A Scalable Multi-FPGA Platform for Complex Networking Applications

Sascha Mühlbach
Secure Things Group
Center for Advanced Security
Research Darmstadt (CASED)
sascha.muehlbach@cased.de

Andreas Koch
Embedded Systems and Applications
Dept. of Computer Science
Technische Universität Darmstadt
koch@esa.cs.tu-darmstadt.de

Abstract—**Ballooning traffic volumes and increasing link-speeds require ever high compute power to perform complex real-time processing of network packets. FPGAs have already been successfully employed in the past to accelerate network infrastructure-operations at these line-speed processing rates. However, much of the prior work concentrated on single-FPGA platforms. To this end, we have studied how to extend an architecture for 10G application-level network processing into a scalable multi-device system. We present a ring-based approach, of which a quad-FPGA implementation will be evaluated on the BEEcube BEE3 computing platform.**

I. Introduction

With the great success of the Internet, network devices have become widespread in both business and personal settings, and their number has grown dramatically in the past years. New services such real-time video and audio and cloud-based computing and storage, combined with the increasing number of users, have lead to ballooning traffic volumes and more tighter quality-of-service requirements.

Since the required performance (e.g., throughput and latency) has proven difficult to achieve by software running on conventional processors, a common solution is to offload operations to dedicated hardware accelerators. This approach has been realized using both programmable fixed-architectures (e.g., network processors, TCP offload engines), as well as reconfigurable architectures (often FPGA-based).

Motivated by advances in FPGA technology and the availability of faster and larger devices, we will show that modern FPGAs have become sufficiently powerful to not only act as partial accelerators, but instead can be used to independently perform even application-level network operations. In case of extremely high performance requirements, as an alternative to using a very large, expensive device, this work demonstrates the scalability of multi-chip implementations composed from smaller, more economical FPGAs. We will employ the platform for application-level network processing to realize the use-case of a honeynet-in-a-box for malware (attack code) collection [1].

The work is organized as follows: After briefly surveying related work, the base architecture and its components are described in Section II. Section III presents specific implementation aspects, followed by evaluation results on the BEEcube BEE3 platform in Section IV. Finally, we close with a conclusion and an outlook towards further research in the last Section.

A. Related Work

Popular research projects for FPGA-based networking are NetFPGA [2] and DynaCORE [3]. In contrast to these often packet-oriented approaches, our own research has always been aiming at higher-level Internet (e.g., TCP, UDP) and application protocol communication. To this end, we created a novel platform [4] optimized for this area. Similar to NetFPGA, our platform also provides a flexible framework for "plugging-in" processing elements (called *Handlers*) for different purposes. It is not organized as a simple linear datapath, though, but also follows the *hierarchy* of the Internet protocol stack, with upper-level handlers building on the functionality of the lower-level ones.

Top-level handlers realize application protocols such as HTTP or SMTP to the degree required by the different use-cases. Our system [4], which has been realized as a single-chip solution so far, is capable of sustained 10G processing and can actually run critical parts of the infrastructure in a 20G burst mode to quickly catch-up after handler-local stalls. This is achieved by pipeline- and task-level parallelism, as well as a flexible, but specialized interconnect scheme more efficient than the general NoC used, e.g., in DynaCORE.

In context of the network security domain, a purely hardware-based approach such as ours has the additional advantage, that no general-purpose software programmable processor is present that could be subverted if the honeypot itself is being attacked.

II. Architecture

As an alternative to the monolithic single-chip system, we now examine how the architecture can be implemented in a scalable manner distributed across multiple devices.

One way to partition the system is to draw the boundaries between the core network processing functionality (basic Ethernet and Internet protocol), which will remain static, and the application-level handlers, which will be more dynamic. Since all handlers will need to both receive and transmit

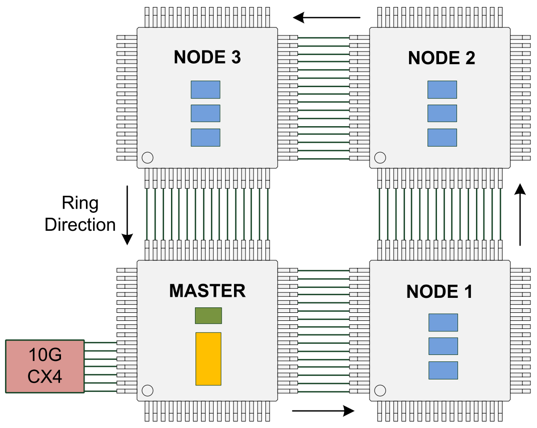

Figure 1. Multi-FPGA network processor in ring topology

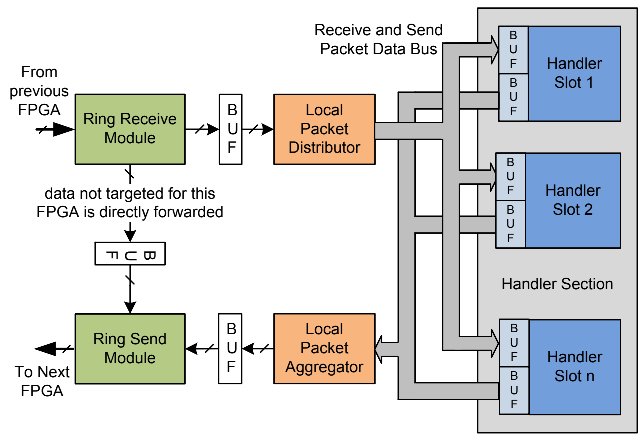

Figure 3. Handler node architecture

data, all of them must be able to communicate with the Ethernet endpoint. One way to achieve this with limited I/O pin requirements (our current constraint is the available connectivity on the BEE3 platform) is a ring structure. Figure 1 shows an example for such a topology, and also indicates how it could map to the four-chip BEE3 platform.

In this ring topology, the FPGA holding the network core and providing the external 10G interface will be referred to as the Master node. It also includes the management interface and controls the operation of the entire system (initialization and destination addressing). The Handlers will be placed into the other FPGAs, referred to as Handler nodes, with each node holding multiple handlers.

The communication ring is unidirectional, packets being forwarded from node to node until they reach their destination. This applies to both incoming as well as outgoing packets, they enter and exit the system at the Master node.

A. Master Node

The Master node has three major parts (see Figure 2): the network core (Fig. 2-a), the management section (Fig. 2-b), and the ring interface (Fig. 2-c). The network core consists of hardware modules, implementing all required protocols to enable autonomous communication in the Internet. Currently, this includes ARP, ICMP, IP, and UDP modules, as well as a specialized high-speed implementation of TCP. The network core is structured as an easily extensible hierarchical design following the actual protocol layers. For further details, please refer to [1], [4].

Header information of incoming network packets is removed inside the core, so that the Handlers have direct access to the application level packet data. Incoming header information required for further processing of the packet (e.g., to create appropriate response headers) is stored within a custom internal control header (ICH). We use this ICH to also carry the internal packet-to-handler routing information of a message on the ring. The addressing is based on the target FPGA and the ID of the Handler slot within that FPGA.

To support flexible configurations, the routing information is stored within a table (Fig. 2-d) that can be managed through the management interface (a network interface that can be accessed over a system-specific protocol we developed for that purpose). The routing table contains the assignment between sockets (packet destination Protocol, IP and Port information) and the corresponding target Handler (FPGA and Slot ID).

B. Handler Node

The Handler nodes (see Figure 3) contain the Handler sections with the different Handlers. All Handlers share the same message-based interface for the reception and transmission of packets. Each Handler has a separate buffer for incoming and outgoing messages. Together with the actual Handler logic, these buffers make up a “Handler slot”. This flexible structure also allows us to easily add partial reconfiguration capabilities to the architecture for replacing Handlers without the need to reconfigure the whole FPGA (which would interrupt the ring communication for short time intervals).

The ring receive module of each Handler Node checks the destination address field in the ICH. If the message is targeted for this FPGA, it is forwarded to the packet distributor module, which forwards the packet to the corresponding Handler slot buffer. If the message is not addressed to the current FPGA, it is directly put into the forward queue and sent to the next FPGA on the ring. The destination address is within the first word of a message, so that this decision can be made immediately when receiving the first bytes of a new packet on the ring.

Response packets of Handlers that need to be transmitted back to clients are put into the corresponding output buffer, where they are picked up by the packet aggregator. Both local data buses (for receiving and sending of messages)

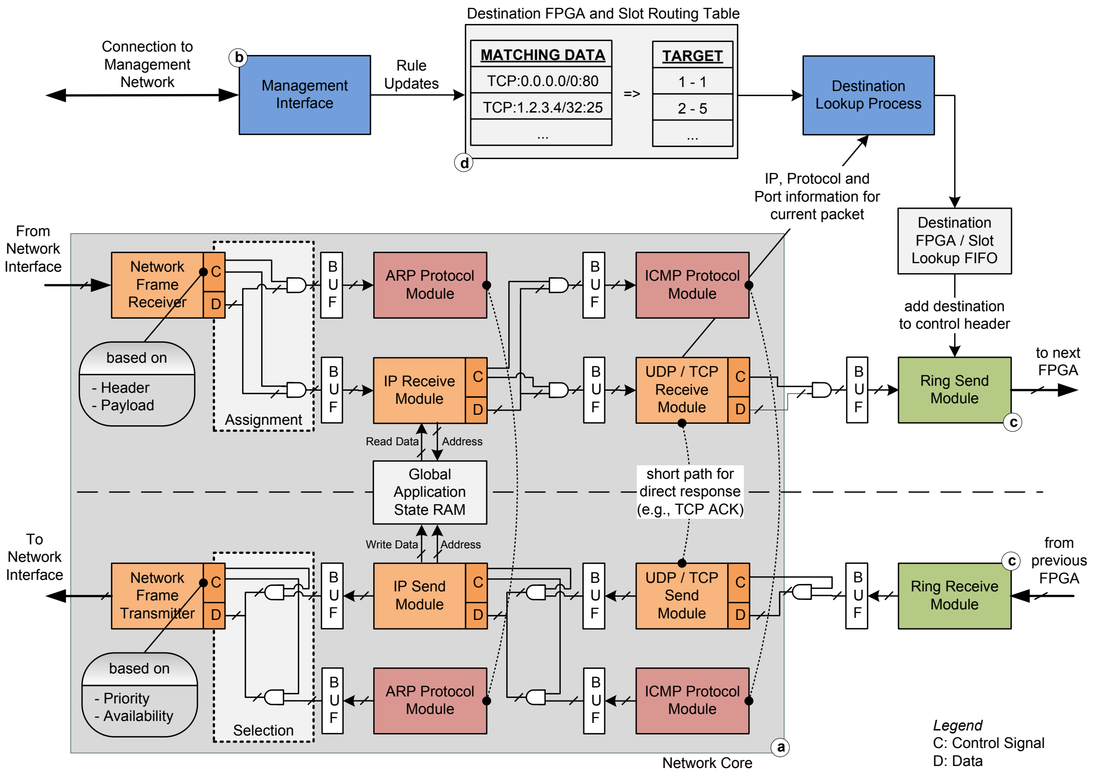

Figure 2. Master node architecture

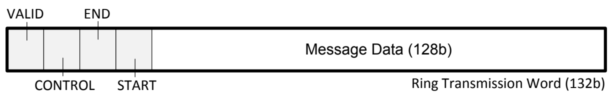

Figure 4. Structure of the ring data word

have a capacity of 16 Gb/s in the current implementation. Handlers can be therefore designed to support operation at 10 Gb/s or higher.

C. Ring Communication

For the communication on the ring we use 66 of the inter-FPGA data lines available on our BEE3. They are run in DDR mode, so that we can transmit 132 bits of data per clock cycle. 128 of them are used as data word for the message and four as status bits (see Figure 4). As data is sent continuously on the ring, a valid flag is used to label valid data words. Two further flags signal the first and the last word of a message. The fourth flag is reserved for future use to denote special ring control messages.

For reliably transmitting data on the 250 MHz inter-chip buses, we perform clock synchronization using [5]. During initialization, each node continuously sends a known training sequence to the next FPGA and this FPGA adjusts its delay until it receives stable data from the previous FPGA.

III. Implementation

The described architecture has been implemented on the BEE3 custom reconfigurable computing platform, which is in our case equipped with four Xilinx Virtex 5 FPGAs: 2x LX155T, 2x SX95T. As all the FPGAs have 10G connections, we decided to place the Master node inside one of the SX95Ts to have the larger LX155Ts available for Handlers.

For evaluating our platform under real conditions, we implemented two example handlers (a simple web- and mail-server emulation) that emulate certain network server functionalities. These emulations allow us to test the functionality and compatibility of our platform using standard networking tools.

The web server emulation contains a ROM with predefined HTML pages to be served to clients. For now, we do not support options such as HTTP keepalive, so after each response the TCP connection is closed. The mail server implements the basic SMTP commands and pretends to be an open relay server (however, it does not actually forward the mails anywhere). In contrast to the web server Handler, the responses of the mail server Handler are small (consisting only of the status code). Therefore by sending data to both Handlers, we can generate a good mixture of small and large packets on our ring. Additionally, the TCP connection of the SMTP handler remains open for multiple

Table I
SYNTHESIS RESULTS FOR MASTER NODE COMPONENTS

Module	**LUT**	**Reg. Bits**	**BRAM**	**Slices**
Network Core	7,138	5,269	68	
Ring Interface	556	1,190	8	
Management	3,823	2,674	19	
Mapped incl. MAC, XAUI and Clocks	**14,735**	**12,044**	**111**	**6,723**
in % of SX95T	25	20	45	45

Table II
SYNTHESIS RESULTS FOR HANDLER NODE COMPONENTS

Module	**LUT**	**Reg. Bits**	**BRAM**	**Slices**
Ring Interface	624	1,455	12	
Handler Section for 32 Slots (w/o Handler)	8,928	8,388	160	
Web Server Handler	842	330	0	
Mail Server Handler	742	363	0	
Mapped	**34,557**	**20,485**	**174**	**13,015**
in % of LX155T	35	21	82	53

request / response packets.

The design was synthesized and mapped using Xilinx ISE 12.4. Each Handler node was configured to include 16 instances of the two example Handlers mentioned above. As every node has a different address and the routing of packets to the Handler slots is configured dynamically, Handlers which are present but not active do not slow-down the system.

IV. RESULTS

The synthesis results for all components are given in Table I and II. For the Handler nodes we give results only for the LX155T, as the results for the SX95T are very similar (in terms of resource requirements). Together with the ring interface and the management functions, the master architecture occupies around 20% of the LUT and 39% of the BRAM resources. This still leaves enough free space to add further functionalities to the network core.

Inside the handler nodes, the logic required to implement the ring interface and the Handler section is negligible. This is not surprising, as the logic of these modules consists of simple state machines that forward messages without performing complex modifications. This leaves nearly the full FPGA available for the actual Handler implementations.

Taking these results, the total number of slots per FPGA is limited by the amount of available BRAMs. Currently, we could have 40 Handler slots per FPGA on both the LX155T and SX95T. Of course, the final number of Handlers is also affected by the sizes of the actual implementations and can vary between 1 and the upper limit.

We used the Ping utility as well as Firefox and Thunderbird to verify the functionality of the core platform and the implemented Handlers. Performance measurements have been done with a 1000B request packet that generates a 1000B response packet. The latency induced by the ring is only around $1.4\mu s$ when assuming empty buffers. In the average case, we expect this latency to be between $10-20\mu s$, depending on the current load.

V. CONCLUSION

With our multi-FPGA platform built around a high-speed hardware implementation of basic Internet communication protocols, we have presented a flexible and extendable environment for the implementation of even complex network applications.

In comparison to our single-chip implementation [4], the multi-FPGA approach is a great improvement of the total processing power of our platform. When using all three Handler node FPGAs completely, we now can have up to 120 Handlers "active" at the same time, which should suffice even for very complex scenarios. While the current implementation uses four FPGAs, the architecture itself supports systems with even more FPGAs acting as Handler nodes (they simply can be plugged-in into the ring). This allows the architecture to scale without the need to fit the entire application on a large and expensive single FPGA.

We will extend the capabilities of the network core by adding new functionalities such as IPv6 compatibility and HTTPS encryption support. Additionally, we will combine our multi-FPGA approach with dynamic partial reconfiguration capabilities into an integrated systems that offers a very high flexibility to the user.

REFERENCES

[1] S. Mühlbach, M. Brunner, C. Roblee, and A. Koch, "MalCoBox: Designing a 10 Gb/s Malware Collection Honeypot using Reconfigurable Technology," in *Proc. of the 20th International Conference on Field Programmable Logic and Applications*. IEEE Computer Society, 2010, pp. 592–595.

[2] J. W. Lockwood, N. McKeown, G. Watson, G. Gibb, P. Hartke, J. Naous, R. Raghuraman, and J. Luo, "NetFPGA–An Open Platform for Gigabit-Rate Network Switching and Routing," in *Proc. of the 2007 IEEE International Conference on Microelectronic Systems Education*, ser. MSE '07. IEEE Computer Society, 2007, pp. 160–161.

[3] C. Albrecht, R. Koch, and E. Maehle, "DynaCORE: A Dynamically Reconfigurable Coprocessor Architecture for Network Processors," in *Proc. of the 14th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*. IEEE Computer Society, 2006, pp. 101–108.

[4] S. Mühlbach and A. Koch, "An fpga-based scalable platform for high-speed malware collection in large ip networks," in *Proc. of the 2010 International Conference on Field Programmable Technology*. IEEE Computer Society, 2010, pp. 474–478.

[5] C. Thacker, "DDR2 SDRAM Controller for BEE3," Microsoft Research, 2008.