Copyright Notice

Institute of Communication Networks and Computer Engineering
Universität Stuttgart
Pfaffenwaldring 47, D-70569 Stuttgart, Germany
Phone: +49-711-685-68026, Fax: ++49-711-685-67983
email: mail@ikr.uni-stuttgart.de, http://www.ikr.uni-stuttgart.de

A Novel Architecture for a High-Performance Network Processing Unit: Flexibility at Multiple Levels of Abstraction

Simon Hauger
Institute of Communication Networks and Computer Engineering (IKR),
Universität Stuttgart, Pfaffenwaldring 47, 70569 Stuttgart, Germany
Email: simon.hauger@ikr.uni-stuttgart.de

***Abstract*—Network processing devices in future, high-speed network nodes have to be capable of processing several hundred million packets per second. Additionally, they have to be easily adaptable to new processing tasks due to the introduction of new services or protocols. Field programmable gate arrays (FPGAs) and network processors are suitable devices fulfilling these requirements: The former offer configurability at register-transfer level providing fine grain adaptability to unforeseen processing requirements and a high processing power. The latter are programmed at the more abstract software level and support high-speed execution of their fixed set of instructions. In this paper, we present a novel architecture for an FPGA-based high-speed network processing unit offering programmable modules at multiple levels of abstraction: register-transfer level, micrcode level, software level and parameter level. A prototypical implementation demonstrates its feasibility with today's field programmable gate array devices offering a throughput of more than one hundred million minimum sized packets per second.**

I. Introduction

The outstanding success of the Internet leads to ever increasing data rates in data and telecommunication networks as well as to a continuously growing number of services and applications. Therefore, in future 100 Gbit Ethernet networks, devices like routers and switches have to process more than one hundred million packets per second—on each single line card. Yet, these networking devices have to be easily adaptable to changed requirements due to new services or protocol extensions.

Application Specific Integrated Circuits (ASICs) provide the required processing speed, but they cannot be adapted easily to new requirements. General purpose processors are very flexible and easy to adapt, however they cannot support the needed packet throughput. For this reason, Field Programmable Gate Arrays (FPGAs) and Network Processors (NPs) are increasingly used in networking devices. They feature both a high and easy adaptability and the required processing capacity.

FPGAs are programmable logic devices. They consist of configurable lookup tables, flip-flops and interconnection lines [1]. The desired functionality is described in a hardware-description language at gate level and register-transfer level. As their capabilities are continuously increasing they are more and more used for packet processing tasks in high-speed network nodes [2]–[4].

NPs are integrated circuits containing multiple processors that are specialized for packet processing tasks as well as co-processors, on-chip memory and fast interfaces [5], [6]. As the processors of an NP are programmed like a general purpose processor, they are easily adaptable to changed requirements. For this reason, they are increasingly used in routers.

FPGA-based systems and NPs offer configurability or programmability at different levels of abstraction. The lower level of abstraction makes designing functions for FPGA-based systems more complex, however it enables highly efficient implementations. The more abstract software level of NPs allows to easily program even complex processing tasks. However, due to constraints of their instruction set, certain special functions cannot be implemented efficiently on NPs.

In this paper, a novel architecture for a network processing unit is presented that combines the advantages of conventional NPs and that of FPGAs. The presented architecture provides programmability at multiple levels of abstraction: at register-transfer level like an FPGA, at software level like an NP, and at micro-code level, which is a level of abstraction between those two. Furthermore, due to the modular structure of our architecture functional blocks can be easily added, removed or exchanged, depending on the current requirements. The possibility to configure or parameterize such functional blocks introduces a fourth level of adaptability. A high-performance pipeline structure is used to achieve both modularity and a high packet processing rate. The pipeline deterministically processes one minimum sized packet in each clock cycle. Due to the above features, our novel architecture is called MIXMAP architecture, as it provides **M**odular**i**ty and fle**x**ibility on **M**ultiple levels of **A**bstraction mapped on a **P**ipeline structure.

To provide this modularity and flexibility a network processing unit implementing this architecture has to be based on FPGA technology. Using the proposed architecture, a throughput of one to two hundred million packets per second is feasible with current FPGA devices, which is sufficient for 100 Gbit Ethernet.

In order to show the realizability of the MIXMAP architecture we prototypically implemented and evaluated its major parts. Our prototype system is based on an Altera Stratix II FPGA that is integrated in a hardware development platform.

The rest of this paper is structured as follows: Section 2

reviews how flexible, high-speed packet processing is achieved on FPGAs and NPs, and gives an overview about their different levels of abstraction. Section 3 presents our novel MIXMAP architecture for network processing units, and section 4 gives some details on our prototypical implementation. Finally, we conclude the paper in section 5.

II. Flexible, High Speed Packet Processing

FPGAs and NPs are commonly used for high-speed packet processing tasks. Both provide some form of flexibility, and yet, are capable of supporting a high packet throughput.

A. Field Programmable Gate Arrays

In recent years the capabilities of FPGAs increased to a large extent. As a result they evolved from simple glue logic modules to full-featured packet processing devices [2]–[4]. Also their power consumption is decreasing due to the use of modern manufacturing processes [7].

An FPGA provides fine grain reconfigurability on hardware level, in particular on gate level and register-transfer level. It mainly comprises up to several hundred thousands of small look-up tables, flip-flops and a large interconnection array. Additionally, more and more specialized components are integrated onto FPGAs. Examples are on-chip static random access memory (SRAM) blocks, useful for e. g. packet buffers or forwarding tables, phase-locked loops for clock signal generation, and even hard-wired processors. By configuring all these elements arbitrary digital systems can be realized.

The functionality of FPGA-based modules is described in a hardware-description language (HDL) like VHDL or Verilog. Software programmers often face difficulties writing such hardware descriptions, as these have to represent the parallel hardware on the chip and not a sequential process. However, these hardware descriptions do not have to be on a low level of abstraction: While operations on bit and gate level are supported also more abstract data types, like integer, enumeration types or record types, can be used. Furthermore, control structures like branches and loops can be used to efficiently describe the hardware structure. Libraries with prebuilt, parameterizable functional units further reduce the complexity of the implementation process. Such libraries contain e.g. arithmetic and logic units, cryptography units, standard interface modules and first-in-first-out (FIFO) buffers.

The development process is concluded with tool-supported synthesis and place & route. These steps map the HDL code to available elements on the chip, place them, route the connection lines between them and finally generate a programming file that is loaded into the FPGA device.

B. Network Processors

Network Processors provide a high flexibility as they are programmable like general purpose processors. They are often used in access routers and base stations, but also high-speed NPs are increasingly deployed on line-cards in high-performance routers [8].

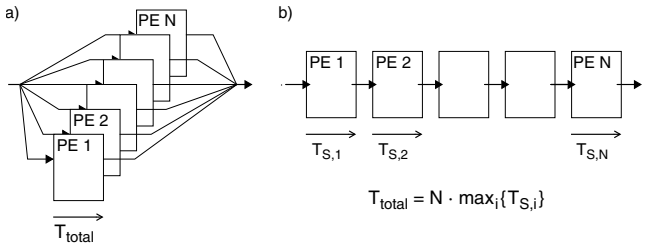

Fig. 1. a) cluster configuration, b) pipeline configuration

An NP usually comprises multiple, rather small, specialized processors. These processors form a pipeline [9], a symmetrical cluster [10], or a combination of both forms [11], [12]. The principal differences between a pipeline and a cluster configuration are discussed in the next section.

To increase the processing throughput the internal processors of an NP are often multi-threaded for hiding memory latencies. Instruction level parallelism is achieved by superscalar or Very-Long-Instruction-Word (VLIW) architectures. Furthermore, NPs usually feature coprocessors and hardware accelerators for special processing tasks as well as several on-chip and off-chip memories for storing lookup tables or buffering packet data.

For programming NPs one has to use specialized languages provided by the manufacturer, low level C, or assembly. Any way, programming NPs is not straightforward and one has to possess a detailed knowledge about the underlying parallel hardware in order to write efficient code.

C. Configuration of Processing Elements

To increase their throughput, all current network processing systems, both NP-based and FPGA-based, make use of parallel processing entities (PEs). In the case of NPs these are its internal optimized processors[1], in the case of FPGAs these are internal processing modules. There are two basic configurations of the PEs, as depicted in Fig. 1: a cluster and a pipeline.

In a cluster configuration all PEs are identical. Each PE performs the entire processing task required for a packet. In a pipeline configuration the processing task is divided into subtasks of equal completion time and each PE is assigned to one such sub-task. All packets thus wander synchronously through all PEs.

Under the ideal assumption that the total processing task of duration T_{total} can be divided equally into the N subtasks of duration $T_{S,i} = T_S = T_{total}/N \quad \forall i \in \{1, 2, ..., N\}$ and that the N PEs of both configurations are identical, both configurations support the same throughput rate R:

$$R_{cluster} = \frac{N}{T_{total}} = \frac{1}{T_S} = R_{pipeline} \tag{1}$$

In a pipeline configuration however, it is hardly possible to define sub-tasks of exactly the same duration. Therefore, the

[1]We do not explicitly consider multiple threads within a processor in this comparison.

throughput is limited by the processing time of the longest sub-task. Accordingly, a cluster configuration performs better than a pipeline made up of the same, identical PEs.

One advantage of the pipeline configuration, however, compensates this drawback by far: its modularity. The modular structure of a pipeline allows that each PE can be optimized for its sub-task. Thus the processing times $T_{S,i}$ of the PEs in the pipeline can be significantly reduced, which increases the throughput of the pipeline. Additionally, the resource and power consumption of the PEs decrease due to this optimization. Furthermore, PEs can be easily added, replaced or removed, in order to change the system's functionality or improve its performance.

A benefit of the cluster configuration is that its throughput can be easily increased by adding further parallel PEs to the system. However, the more PEs work in parallel the more probably it may happen that two or more PEs execute the same sub-task at the same time. This may provoke access conflicts if the PEs share resources required for this sub-task. In this case synchronization or arbitration mechanisms have to be used, and the advantage of parallelism vanishes [13]. Additionally, connecting many parallel PEs to shared resources becomes increasingly complex.

In a pipeline configuration, in contrast, one sub-task is always executed in only one PE and no resource conflicts within a sub-task can occur. Additionally, resources can be more easily confined to one PE either architecturally or by programming. Due to this, the time a PE needs for each packet is deterministic. Thus, a pipeline can guarantee a deterministic throughput, making it ideally suited to high-performance network processing tasks [9], [11], [14].

Due to these reasons our novel MIXMAP architecture is based on a pipeline configuration of PEs featuring both modularity and deterministic throughput.

D. Design at different levels of abstraction

Abstraction hides complex details that we do not have to be aware of in a certain context. Therefore, designing at a certain level of abstraction helps building large systems, as the abstraction hides low-level complexities from the designer.

In the domain of computer engineering several levels of abstraction are used [1]. At the level of device physics semiconductor physicists design ever faster and smaller transistors. Based on those designers of integrated circuits build basic logic gates, flip-flops and memory cells. This level is called transistor (or circuit) level. It is almost impossible to design entire packet processing systems at these levels of abstraction. The following levels of abstraction are the gate level, the register-transfer level, the microcode level, and the software level. They are commonly used to design digital systems.

Gate level: Basic elements on this level are logic gates, like AND, OR, and NOT, as well as single-bit flip-flops. They are used for building e.g. specialized arithmetic units, decoders, comparators or finite state machines.

Register-Transfer level: On this level operations on bit vectors are performed. Registers and memory blocks save their state and combinatorial units (e.g. arithmetic units, logic units, de-/encoders) process their contents. Elements for transferring the bit vectors between these elements are busses, multiplexers, and tristate buffers. The behaviour of all elements is governed by control signals, stemming from e.g. a decode unit. FPGA-based systems are designed at register-transfer level and at gate level.

Microcode level: Implementations at this level make use of a given infrastructure built up by elements of the register-transfer level. These elements are used for processing, transfering and storing bit-vectors. Their temporal behaviour is programmed in a very low-level language, called microcode. Microcode is composed of micro-operations that can define any basic operation in the system by setting the respective control signals. Depending on the given infrastructure several micro-operations may be performed in parallel. Microcode has been used extensively to emulate complex instructions with a given simple hardware in Complex Instruction Set Computers (CISC). Currently it is re-discovered for making network processing devices adaptable [15], [16].

Software level: This level of abstraction further hides the complexity of the underlying hardware. Systems on this level are implemented using assembly language or a higher level language like C or Java. On assembly level still a very limited view of the underlying hardware is given, consisting of a register file, a memory model and a set of supported instructions. An assembly program is a sequence of such instructions performing operations on the data in registers and memory. Higher level languages build up on the assembly language. Programming is simplified by abstracting even more from the underlying hardware and providing abstract data types and control structures. These are finally mapped to the assembly language by a compiler. Applications on NPs are implemented in either assembly or a higher level language.

A further abstraction that can be applied to any of these levels is to define parameterizable functional units. The behaviour or structure of these units can then be defined by setting its parameters accordingly. In the following, we call this level of abstraction *parameter level*.

Usually, one is fixed to a certain level of abstraction by the choice of a certain device. NPs are programmed at software level, while FPGAs are described at register-transfer level (and partly also at gate level). However, when designing at a certain level of abstraction, one sometimes misses features that are concealed by the used abstraction. Furthermore, the elements or the given infrastructure at a certain level of abstraction may be inadequate for certain functionalities. Emulating the missing feature can be awkward, more complex, less efficient or even impossible. Then, switching to a lower level of abstraction would evade this obstacle, as the missing feature could be implemented according to one's needs. There are several examples for this: bit level operations are cumbersome and inefficient in assembly but easy to describe and fast at register-transfer level; also, at software level certain unsupported atomic operations on shared resources can only be emulated by complex locking mechanisms, however, are

easily described at register-transfer level.

Our novel MIXMAP architecture for an FPGA-based network processing unit therefore provides the possibility to design functions at register-transfer level, at microcode level, at software level, and at parameter level.

E. Related work

Several proposals exist for NPs that make use of the reconfigurability of FPGAs: [17] proposes FPGA-based reconfigurable coprocessors, [18] presents an NP architecture with reconfigurable data paths. In [19] a reconfigurable network processing platform is described, and [20] proposes to use soft multiprocessor systems on FPGAs for network processing tasks. All these architectures, however, are designed for access and enterprise networks and do not support high-performance packet processing as needed in core networks.

TPACK offers adaptable FPGA-based high-speed network processing units [4], but does not provide the source code of its implementations to its customers for adaptation. NetFPGA [2] is an open platform for science and education to build FPGA-based high-performance networking systems. Its four Gigabit Ethernet interfaces as well as the used FPGA confines its use to lower speed networks. With 10 Gigabit Ethernet interfaces and high-speed FPGAs, however, it could be used as a platform for our architecture.

III. Architecture with Multiple Design Levels

In this section we present our novel MIXMAP architecture for a network processing unit. First, we introduce our design objectives, then we describe the architecture's basic pipeline structure and its modularity, before we detail on the different modules supporting the design of functions at several levels of abstraction.

A. Objectives

Three main objectives have led us to our novel MIXMAP architecture:

- *High packet throughput*: The architecture of our network processing unit shall be capable to process packets at very high rates.
- *Modularity*: The architecture shall be modular, so that functional units can be added, replaced or removed independently from each other.
- *Design at different levels of abstraction*: Various modules shall be provided, that can be adapted, programmed or configured at different levels of abstraction.

To achieve these objectives we chose a pipeline as the basic structure of our MIXMAP architecture. We dimensioned the pipeline such that the highest possible packet throughput can be achieved for minimum sized packets. Modularity is achieved by supporting modular, functional units. The functionality of each of these units can be designed or adapted in modules of different levels of abstraction. The following sections go into more detail about all of these topics.

B. Pipeline Architecture

The basic structure of our architecture is a pipeline, because this structure can provide modularity and a deterministic throughput, as we detailed in section II-C. We dimensioned the pipeline such that a maximum packet throughput can be achieved. Therefore we minimized the processing time T_S per stage and maximized the number of bytes that are processed within a stage.

The minimum processing time is one clock cycle within a digital system, thus we set $T_S = T_{clk} = 1/f_{clk}$. So, the pipeline resembles a pipeline of a classic RISC processor, forwarding its data each clock cycle to the next stage.

When dimensioning the number of bytes that the pipeline processes in each stage, several aspects were taken into consideration: Firstly, in order to achieve a maximum packet throughput, the pipeline should allow to process packets of maximum size in each stage. In an Ethernet network the maximum frame length is 1500 bytes. With current technology, however, that large data words cannot efficiently be processed and forwarded in parallel within an integrated circuit. Secondly, about half of all packets of the Internet traffic are minimum sized packets. So, dimensioning the pipeline for maximum sized packets, leads to an inefficient use of large parts of the pipeline. Thirdly, high-performance network nodes usually process only the header fields lying within the first twenty to sixty bytes of a packet. Therefore, not the entire packet has to be transfered through the pipeline.

Considering these aspects we decided to dimension the number of bytes processed in each stage to the minimum supported packet size, i.e. 64 bytes for an Ethernet network. We call this amount of data in the following *one (data) word*. Additionally, we give two architectural options: With option *FP* (full packet) the complete packet traverses the pipeline and operations on any part of the packet can be performed. With option *1W* (1 word) only the first (large) word of each packet traverses the pipeline and the rest is buffered within a small internal memory. This option offers the maximum possible throughput and is preferable if only operations on the header fields have to be performed.

The uniform behavior of the pipeline allows to specify the packet throughput deterministically. For option *1W* the packet rate is always equal to the clock rate of the system, i.e. with a clock rate of 200 MHz the packet rate equals 200 million packets per second. For option *FP* this throughput is only valid for minimum sized packets and decreases for larger packets.

The supported line data rate of the architecture is at least $D_0 = f_{clk} \cdot w$ for architectural option *1W*, with f_{clk} being the clock rate and w the word width of the pipeline. For option *FP* the internal bit rate varies between D_0 and $D_0/2$, due to partialy filled last data words.

C. Modularity by Functional Units

On top of the basic pipeline structure, the architecture is structured into functional units, as depicted in Fig. 2. A functional unit contains one or several pipeline stages and

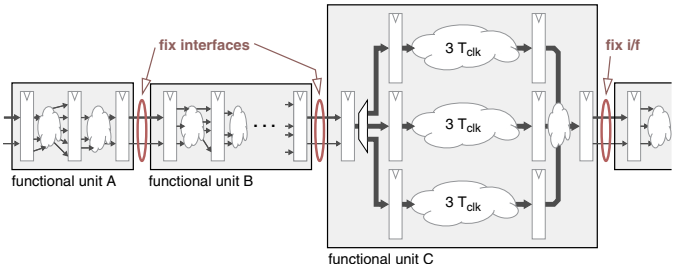

Fig. 2. The MIXMAP architecture is structured into functional units that can be arbitrarily added, exchanged or removed.

performs a certain packet processing function, e.g. a checksum computation, or an address lookup.

It shall be possible to easily add arbitrary functional units to a system, replace or remove units, in order to change the functionality of the network processing unit. Therefore, all functional units have to fulfill strict interface specifications.

The interface of a functional unit is defined by its input and output signals and their temporal behaviour. The interface signals are the data word containing the bytes of the processed packet, some control signals and meta-data.

The meta-data carries additional information from external devices and internal functional units that might be needed by other functional units. Each piece of meta-information has to be at a fix position, so that all functional units can properly access this data. Examples of meta-information are the ingress port of the current packet, the packet size, classification results, the next-hop address, and the egress port. The size of the meta-data should be large enough to allow the definition of fields with additional information.

All interface signals have to obey the temporal requirements given by the pipeline: In each clock cycle one data word has to be accepted and one processed data word has to be sent out. The way how each functional unit is organized internally is irrelevant. A functional unit can be organized as a pipeline or a cluster of processing entities, as long as the temporal interface requirements are fulfilled (cf. Fig. 2).

Each functional unit can be based on any of various module types: Modules whose functionality is designed at register-transfer level, modules that are micro-programmed, modules that are implemented at software level, and modules that only have to be parameterized. These module types are described in the following sub-sections.

In order to provide this modularity, the technology for a network processing unit implementing this architecture has to support a fine grain reconfigurability like an FPGA. Reconfigurability is required on gate and register-transfer level to support the design of functional units at this level, and on module level to support adding and removing of functional units. We therefore intend an FPGA as the base technology for implementing the MIXMAP architecture.

Functional units are implemented independently of other units using one of the provided modules. After that, the module is embedded into the pipeline structure. Then, the synthesis and place & route processes are performed to map the entire network processing unit on the FPGA device. Optionally, also partial reconfiguration might be used.

D. Register-Transfer Level Module

Register-transfer level modules allow to design functional units at the lowest level of abstraction possible within the MIXMAP architecture. Such modules offer the highest freedom to design specialized functions and are the most efficient in terms of chip area consumption. On the downside, designing a functional unit at this level requires hardware design knowledge in order to exploit the benefits of the inherent parallelism. Furthermore, one has to take care that the temporal interface requirements are met.

For realizing a functional unit based on such a module, one starts with an "empty" HDL-file containing only the declaration of the required input and output signals. Based on this, one implements the functionality using the respective hardware-description language. The functionality can be validated using common HDL-simulation tools (e.g. Modelsim).

E. Microcode Level Module

Functional units can be also implemented using microcode. Microcode level modules provide a pre-defined infrastructure that ideally suits the pipeline structure. By this the programmer of the module neither has to think about observing the temporal interface requirements nor any internal timing constraints. Instead, one can concentrate on the actual functionality.

The pre-defined infrastructure is built up by a sequence of multi-purpose pipeline stages containing multiple multiplexers and arithmetic and logic units. These are controlled by user-programmed microcode that is saved locally in each stage. The user, i.e. the programmer, can describe the functionality of the module by micro-operations. In each pipeline stage several micro-operations can be executed in parallel. Possible micro-operations are arithmetic, logic and shift operations that work on almost arbitrarily-sized slices of bit-vectors. Also conditional operations are supported.

The microcode is saved in on-chip memory and can be loaded on the system during operation.

F. Software Level Modules

The third group of modules is software-controlled. Programs can be either written in low-level assembly language in order to use the underlying hardware the most efficiently, or in a language like C to be able to use the benefits of a high level programming language.

All functional units have to accept and send one data word in every clock cycle, so one single processor in a module could process each data word for one clock cycle only. Therefore multiple processors have to be within such a module. There are two variants, how to implement this kind of module:

The first variant is to place a cluster of N processors that work in parallel into the module as illustrated in Fig. 3. These processors can be highly optimized (possibly even hard-wired), so a higher clock frequency than in the system pipeline may be supported. Thus, having N processor cores working

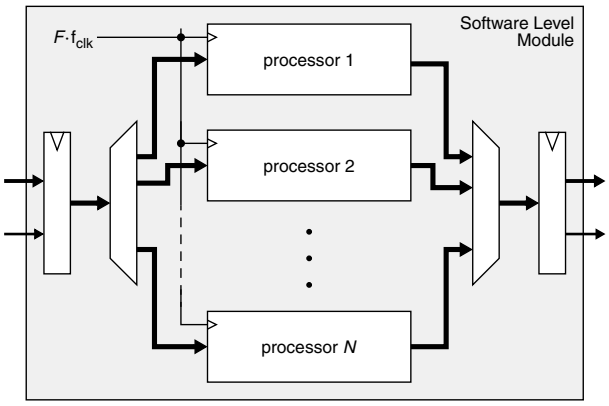

Fig. 3. A software level module of variant 1 contains a cluster of N parallel processors with an F times faster clock rate than the system pipeline.

with an F times faster clock frequency, a program is allowed to execute $N \cdot F$ clock cycles. E.g. with N=16 processors with an F=4 times faster clock, programs with up to 64 clock cycles are possible.

This small number still seems to be too less for reasonable programs. However our analysis of existing program code for the Intel IXP2400 NP [12] showed, that many functions can be performed that quickly by using an optimized instruction set and internal memory accesses only.

The second variant is to implement a system with a synchronous data-flow architecture [21] comparable to the Xelerated NPs [9]. Such a system consists of a pipeline of processing units through which the packet- and meta-data flows together with an instruction pointer. Here, too, the number of instructions is limited to the number of processing stages multiplied by a possible speed-up factor, if a higher clock rate is used.

G. Parameter Level Modules

Finally, readily-build functional units that provide editable parameters are a fourth way to adapt the functionality of the network processing unit to new or changed requirements. Obviously only small adaptations that are thought of a priori can be made.

IV. Prototypical Implementation and Evaluation

In order to check the feasibility of the MIXMAP architecture, we designed and implemented a prototypical system featuring our novel architecture. For this system we realized several functional units based on modules of different levels of abstraction. Finally, we performed several tests on our FPGA-based Universal Hardware Platform [22].

Our prototypical system possesses the pipeline structure as described in the previous section. The complete packet traverses the pipeline (option *FP*), so there is no need for a packet buffer. To show the design of functional units at different levels of abstraction we implemented modules for the design at register-transfer level and at software level. (Modules for the design at microcode level are going to be implemented soon.)

As described in the previous section, the register-transfer level modules do not provide any fix infrastructure except for the standardized interface signals on ingress and egress side.

Our software-level modules are realized as a cluster of N processors (i.e. variant 1) that offer an optimized instruction set for packet processing tasks. The processors are clocked with an F times higher rate than the system pipeline (N and F can be parameterized according to the requirements). A control unit distributes the incoming data in a round-robin fashion each system clock cycle to one of the N processors, and re-collects it respectively N system clock cycles later. Within the processors this data is stored in the upper half of the register file.

We exemplarily implemented functional units of all basic fast-path router functions using register-transfer level modules and some using software level modules: Ethernet header processing, IP (Internet Protocol) header validation, IP route lookup, IP header update, Ethernet header update. Furthermore, we also realized several router-assisted congestion control functions possibly needed in future Internet routers at both levels of abstraction. This shows the easy adaptability to new requirements. Parameters for configuring the functional units add some adaptability at parameter level, too.

We synthesized several configurations of our prototype system and put it into operation on our Universal Hardware Platform [22]. This platform features an Altera Stratix II EP2S60F1020C3ES FPGA, up to six Gigabit Ethernet interfaces as well as mezzanine cards that offer additional memory or ternary content addressable memory (TCAM).

Due to the rather small FPGA used in our tests, we could only place nine processors in our software module and had to clock the system pipeline with a lower clock frequency in order to be able to execute our programs. However, modern FPGAs have a much higher capacity, so that more than one hundred processors should fit into one device.

Other configurations, which were solely based on register-transfer level modules, used a clock rate of 115 MHz. Therefore, such a system could process up to 115 million minimum sized Ethernet packets per second. This corresponds to a pipeline throughput of almost 60 Gbit/s (option *FP*)—achieved on our four year old FPGA. If only the first word of each packet is processed by the pipeline (option *1W*) and assuming an average frame length of 200 bytes, a line rate of 200 Gbit/s could be achieved—given a fast enough packet buffer.

Obviously we could not test the full throughput using the Gigabit Ethernet interfaces. Therefore, we validated the correct functionality by tests under full load using packets that were both generated and, after processing, verified on chip.

V. Conclusion

In this paper we proposed a novel architecture for an FPGA-based high-performance network processing unit. Network nodes in future high-speed packet networks, e.g. in a 100 Gbit Ethernet network, have to process more than one hundred million packets per second. Additionally, the processing units

have to provide a certain degree of flexibility to be easily adaptable to upcoming new services or protocols.

We reviewed FPGAs and NPs, which are adaptable devices currently used for network processing on high-performance router line-cards. We discussed the principal differences in the organization of processing entities as a pipeline or in a cluster and compared the different levels of abstraction that are used when implementing functions on FPGAs and NPs. Implementing certain functions at a lower level of abstraction may be more efficient, while a higher level of abstraction simplifies the design process.

Therefore we designed our MIXMAP architecture with the objective to provide the possibility to design functions at different levels of abstraction, featuring functional units based on modules on register-transfer level, microcode level, software level, and parameter level. The strict interface requirements and the architecture's modular structure allows to easily add, replace or remove functional units. To achieve the objective of a very high packet throughput suitable for core network nodes, the architecture is based on a high-performance pipeline structure. This pipeline is capable of processing one minimum sized packet in each clock cycle. This is achieved by the use of extremely large pipeline registers in each stage and a processing time of only one clock cycle per stage.

Finally, we presented a prototype implementation featuring our novel MIXMAP architecture. Our exemplary implementation of functional units providing both functions of current and of future Internet routers validated the usability and flexibility of our modular architecture. Synthesis results as well as functional and performance tests showed that today a high-performance MIXMAP network processing system capable to process more than hundred million packets per second is technologically feasible, and even higher packet rates will become possible with future FPGA devices.

ACKNOWLEDGMENT

The author would like to thank his colleagues Oswin Horvath and Arthur Mutter for valuable discussions, and Florian Mück, Micha Klingler and Domenic Teuchert for their help in implementing the MIXMAP architecture.

REFERENCES

[1] J. F. Wakerly, *Digital Design: Principles and Practices*, 4th ed. Upper Saddle River, NJ, USA: Pearson Education, Inc., 2006.

[2] G. Gibb, J. Lockwood, J. Naous, P. Hartke, and N. McKeown, "NetFPGA – an open platform for teaching how to build gigabit-rate network switches and routers," *IEEE Transactions on Education*, vol. 51, no. 3, pp. 364–369, Aug. 2008.

[3] A. Mutter, M. Köhn, and M. Sund, "A generic 10 Gbps assembly edge node and testbed for frame switching networks," in *Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities (TridentCom2009)*, 2009.

[4] TPACK A/S, "SOFTSILICON for flexible packet transport," Jan. 2008. [Online]. Available: http://www.tpack.com/fileadmin/user_upload/Public_Attachment/SOFTSILICON_for_Flexible_Packet_Transport_web_v1.1.pdf

[5] D. Comer, *Network Systems Design Using Network Processors*. Upper Saddle River, NJ, USA: Pearson Education, Inc., 2006.

[6] N. Shah, "Understanding network processors," Master's thesis, University of California, Berkeley, Sep. 2001.

[7] Altera Corporation, "Power-optimized solutions for telecom applications," White Paper WP-01089-1.0, Jan. 2009. [Online]. Available: http://www.altera.com/literature/wp/wp-01089-power-optimized-telecom.pdf

[8] Cisco Systems, Inc., "Cisco CRS-1 Carrier Routing System," 2006. [Online]. Available: http://www.cisco.com/en/US/prod/collateral/routers/ps5763/prod_brochure0900aecd800f8118.pdf

[9] Xelerated AB. (2009) Xelerated HX300 family. [Online]. Available: http://www.xelerated.com/templates/page.aspx?page_id=329

[10] Hi/fn, Inc., "HIFN 5NP4G network processor," product brief, 2008. [Online]. Available: http://www.hifn.com/uploadedFiles/Library/Product_Briefs/5NP4G_pb_v1.pdf

[11] EZchip Technologies, Inc., "Np-3," product brief, 2007. [Online]. Available: http://www.ezchip.com/Images/pdf/NP-3_Short_Brief_online.pdf

[12] Intel Corporation, "Intel IXP2400 network processor," data sheet, Feb. 2004. [Online]. Available: http://download.intel.com/design/network/datashts/30116411.pdf

[13] S. Hauger, M. Scharf, J. Kögel, and C. Suriyajan, "Quick-Start and XCP on a network processor: Implementation issues and performance evaluation," in *Proceedings of IEEE High Performance Switching and Routing (HPSR)*, May 2008.

[14] S. Hauger, T. Wild, A. Mutter, A. Kirstädter, K. Karras, R. Ohlendorf, F. Feller, and J. Scharf, "Packet processing at 100 Gbps and beyond—challenges and perspectives," in *Proceedings of the 10. ITG Symposium on Photonic Networks*, May 2009.

[15] S. Vassiliadis, S. Wong, and S. Cotofana, "Microcode processing: Positioning and directions," vol. 23, no. 4, pp. 21–30, 2003.

[16] D. Saß, S. Hauger, and M. Köhn, "Architecture and scalability of a high-speed traffic measurement platform with a highly flexible packet classification," *Computer Networks*, vol. In Press, Corrected Proof, pp. –, 2008. [Online]. Available: http://www.sciencedirect.com/science/article/B6VRG-4V2NP6F-1/2/9d39e95cefff3940bb67a1041bc1cd36

[17] C. Albrecht, R. Koch, and E. Maehle, "Dynacore — a dynamically reconfigurable coprocessor architecture for network processors," in *Proceedings of the 14th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP '06)*. Washington, DC, USA: IEEE Computer Society, 2006, pp. 101–108.

[18] M. Meitinger, R. Ohlendorf, T. Wild, and A. Herkersdorf, "FlexPath NP - a network processor architecture with flexible processing paths," *System-on-Chip, 2008. SOC 2008. International Symposium on*, pp. 1–6, Nov. 2008.

[19] C. Kachris, "Reconfigurable network processing platforms," Ph.D. dissertation, Technische Universiteit Delft, Dec. 2007.

[20] K. Ravindran, N. R. Satish, Y. Jin, and K. Keutzer, "An FPGA-based soft multiprocessor system for IPv4 packet forwarding," in *15th International Conference on Field Programmable Logic and Applications (FPL-05)*, Aug. 2005, pp. pp 487–492.

[21] J. Carlstrom and T. Boden, "Synchronous dataflow architecture for network processors," *IEEE Micro*, vol. 24, no. 5, pp. 10–18, Sept.-Oct. 2004.

[22] Institute of Communication Networks and Computer Engineering, Universität Stuttgart, "The Universal Hardware Platform (UHP)," 2005, http://www.ikr.uni-stuttgart.de/Content/UHP/.