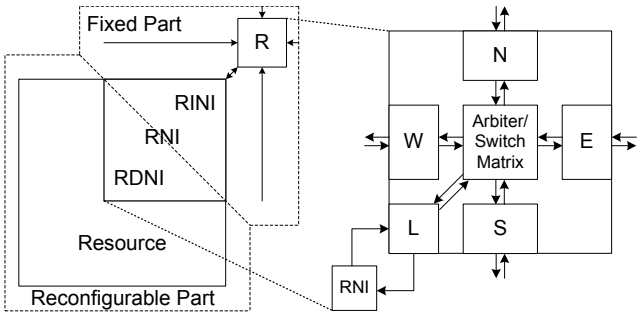

Fig. 5. Structure and components of the NoC grid and an NoC router

A. Implemented Features

An NoC feature is a mechanism that either provides basic functionality or enhances QoS or other properties of the NoC. Regarding the MATMUNI system, *performance* and *modularity* are of special interest. Sufficient performance in terms of throughput is required for non-blocking operation to avoid packet drops. Since networking hardware always generates delay in the network, performance in terms of low end-to-end latency within the NoC is secondary. Modularity, which supports IP reuse and future modifications, is inherently given in NoCs.

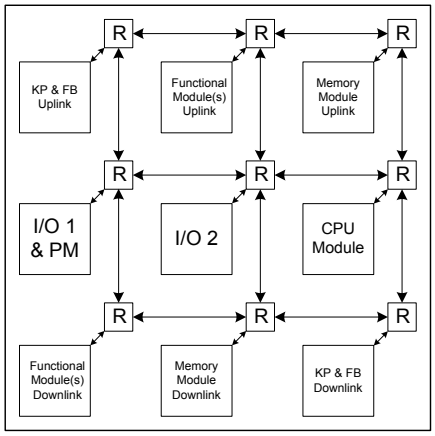

Fig. 6. A 3x3 NoC & mapping of MATMUNI's submodules

From our point of view, typical NoC features and mechanisms can be classified into three categories: minimum requirements, additional features to optimize an NoC's behavior, and features or mechanisms that are for free, e.g., given by a-priori knowledge during the design phase like in the discussed packet processing scenario. Thereby, NoC features are assessed by their necessity and costs with regard to simplicity. This scheme *does not* have to apply to other applications and is far from being exhaustive. But it helped us to choose relevant aspects. As most of the NoC features are well-known, we will not go into detail.

For our own implementation, we selected features that are minimum requirements for the operation of a basic NoC. These features conform to the KISS principle. We omitted sophisticated optimizations, which are not a stringent requirement for NoC operation, e.g., Virtual Channels (VC). To enhance the performance of the NoC-based system and to provide some QoS, we prefer using features that enhance QoS at *no* or just *moderate* costs. Suchlike approaches are QoS by design [15], [16], [17], advantageous module mapping, and the use of *existing* FPGA resources like multiple clock networks. Every application shows typical traffic patterns. Using CTGs and APCGs, the main data paths and bottlenecks of a system can be identified. This a-priori knowledge allows for optimizations and specific design decisions that do not need additional logic resources.

We implemented three different router versions and chose the following features to be included in the routers:

Routing & Arbitration A simple XY-routing algorithm was chosen. It is inherently deadlock-free and thus optimal for our NoC. The routing and arbitration logic need at least 4 clock cycles within each router. This is the minimum delay per node. The arbiter uses a simple Round-Robin (RR) algorithm when multiple requests occur.

Switching Two router versions use wormhole switching (WHS)—one with registered (WHS) and one with combinational acknowledge signals (WHS*). The third router version features virtual-cut-through switching (VCTS). The WHS versions have a buffer for one flit per port whereas the VCTS version contains a FIFO buffer for one maximum transmission unit (MTU) per port. Output port buffering overcomes difficulties of head-of-line blocking.

Flow control The same data link layer interface is used for all router versions as shown in Figure 7. The difference is in the flow control schemes, which are also illustrated in 7: First, the header flit is assigned (DATA) together with request (REQ) and start-of-frame (SoF). When routing and arbitration have finished (after at least 4 cycles), the acknowledgment flag (ACK) is set from the rx-port to show its readiness to receive data. With WHS, every flit needs to be acknowledged to stop the "worm" at its current position in case the header is blocked. In the WHS version, every flit needs at least two clock cycles for a single hop. In the WHS* version, one flit can be transmitted per clock cycle. With VCTS, the remaining flits are streamed into the output buffer of the target port after the header flit has been acknowledged. Only the header flit needs to be acknowledged due to sufficient buffer space. End-of-frame (EoF) indicates the last flit and frees the occupied port. Byte-enable (BE) defines the valid bytes.

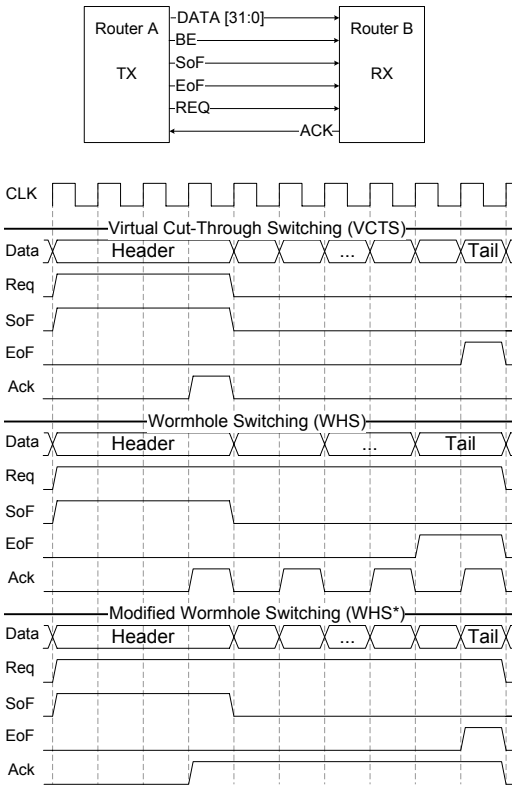

Fig. 7. NoC data link layer interface and flow control

B. Synthesis Results

The three router versions have been synthesized for a Virtex-4 FX20 (V4) and a Virtex-2P VP30 (V2). Table I contains the synthesis results. V2 synthesis results are used for comparison with a router presented in [18]. The (theoretically) maximum link bandwidth (LBW) for a single unidirectional channel between adjacent routers is given in $\mathrm{Gbit/s}$ according to the flow control schemes illustrated in Figure 7.

TABLE I
SYNPLIFY PRO SYNTHESIS RESULTS FOR A VIRTEX-4 FX20 (V4) AND A VIRTEX2P XC2VP30 (V2)

NoC version	f [MHz]		FFs		BR		LUTs		LBW [Gbit/s]	
	V4	V2	V4	V2	V4	V2	V4	V2	V4	V2
WHS router	240	178	295	298	—	—	809	825	3.84	2.84
WHS* router	193	143	315	299	—	—	925	862	6.17	4.57
VCTS router	215	182	225	248	5	5	1117	1139	6.88	5.82

A WHS router requires 809 lookup tables (LUTs) and 295 register flip flops (FFs). It achieves a frequency of up to 240 MHz and an LBW of up to 3.84 Gbit/s. The modified WHS* router requires slightly more resources: 925 LUTs and 315 register FFs. The maximum frequency is 193 MHz, which is less than for WHS. But the LBW increases to 6.17 Gbit/s because only one clock cycle is needed to transmit a flit. For WHS and WHS*, the buffers are realized with FFs. The VCTS router requires ca. 1117 LUTs and nearly the same number of FFs due to a more complex buffering logic. 5 block rams (BR) are used to buffer incoming flits. A maximal frequency of 215 MHz can be achieved. Though this is less than the WHS router's frequency, the LBW nearly doubles to 6.88 Gbit/s because of VCTS' flow control scheme.

A comparable small-size NoC router [18] has similar synthesis results as our router for a V2 device (see Table I). But it reaches only a maximal frequency of ca. 33 MHz, which is quite low for this FPGA. In [19], similar NoC routers have been synthesized for a Virtex-2 1000 FPGA, which also feature WHS and XY-routing. Additionally, VCs are implemented. Three versions use either RR-arbitration, or priority traffic differentiation, or circuit switching mechanisms. Footprints range from 1622–1984 LUTs and 467–513 FFs. These routers require much more resources compared to our routers (synthesis values for this FPGA are not given in Table I since they are similar to the V4's values except for the frequency).

C. Simulation

Basal traffic patterns have been simulated for the different NoCs for functional verification. As these types of NoCs are already extensively tested with regard to performance and throughput as for example in [18], [20], or [21], a very similar behavior can be expected for the presented NoC versions. With increasing flit injection rate, the NoCs become saturated and cannot accept more flits and packets. At the same time, the average packet latency increases. This depends on parameters like buffer space and packet size, but the principle behavior is the same.

Furthermore, we used E-Core [22] to emulate MATMUNI's traffic pattern in the NoC. E-Core is used for application-specific performance evaluation. It is flexibly configurable and can emulate the typical behavior of IP cores in an NoC. E-Core is especially feasible for applications with streaming types of traffic alike MATMUNI. Basically, an instance of E-Core can be configured as data source, data sink, or transceiver together with a variety of parameters such as injection rate or packet sizes. This way, a quick performance estimation of the NoC-based MATMUNI system is possible without actually implementing its functionality. Figure 8 shows the basic structure of an E-Core instance. It consists of a special RNI, which connects to the NoC router, and of a processing core, which emulates the desired behavior.

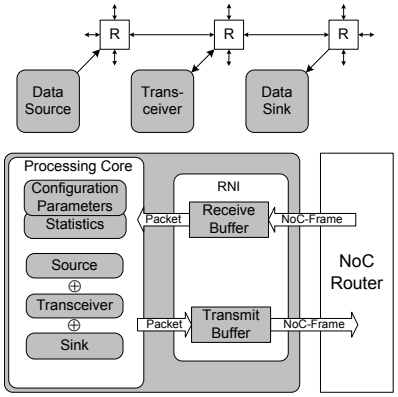

Fig. 8. Coarse structure of E-Core

VI. DISCUSSION

A. Implications concerning the Redesign of MATMUNI

The feasibility of an NoC architecture for MATMUNI depends on how well MATMUNI's requirements are fulfilled by the NoC. Congestions and locked resources limit the BW that can actually be utilized depending on the load and the traffic patterns. Since sophisticated features for QoS have been omitted in the routers, we are still in need of methods that can provide QoS to meet the requirements of MATMUNI. Knowing MATMUNI's internal communication dependencies as described in Section IV-D, we optimized the mapping of the IP cores onto the NoC. Figure 6 shows the mapping of the application onto a 3x3 NoC derived from its APCG in Figure 4. Two different aspects have been considered: on one hand, the location of the hard-wired off-chip I/O primitives within the target FPGA (XC4VFX20) and, on the other hand, reducing concurrency on the NoC links. The maximum LBW can nearly be exploited because congestions and locked resources due to concurrency are largely prevented by the chosen arrangement of the IP cores. Multiple E-Core instances have been configured to emulate the traffic pattern of the application scenario. The simulation showed that the NoC can handle the traffic generated by MATMUNI. The main data paths require at least a BW of 1 Gbit/s in each direction. The link BW of a single unidirectional channel in our NoCs ranges from 3 to 6 Gbit/s as shown in Table I. Even

when not driven at its maximum speed, the NoC provides sufficient bandwidth to handle 1 $\mathrm{Gbit/s}$. Thus, an NoC is a feasible approach to tackle the drawbacks of MATMUNI's current architecture. The trade-off is the overhead for the NoC infrastructure itself. Although the Virtex4 FX20 is a moderately sized device, approximately the half of its logic resources would be used for the 3x3 NoC itself. But diverse submodules of MATMUNI's current architecture *need not* to be implemented in the NoC-based redesign. The buffering functionality of the synchronization FIFOs for off-chip I/O as well as so-called local buffer modules can be omitted, since the RNIs and NoC routers do buffering anyway. Particularly in the VCTS version, the routers contain FIFO buffers with independently clocked read and write ports. Thus, the NoC is not only used for inter-module communication but also becomes the buffer. Considering routers with low hardware footprint as presented in Section V, the resource overhead can be minimized. Applying low-cost optimizations through a-priori knowledge exploits the NoCs potential. To further enhance performance, a GALS architecture promises to be a viable approach. IP core can be driven with individual clocks to operate at their maximum speed or to reduce idle cycles. Since multiple clock networks already exist in current platform FPGAs, the use of more than one clock is nearly for free (The FPGA vendors already paid for it!).

B. General Implications

Various sophisticated features exist on different levels that can provide QoS, e.g., adaptive routing, VC, and traffic differentiation. These mechanisms show an interesting discrepancy. On the one hand, they are used to manage the available bandwidth as well as to guarantee bandwidth in times of high traffic load. But when a network is already congested, only reserving a fraction of the bandwidth can help. On the other hand, they are expensive in terms of valuable logic resources in FPGAs and silicon area in ASICs. Router complexity increases and router and NoC speeds decrease. Thus, they also reduce the maximum physical bandwidth—they bite the hand that feeds them. Some of the features even interact either by negating their individual benefits or by requiring each other to be fully exploited as discussed in [23]. Furthermore as stated in [19], these features do not show the expected performance gains, especially when the network traffic is *not* known in advance. But when the network traffic or the application *is* known in advance, inexpensive mechanisms can be used instead to provide QoS—assuming that the NoC is not used at its full capacity—because shaping network traffic has most impact on performance parameters. Thus, the benefits of sophisticated QoS features do not justify their costs for the application domain, which is presented here.

But the statement above does not yet answer the question what QoS-features really are (see Section III). The use of these mechanisms is driven by the fact that bandwidth never seems to be sufficient. Until now, SoC buses and NoCs are mostly used at their limit. It is a back and forth between providing physical bandwidth and applications that consume this bandwidth. Arbitration and scheduling features mitigate or bypass QoS problems that exist due to the permanent wish for more and more bandwidth. Hence, most of the features are just workarounds until enough bandwidth is available.

As we are looking for a solution for these problems, the Internet and its success can serve as a paradigm providing long-time experiences. Alike the transition from ATM to Ethernet as outlined in Section II, a transition from expensive, sophisticated mechanisms to more simple approaches *is* feasible in NoCs as well since both serve similar networking principles. By shifting QoS into the design process or into higher layers, e.g., into the RNI or the IP core itself, complexity is decentralized and the actual NoC remains compact and fast. Hence, we use NoCs in their original sense—to mitigate parasitic effects, to handle increasing SoC complexity, and to provide a high performance communication grid.

C. Different Target Platforms for NoCs

Today, most NoC are used in ASICs designs. ASIC designs are up to 3.2 times faster than the same circuit implemented on an FPGA as analyzed in [24]. But costs for ASIC designs are clearly higher in terms of time and money as for FPGA designs. FPGAs are more flexible due to their reconfigurability and feature a simple push-button design flow. Unfortunately, NoCs have the drawback of utilizing a distinct portion of the FPGA's valuable logic resources. As outlined in [25], hardwired general purpose NoCs on FPGAs seem to be a valid approach to benefit from the best of both worlds. Regarding the current development of platform FPGAs, one has only to continue this evolution. Special purpose FPGAs already exist, for instance in the Xilinx Virtex-4 family. These devices are either specialized for digital signal processing, embedded processing and connectivity, or provide extensively large amounts of logic. In [24], the integration of basal, reusable blocks in FPGAs as embedded hardwired macros was confirmed to be beneficial from different perspectives. Thus, FPGA vendors should follow their strategies to support customers with general, ready-made macros to narrow the design-productivity-gap. With suchlike macros—customizable Networks-on-Chip resources in this case—engineers can focus on the important aspects of their designs. Actually, separation of computation and communication is said to be one of the prime advantages of NoCs. However, while we are just now and here debating the "Look and Feel" of the wires and routers instead of concentrating on the application itself and as it is also done in many other studies, there is no true separation. In fact, this separation exists—but only from a theoretical point of view. As long as we search the multifarious NoC design space to finally hit the sweet spot, the benefits of separating computation and communication have not yet found their way into the design process of NoC-based applications. Especially for industrial applications and commercial embedded systems, this is an important aspect. Only with matured NoC technology, NoCs can efficiently and economically justifiable be exploited for real-world applications and in a wider scale be applied in commercial systems.

VII. Conclusion

This article is an application study addressing the redesign of MATMUNI—a previously developed packet processing system. Due to scalability and performance drawbacks of that system's current architecture, a NoC-based approach was envisaged. The packet processing system's internal dependencies have been analyzed to map its functionality onto the NoC. Therefore, a simple and compact NoC was developed with regard to costs and simplicity. Preliminary FPGA synthesis results and simulation results have been discussed with respect to MATMUNI's projected redesign. Future work comprises specification and implementation of a special communication protocol that reproduces the current, signal-based communication on the message level and synthesis of the NoC-based MATMUNI system on the target FPGA.

Furthermore, various questions on NoCs in general have been discussed, e.g., issues related to the necessity of QoS in application-specific NoCs. The main statement of this paper is that QoS can be achieved by design—in particular for similar applications with streaming traffic. Therefore, a-priori knowledge on an application's typical communication characteristics is beneficial. As outlined in the beginning, there will always be the need for some QoS. However, expensive QoS features can mostly be omitted or shifted into higher protocol layers to keep the NoC grid simple and smart in itself. Rather, we eschew the use of sophisticated QoS mechanisms by virtue of their downsides such as increased area, complexity, and logic requirements as well as decreased speed and bandwidth. NoCs should be used in their original sense—as wire-replacement. Thus, NoCs should not be over-specified. Generally speaking, we propose...

- ...to follow the path of increasing the internal communication bandwidth of NoCs, which results in lasting performance gains...
- ...and to apply simple QoS features on top, which are easy to implement and evaluate.

Since there are also applications that strictly require QoS on different levels, e.g., real-time applications, automotive embedded systems, or multi-processor systems, these statements cannot be generalized. But for the targeted application domain just "KISS" or to put it in a more cultured way:

> Simplicity is the ultimate sophistication.
>
> *(Leonardo da Vinci)*

Acknowledgment

We thank Nokia Siemens Networks, location Greifswald, Germany, for supporting the MATMUNI project.

References

[1] W. J. Dally and B. Towles, "Route Packets, not Wires: On-chip Interconnection Networks," in *Proc. of the 38th DAC*, Las Vegas, Nevada, USA, June 2001.

[2] L. Benini and G. D. Micheli, "Networks on Chips: A New SoC Paradigm," *Computer*, vol. 35, no. 1, pp. 70–78, 2002.

[3] T. Bjerregaard and S. Mahadevan, "A Survey of Research and Practices of Network-on-chip," *ACM Computing Surveys*, vol. 38, no. 1, 2006.

[4] E. Rijpkema et al., "Trade Offs in the Design of a Router with Both Guaranteed and Best-Effort Services for Networks on Chip," in *Proc. of the DATE 2003*, Munich, Germany, March 2003.

[5] M. Millberg, E. Nilsson, R. Thid, S. Kumar, and A. Jantsch, "The Nostrum Backbone - a Communication Protocol Stack for Networks on Chip," in *Proc. of the 17th Intl. Conf. on VLSI Design*, Mumbai, India, January 2004.

[6] J. Xu, W. Wolf, J. Henkel, and S. Chakradhar, "A Design Methodology for Application-specific Networks-on-Chip," *Transactions on Embedded Computing Systems*, vol. 5, no. 2, pp. 263–280, 2006.

[7] M. B. Stensgaard, T. Bjerregaard, J. Sparso, and J. H. Pedersen, "A Simple Clockless Network-on-Chip for a Commercial Audio DSP Chip," in *Proc. of the 9th EUROMICRO Conf. on Digital System Design*, Dubrovnik, Croatia, August/September 2006.

[8] Deutscher Commercial Internet Exchange, http://www.decix.de.

[9] U. Ogras, J. Hu, and R. Marculescu, "Key Research Problems in NoC Design: A Holistic Perspective," in *Proc. of CODES+ISSS*, Jersey City, NJ, USA, September 2005.

[10] R. Hecht, S. Kubisch, H. Michelsen, E. Zeeb, and D. Timmermann, "A Distributed Object System Approach for Dynamic Reconfiguration," in *Proc. of the 13th RAW*, Rhodes Island, Greece, April 2006.

[11] J. Cioffi et al, "Vectored DSLs with DSM: The Road to Ubiquitous Gigabit DSLs," in *Proc. of the World Telecommunications Congress 2006 on CD-Rom*, Budapest, Hungary, April/Mai 2006.

[12] S. Kubisch, H. Widiger, D. Duchow, D. Timmermann, and T. Bahls, "Wirespeed MAC Address Translation and Traffic Management in Access Networks," in *Proc. of the World Telecommunications Congress 2006 on CD-Rom*, Budapest, Hungary, April/Mai 2006.

[13] H. Widiger, S. Kubisch, D. Duchow, D. Timmermann, and T. Bahls, "A Simplified, Cost-Effective MPLS Labeling Architecture for Access Networks," in *Proc. of the World Telecommunications Congress 2006 on CD-Rom*, Budapest, Hungary, April/Mai 2006.

[14] H. Widiger, S. Kubisch, D. Timmermann, and T. Bahls, "An Integrated Hardware Solution for MAT, MPLS-UNI, and TM in Access Networks," in *Proc. of the 31st Annual IEEE Conf. on Local Computer Networks*, Tampa, FL, USA, November 2006.

[15] U. Y. Ogras and R. Marculescu, "'It's a Small World After All': NoC Performance Optimization Via Long-Range Link Insertion," *IEEE Transactions on VLSI Systems*, vol. 14, no. 2, July 2006.

[16] J. Hu and R. Marculescu, "Application-specific Buffer Space Allocation for Networks-on-Chip Router Design," in *Proc. of the IEEE/ACM Intl. Conf. on Computer-aided Design*, San Jose, CA, USA, November 2004.

[17] Z. Guz et al., "Efficient Link Capacity and QoS Design for Network-on-Chip," in *Proc. of the DATE 2006*, Munich, Germany, March 2006.

[18] B. Sethuraman et al., "LiPaR: A Light-Weight Parallel Router for FPGA-based Networks-on-Chip," in *Proc. of the 15th ACM GLSVLSI*, Chicago, ILL, USA, April 2005.

[19] A. Mello, L. Tedesco, N. Calazans, and F. Moraes, "Evaluation of Current QoS Mechanisms in Networks on Chip," in *Proc. of the Intl. Symp. on System-on-Chip*, Tampere, Finland, November 2006.

[20] L. Tedesco, A. Mello, D. Garibotti, N. Calazans, and F. Moraes, "Traffic Generation and Performance Evaluation for Mesh-based NoCs," in *Proc. of the 18th Annual Symp. on Integrated Circuits and System Design*, Florianopolis, Brazil, 2005, pp. 184–189.

[21] D. Wu, B. M. Al-Hashimi, and M. T. Schmitz, "Improving routing efficiency for network-on-chip through contention-aware input selection," in *Proc. of the Conf. on Asia South Pacific Design Automation*, Yokohama, Japan, 2006, pp. 36–41.

[22] S. Kubisch, H. Widiger, C. Cornelius, D. Timmermann, and A. Strzeletz, "E-Core – A Configurable IP Core for Application-specific NoC Performance Evaluation," in *DATE 2007, Proc. of the Workshop on Diagnostic Services in Network-on-Chips*, Nice, France, April 2007.

[23] A. S. Vaidya, A. Sivasubramaniam, and C. R. Das, "Impact of Virtual Channels and Adaptive Routing on Application Performance," *IEEE Transactions on Parallel and Distributed Systems*, vol. 12, no. 2, pp. 223–237, February 2001.

[24] I. Kuon and J. Rose, "Measuring the Gap between FPGAs and ASICs," in *Proc. of the 14th ACM/SIGDA Intl. Symp. on FPGAs*, February 2006.

[25] R. Hecht, S. Kubisch, A. Herrholtz, and D. Timmermann, "Dynamic Reconfiguration with hardwired Networks-on-Chip on future FPGAs," in *Proc. of the 15th FPL*, Tampere, Finland, August 2005.