FlexPath NP – A Network Processor Concept with Application-Driven Flexible Processing Paths

Rainer Ohlendorf, Andreas Herkersdorf, Thomas Wild
Munich University of Technology
Arcisstrasse 21
D-80290 Munich, Germany
+49 89 289-23872
Rainer.Ohlendorf@tum.de

ABSTRACT

In this paper, we present a new architectural concept for network processors called FlexPath NP. The central idea behind FlexPath NP is to systematically map network processor (NP) application sub-functions onto both SW programmable processor (CPU) resources and (re-)configurable HW building blocks, such that different packet flows are forwarded via different, optimized processing paths through the NP. Packets with well understood, relatively simple processing requirements may even bypass the central CPU complex (AutoRoute). In consequence, CPU processing resources are more effectively used and the overall NP performance and throughput are improved compared to conventional NP architectures. We present analytical performance estimations to quantify the performance advantage of FlexPath (expressed as available CPU instructions for each packet traversing the CPUs) and introduce a platform-based System on Programmable Chip (SoPC) based architecture which implements the FlexPath NP concept.

Categories and Subject Descriptors

C.0 [**GENERAL**]: *System architectures*

C.1.3 [**Other Architecture Styles**]: *Adaptable architectures, Data-flow architectures, Pipeline processors*

C.2.6 [**Internetworking**]: *Routers*

General Terms

Performance, Design

Keywords

Network processors, application-specific architectures, hardware accelerators, dynamically reconfigurable processors, IP networking

1. INTRODUCTION

Network processors (NPs) are programmable VLSI (Very Large Scale Integration) components with instruction set extensions, dedicated co-processors and network interfaces to support packet forwarding and processing. NPs attempt to merge the flexibility characteristics of general purpose CPUs with the performance of networking ASICs (Application Specific Integrated Circuit) in terms of packet throughput (Gbit/s) and latency (< few ms). The requirements for high flexibility and performance result from a rapid growth in new networking applications on the one hand, and the fact that these new protocols and services are deployed and tested in the market before being finally standardized. Example application areas for NPs include QoS (Quality of Service)-based IP routers [1, 2], secure VPN (Virtual Private Network) gateways [3], multimedia IP service integration in wireless (GSM / UMTS) base station controllers [4], alternative VoIP telephony networks [5] and broadband IP service provisioning to the home via DSL (Digital Subscriber Line) or Cable Modem technologies. NPs in various performance classes will evolve as a key technology component for future generations of packet switching systems deployed in both private and public communication networks. A detailed overview and comparison among various NP architectures is found in [6].

Programmable components – such as NPs – have the inherent advantage over ASICs that they can be easily adapted in the field to specification changes or functional enhancements by means of software upgrades. The main challenge for these programmable solutions is to keep up with the performance of ASICs. In this paper we introduce a system architecture level contribution to NPs with the objective to mitigate the flexibility/performance dilemma. Configurable, application-driven processing paths enable a flexible, packet flow-specific traversal of NP (hardware and software) building blocks. Packets which predominantly can be handled by NP hardware units consume only minimal (or no) CPU processing resources, packets for which the provisioned NP hardware is insufficient are dominantly processed by the CPUs.

The paper is structured as follows: section 2 provides a brief conceptual overview on state-of-the-art NP architectures. Section 3 presents the fundamental concepts of the FlexPath NP and its AutoRoute operation mode. The performance estimation to quantify the expected benefits of FlexPath versus a conventional NP architecture is subject of section 4. As the FlexPath project is "work-in-progress", we describe in section 5 the SoPC (System on Programmable Chip) architecture of a demonstrator to proof the validity of the FlexPath concept in a real network. We conclude with an outlook to future work in section 6.

2. BACKGROUND AND RELATED WORK

Today's relevant network processor architectures [7] are based on a "processor-centric" approach as depicted in figure 1. Processor-centric means that after packet reception and storage in the packet data memory sub-system, the (multi-processor) CPU cluster takes control over the packet handling. This includes determining the sequence in which dedicated co-processors and hardware accelerators for networking related functions (e.g. address lookup, cryptography, traffic management, etc.) are traversed. Thus, the control over the packet can change multiple times between CPU cluster and co-processors before the packet is retransmitted over standard network interfaces (e.g. 10/100 or Gbit Ethernet, SONET/SDH Utopia, etc.).

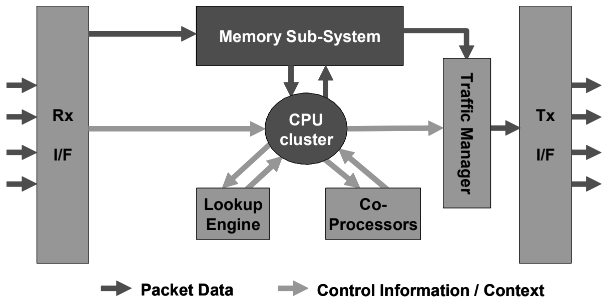

Figure 1: Classical processor-centric NP concept

Although the NP implementations of various vendors differ greatly with respect to the realization of the processor complex (SMP (Symmetric Multi-Processing) RISC cores, pipelined micro-engines, proprietary ASIPs), memory hierarchy and the amount of co-processor support; each and every packet, irrespective of its application-specific processing demands, starts its processing path at the CPU and typically traverses the CPU cluster multiple times [8, 9].

Assuming an aggregate ingress link capacity of 1.244 Gbit/s (2× OC-12) and a worst-case scenario of consecutive shortest packets of 64 Bytes (referring to the minimum frame size in Ethernet) this results in a packet rate of $\frac{1.244 Gbit/s}{64 \times 8 bit/packet} = 2.4 Mpps$ (million packets per second) or a new packet arrival every 417 ns. The packet rate is the event rate each processor in a pipeline structure, or the SMP cluster as a whole, has to cope with. If packet control switches several times between CPUs and co-processors, the event rate (or thread context changes) multiplies accordingly. While dedicated co-processors significantly offload the CPU cluster from processing intensive tasks, they cannot reduce the packet (event) rate in the above described processor-centric NP architecture concept, but will even increase it instead.

3. THE FlexPath NP CONCEPT

Under the assumptions that

- high bit rate network connections (such as Gbit Ethernet or OC-12 and higher SONET/SDH links) consist of a mixture of many (thousands of) individual packet flows with very diverse processing requirements and
- packet flows with relatively low processing complexities can even be completely processed and forwarded by application-specific hardware units

we devised a new NP concept called FlexPath NP (see figure 2). In a FlexPath NP, a variety of alternative processing paths (i.e. NP building block traverse sequences) are conceivable. For example:

- Packets or packet contexts may be sent to a number of co-processors (e.g. next hop address lookup, authentication or decryption) before being handed over to the CPU along with the obtained results. The computing amount for the CPU is reduced, while the packet event rate is not increased as in the processor-centric reference scenarios.
- Packets with relatively simple processing requirements can directly be sent to the Traffic Manager unit on the egress side of the NP. As packets bypass the CPU cluster in this operation mode, we call this processing path **AutoRoute**. The "saved" CPU computing capacity will instead be preserved for packets remaining on the CPU path. We refer to section 4.2 for a discussion of viable AutoRoute scenarios.
- Packets or packet contexts may directly be sent to the CPU cluster as in the "processor-centric" case. Actually, the case where all packet flows use the CPU path will serve later as a reference for evaluating the benefits of the other FlexPath alternatives.

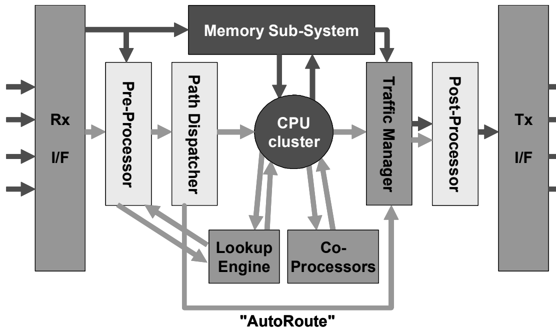

Figure 2: FlexPath NP concept with dynamically reconfigurable packet paths

The decisions about the most suitable processing path have to be made on a per-packet basis (remember the inter-arrival time of 417 ns in case of 1.244 Gbit/s link rate). In the FlexPath NP, this decision function has been eliminated from the software process running on the CPU cluster. It is now performed by a hardware assist called **Path Dispatcher** that contains reconfigurable rule-tables. Based on these rules and the incoming packet context the processing path selection is made. If no matching rule can be found, the default processing path, which traverses the CPU cluster, is chosen. Taking an alternative processing path requires explicit permission in form of a matching rule entry. Configuration and updating of the rule-tables is the task of a software process (called Path Manager) running on the NP control point CPU. Updating the rule-tables is not timing critical as it only requires intervention when a packet flow class terminates, newly initiates or changes its processing path during its lifetime (Note, that we currently anticipate several hundred to a few thousand rule-table updates per second [10,11]).

Besides the Path Dispatcher, a packet Pre-Processor and a packet Post-Processor unit are necessary prerequisites for alternative processing paths in FlexPath.

The **Pre-Processor** unit is running in parallel to the packet data storage functionality in the ingress path of the NP. It performs basic functions like packet header parsing, extraction of pertinent header information (destination/source addresses, L4 port numbers, priority bits, protocol IDs), alignment of extracted header fields to CPU word length (32 bit), packet prioritization and classification and calls the address lookup or other co-processor(s). Thus, the Pre-Processor retrieves sufficient information about the incoming packet (and builds the so-called packet context) which enables the Path Dispatcher to make decisions about the further handling of the packet.

Packets still traversing the CPU cluster nevertheless benefit from the pre-processing. CPUs do not have to build the packet contexts themselves and save thread switches or stall periods due to co-processor invocation.

The **Post-Processor** performs at minimum basic egress packet modification operations like MAC address insertion, TTL decrement, and IP checksum calculation. These functions are mandatory for finishing packet processing on simple AutoRoute cases. The Post-Processor performs these operations by extracting corresponding commands out of the packet context. Post-Processor commands are added to the packet context either by the Pre-Processor or CPU software which has to be aware of the Post-Processor capabilities.

Thus, packets traversing the CPU cluster benefit again from the FlexPath Pre-/Post-Processor units. The CPUs can take advantage of the offered features and are not obliged to perform the necessary bit-level modifications by themselves.

The FlexPath NP concept allows for a maximum of flexibility in NP packet processing as packet flow classes can individually and dynamically change their processing paths anywhere between the classical "processor-centric" and the aggressive AutoRoute modes. Of particular interest are packet flows where the processing path dynamically changes with different phases of the packet flow's lifetime. For example, during connection setup the packets traverse the CPU cluster. If the CPU identifies that there is sufficient functionality in the ingress and egress processing units for this flow (Pre-/Post-Processors), the Path Dispatcher rule-tables are updated to either invoke co-processors before passing control to the CPU or AutoRoute subsequent packets of this flow. If in a later stage the Pre-Processor identifies unknown options or error conditions in packets of the particular flow, it is necessary to change the forwarding path back towards the CPU cluster. When performing these dynamic transitions, it must be insured that no packets get lost and that the sequence of packets belonging to the same flow is preserved.

4. PERFORMANCE ESTIMATION

By means of analytical performance estimations, we now quantify the gain in CPU processing performance (expressed as available CPU instructions per packet (IPP)) depending on the fraction of traffic (b) taking the AutoRoute path. An aggregate NP link rate (R) of 1.244 Gbit/s and a packet size (s) of 64 Bytes is assumed (2.4 Mpps). Network processing data plane applications vary in terms of processing requirements between a few hundred to several thousand IPP [12, 13]. Simple packet forwarding (ATM / Ethernet switching, IPv4 trie forwarding) requires a minimum of 200 – 500 IPP, MPLS or QoS-enabled packet forwarding between 500 – 700 IPP, VPN and traffic load balancing applications in the order of 1500 – 2000 instructions and 3000 and more instructions (per 64 byte packet segment) are required for intrusion detection and virus scanning. Note that the latter examples are payload processing applications where the IPP is correlated to the packet length and can, for large packets with hundreds of bytes, amount to several ten thousand IPP. Regarding the case of very simple forwarding applications, the minimum processing requirement already amounts to $2.4 Mpps \times 300 \frac{Instr.}{packet} = 720 MIPS$.

Three different system alternatives with variable processor resources are compared. Table 1 summarizes the key system and CPU performance parameters. The three system alternatives are:

1. **Processor-Centric** (as the reference scenario): We choose an architecture with 6 ASIP cores, each running at 232 MHz [14]. As the ASIP cores support simultaneous multithreading, an (optimistic) CPI (clocks per instruction) of 1 was considered yielding a nominal processing performance of 1392 MIPS for the ASIP cluster. The average available IPP(0) in case of b=0% is 573 instructions.
2. **FlexPath_1**: The FlexPath_1 configuration consists of a CPU cluster with one standard, single-threaded embedded RISC core (e.g. PowerPC 440 or MIPS). Such RISC cores are available as hard macros in ASIC libraries and typically run at up to 667 MHz in 130nm CMOS technologies [15]. Since single-threaded CPUs stall during memory and coprocessor accesses, we considered a CPI of 1.4 for this system alternative. The nominal processor performance now amounts to 476 MIPS and the average IPP(0) is 196 instructions.
3. **FlexPath_2**: FlexPath_2 has twice the processing power as FlexPath_1. Two embedded RISC cores with a total performance of 953 MIPS are employed yielding an average IPP(0) of 392 instructions.

Table 1: System Parameters Processor-Centric vs. FlexPath

	Centric	FlexPath_1	FlexPath_2
CPU clock [f]	232 MHz	667 MHz	667 MHz
Packet size [s]	64 Byte	64 Byte	64 Byte
NP bit rate [R]	1.24 Gbit/s	1.24 Gbit/s	1.24 Gbit/s
CPU count [N]	6	1	2
CPI [CPI]	1.0	1.4	1.4
Nom. MIPS [M]	1392	476	953
Avg. IPP (b=0%)	573	196	392

In the following IPP estimations, we will, for the sake of a conservative analysis, exclude the Pre-/Post-Processor MIPS but keep in mind that the FlexPath ingress and egress hardware functionality, together with a MAC/IP address lookup engine shall be adequate to perform simple packet switching and forwarding.

4.1 Potential AutoRoute Gain

The first question to investigate is: "How large is the available CPU processing budget (expressed as instructions per packet) for packets traversing the CPU cluster if a variable percentage (b) of packets take the AutoRoute path?"

Given the system parameters from Table 1, one can compute the available CPU IPP1(b) for all three system alternatives with the following formula: $IPP1(b) = \frac{N \times f \times s}{CPI \times R}\left(\frac{1}{1-b}\right)$.

Figure 3 illustrates the resulting instruction budgets with IPP1 drawn on the Y-axis in logarithmic scale.

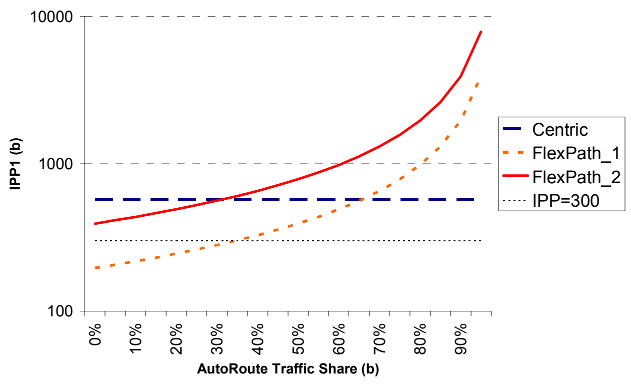

Figure 3: IPP1(b): Processor-Centric vs. FlexPath_1 and FlexPath_2

IPP1 for the processor-centric alternative is – by definition – independent of b (processor-centric means 0% bypass) and has a constant value of 573 instructions per packet, which is sufficient for QoS or DiffServ forwarding on the entire link capacity.

With respect to the two FlexPath alternatives, the following observations can be made:

- For b=0% (i.e. all packets traverse the CPU cluster) the respective IPP1(0) starts off in case of FlexPath_1 at 196, and in case of FlexPath_2 at 392. Thus, for b=0%, FlexPath_1 is not a viable alternative for any networking application under the link rate assumptions made. FlexPath_2 at b=0% fulfills the required CPU performance for simple packet switching and forwarding.
- However, both curves show a more than exponential increase of IPP1(b) over b. The break even points with the processor-centric alternative are at b_{BE}=65.8% for FlexPath_1 and b_{BE}=31.5% for FlexPath_2. The break even points indicate that if b_{BE} percent of the traffic can be forwarded by the FlexPath hardware assists, the remaining (100–b_{BE}) percent of the traffic receive the same service in the RISC CPU(s) as packets in the processor-centric alternative. Expressed differently, the CPU IPPs saved on the AutoRoute packets are now available to provide higher CPU processing capacities for those packets still traversing through the CPU cluster.

For even higher values of b in the range of 80% to 90%, IPP1 values between 2000 and 3000 can be achieved with FlexPath_2, allowing the execution of complex tasks for the remaining packet share.

4.2 Viability of AutoRoute Scenarios

The critical question to be answered for the viability of the FlexPath concept is: "How realistic is it to assume that 40% or more of the traffic volume on high-speed links can take the AutoRoute path?"

Consider a typical traffic profile as it is found in the internet today [16]. The measured results on Sprint's OC-48 (2.5 Gbit/s) internet backbone links show that 90% of the traffic uses the TCP/IP stack. Another 6.2% of the traffic is UDP/IP traffic belonging to streaming applications (video, audio or VoIP). A well-known property of the TCP protocol is that every packet needs to be acknowledged. This is achieved by sending another TCP packet from the receiver to the sender of the original message with the ACK bit set in the TCP header. Although "piggybacking" of these acknowledgements into packets carrying other payload information is possible, in most cases there is no data in the acknowledgement packets. TCP ACKs need no sophisticated processing but just simple forwarding to reach their destination. Roughly 45% of all packets in the network are such simple, minimum sized acknowledgement packets that can be handled by the FlexPath AutoRoute concept and we have identified one common example where shortest length packets in the internet can be processed without CPU intervention.

Let's look at another application example from the wireless access network. 3G base stations or UMTS radio access network gateways is network equipment sitting between wireless end users and the wire-line telephony and data networks. Base stations and access gateways typically support a wide range of networking interfaces: OC-3/OC-12 SONET/SDH Utopia and T1/E1 (1.5 Mbit/s, 2Mbit/s) TDM (time division multiplex) channels. The ratio between "forwarding traffic" among daisy-chained base stations via the OC-3/OC-12 links, and traffic to/from mobile terminals via the T1/E1 channels, can get as high as 10:1. Here, only 10% mobile user traffic needs high flexibility and processing complexity (for functions like header compression, protocol conversion etc), while 90% of the traffic can be forwarded based on the L2/L3 information (thus, being a candidate for the AutoRoute path). In such an application, an IPP1 of 2000 – 3000 creates valuable headroom for the computationally intensive functions on up to approximately 120 Mbps capacity, which is equivalent to roughly 60 E1 channels.

The above list of AutoRoute-"friendly" applications is not exhaustive. Streaming media and real-time applications, such as VoIP or UDP/RTP, could also benefit from the low latency, low jitter variation AutoRoute forwarding path that is fully implemented in hardware. Summing up the above mentioned cases, we see a significant share of the traffic as premier candidates for the AutoRoute forwarding path. The remaining packet processing functions may then be realized on a reasonably smaller and less stressed CPU cluster.

4.3 NPs under Variable Load Conditions

An interesting perspective on processor-centric NP architectures is gained with the following investigation: "What is the remaining CPU processing budget IPP2 in a processor-centric NP architecture if d% of the packets have a processing demand of Z CPU instructions per packet?". The corresponding formula is:

$$IPP2(d) = \left(\frac{N \times f \times s}{CPI \times R} - d\,x\,Z \right) \left(\frac{1}{1-d} \right)$$

Figure 4 shows the corresponding set of curves for selected Z values of 200, 300, 600, 1000, 1500 and 2000 instructions per packet in dependency of d varying between 0 and 95% of the aggregate NP bit rate R. We see that for Z<IPP1(0)=573, the IPP2(d) processing budget increases with a rising fraction of d. Here, like in the AutoRoute case, spare CPU cycles saved on low complexity packet flows free up higher processing budgets for more demanding packet processing. For packet workloads Z>IPP1(0), we observe a rapid decline of the available processing budget for the remaining traffic already with moderate values for d. For example, with Z=1500 IPP and d=20%, the remaining 80% of the NP traffic has a processing budget of about 300 IPP, which is barely enough to cope with simple packet forwarding. Hence, the conclusion is that when a processor-centric NP architecture is exposed to heavy processing demands for a moderate fraction of its capacity, the lion share of the traffic can not receive a service level which exceeds basic forwarding functionality.

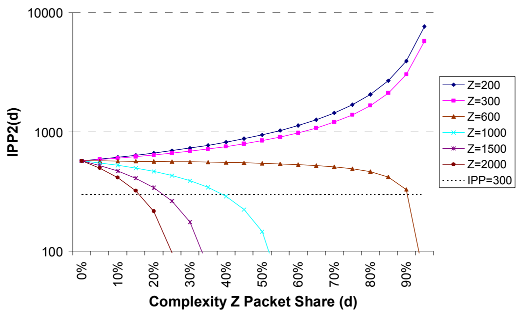

Figure 4: Processor-centric NP under variable workloads

A comparison to the FlexPath_2 case in figure 3 reveals that with 80% AutoRoute – i.e. 80% of traffic delegated to HW-assisted simple forwarding capabilities – the processing budget for the remaining 20% of traffic is also in the range of 1500 IPP. Both NPs deliver about the same processing budget to the heavy workload traffic: The processor-centric case with a specialized six-fold processing engine, the FlexPath NP with two general purpose RISC cores with nominally 40% less processing capacity but Pre-/Post-Processing and Path Dispatcher assists.

The question we declared to be critical for the viability of the FlexPath AutoRoute concept – "Is there a sufficient fraction of traffic on the high-speed link that can take the AutoRoute path?" – is equally critical for the processor-centric NP architecture. Without sufficient low processing complexity packet flows, the processor-centric NP runs "out of steam" to cope with a moderate fraction of processing intensive packet flows.

The above analysis is also applicable to FlexPath when considering the percentage of AutoRoute traffic b in the formula:

$$IPP2(d,b)=\left(\left(\frac{N\times f\times s}{CPI\times R}\right)-d\ x\ Z\right)\left(\frac{1}{1-b-d}\right)$$

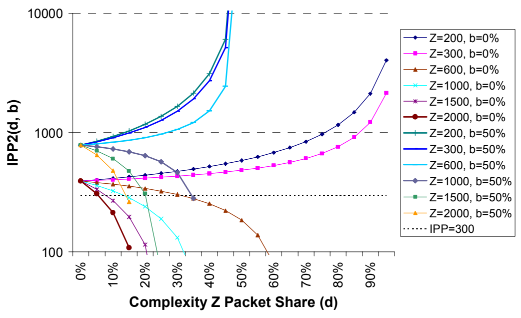

Figure 5: IPP2(d, b) for two FlexPath_2 scenarios

d now refers to a fraction of the total traffic that traverses the CPU cluster, i.e. d≤(1-b). Figure 5 shows two scenarios: FlexPath_2 with b_1=0% (lower curve set) and b_2=50% AutoRoute (upper curve set).

5. FlexPath ARCHITECTURE

We currently develop a demonstrator of the FlexPath_1 NP on an off the shelf, commercially available FPGA development board [17]. The board hosts a Xilinx Virtex-II Pro 30 component featuring two embedded 405 PowerPC hard macros running at up to 300 MHz, has 256 MB SDRAM for packet and context memory and a 10/100 Ethernet PHY. G-bit Ethernet PHYs can be connected to the FPGA via a small daughter board.

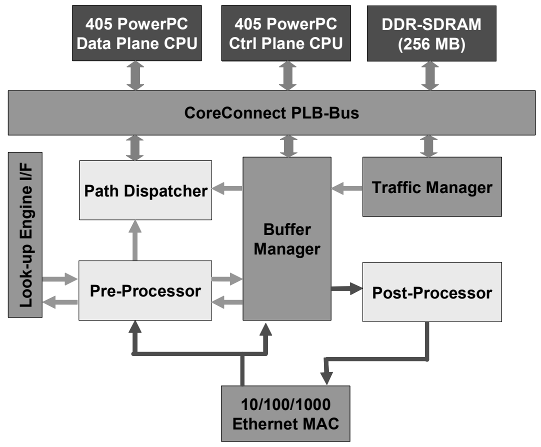

Figure 6: Architecture of the FlexPath Reference Design

Figure 6 shows the architecture block diagram of our FlexPath_1 demonstrator. It follows a standard, platform-based System on Chip (SoC) architecture with two PowerPC cores (one for data plane processing and one for control plane processing) and one SDRAM memory controller attached to the on-chip 64 bit wide CoreConnect PLB (processor local bus) running at 100 MHz clock rate. All of the above mentioned building blocks, plus the 10/100 Ethernet MAC unit, are either provided with the development environment or through the Xilinx Core Alliance IP libraries. Apart from system integration, they do not require any development effort. The Buffer Manager, which performs packet segmentation/reassembly and storing/retrieving of packet segments to/from SDRAM memory, and the Traffic Manager, which schedules dual priority queues per physical NP port, are designs reused from other projects pursued in our group. The Pre-Processor accesses an external Lookup Engine for next-hop destination port determination.

The development effort for the FlexPath_1 demonstrator is limited to the light gray blocks: Pre-Processor, Path Dispatcher and Post-Processor, which were introduced in section 3. Objectives of the FlexPath_1 demonstrator are:

1. Prove that AutoRoute packet forwarding can be accomplished for conventional packet forwarding with the described Pre-/Post-Processor and Path Dispatcher functions in an otherwise standard RISC-based SoC environment.
2. Validate that a FlexPath-based NP architecture exhibits an IPP1(b) behavior comparable to the analytically derived curve as depicted in figure 3.
3. Prove that the FlexPath-specific building blocks can be implemented for high data rates (up to 2 Gbit/s throughput in FPGA technology).

Although our FlexPath_1 NP demonstrator will not achieve a system-level packet throughput corresponding to 2 Gbit/s (a single 405 PowerPC core and the 64-bit / 100 MHz PLB bus capacity in the FPGA are not adequate for such a performance), we want to demonstrate that the Pre-/Post-Processor and Path Dispatcher units can be designed and developed for high speed rates in FPGA and standard cell CMOS technologies and thus will not become a potential NP throughput bottleneck. With a 32 bit data path, 2 Gbit/s translates into a system clock rate of 62.5 MHz for the FlexPath units.

The Pre-Processor has been designed and implemented in VHDL and covers the following functions: packet header validity checking, source, destination address extraction, protocol stack determination, ARP, ICMP and TCP/ACK packet flagging. The Pre-Processor aligns all pertinent header fields onto CPU word boundaries and builds the packet context. The outcome of logic synthesis is that the Pre-Processor consumes a total of 588 (of 13,696 available) slices using 715 flip-flops and 993 4-input LUTs, which is roughly 4.3% of the FPGA's logic resources. The Post-Processor and Path dispatcher have not been designed yet. If we assume in a first order estimation that the Post-Processor is twice as complex as the Pre-Processor (due to obligation to not only create but also interpret packet contexts) and the Path Dispatcher dominantly consists of rule tables mapped on FPGA internal block RAMs (each block RAM is 18 kbit in size and there are a total of 136 such block RAMs in the Virtex-II Pro 30), the overall complexity introduced with the FlexPath concept will be bounded by around 15% of the FPGA's capacity.

6. CONCLUSIONS AND OUTLOOK

A new network processor concept and corresponding RISC-based platform SoC architecture have been introduced. Analytical performance estimations show that substantial gains in NP performance can be achieved by a flexible NP function traversal on an appropriately balanced hardware and software architecture.

The obtained results for performance analysis and prototype implementation of a FlexPath architecture support our hypothesis that conventional embedded RISC processor cores are eligible for network processing applications when supported by Pre-/Post-Processor hardware assists which adopt functions, e.g. bit-level manipulations, that general purpose CPUs are not optimized for.

Following a standard platform-based SoC design approach for NP development has the inherent advantage of accessibility to a huge portfolio of existing IP core libraries and a corresponding software development and debugging tools suite. Nevertheless, the presented concept may also be considered in conjunction with optimized NP ASIP cores, which would as well benefit from the flexible paths, possibly increasing the overall NP throughput.

Future work comprises: Completing the Post-Processor and Path Dispatcher designs and implementations, full system integration and testing in a real network environment and investigating additional scenarios and applications that will benefit from the AutoRoute concept. Of particular interest are applications which require dynamic function path reconfiguration during run-time. For example, packets flow through the CPU during a connection setup phase, the packet flow is then redirected to an optimized function path involving one or several co-processors and has to be switched back to the CPU in case of exception identification or proper packet flow termination. All path modifications are initiated and supervised by a central control processor, while the time critical packet-by-packet forwarding happens under Path Dispatcher control without packet loss or sequence corruption.

7. ACKNOWLEDGEMENTS

We would like to thank the German Research Foundation (DFG) for supporting the FlexPath project and the Xilinx University Partnership (XUP) program for donating the FPGA development board, ISE/EDK tools license and access to Core Alliance macros.

8. REFERENCES

[1] Kumar, V. P., Lakshman, T.V., Stiliadis, D.: "Beyond best effort: Router architectures for the differentiated services of tomorrow's internet", IEEE Communications Magazine, vol. 36, no. 5, pp. 152-164, May 1998

[2] Blake, Black, Carlson, Davies, Wang, Weiss: "An Architecture for Differentiated Service", RFC 2475, December 1998

[3] Ying, Q., Zhigang, Z., Biswas, J.: "Programmable Security Devices for the Network Edge – IP Security on a Network Processor", ICACT2002, International Conference on Advanced Communications Technologies, pp. 873-880, 2002

[4] Subbiah, B., Raivio, Y.: "Transport architecture evolution in UMTS/IMT-2000 cellular networks", International Journal of Communication Systems, Vol. 13, issue 5, pp. 371-385, August 2000

[5] Zeadally, S., Siddiqui, F., Kubher, P.: "Voice over IP in intranet and Internet environments", IEE Proceedings Communications, vol. 151, issue 3, pp. 263-269, June 25th, 2004

[6] Shah, N.: "Understanding Network Processors" – In: Berkeley Technical Report, September 2001

[7] Lawton, G.: "Will Network Processor Units Live up to their Promise?", IEEE Comp. Magazine, pp. 13-15, April 2004

[8] Kulkarni, C., Gries, M., Sauer, C., Keutzer, K.: "Programming Challenges in Network Processor Deployment", Int. Conference on Compilers, Architecture and Synthesis for Embedded Systems (CASES), October 2003

[9] Allen Jr., J. R., et al: "IBM PowerNP network processor: Hardware, software and applications", IBM Journal of R&D, vol. 47, no. 2/3, March/May 2003

[10] Spirent PPPoE AX/4000 Broadband (DSLAM) Test System, Press Release, http://www.spirentcom.com/news/press.cfm?id=965, Feb 18, 2003

[11] DSL Forum Technical Report TR-092, "Broadband Remote Access Server Requirements Document", August 2004

[12] Ramaswamy, R., Wolf, T.: "PacketBench: A Tool for Workload Characterization of Network Processing", IEEE 6th Annual Workshop on Workload Characterization (WWC-6), pp. 42-50, Austin, TX, October 2003

[13] Jenkins, C.: "NPU Co-Processors", Presentation at Network Processor Conference, San Jose, CA, August 2000

[14] Intel IXP1200 Network Processor Family, http://www.intel.com/design/network/prodbrf/27904001.pdf

[15] IBM PowerPC 440 Product Brief, March 24, 2004, http://www-306.ibm.com/chips/techlib

[16] IP Monitoring Project, http://ipmon.sprint.com, Traffic Profile gathered on February 6th, 2004 at the San Jose (sj-25) OC-48 (2.5 Gbit/s) link

[17] ML310 Embedded Development Platform, Xilinx development boards, http://www.xilinx.com/ml310/