Prototyping RISC Based, Reconfigurable Networking Applications in Open Source

Jong Hun Han*, Noa Zilberman*, Bjoern A. Zeeb*, Andreas Fiessler†, Andrew W. Moore*
*University of Cambridge Email: *firstname.lastname*@cl.cam.ac.uk
†genua GmbH Email: *andreas*@leyanda.de

***Abstract*—In the last decade we have witnessed a rapid growth in data center systems, requiring new and highly complex networking devices. The need to refresh networking infrastructure whenever new protocols or functions are introduced, and the increasing costs that this entails, are of a concern to all data center providers. New generations of Systems on Chip (SoC), integrating microprocessors and higher bandwidth interfaces, are an emerging solution to this problem. These devices permit entirely new systems and architectures that can obviate the replacement of existing networking devices while enabling seamless functionality change. In this work, we explore open source, RISC based, SoC architectures with high performance networking capabilities. The prototype architectures are implemented on the NetFPGA-SUME platform. Beyond details of the architecture, we also describe the hardware implementation and the porting of operating systems to the platform. The platform can be exploited for the development of practical networking appliances, and we provide use case examples.**

I. Introduction

Transformational technology [8], [13], combined with computing bottlenecks [27] has led to a rethinking of the hardware-software designs of modern computing systems. Undoubtedly, many of these changes had been driven by the growth in networking data created and consumed by user applications. The increasing amount of networking data also leads architects to focus on data-centric computational architectures able to meet the demand of high I/O performance. Systems-on-Chip (SoC) integrating high bandwidth network interfaces also emerge on high performance server systems [11], [15].

In this paper, we explore architectures embedding RISC based processors on the NetFPGA-SUME platform [5]. RISC processors are dominant in the mobile and portable electronics appliances markets due to their high power efficiency [10]. While building a high-performance server based on 64-bit RISC CPUs, embedded within a SoC alongside high bandwidth integrated networking and I/O interfaces are also available [20], [16], While the computing performance of x86 based servers is still better, it is believed that RISC based servers will eventually prevail thanks to their superior power efficiency [19].

Soft processors provide a high degree of freedom for both the processor design and the architecture at a system level. While free open-source approaches, in contrast to commercial RISC processors (e.g. ARM-based), have an appealing low-bar to entry. Extensibility and cost have lead to open-source CPU projects [7], [4], [1] being increasingly attractive to users and developers in both academia and industry.

The configurability available in commercial processors embedded in FPGAs (e.g. Xilinx's ARM-based Zynq, Altera's Nios) is not sufficient to enable the introduction of new processor-level functionality or for tailoring performance. In contrast, flexibility can almost verge on excessive with open source processor projects varying significantly: from implementations of an instruction set architecture (ISA), through to a soft-core processor, to a full SoC implementation. While many of these open source projects provide tool chains for hardware and software implementations, a considerable engineering effort is still required to build a system on a newly customized platform. Even when hardware is already implemented, porting software to a platform is far from being a trivial task.

Two open source RISC processor implementations are demonstrated on the NetFPGA-SUME platform: RISC-V [22], [4] and BERI [23]. In each case a significant amount of software operations is executed, including, but not limited to device-drivers, operating-system ports, and a range of application programmes.

A high performance networking platform with an embedded processor adventures to both industry and academic research. As an example, a network switch is a key building block in network systems. In the past, the design of a switch was focused on the forwarding engine. However, network architects are pursuing an increasing level of flexibility without comprising performance. An embedded processor within the switch enables fully fledge control and adoption of a switch functionality in field, such as the installation of new protocol stacks on the fly. However, multi-core processor cannot replace the FPGA, as FPGAs provide higher data plane performance [28]. Although FPGA devices have inherent limitations to achieving high performance compared to ASICs, they allow implementation of network fabrics capable of more than 100 Gbps (e.g. [25]). Additionally, FPGAs allow verification of line-rate hardware implementations of system functions without incurring tremendous tape-out costs on state of the art silicon technology.

The contribution of this work is several-fold: (i) We

demonstrate the development of fully-featured network-systems, based on an open source RISC based CPU, implemented as an FPGA-based prototype. (ii) We compare and contrast two different open-source processor architectures as the processing cores for the networking systems and (iii) we demonstrate porting and running standard operating systems on top of these fully programmable systems. An integrated open source platform such as this can provide a natural evaluation environment for networking systems as an enabler for their rapid implementation. (iv) We make the infrastructure available as an open source contribution to the NetFPGA ecosystem.

II. Motivation

The NetFPGA-SoC (NetSoC) is a platform for the research community that enables research into current and future approaches to network-based systems. To understand the challenge the platform tackles, consider the following use model:

New network stacks and operating systems for datacenter and cloud computing (e.g. [9], [11], [17]) seek high throughput, low latency and increased application performance. These works take advantage of advances in networking, such as network hardware functionality, flexibility from software defined networks, and novel network fabric topology. However, all these solutions are limited by the architecture imposed by their commodity CPUs. Examples of such limitations include limited support for resource allocation and isolation in the CPU's hardware for I/O or interconnect.

To study possible solutions to application limitations, a demand is created for platforms that support modifying and extending CPU/systems architectures. This means closed commodity CPU based systems are not suitable. Furthermore, using FPGAs with embedded CPUs makes it hard to port any successful solution to other platforms or adapt it for commercial ASIC use. This calls for an open source CPU based platform.

The CPU is only one part of the equation, as extensive networking knowledge is required to create high performance networking platforms. Processors and networking devices have different performance metrics: IPC vs. packets per second, throughput vs. bandwidth and latency. The work on NetSoC brought together researchers from both practices, and through close integration bridged the gap between the fields.

NetSoC is tailored for scalability studies: if, e.g., there is a $10\times$ frequency ratio between an FPGA and silicon-based CPU, then a design that supports 10GbE on an FPGA will support 100GbE using an ASIC with an embedded CPU. NetSoC can be used to analysis bottlenecks by reducing only the CPU and data path pipeline frequency without scaling down the DRAM frequency. This changes the relative rates and replicates the impact of improved performance RAM relative to a stable speed of CPU and data-pipeline. In turn, this can assist in detecting hidden architecture constraints, or limitations mistaken to be the memory wall.

TABLE I: Summary of open source RISC processors.

	BERI [23], [24]	RISC-V [22], [7]
ISA	MIPS R4000	RISC-V ISA
OS	FreeBSD	Linux
Design	Bluespec SystemVerilog (BSV)	Chisel
FPGA Speed	120MHz	50MHz

In NetSoC, we support two types of RISC CPUs: BERI and RISC-V (described in section III). Each of the CPUs supports a different operating system: FreeBSD and Linux, respectively. This achieves three goals: making both the BERI and RISC-V architectures available to networking researchers, making both FreeBSD and Linux available to networking researchers focused on operating systems and network stack design, and enabling comparative performance studies of RISC-V and BERI over the same hardware platform.

III. System Architecture

Our networked system is prototyped over NetFPGA-SUME [26], a low-cost, PCIe host adapter card able to support 40 Gbps and 100 Gbps applications. Two open source RISC processors known as BERI [23] and RISC-V [7], summarized in Table I, are implemented respectively on the NetFPGA-SUME platform to explore their performance and costs.

We use an identical system-level architecture for both RISC processors, as illustrated in Figure 1. All peripherals have address mapped registers, exposed to the processors as well as an external host (e.g. PC) via a PCI-Express (PCIe) interface. In addition to the processors, the platform integrates multiple NetFPGA modules, such as the networking fabric, as well and Xilinx peripheral IPs (e.g. DDR memory, 10G port). In the following subsections, we provide a detailed description of the integrated modules.

A. BERI and RISC-V

BERI [23] and RISC-V [7] are two diverse architectures. While BERI [23] is designed based on MIPS R4000 Instruction Set Architecture (ISA), RISC-V is created based on new open source ISA. In [24], BERI is running at a clock speed of 100 MHz, with a 256-bit wide memory interface and 16 KB L1 and 64 KB L2 caches. In this work, we use the BERI1 flavour of the processor, which is a mature and higher-performance variant for advanced research [23], and implement it using 120 MHz, clock frequency on NetFPGA-SUME. We integrate BERI modules generated by a Bluespec SystemVerilog (BSV) compiler into the architecture shown in Figure 1.

Unlike BERI, RISC-V is designed in Chisel [2], which is an open source hardware construction language that supports layered domain-specific hardware language. A RISC-V ASIC implementation runs at a clock speed of 1.3 GHz [14]. In this paper, we adopted the RISC-V Verilog core variant used in [4]. In Figure 1, the SPI controller used to access the SD

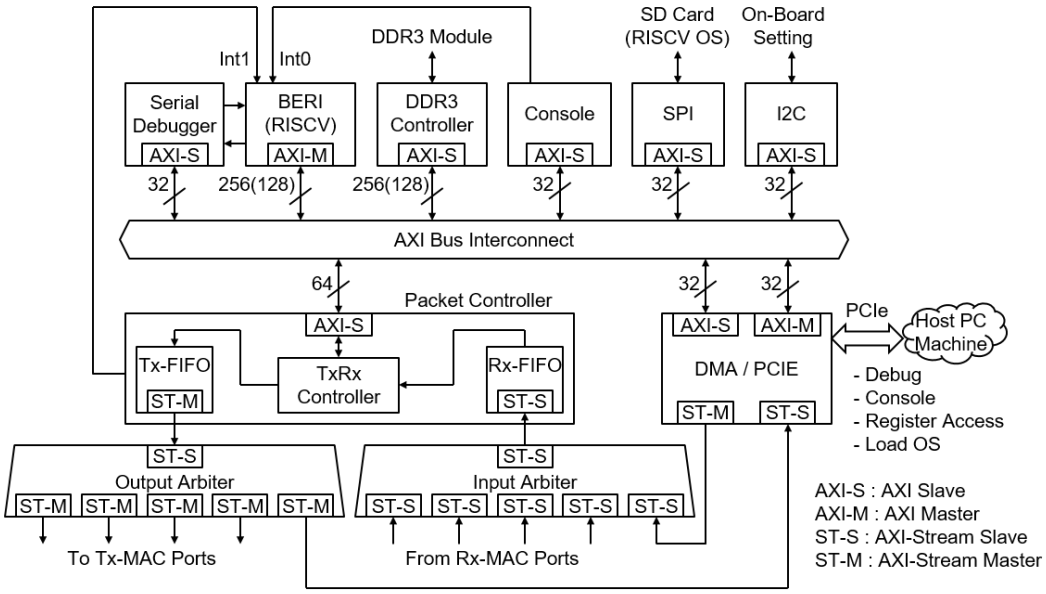

Fig. 1: RISC processor based FPGA platform architecture.

card storing a bootloader and a Linux kernel is used only by the RISC-V processor.

As a stand alone computing unit, NetSoC requires a large memory, from which the operating system and applications can be executed. To this end, we connect the processors to the external DDR3 SODIMM module running at 1866 MT/s. The density of each memory module is 4 GB and it can easily accommodate the small size of operating system we use. The external memory on the platform can be extended up to 32 GB and used also for other purpose.

B. SoC Interconnect

An AMBA AXI protocol is used across the design both for the *data-plane* (using an AXI-Stream protocol) and the *control-plane* (using AXI4 and AXI4-lite protocols). The AXI bus interconnect in Figure 1 has two master interfaces[1]: the processor(s) and the direct memory access (DMA) engine. The DMA engine is the communication module with the host machine, using a PCIe interface. Consequently, all the modules connected to the AXI interconnect can be accessed both by the processor and the host machine. Therefore, the memory and peripherals can be monitored and debugged on the host machine side.

The AXI-stream protocol is used for Ethernet packet data transactions across the architecture. Point-to-point transactions between AXI-stream master and slave (ST-M and ST-S, respectively) allow to easily handle bursts of data without compromising the line rate.

[1]A RISC-V core has two master interfaces separated into AXI4 and AXI4-lite for memory and peripheral accesses, respectively. One is omitted in Figure 1.

C. Networking Modules

The networking modules used in the architecture are different from the reference modules provided by NetFPGA-SUME. While there is some similarity in roles and functionality, the RISC embedded architecture requires a different implementation, as illustrated in Figure 1. The networking modules consist of four module types: 10 GbE ports, Input Arbiter (IAR), Output Arbiter (OAR) and Packet Controller (PAC). Unlike a PC handling high bandwidth networking through PCIe a board, SoC can enable tightly binding CPU, MMU, and networking interface to improve performance in terms of data rate and latency.

The 10 GbE port includes basic Ethernet layers one and two functionality. Every 10 GbE port module (omitted from Figure 1), is a combination of an incoming port and an outgoing port, which create a single physical port. Meta data indicating packet length and source port information is appended to each packet within this module as well.

IAR module arbitrates in a round robin manner between all the 10 GbE input ports to the device, whereas OAR arbitrates between all output ports while sending outgoing packets. Both IAR and OAR use an AXI-Stream interface to connect to the 10 GbE port through the MAC core.

PAC serves two roles: first, it converts all AXI-Stream transactions from the network to AXI-Lite transactions towards the AXI interconnect, and second it controls all the transactions from the network to RISC and the other way around. Any incoming packet arriving from the input arbiter triggers an interrupt to the CPU, which in turn reads the packet from the PAC (through the AXI interconnect). Outgoing packets, from the processor to the network, also pass through the PAC to the Output Arbiter, an from there to the 10 GbE ports.

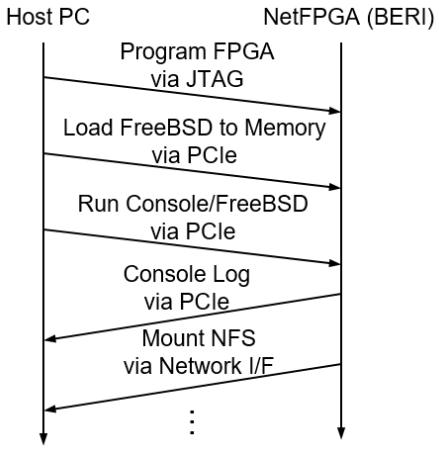

Fig. 2: BERI FPGA programming and boot sequences.

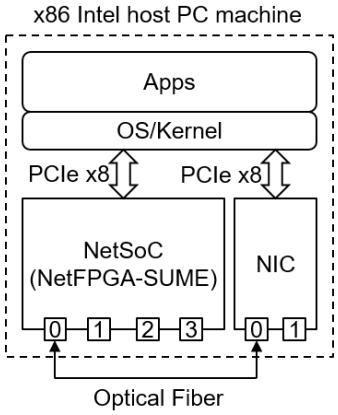

Fig. 3: Experimental setup for the evaluation.

In NetSoC, no packet processing is done by the networking modules, as this functionality is executed by the kernel driver running on the processor. This feature enables high programmability in the NetSoC platform, as any protocol or functionality can be programmed into the processor per use case.

D. Debugger and Console Modules

BERI processor operations can be traced and debugged using a built-in debugging unit [23]. The debugging unit traces the processor operations and provides access to local registers, interfacing with the serial debugger in Figure 1 by sending and receiving serial byte data. The serial debugger, connected to the AXI interconnect, is driven by a host PC through the PCIe interface. Using the serial debugger, the processor can be paused, resumed and transfer a dump of the operated instructions for debug purposes. It is also used to trigger the boot process of the BERI core after loading the FreeBSD kernel into the DDR3 memory.

A console module is designed to emulate a serial interface peripheral (e.g UART). It is mapped onto the same address range as the BERI serial interface implemented in [12]. The console module can be used as a terminal of the BERI operating system.

E. Operating System and Kernel Network Driver

Both processor implementations are running standard operating systems: FreeBSD for BERI, and Linux for RISC-V.

While Linux for the RISC-V is used "as is" from [7], with FreeBSD for BERI the process was different. We started with the BERI infrastructure described in [12]. Our porting process followed the adjustments common to bring-up of a new embedded or SoC target device using an already supported CPU. We had to update the device tree source, describing the hardware available, and the boot loader, for the memory layout and the peripherals. A new kernel configuration file was added including BERI specific peripherals and setting platform specific options. In addition to the aforementioned console modules, a network interface driver was written for interconnecting with the platform, which is already publicly available in [6]. One advantage of the FreeBSD operating system, is its existing support of on the fly installation of network protocols [21], which largely advantages networking appliances. Without a DMA engine on the BERI side, the kernel network driver implements a classic register-based programmed I/O (PIO) interface talking to an input and an output FIFO. The evaluation results in Section IV are obtained using this driver.

Due to differences in peripherals and drivers available in the OSes for BERI and RISC-V architectures, their initialization sequences are designed differently. While RISC-V loads an image of the Linux kernel from the SD card, BERI requires several steps to load its kernel, without using the on-board storage. Figure 2 illustrates the BERI initialization sequence during the evaluation (Section IV), from programming the FPGA to mounting the network file system (NFS). The NFS is set between BERI and the host over the 10 GbE ports shown in Figure 3.

After the OS kernel is loaded, any application can run on the target processor, as with any host running FreeBSD or Linux. An immediate example is using the network driver running on FreeBSD to configure and enable the BERI network interfaces. Other examples are using *scp* and *vi*.

IV. Evaluation

We implemented the presented BERI and RISC-V architectures for a Virtex-7 690T FPGA used on NetFPGA-SUME[2], using Xilinx Vivado 2014.4 EDK, following the conventional Xilinx tool chain design flow. The BERI and RISC-V based platforms run at 120 MHz and 50 MHz, respectively. Although a single-core implementation is used to port the OS kernels, we also investigate the number of BERI and RISC-V cores that can be simultaneously instantiated on the platform.

[2]This work will be available as a contributed project for the NetFPGA-SUME from the NetFPGA (http://www.netfpga.org) project.

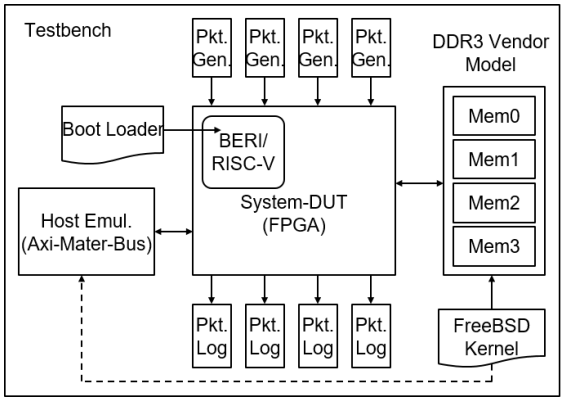

Fig. 4: RTL level simulation environment for verification.

A. Simulation Environment

Figure 4 presents the simulation environment used to verify the architecture at system level. We conducted a RTL level simulation, using the same 32 MB FreeBSD kernel used in the booting process of the actual hardware. The Design-Under-Test (DUT) is the top level module implemented on the FPGA. To simulate interaction with the external memory, the memory controller is connected to a DDR3 SODIMM memory model [3], which represents the same memory as it is used on the SUME board.

While the DRAM memory is initialized in the hardware by the host via the AXI bus interface, the initialization is not practical in simulation due to the duration of the simulation processing time. Instead, the memory model is initialized with the kernel by using a dedicated testbench task. The task converts the data to a required format, splits the kernel to files and distributes it as required. In this manner, we can observe in the simulation the entire core initialization sequence, from BERI entering the kernel to logging data out into the console module. The RTL level simulation was performed using the Mentor Questa Sim 10.4 simulator.

The simulation environment for the RISC-V architecture is similar to the Figure 4. In this case, we verified the sequence until the boot loader accesses the SD card via the SPI controller, and the Linux kernel is copied into the memory. The simulation environment is also available as open-source as part of the released project.

B. Multi-Core Implementation Results

Many-core implementations can benefit a multitude of applications. We explore multi-core implementations of both processor architectures, and evaluate their resource usage and scalability. Clock speeds of 120 MHz and 50 MHz are used for the single- and multi-core BERI and RISC-V implementations, respectively. We find that four (quad) BERI cores and eight (octa) RISC-V cores can be implemented on the platform at a time. Table II summarizes the implementation cost of BERI and RISC-V, showing the amount of resources taken by the RISC cores, AXI Interconnect (Inter), Peripheral and Networking modules (Peri) and the remaining unused resources (Unused). The RISC-V implementation is significantly smaller than BERI; this allows twice the number of cores on the same platform. Differences in the architecture of every processor affects the surrounding peripherals and interconnects, leading to a difference in the number of LUT and FF. In addition, AXI-interconnect (Inter) is affected by the number of cores due to the increased number of the cores' master interfaces.

TABLE II: Comparison of FPGA implementation cost.

	Single-BERI LUT/FF	Quad-BERI LUT/FF	Single-RISC-V LUT/FF	Octa-RISC-V LUT/FF
RISC(s)	72.1K/29.8K	289K/118K	40.8K/16.7K	326K/133K
(%)	16.6%/3.4%	66.7%/13.6%	9.4%/1.9%	75.3%/15.4%
Inter	16.9K/18.7K	21.0K/19.0K	6.1K/6.2K	16.9K/13.4K
(%)	3.9%/2.1%	4.9%/21.9%	1.4%/0.7%	3.9%/1.6%
Peri	47.8K/47.2K	47.5K/47.3K	35.8K/35.4K	35.6K/35.4K
(%)	11.0%/5.4%	10.9%/5.4%	8.2%/4.1%	8.2%/4.1%
Unused	296K/770K	75.7K/681K	350K/808K	54.1K/683K
(%)	68.4%/88.9%	17.4%/78.7%	80.8%/93.2%	12.5%/78.9%

This experiment provides two important insights: First, when a computing-intensive networking application is required, increasing the number of cores in order to achieve better compute performance is possible. Second, when a networked data-intensive application is required, a single-core implementation has plenty of resources (70%-80%) available to implement networking and data processing modules, increasing the bandwidth performance.

C. Example: Network Stack Evaluation on BERI FreeBSD

The most likely use case of NetSoC is closely integrating new CPU features with the network. Our experiment therefore uses the BERI processor and network utilities running on FreeBSD, with the goal of showing that NetSoC performance scales comparably with CPU frequency. Figure 3 shows an experimental setup for the performance evaluation. The PC machine used for the experiment ran an octa-core i7-960 CPU, with 24 GB of RAM, equipped with NetFPGA-SUME and a dual port SolarFlare 10G NIC card. As illustrated in Figure 3, NetSoC and the host PC can communicate through the 10 GbE ports over an optical fibre connection. In the experiment, we evaluated how BERI core clock frequencies affect the `ping` packet latency between NetSoC and PC machine, as shown in Figure 5 where "PC-NICs" is the latency between identical PC machines using the identical 10 GbE NIC cards. "HW" refers to a FPGA-based hard-coded implementation of `ICMP echo reply`, implemented by the NAAS-Emu project over NetFPGA-SUME[3], with a known latency of 1.27us and a jitter of 100ns. The latency marked as "HW" is the ping round trip latency between the PC and this implementation. It provides us a reference for the latency caused by the PC, fiber and 10 GbE ports.

[3]Included in NetFPGA SUME release 1.4.0

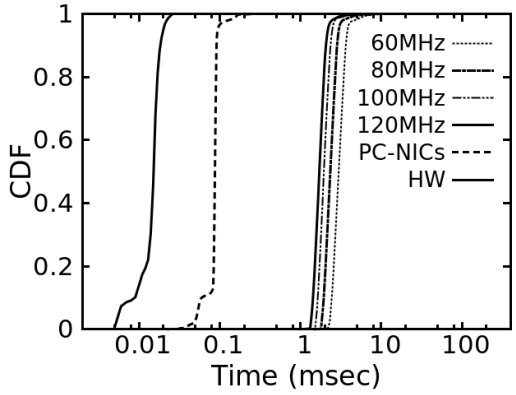

Fig. 5: CDF of ping latency result.

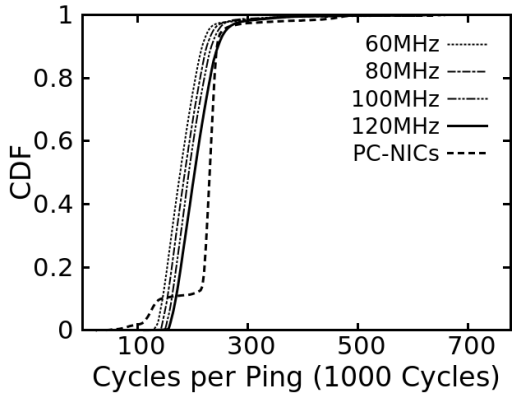

Fig. 6: CDF of cycles per ping, in thousands of cycles.

As we consider the scaling of frequency from FPGA to ASIC, we compare the latency of NetSoC with the latency of PC-NICs in CPU cycles. To improve the accuracy, we only consider the NetSoC latency, and subtract the median accumulated latency of the requesting PC, fiber and 10 GbE port latency, represented by "HW" (15 us). The results are presented in Figure 6: considering the median, there is less than 15% difference in cycles between NetSoC default frequency (120MHz) and minimal tested frequency (60MHz), and there is less than 20% difference in cycles between PC-NICs and NetSoC running at 120MHz. The higher cycles count of the host is contributed partly to unaccounted contributors, such as the PCIe interconnect.

We also evaluated TCP and UDP performance on NetSoC using `iperf`. The application uses packet processing modules with a simple register-based PIO and achieves only 6.9 Mbps for both TCP and UDP. These results are comparable with FreeBSD running `netsend` results shown in [18]. Similarly, we expect to achieve a much higher performance by using userspace applications for packet generation. Beyond those the achievable data rate can be significantly improved by enhancing the packet processing module in the networking part of the design, adding features such as packet DMA to the local RAM, checksum validation, segmentation offload and more, which are under development for our architecture.

V. Conclusions and Future work

We presented an open source, RISC based, SoC architectures for networking applications implemented on the NetFPGA-SUME platform. The system was tested using common user applications running on FreeBSD and Linux operating systems. We showed that the integrated RISC processor is a feasible solution for networking appliances and that scalable CPU designs can take leverage of NetSoC. Furthermore, the flexibility and programmability provided by the platform open new directions for networking research.

Although the frequency of FPGA devices is limited in comparison with ASICs, their parallel implementation allows evaluating scalable prototypes and high data rate networking fabrics close to practical systems. The processors and peripherals presented in this paper require further optimization through hardware and software co-designing. The *data-* and *control-plane* designs in the architecture can be considered relatively independent, and we intend to explore reciprocation within the integrated architectures. We plan to further extend our work, developing novel networking fabric solutions as proposed in [27].

VI. Acknowledgements

We would like to thank the many people who contributed to this paper. We would like to thank Salvator Galea, from the EPSRC NAAS project (EP/K034723/1), who contributed the ICMP echo reply hardware implementation. This work was jointly supported by the European Union's Horizon 2020 research and innovation programme 2014-2018 under the SSICLOPS (grant agreement No. 644866), ENDEAVOUR (grant agreement No. 644960), the Leverhulme Trust Early Career Fellowship ECF-2016-289, the Defense Advanced Research Projects Agency (DARPA) and the Air Force Research Laboratory (AFRL), under contract FA8750-11-C-0249. The views, opinions, and/or findings contained in this article/presentation are those of the author/ presenter and should not be interpreted as representing the official views or policies, either expressed or implied, of the Department of Defense or the U.S. Government.

References

[1] BERI. http://www.beri-cpu.org.
[2] Chisel. http://github.com/ucb-bar/chisel.
[3] DDR3 Verilog Model. http://www.micron.com/parts/dream/ddr3-sdram/mt41k512m4hx-15e.
[4] LowRISC. https://www.lowrisc.org.
[5] NetFPGA-SUME-live. https://github.com/NetFPGA.
[6] NF-MAC-DRIVER. https:/github.com/freebsd/freebsd/blob/master/sys/dev/netfpga10g/nf10bmac/if_nf10bmac.c.
[7] RISC-V. https://riscv.org.
[8] K. Bailey, L. C. Steven, D. Gribble, and H. M. Levy. Operating system implications of fast, cheap, non-volatile memory. In *13th USENIX conference on Hot topics in operating systems*, 2011.
[9] A. Belay, G. Prekas, A. Klimovic, S. Grossman, C. Kozyrakis, and E. Bugnion. IX: A protected dataplane operating system for high throughput and low latency. In *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*, pages 49–65. USENIX Association, Oct. 2014.

[10] E. Blem, J. Menon, and K. Sankaralingam. Power struggles: Revisiting the risc vs. cisc debate on contemporary arm and x86 architectures. In *IEEE High-Performance Computer Architecture (HPCA)*, 2013.

[11] P. Costa, H. Ballani, K. Razavi, and I. Kash. R2c2: a network stack for rack-scale computers. In *ACM SIGCOMM Computer Communication Review*, volume 45, pages 551–564. ACM, 2015.

[12] B. Davis, R. Norton, J. Woodruff, and R. N. M. Watson. How freebsd boots: a soft-core mips perspective. In *AsiaBSDCon*, 2014.

[13] K. Keeton. The machine: An architecture for memory-centric computint. Workshop on Runtime and Operating Systems for Supercomputers (ROSS), 2015.

[14] Y. Lee, A. Waterman, R. Avizienis, H. Cook, C. Sun, and V. Stojanovic. A 45nm 1.3GHz 16.7 Double-Precision GFLOPS/W RISC-V Processor with Vector Accelerators. In *IEEE European Solid State Circuits Conference (ESSCIRC)*, 2014.

[15] S. Legtchenko, N. Chen, D. Cletheroe, A. Rowstron, H. Williams, and X. Zhao. Xfabric: A reconfigurable in-rack network for rack-scale computers. *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*, 2016.

[16] S. Li, K. Lim, P. Faraboschi, J. Chang, P. Ranganathan, and N. P. Jouppi. System-level integrated server architectures for scale-out datacenters. In *IEEE/ACM Micro-44*, 2011.

[17] S. Peter, J. Li, I. Zhang, D. R. Ports, D. Woos, A. Krishnamurthy, T. Anderson, and T. Roscoe. Arrakis: The operating system is the control plane. *ACM Transactions on Computer Systems*, 33(4):11, 2016.

[18] L. Rizzo. netmap: a novel framwork for fast packet i/o. In *21st USENIC Security Symposium*, 2012.

[19] J. Shuja, K. Bilal, S. A. Madani, M. Othman, R. Ranjan, P. Balaji, and S. U. Khan. Survey of techniques and architectures for designing energy-efficient data centers. *IEEE Systems Journal*, 10(2):507–519, June 2016.

[20] C. Singh, G. Favor, and A. Yeung. Xgen-2:28nm scale-out processor. In *Hot Chips: A Symposium on High Performance Chips*, 2014.

[21] R. Stewart. *TCP Stack Modularity*. FreeBSD, 2016.

[22] A. Waterman, Y. Lee, D. Patterson, and K. Asanovic. The RISC-V Instruction Set Manual, Volume I: Base User-Level ISA. Technical Report UCB/EECS-2011-62, University of California, Berkeley, May 2011.

[23] R. N. M. Watson, J. Woodruff, D. Chisnall, and Others. Bluespec Extensible RISC Implementation: BERI Hardware reference. Technical Report UCAM-CL-TR-868, University of Cambridge, Computer Laboratory, Apr. 2015.

[24] J. Woodruff, A. T. Markettos, and S. W. Moore. A 64-bit MIPS processor running freebsd on a portable FPGA tablet. In *Internation Conference on Field-programmable Logic and Applications*, 2013.

[25] Xilinx. *"Xilinx Highlights All Programmable Solutions for 400GE Applications at WDM Nice 2014"*. http://press.xilinx.com/2014-06-17-Xilinx-Highlights-All-Programmable-Solutions-for-400GE-Applications-at-WDM-Nice-2014, [Online].

[26] N. Zilberman, Y. Audzevich, G. Covington, and A. W. Moore. NetFPGA SUME: Toward 100 Gbps as Research Commodity. *IEEE Micro*, 34:32–41, Sept. 2014.

[27] N. Zilberman, A. W. Moore, and J. A. Crowcroft. From Photons to Big Data Applications: Terminating Terabits. *Royal Society Philosophical Transactions A*, 2016.

[28] N. Zilberman, P. M. Watts, C. Rotsos, and A. W. Moore. Reconfigurable network systems and software-defined networking. *Proceedings of the IEEE*, 103(7):1102–1124, 2015.