Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Institut für
Technische Informatik und
Kommunikationsnetze

Richard Huber

A Dynamic Hardware Architecture for Future Networks

Master Thesis MA-20XX-XX
December 2011 - June 2012

Advisor: Ariane Keller, ariane.keller@tik.ee.ethz.ch
Co-Advisor: Dr. Stephan Neuhaus, stephan.neuhaus@tik.ee.ethz.ch
Co-Advisor: Daniel Borkmann, daniel.borkmann@tik.ee.ethz.ch
Professor: Prof. Dr. Bernhard Plattner, plattner@tik.ee.ethz.ch

Contents

Chapter 1

Introduction

The increasing number and diversity of networking devices is challenging the network research community. More and more applications and protocols are developed to address the needs of different platforms and use cases. For performance reasons, these protocols and applications are often tightly bound to each other. The resulting systems may then perform well for a specific application, but lack of flexibility in the underlying architecture. This inflexibility is well demonstrated by the still ongoing switch from IPv4 to IPv6 in the TCP/IP protocol stack of the current internet.

The Autonomic Network Architecture (ANA) [1] targets this problem with a fundamental change of paradigms. In ANA, all functionality is encapsulated in so called functional blocks. The size of functional blocks is a priori not defined, it may range from a minimalistic checksum calculation engine up to a huge holistic protocol stack. ANA provides a set of abstraction models and generic communication methods that the functional blocks use to interact with each other. This framework allows a completely isolated development of the algorithms running in the functional blocks. The networking devices then interconnect the functional blocks and build up an individual protocol stack. The device chooses the functional blocks for this protocol stack depending on the capabilities of the platform and the requirements of the running applications. The result is a flexible and dynamic protocol stack, that is able to adapt itself to any given situation and may change at runtime. Figure 1.1 shows such a dynamic protocol stack and compares it to the strongly layered protocol stack of the current internet.

The level of indirection introduced by ANA trades some performance optimization features of the application for a higher flexibility and modularity of the overall system. In order to compensate this performance penalty, a hardware acceleration for ANA is desired. It is not feasible to use application specific integrated circuits (ASIC) for this purpose, since the functionality of these devices must be completely specified at design time. This static characteristic of ASICs would wipe out ANA's effort towards flexibility and adaptability.

The evolving partial reconfiguration capabilities of modern field pro-

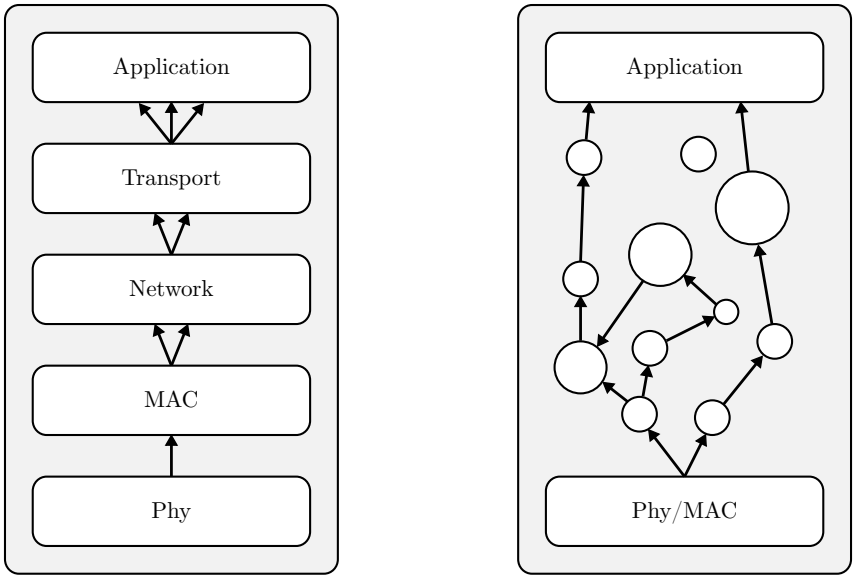

Figure 1.1: a) The strongly layered static protocol stack of the current internet, b) The dynamically adaptable protocol stack of an ANA node

grammable gate arrays (FPGA) combine the flexibility of a software implementation with the high-performance acceleration features of ASICs. [6] and [3] therefore came up with the idea of porting ANA to a reconfigurable system on chip (rSoC). On these devices, a microprocessor core (either soft- or hardcore) is combined with a programmable logic fabric. The software running in the microprocessor may then decide to shift the execution of computational intensive and frequently used functional blocks from the microprocessor to the programmable logic. Based on the partial reconfiguration features, this mapping of functional blocks to hardware or software can be adapted at runtime.

In order to reduce the engineering effort, [6] and [3] further suggest to use ReconOS [5] as base system. In ReconOS, partially reconfigurable logic slots are used as so called hardware threads. These hardware threads appear to the software like normal software threads. ReconOS enables the hardware threads to access operating system objects like semaphores, message boxes and shared memory. The combination of these mechanisms simplifies the interaction of software and hardware elements. The suggested approach is now to benefit from the hardware acceleration provided by ReconOS by implementing the functionality of some functional blocks as hardware threads. Figure 1.2 shows the architecture of ReconOS and how the functional blocks are mapped to the hardware threads.

The major problem of this approach is the shared bus architecture of ReconOS. Especially with a data intensive application like ANA the shared bus

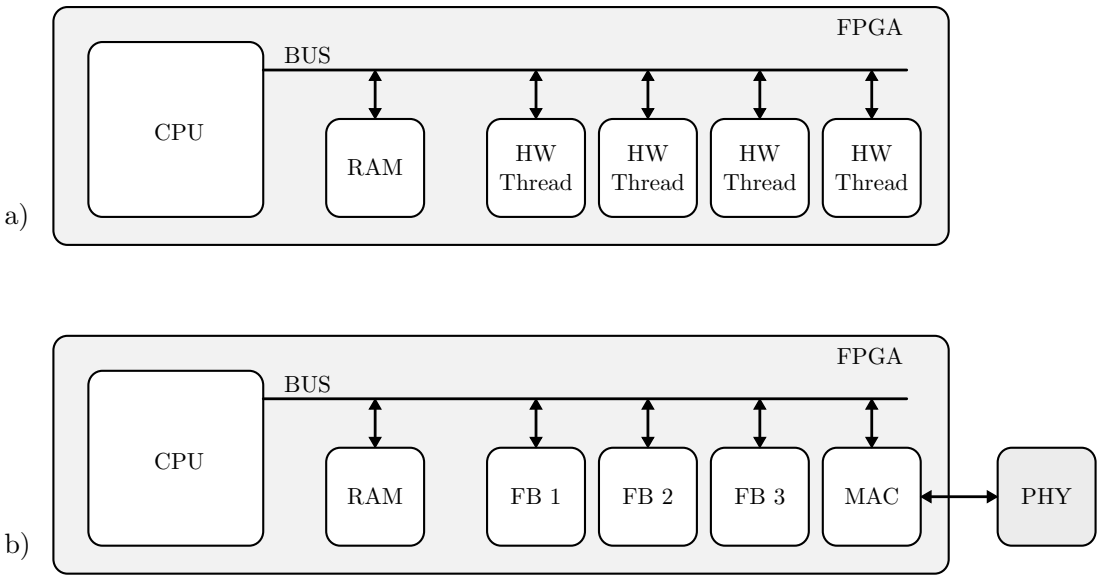

Figure 1.2: a) The ReconOS architecture: partially reconfigurable logic slots are used as hardware threads. b) Using ReconOS for ANA: a set of functional blocks is executed in hardware threads.

soon becomes the bottleneck of the system. In this thesis we present an extension of the ReconOS framework with a high throughput communication infrastructure shown in figure 1.3. This infrastructure contains a packet based network on chip (NoC) that guarantees each hardware thread a worst case incoming and outgoing data throughput. Additionally, a ring buffer based gateway was designed to speed up the transfer from software to hardware and vice versa.

The reminder of this thesis is organized in the following way: TODO!

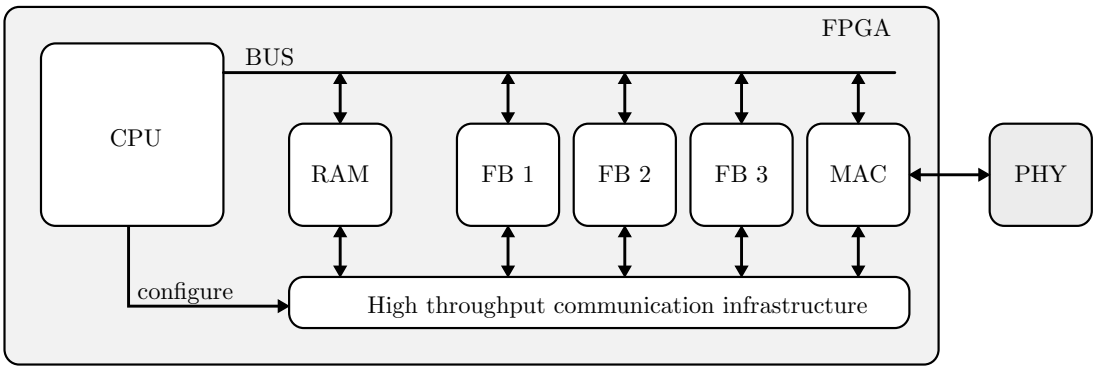

Figure 1.3: The high throughput communication interface guarantees the functional blocks a worst case incoming and outgoing bandwith.

Chapter 2

Design

In this chapter we present... TODO!

2.1 Design Targets

In ReconOS, multiple hardware threads execute tasks on behalf of the CPU. To this end, the hardware threads and the software running in the CPU need to exchange data with each other. ReconOS provides different mechanisms for this purpose, most notably message boxes and shared memory. All of these mechanisms have in common that they use a shared bus for the data transmission. This shared bus architecture is not well suited for a networking application like ANA. Imagine a packet that has to be processed by a set of functional blocks, all of them mapped to a hardware thread. In this case, the bus has to repeatedly forward the whole packet from one functional block to the next. The overall data transfer on the bus therefore increases linearly to the number of participating functional blocks.

In this thesis, we want to extend the ReconOS architecture with a high throughput communication infrastructure. The data rate at which each functional block can send data over the communication infrastructure to a specific destination should be independent of the traffic in the communication infrastructure targeting a different destination.

We distinguish three different applications of data transfer:

- Data transfer from a hardware thread to a software thread
- Data transfer from a software thread to a hardware thread
- Data transfer from one hardware thread to another hardware thread

The case of data transfer from one software thread to another software thread is out of the scope of this thesis.

ANA distinguishes two directions in which a specific packet could be processed. *Ingress* packets arrive at the physical interface, traverse the protocol stack upward and are then passed to the corresponding application. *Egress* packets are produced by an application, traverse the protocol stack downwards and then transmitted on the physical interface.

2.1.1 Throughput

In this thesis, we assume that the size of a packet does not significantly change on its way through the protocol stack. We can therefore upper bound the total data rate at the input and output port of each functional block with the data rate on the physical interface. In our development environment, the Xilinx ML605 evaluation kit [2], this physical interface is a Gigabit Ethernet module.

We therefore expect form the communication infrastructure to forward 1 GB/s of data to each hardware mapped functional block, independent of the origin of the data, as long as the outgoing data rate of each hardware thread is upper bounded by 1 GB/s.

2.1.2 Latency

Data transfers that cross the hardware/software boundary cause an orders of magnitudes higher latency than data transfers in the hardware domain [3]. This is caused by the shared memory access and by the high overhead of the unavoidable interrupts. The total latency of a packet is therefore mainly influenced by the performance of the hardware to software or software to hardware data transfer, respectively. We therefore neglect latencies caused by the communication infrastructure between hardware threads and focus on the latencies induced when crossing the hardware/software boundary.

2.1.3 Hardware Resource Utilization

The desired communication infrastructure is only a helper construct for the actual application running on the device. We therefore want to keep it as small as possible, so that there is a maximum of programmable logic still available for the application.

2.2 Evaluation of Design Principles

In this section, we discuss possible design principles for the communication infrastructure. The discussion is based on the results of [4]. In subsection 2.2.4 we then conclude which design principle is best fit to the design targets stated in section 2.1.

2.2.1 Point to Point Interconnection (P2P)

In a P2P architecture, all hardware threads are directly connected to all other hardware threads. The data transmission rate and the latency is optimal, since the packets do not share the resources with any other transmission. The receiving hardware thread is responsible for the serialization of the packets concurrently arriving on the different input ports. The logic and routing resource utilization of this architecture scales very bad with an increasing number of hardware threads.

2.2.2 Multibus

The multibus architecture is a mixture of the original shared bus architecture of ReconOS and the P2P architecture. In this architecture, all hardware threads read from their own individual bus. In order to send a packet to a hardware thread, the source hardware thread writes the packet to the bus of the destination hardware thread. An individual bus arbiter is required for each hardware thread.

The data rate on this architecture is optimal, since each hardware thread can receive data with the full data rate of his bus. The latency depends on the response time of the bus arbiter. This architecture has a high routing resource utilization. Additionally, the parasitic capacities of the high impedance bus interfaces might limit the clock rate on the bus when the number of hardware thread increases.

2.2.3 Network on Chip (NoC)

2.2.4 Conclusion

Appendix A

Original Task Description

A.1 Introduction

This master thesis is in the context of the EPiCS project. The goal of the EPiCS project is to lay the foundation for engineering the novel class of proprioceptive computing systems. Proprioceptive computing systems collect and maintain information about their state and progress, which enables self-awareness by reasoning about their behaviour, and self- expression by effectively and autonomously adapting their behaviour to changing conditions. Concepts of self-awareness and self-expression are new to the domains of computing and networking; the successful transfer and development of these concepts will help create future heterogeneous and distributed systems capable of efficiently responding to a multitude of requirements with respect to functionality and flexibility, performance, resource usage and costs, reliability and safety, and security.

In this thesis we focus on the networking aspect of EPiCS which we call *EmbedNet*. EPiCS uses the network architecture developed in the ANA project as a basis. The ANA network architecture is a novel architecture that enables flexible, dynamic, and fully autonomous formation of network nodes. In the EPiCS project we develop the ANA architecture further. On the one hand we will develop mechanisms to adapt the functionality provided by the protocol stack at runtime, on the other hand we will develop mechanisms that map the networking functionality dynamically to either hardware or software.

The objective of this Masters Thesis is to refine the hardware architecture and the hardware/software interface of EmbedNet.

A.2 Assignment

This assignment aims to outline the work to be conducted during this thesis. The assignment may need to be adapted over the course of the project.

A.2.1 Objectives

The goal of this Master thesis is to develop a hardware architecture for future dynamic protocol graphs. The developed architecture has to partition networking functionality into individual blocks. The architecture should allow for the dynamic reconfiguration of whole networking blocks as well as for the runtime configuration of communication parameters in the networking blocks. An interconnect between the functional blocks should be provided that allows for the forwarding of packets at line rate. The transmit data path should not interfere with the reception data path.

A.2.2 Tasks

This section gives a brief overview of the tasks the student is expected to perform towards achieving the objective outlined above. The binding project plan will be derived over the course of the first three weeks depending on the knowledge and skills the student brings into the project.

Familiarization

- Study the available literature on ANA, EmbedNet and Reconos [1, 2, 3, 4].
- Setup a FPGA development environment with the Xilinx tools 12.3 (exact) and ModelSim SE version 6.1f (or higher).
- Familiarize yourself with git/github and clone the reconos source code repository [5].
- Install the eCos source code and cross compile toolchain [6].
- Verify your toolchain by running first the sort_demo_thermal and then the pr_demo.
- In collaboration with the advisor, derive a project plan for your master thesis. Allow time for the design, implementation, evaluation, and documentation of your software.

Architecture and hardware design

- Develop a dedicated communication interface between the different hardware networking blocks. Adapt the hardware blocks where needed. The interface should be able to forward data at line rate (1Gbit/s). Transmission and reception direction should be treated independently. The interface should support different priorities of either sending networking blocks or individual packets.

- Develop a configuration interface for the hardware networking blocks. Each hardware block needs to be configured with the information required for forwarding the individual packets to the next protocol. The configuration interface should be able to set, delete, and replace configuration entries. The configuration should be initiated by a software program.

- Develop a data path for sending data between hardware and software networking blocks. Consider using two ring buffers per networking block, one for packets to be transmitted and one for packets to be received. The reader/writer synchronization should be handled properly. An interface should be developed to send and receive packets from software.

- Optional: Adapt the data path to work with the LANA protocol stack [7].

- Optional: Develop a monitoring framework for the measurement of hardware parameter such as temperature, utilisation or power consumption. The framework should collect the data in hardware and make them accessible to a software program.

Implementation

- Determine an appropriate version control system. The EPiCS project is hosted at github. You might want to put your code in the same repository.

- Implement a static version of the dedicated communication interface together with some dummy networking blocks for validation purposes.

- Add the possibility for partial reconfiguration of hardware networking blocks to the communication interface.

- Implement the configuration interface. Depending on the overall EPiCS project status use either the eCos operating system or the Linux operating system (preferred).

- Implement the data path for the hw/sw interface. Depending on the overall EPiCS project status use either the eCos operating system or the Linux operating system (preferred).

- Optional: Implement the interface to LANA.

- Optional: Implement the monitoring framework.

Validation

- Validate the correct operation of your implementation after each implementation step. Use for your evaluation different packet sizes (short, long, even or odd number of bytes, etc.).
- Check the resilience of the implementation, including its configuration interface, to uneducated users.

Evaluation

- Do a performance evaluation of your implementation.
- Optional: Determine the bottlenecks of your implementation.
- Optional: Do a performance comparison between packet forwarding for different combinations of hardware and software networking functional blocks.

Documentation

- Appropriate source code documentation.
- Write a step-by-step how to that describes the compilation of your code, the loading of the code into the hardware and the execution of your code.
- Write a documentation about the design, implementation, validation and evaluation of your work.

A.3 Milestones

- Provide a ”project plan” which identifies the mile stones.
- Two intermediate presentations: Give a presentation of 10 minutes to the professor and the advisors. In this presentation, the student presents major aspects of the ongoing work including results, obstacles, and remaining work.
- Final presentation of 15 minutes in the CSG group meeting, or, alternatively, via teleconference. The presentation should carefully introduce the setting and fundamental assumptions of the project. The main part should focus on the major results and conclusions from the work.
- Any software that is produced in the context of this thesis and its documentation needs to be delivered before conclusion of the thesis. This includes all source code and documentation. The source files for the final report and all data, scripts and tools developed to generate the figures of the report must be included. Preferred format for delivery is a CD-R.

- Final report. The final report must contain a summary, the assignment, the time schedule and the Declaration of Originality. Its structure should include the following sections: Introduction, Background/Related Work, Design/Methodology, Validation/Evaluation, Conclusion, and Future work. Related work must be referenced appropriately.

A.4 Organization

- Student and advisor hold a weekly meeting to discuss progress of work and next steps. The student should not hesitate to contact the advisor at any time. The common goal of the advisor and the student is to maximize the outcome of the project.
- The student is encouraged to write all reports in English; German is accepted as well.
- The core source code will be published under the GNU general public license.

A.5 References

[1] ReconOS: Multithreaded Programming for Reconfigurable Computers: Description of the hardware/software architecture
[2] Reconifgurable Nodes for Future Networks: Description of how we would like to use the hw/sw architecture to build reconfigurable networks
[3] The Autonomic Network Architecture (ANA): Description of the ideas and sw prototype for configurable networks
[4] https://github.com/EPiCS/epics-org/blob/master/deliverables/D3-1_Architecture_And_Tool_Flow/D3-1_Architecture_And_Tool_Flow.pdf
[5] https://github.com/EPiCS/reconos/
[6] webpage: ecos.sourceware.org, a version that is compliant with reconos is in the github repository, cross compilation tools: gcc-4.1.2 glibc-2.3.6
Webpages:
http://www.ana-project.org
http://www.epics-project.eu
http://www.reconos.de

Bibliography

[1] G. Bouabene, C. Jelger, C. Tschudin, S. Schmid, A. Keller, and M. May. The autonomic network architecture (ana). *Selected Areas in Communications, IEEE Journal on*, 28(1):4 –14, january 2010.

[2] Xilinx Inc. Ml605 evaluation kit product site http://www.xilinx.com/products/boards-and-kits/ek-v6-ml605-g.htm, May 2012.

[3] A. Keller, B. Plattner, E. Lübbers, M. Platzner, and C. Plessl. Reconfigurable nodes for future networks. In *GLOBECOM Workshops (GC Wkshps), 2010 IEEE*, pages 357 –361, dec. 2010.

[4] Hyung Gyu Lee, Naehyuck Chang, Umit Y. Ogras, and Radu Marculescu. On-chip communication architecture exploration: A quantitative evaluation of point-to-point, bus, and network-on-chip approaches. *ACM Trans. Des. Autom. Electron. Syst.*, 12(3):23:1–23:20, May 2008.

[5] Enno Lübbers and Marco Platzner. Reconos: Multithreaded programming for reconfigurable computers. *ACM Trans. Embed. Comput. Syst.*, 9(1):8:1–8:33, October 2009.

[6] Enno Lübbers, Marco Platzner, Christian Plessl, Ariane Keller, and Bernhard Plattner. Towards Adaptive Networking for Embedded Devices based on Reconfigurable Hardware. In *International Conference on Engineering of Reconfigurable Systems and Algorithms (ERSA'10)*, pages 225–231, Las Vegas, NV, USA, 2010. CSREA Press.