FACULTY OF SCIENCE
UNIVERSITY OF COPENHAGEN

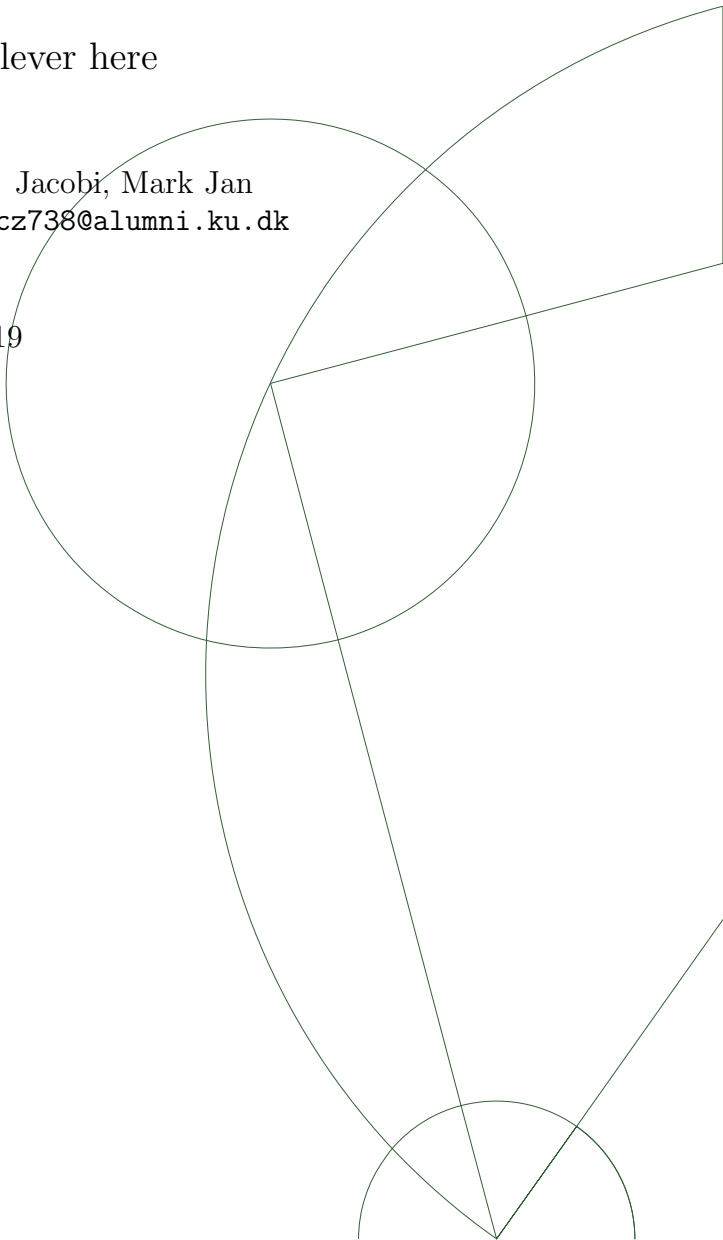
TCP/IP in hardware using SME

Write something clever here

Meznik, Jan
pzj895@alumni.ku.dk
jan@meznik.dk

Jacobi, Mark Jan
dcz738@alumni.ku.dk

June 10, 2019



Contents

Chapter 1

Introduction

This thesis describes the design and implementation of an efficient, high-speed TCP/IP network stack intended to run on custom hardware where performance, responsiveness, and throughput is crucial.

As is the trend with modern automation, computerization, and mechanization, new devices are steadily invented to handle this increasing demand for data and control. With the ever-increasing sophistication of machines generating immense amount of information, the data needs to be transmitted to numerous other machines for further processing, or even simply storage. The most common and the most convenient way of linking multiple devices together is using the internet, and its underlying protocols. However, the networking stack supplied with most major operating systems, while heavily optimised, suffers from considerable penalties due to complexities of a standard computer architecture. For example, heavy network traffic utilizes the computers' internal busses, allocates vast amounts of memory, and spend precious CPU clock-cycles with polling and interrupts. This prevents the machine from using these resources for actual computing tasks.
These issues have been identified and solved by hardware manufacturers by adopting dedicated Network Interface Controllers (NIC) which would employ various techniques to offload the processing. One such offloading technique is called the TCP offload engine (TOE), which usually takes care of the essential parts of networking involved – the Internet Protocol and the Transmission Control Protocol [10].

Modern hardware manufacturers can produce NICs boasting network throughput speeds as high as 100 Gigabits [5]. Unfortunately, these cards are highly specialized for certain applications, and even though they provide basic programmability, they are rarely suitable for rapid prototyping of applications and other custom hardware devices. Furthermore, each NIC manufacturer has a diverse set of hardware with varying interfaces, making it hard to combine and swap and test these cards. Licensed software solutions in the form of IP blocks exist as well. Unfortunately, these blocks are usually distributed as black-boxes of VHDL code, which is hard to maintain, and even harder to debug and extend.

In this thesis, we bridge the gap between the blazingly-fast network offloading devices and their more flexible and malleable software counterparts.
This networking stack is implemented in a fully self-contained fashion so that it is completely independent of any other software running on the machine, while utilizing the performance advantages gained from the lack of overhead in conventional implementations. The use of a high-level programming language in combination with the modern Synchronous Message Exchange model makes the network stack a very versatile implementation with ease of use, debugging, and even extension.
MORE TO COME!!

Chapter 2

Background

In this chapter, we will introduce the basic concepts of the Internet Protocol Suite, briefly describe its origin, semantics, and its reference implementation in the early BSD systems. Furthermore, SME will be introduced as a basis for the implementation.

2.1 Field Programmable Gate Array (FPGA)

Field Programmable Gate Arrays, or FPGA for short, are devices containing integrated circuits (ICs) consisting of arrays of logic blocks. These ICs can be reprogrammed at any time for a desired application or functionality [7], making the devices very flexible and extensible, even after manufacturing.
Unlike conventional processors with a very sequential nature, the logic blocks in FPGAs are truly parallel in nature. Given the right programming, an FPGA can allocate dedicated sections of the chip for each independent subtask, enabling the circuitry to perform numerous independent calculations at once [7]. Unfortunately, this universality of FPGAs comes at a cost to their performance. Whereas conventional processors are heavily optimise based on the predetermined circuitry, FPGAs programmers must ensure to utilize the parallel nature of the device in order to secure best possible performance.
Still, with innovations and steady improvements in modern FPGAs, the devices can easily reach a clock higher than 500 MHz [6].

2.2 Internet Protocol Suite (TCP/IP)

Internet Protocol Suite, better known as simply TCP/IP, is a conceptual model providing end-to-end communication between computers. It consists of a collection of protocols specifying the communication between multiple Internet systems [4]. The very early research and development on what would later become the Internet Protocol Suite began in the late 1960s by the Defense Advanced Research Project Agency (DARPA), and was being adopted by DARPA, as well as the public, since 1983 [14]. Although the Internet Protocol Suite predates the newer, arguably more refined Open Systems Interconnection model (OSI model), TCP/IP still remains the popular choice in modern systems. As opposed to OSI 7-layer model [8], the collection of protocols in TCP/IP are organized into 4 abstraction layers, each related to their scope of networking involved.

2.2.1 Link Layer

The link layer is the lowest, bottom-most layer in the Internet Protocol Suite. Link layer addresses methods and protocols operating on the link that the host is physically connected to[1]. Contrary to the OSI model, this lowest layer in TCP/IP does not regard the standards and protocols of the physical mediums used (the pin layout, voltages, cable specifications etc.), making TCP/IP hardware-independent. As a result, TCP/IP can in theory be implemented

[1] Wireless connections are also included under this category.

on virtually any hardware configuration, emphasizing the flexibility of the model.

2.2.2 Internet Layer

The internet layer mainly concerns itself with sending data from the source network to the destination network. This seemingly simple task requires multiple functions from the layer:

- Addressing and identification
- Packet routing
- *Basic* transmit diagnostic information
- Carrying data for various upper layer protocols

2.2.3 Transport Layer

The transport layer establishes end-to-end data transfer between hosts. Protocols in the transport layer can provide additional services to the user, such as reliability, ordering, error- and flow-control, application addressing (port numbers), error-checking, and so on.
While it is possible to bypass the protocols in this layer on most modern network stacks, the protocols in the transport layer provide such essential and useful services that it hardly ever makes sense to implement in the application layer.

2.2.4 Application Layer

The application layer protocols are used by applications and services to exchange information over the network. A few of the well-known application layer protocols are the Hypertext Transfer Protocol (HTTP) [2], File Transfer Protocol (FTP) [3], and Simple Mail Transfer Protocol (SMTP) [11].
This layer is usually implemented by the applications themselves, and therefore are not strictly required to actually run a TCP/IP network.

2.3 Synchronous Message Exchange

The Synchronous Message Exchange model (SME) is a messaging framework created in order to help model hardware descriptions [12]. It was conceived once the flaws of using Communicating Sequential Processes (CSP) was identified during the modelling of a vector processor with CSP using PyCSP [9]. It turned out that there is a major discrepancy between the way data is propagated in hardware opposed to that of the CSP model. While CSP does not pose any requirements on the communication between processes, in digital hardware, all communication has to be synchronized, driven by a clock. To combat this in the CSP model, a global clock process needed to be implemented, which was connected to all other processes. Additionally, latches had to be introduced in order to not overwrite values during a cycle. This caused an explosion of both channels and latches in the final design, making CSP a much less viable framework for hardware modelling [12].

2.3.1 The model

The SME model consists of only a few fundamental concepts. Each SME model is a *network* consisting of one or more *processes*. These processes do not share any memory or storage, but are interconnected with *busses*. These busses are perhaps the most interesting units in SME model, as they not only propagate information between processes using the underlying *channels*, but also introduce an implicit clock between the processes.

2.3.2 Process execution flow

The execution flow of a process is fairly simple, and relates very closely to that of the actual hardware. At the beginning of a clock-cycle, the input-ports are read into the busses they are connected to. Then, the process executes its "compute" stage, and the results, if any, are written to the

output-port, which will be read by the following bus. A visualization of the execution flow can be seen on figure 2.1. It is im-

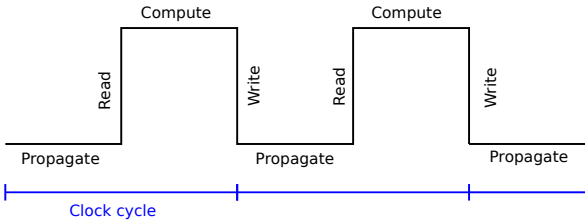


Figure 2.1: An illustration of a typical SME clock-cycle

portant to note that although certain channels might be written earlier than others in a process clock, the subsequent processes connected to said bus will first see the values change in the beginning of the next clock cycle.

2.3.3 Using SME

SME has undergone multiple iterations, reworks, and extensions. While it is still under very active testing and development, its core functionalities and features are well-established and stable [13].
SME has concurrent implementations in the C# and Python languages, with promising efforts to unify these under a common intermediate domain-specific language SMEIL [1]. The C# version has exhibited various advantages over the Python counterpart, such as the more error-prone strong typing system, which better reflects the functionality of the hardware, as well as making the code more readable to the progrmamer. At the time of writing, the C# implementation currently enjoys the most recent features of the SME model, as it is being the most actively developed version.

Chapter 3

Design

3.1 Overview

The networking stack introduced in this thesis is implemented in the C# programming language with SME. The aim of its design is to capacitate performance, flexibility, and ease of use. In this chapter, the design principles are described, the architecture of the solution is outlined, and the components are outlined.

3.1.1 Design principles

As briefly mentioned in the introduction, the proposed network stack is to provide an alternative to the existing proprietary network offloading engines. While the main goal of this thesis is to research and study the suitability of SME for implementing a TCP/IP stack on an FPGA, there are many other aspects of the system to be studied.

3.1.2 Initial requirements

Following our design principles, initial requirements and goals for the networking stack are set so that these can be tested and improved upon.

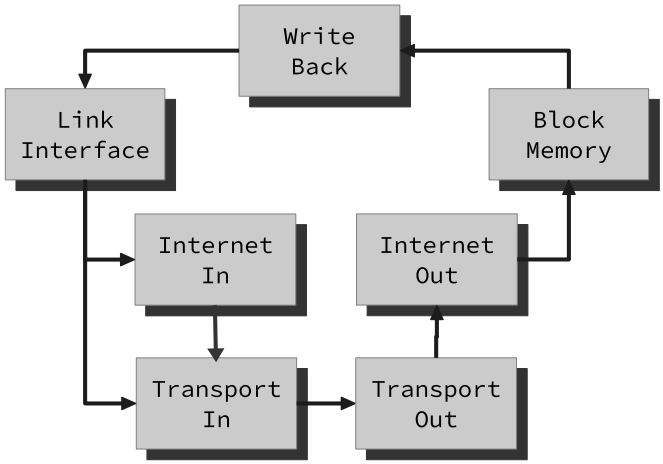
- **Essential protocols only**
  Considering that the SME project is still fairly early in its development, and considering the sheer number of protocols in the internet protocol suite, the networking stack in this thesis is to support only the absolutely essential protocols required to provide the users with a meaningful interface to the internet. These protocols should be picked such that the system can provide the end-user with a network data-stream, which can transport information to and from a remote computer.
  The initial protocols chosen may be implemented and supported partially, but they must not deviate from the standard specifications.

- **Support an interface for the end-user**
  The system must be controlled by an end-user on the FPGA. Such an interface is very unique in its own way, compared to standard software interfaces, like the ones defined in the POSIX collection of specifications. By supporting such an external interface gains insight in the way such a networking stack will be used, and which measures must be taken in order to provide the best possible integration and performance considerations.

- **Independent of underlying physical hardware**
  By using SME, the underlying hardware description language code can be abstracted away from the actual implementation. This will later provide developers to easily modify and tweak the networking stack without having to consider the target hardware.
  Likewise, the networking stack may not rely on using a certain physical layer hardware, and must be designed to be independent of the underlying hardware used for the physical connections. This will ensure that the target hardware can easily swap be-

tween physical connectors, such as going from ethernet cables to wireless, or even another FPGA.

3.2 The architecture



3.3 Link

3.4 Internet

3.5 Transport

3.6 Interface

Appendices

Bibliography

[1] Truls Asheim. A domain specific language for synchronous message exchange networks. 2018.

[2] Tim Berners-Lee, Roy T. Fielding, and Henrik Frystyk Nielsen. Hypertext transfer protocol – http/1.0. RFC 1945, RFC Editor, May 1996. http://www.rfc-editor.org/rfc/rfc1945.txt.

[3] A.K. Bhushan. File transfer protocol. RFC 114, RFC Editor, April 1971.

[4] Robert Braden. Requirements for internet hosts - communication layers. STD 3, RFC Editor, October 1989. http://www.rfc-editor.org/rfc/rfc1122.txt.

[5] Xilinx Inc. 100g nic with pp integration. https://www.xilinx.com/applications/data-center/network-appliances/100g-nic-pp-integration.html. [Online; accessed 2019-05-13, Archived by WebCite ®at http://www.webcitation.org/78L0it9n0].

[6] Xilinx Inc. What is an fpga? https://www.xilinx.com/products/silicon-devices/fpga/what-is-an-fpga.html, 2019. [Online; accessed 2019-05-12, Archived by WebCite ®at http://www.webcitation.org/78K4ocp7U].

[7] National Instruments. Fpga fundamentals. http://www.ni.com/da-dk/innovations/white-papers/08/fpga-fundamentals.html, March 2019. [Online; accessed 2019-05-12, Archived by WebCite ®at http://www.webcitation.org/78K4kDEjB].

[8] TELECOMMUNICATION STANDARDIZATION SECTOR OF ITU. Open systems interconnection - basic reference model: The basic model. Std, ITU, November 1994. [Online; accessed 2019-04-29, Archived by WebCite ®at http://www.webcitation.org/77zmViPac].

[9] John Markus Bjørndalen, Brian Vinter, and Otto J. Anshus. Pycsp - communicating sequential processes for python. volume 65, pages 229–248, 01 2007.

[10] Jeffrey C. Mogul. Tcp offload is a dumb idea whose time has come. USENIX Association, 05 2003.

[11] J. Postel. Simple mail transfer protocol. RFC 788, RFC Editor, November 1981.

[12] Brian Vinter and Kenneth Skovhede. Synchronous message exchange for hardware designs. 08 2014.

[13] Brian Vinter and Kenneth Skovhede. Bus centric synchronous message exchange for hardware designs. 08 2015.

[14] Mitch Waldrop. Darpa and the internet revolution. https://www.darpa.mil/attachments/(2015)%20Global%20Nav%20-%20About%20Us%20-%20History%20-

%20Resources%20-%2050th%20-%20Internet%20(Approved).pdf. [Online; accessed 2019-04-29, Archived by WebCite ®at http://www.webcitation.org/77zkNxkL8].