Dissertation

submitted to the
Combined Faculties of the Natural Sciences and Mathematics
of the Ruprecht-Karls-Universität Heidelberg, Germany
for the degree of
Doctor of Natural Sciences

presented by
Dipl.-Phys. Eric Christian Müller
born in Heidelberg, Germany

Date of oral examination: 2014-12-17

Novel Operation Modes of Accelerated Neuromorphic Hardware

Eric Christian Müller

2014

Referees:	Prof. Dr. Karlheinz Meier
	Prof. Dr. Ulrich Brüning

Novel Operation Modes of Accelerated Neuromorphic Hardware

The hybrid operation mode relies on a combination of conventional computing resources and a neuromorphic, beyond von Neumann system to perform a joint real-time experiment. The interactive operation mode provides prompt feedback to the user and benefits from high experiment throughput. The performance of a custom transport-layer protocol is evaluated connecting the accelerated neuromorphic system and the computer cluster. Wire-speed performance is achieved between host and eight FPGAs ((846.7 ± 1.2) MiB/s, 94% wire speed), and between two hosts using 10-Gigabit Ethernet ($> 99\%$) as well as 40GbE ($> 99\%$) to explore scaling behavior. The software architecture to process neuronal network experiments at high rates is presented including measurements which address the key performance indicators. During hybrid operation, the tight coupling between both resources requires low-latency communication. Using a custom-developed software framework, an average one-way latency between two host computers connected via 10GbE is found to be (2.4 ± 0.2) µs and (8.5 ± 0.4) µs to the neuromorphic system. A hybrid experiment is designed to demonstrate the hardware infrastructure and software framework. Starting from a conventional neuronal network simulation, the experiment is gradually migrated into a time-continuous experiment which interacts between a host computer and the neuromorphic system in real time. Results of the intermediate steps and the final, hybrid operation are evaluated.

Neurartige Betriebsarten beschleunigter neuromorpher Hardware

Der hybride Betriebsmodus beruht auf der Kombination aus konventionellen Rechnern und eines neuromorphen Systems, das nicht auf der von Neumann Architektur beruht, um ein Experiment gemeinsam in Echtzeit durchzuführen. Die interaktive Betriebsart ermöglicht eine schnelle Ergebnisrückmeldung an den Benutzer und profitiert von einem hohen Experimentdurchsatz. Die Leistung eines speziell entwickelten Transportprotokolls, welches das beschleunigte neuromorphe System und den Rechnercluster verbindet, wird evaluiert. Die nominelle Übertragungsgeschwindigkeit wird zwischen Hostrechner und acht FPGAs ((846.7 ± 1.2) MiB/s, 94% der Maximalgeschwindigkeit), sowie zwischen zwei Rechnern über 10GbE ($> 99\%$) sowie 40GbE ($> 99\%$) erreicht. Die Softwarearchitektur, die nötig ist um hohe Experimentraten zu erreichen, wird vorgestellt und anhand von Messungen werden die Schlüsselgrößen untersucht. Die enge Kopplung der Resourcen, die der hybride Betrieb benötigt, stellt Anforderungen an die Kommunikationslatenz. Diese wird mittels eines speziellen, selbstentwickelten Softwarepakets gemessen. Für die Interhostlatenz werden (2.4 ± 0.2) µs und für die Anbindung des neuromorphen Systems (8.5 ± 0.4) µs ermittelt. Ein hybrides Experiment wird entworfen, das als Demonstration der Hard- und Softwareinfrastruktur dient. Beginnend mit einer konventionellen neuronalen Netzwerksimulation wird das Experiment schrittweise in Richtung einer zeitkontinuierlichen Echtzeitinteraktion zwischen Kontrollrechner und neuromorphem System entwickelt. Zwischenschritte sowie das Endresultat werden präsentiert und ausgewertet.

Contents

List of Figures

List of Tables

Introduction

Understanding the brain is the goal of neuroscience. A commonly used approach in computational neuroscience are software simulations of spiking neuronal networks. Recently, a large-scale simulation[1] – covering one percent of the size of the human brain – was performed on the K computer, a supercomputer[2] which is ranked fourth in the current TOP 500 [2014]. This simulation already reaches three types of technical limitations. Simulating $O(10^{12})$ synapses exhausts the available memory. The power consumption of such supercomputers is tremendious; the K computer consumes 12.6 MW. Finally, the slow simulation speed – the previously mentioned simulation required 40 min runtime to simulate 1 s, a speed-down of 2400 – limits the number of simulations which can be performed within the assigned maximum computing time.

Yet, there is an alternative to simulation: emulation – neuromorphic hardware systems implementing neurons and synapses emulate the behavior of the biological counterparts. Using mixed-signal VLSI, many neurons and synapses can be assembled on single chips to build neuronal networks. Within the Human Brain Project (HBP), this endeavor is pursued to build multiple interconnected wafer-scale systems, the so-called NM-PM1. The NM-PM1 features almost $2 \cdot 10^5$ neurons and $4.4 \cdot 10^7$ synapses per silicon wafer. It implements a leaky integrate-and-fire neuron model enhanced by an exponential term as well as an adaptation term. The synapses are plastic, a long-term learning mechanism is implemented on chip.

However, this approach also faces three limitations. Noise influences analog circuits which induces non-deterministic behavior. Compared to freely programmable software simulations, the configurability of the hardware implementation is limited to a specific model and to certain parameter ranges. Finally, the maximum neuronal network size is limited by the size of the system. The NM-PM1 system consists of twenty wafers, i.e. $4 \cdot 10^6$ neurons and 10^9 synapses which is three orders of magnitude smaller than the simulation mentioned before.

On the other hand, there are clear advantages. The power consumption of neuromorphic systems bridges the gap to biology for the first time. For example, the K supercomputer simulation supported approximately 340 synapses per Watt[3]. In constrast, the NM-PM1 wafer module provides $440 \cdot 10^6$ synapses per Watt[4]. In these terms, the NM-PM1 is six orders of magnitude more efficient than the K

[1]NEST simulation of $1.73 \cdot 10^9$ neurons with 1 Hz firing rate on average, $10.4 \cdot 10^{12}$ synapses, 40 min for 1 s simulated time [Eppler, 2014; RIKEN, 2013].

[2]LINPACK $R_{\max} = 11$ PFlop/s at 12.6 MW power consumption.

[3]Power consumption of 12.6 MW, 40 min to simulate 10^{12} synapses for 1 s simulated time.

[4]Power consumption of 1 kW, 1 s to simulate $4.4 \cdot 10^7$ synapses for 10 000 s emulated time.

supercomputer.

Secondly, neuromorphic hardware stands for a new computer architecture. In contrast to the conventional von Neumann machines, it inherits properties from biology like fault tolerance and unsupervised learning. This potentially allows for robust and adaptive systems without preprogrammed patterns of behavior.

Finally, the main advantage for the user is the speed of accelerated neuromorphic systems. Compared to the biological time domain, typical large-scale simulations run slower than real-time; the K supercomputer simulation exhibited a speed-down of 2400. In contrast to software simulations, the presented neuromorphic system exhibits a speedup[5] of typically 10^4 relative to biology. Due to this highly accelerated operation, a large range of experiment time scales can be explored. On the one hand, large time scales are required for experiments which explore the temporal dynamics of learning (days), development (years) and evolution. On the other hand, parameter sweeps require a large set of short-running experiments; the accelerated operation reduces the overall run time.

However, this hardware feature needs to be conveyed to the user meaning that the support infrastructure needs to be capable of coping with the high emulation speed. Intermediate hardware layers as well as the software stack need to be optimized for this task. The exact requirements depend on the application which can be categorized into three modes of operation.

Neuronal network experiments can be self-contained; that means the individual experiment stimulus is pre-defined and not depending on network output. Such experiments run in larger *batch*es or interactively. Both modes benefit from high experiment rates; however, the interactive mode additionally requires a low configure-to-result latency to allow for *interactive* updating of the result. For intuition-guided exploration of parameter spaces update rates in the order of 1 Hz, or faster, are required.

Finally, experiments that are not self-contained require some interaction between the neuronal network and the entity which creates the stimulus. This so-called *closed-loop* operation is particularly difficult to achieve on accelerated neuromorphic systems as the speedup increases the visible external reaction delay in the emulation time domain.

The self-contained experiments depend on a fast data flow between the user and the system. The NM-PM1 system provides 1 GiB/s per wafer module. To saturate this bandwidth, multiple requirements have to be met: a fast communication protocol, pipelined software operation, and a software architecture providing efficient data structures. Regarding the closed-loop operation, timing constraints have to be met: a low-latency communication channel that uses low-level operating system methods to provide latencies in the µs range. Lastly, the neuroscientific modeling community can only benefit from this system if user-friendly operation is a matter of course.

Software development does not happen in isolation but in an area of diametral

[5]The time constants can be configured to yield speedups in the range around 10^4.

interests. One challenge is the continuous influx of new components enhancing the system. New hardware components always require new software or modifications to existing implementations. Related to the progress of neurocomputational research, the user demands are changing: for example, the user interfaces necessitate continuous development to keep up with the current de-facto standards in computational neuroscience. Other challenges include support for increasingly sophisticated experiments and the integration of external software packages[6]. The Human Brain Project (HBP) aims for a holistic integration of data resources, neuroscientific modeling, data analysis and simulation. A toolkit to bring data providers and users together, with special focus on the aspect of large-scale simulations. In this perspective, the integration of the NM-PM1 system into this toolkit, the so-called *unified portal*, is of primary concern.

Structure of the Thesis

The thesis at hand is split into three chapters. In the first chapter (chapter 1) the neuromorphic systems are described. Starting with the wafer module, the relevant system components are introduced. Subsequent sections focus on the architecture of the conventional part. Both, the cluster architecture and the data network architecture have been designed by the author. An evaluation of the main properties is provided.

The second chapter (chapter 2) covers efforts to support the typical usage mode of accelerated neuromorphic systems: isolated experiments in quick succession. The figure of merit is the throughput which has two meanings: data throughput (in B/s) and the number of experiments per unit of time.

Finally, the third chapter (chapter 3) concentrates on hybrid operation of the Hybrid Multiscale Facility (HMF). This means the real-time interaction between a conventional software simulation and the neuromorphic hardware. In particular, this mode has requirements regarding communication latencies. The figure of merit has two facets: data communication latency and reaction delay.

Every chapter provides a small summary at the end to collect the essential parts. A final discussion is provided at the end.

Naming Conventions

This thesis focuses on the neuromorphic hardware systems that have been developed during the BrainScaleS and Human Brain Project projects. The core of the system are the neurons and synapses which are implemented as physical models. In constrast to software *simulations*, the model equations are not numerically solved, but rather *emulated*.

[6]For example, transistor-level simulators to support a common work flow accessing hardware or simulation.

An unique feature of the system is the accelerated operation. That means, the time constants of the model's hardware implementation are smaller when compared to real time. This is called *speedup*.

The ambiguous term *network* is always used in conjunction with *data* to separate its meaning from the *neuronal network*.

Hardware Components and Abbreviations The neuromorphic hardware system presented in this thesis is complex. For example, a single Wafer Module comprises over 100 printed circuit boards. This is why only the parts relevant for this thesis have been described; a complete hardware specification is available [HBP SP9 Specification, 2014]. This specification describes the hardware state planned for month 30 of the Human Brain Project (HBP) project. However, the specification will be updated to include later modifications. At the end of this document, a glossary collects all abbreviations used in the text.

International System of Units The author adheres to the international system of units. However, in computer science the base-2 notation for units of digital information is popular. To resolve this issue, a set of binary prefixes has been defined by the International Electrotechnical Commission (IEC). The standard SI prefixes are modified by appending *i* to indicate base-2 notation. For example, the author uses $2^{10}\,\mathrm{B} = 1024\,\mathrm{B} = 1\,\mathrm{KiB}$ – the *i* in kB denotes the non-SI prefix (called *kibi*, i.e. kilo-binary, instead of *kilo*). The same is true for MiB, GiB, and TiB. All data sizes stated without *i* indicate SI-sizes, e.g., $1\,\mathrm{MB} = 10^6\,\mathrm{B}$.

Supervised Work

The author coordinated and supervised the work of Moritz Schilling, Vitali Karasenko, Kai Husmann and David Hinrichs. During his diploma thesis, Moritz Schilling worked on the HostARQ protocol (HostARQ) software implementation presented in section 2.1.1.2. Vitali Karasenko worked on the FPGA implementation of the HICANN-ARQ protocol and HostARQ protocol presented in section 2.1.1.3 and section 2.1.1.3. He performed preparatory work in his bachelor thesis, and implemented both components in his master thesis. Two more bachelors worked on the software stack: Kai Husmann evaluated a key component of the pipelined software architecture presented in section 2.2.5. David Hinrichs worked on the integration of a transistor-level simulation into the software flow presented in section 2.2.1.4.

1. System Architecture

This chapter describes hardware components of the Hybrid Multiscale Facility (HMF) and NM-PM1 systems which are relevant for this thesis. In the first section the neuromorphic part is described. The conventional part – which has been designed by the author – is described in the remaining sections. A complete hardware specification can be found in HBP SP9 Specification [2014].

Figure 1.1 presents an illustration of the Human Brain Project (HBP) NM-PM1 system. At the time of writing, the compute cluster and some parts of the data network have already been installed.

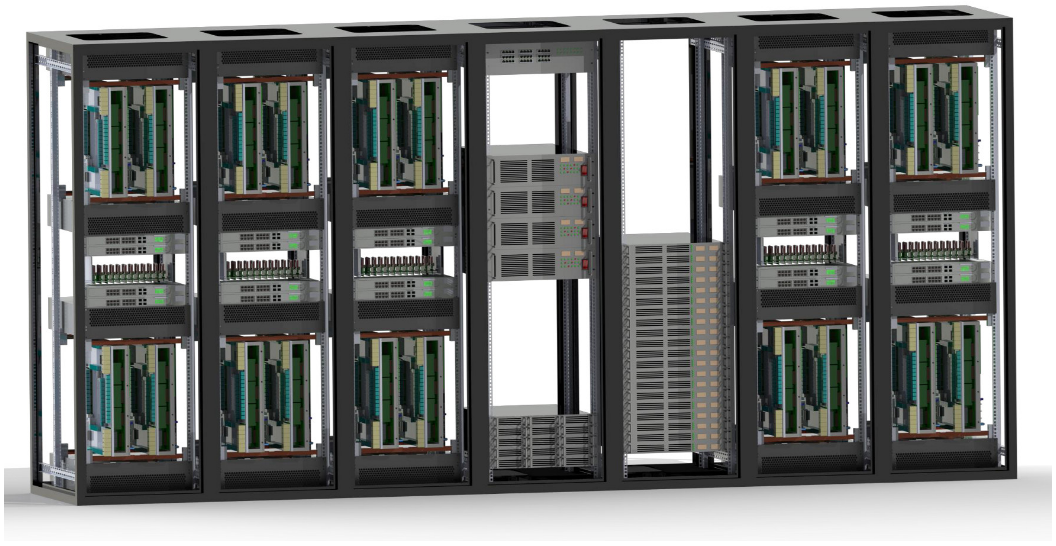


Figure 1.1.: Illustration of the HBP NM-PM1 system. The neuromorphic part is distributed over five racks containing four wafer modules each. A 20-node compute cluster including one storage server as well as two front-end nodes is located in the fourth rack (from the left); in the rendering, the compute nodes are located in the fifth cabinet. Communication between the individual wafer modules and compute nodes relies on multiple Gigabit Ethernet links that are aggregated to 10-Gigabit Ethernet. A backbone switch located on top of the compute cluster interconnects the compute nodes, the wafer switches and provides an uplink to the institute. Image by Dan Husmann.

1.1. The Wafer Module

The core of the HMF and NM-PM1 system is the neuromorphic part; its design started during the FACETS project. This *neuromorphic* unit, the so-called Wafer Module, contains a 20 cm wafer produced in a standard 180 nm CMOS process technology. On a single wafer are 48 groups, the so-called *reticles*, of eight *HICANN* (high-input count analog neuronal network) chips each – 384 HICANN chips in total. In contrast to typical fabrication, the wafer is not cut into many dice but stays complete; an interconnection step during post-processing provides on-wafer connectivity between the reticles. This explains why the system carries the attribute *wafer-scale*. One HICANN chip provides up to 512 neurons and 224 synapses. This results in approximately $2 \cdot 10^5$ neurons and $4.4 \cdot 10^7$ synapses per wafer. Neurons are emulated by an analog circuit that implements the adaptive exponential integrate-and-fire neuron model [Brette and Gerstner, 2005]. Up to 64 neuron circuits can be combined to form a single neuron supporting more than 224 synapses. By combining 64 neuron circuits a total number of 14336 synapses can be used. Figure 1.2 presents a schematic, exploded view of a single wafer module. A single wafer module consists of multiple parts. These are support electronics for power monitoring and control as well as the FPGA-based[1] communication printed circuit boards (PCBs)[2], the so-called FPGA communication PCBs (FCPs). The latter provides inter-wafer and host connectivity. Associated with a single wafer module are two more types of components: power control is performed by power management units (PMUs), and the analog readout modules (AnaRMs) offer analog recording capabilities. All AnaRMs modules of four wafer modules, i.e. a *rack*, get aggregated by one analog readout module aggregator node (AnaRMAN).

1.1.1. Internal Connectivity

A wafer comprises many identical HICANN chips. During fabrication, the step-and-repeat method cycles through all possible positions and replicates the same physical structures, i.e. groups of eight HICANN chips or reticles, onto the wafer. A later post-processing step adds connections between the otherwise unconnected reticles. These connections are used for a bus network, the so-called Layer 1 (L1) network, providing on-wafer spike-event transport. All HICANN chips of one row or column share 256 *vertical* and 64 *horizontal* buses. Two buses can be interconnected at specific connection points using so-called crossbar switches. However, not every switch position is physically available because the presence of the switch itself increases the parasitic capacity of the bus lane up to a point where successful data transmission is no longer possible. For a detailed description of the bus system see HBP SP9 Specification [2014]. The specified bus event[3] rate at maximum clock

[1]field-programmable gate array (FPGA)
[2]printed circuit board (PCB)
[3]The raw symbol rate is $R = 2\,\text{Gbit/s}$, events are encoded using 6 bits plus start and stop bit. After every frame a pause of the same length is inserted.

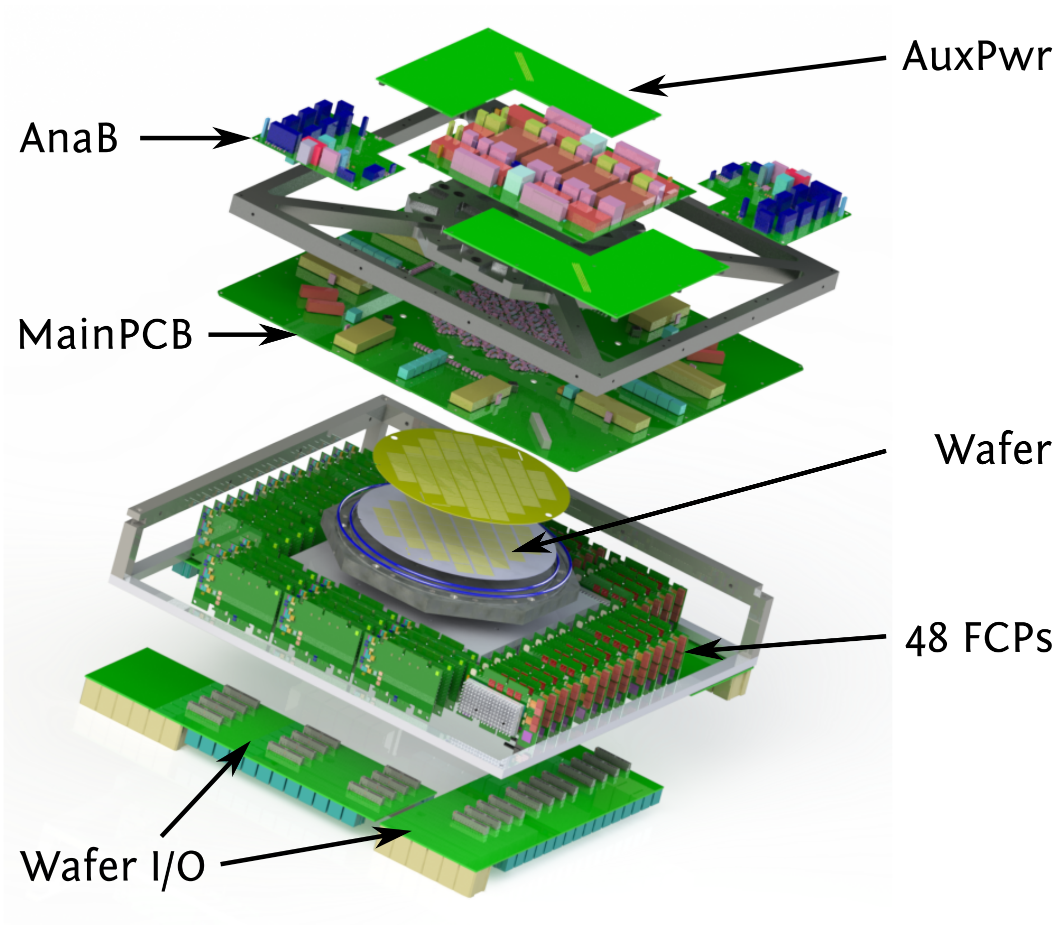


Figure 1.2.: A single wafer module consists of many components. Core of the wafer module is the wafer which carries neurons, synapses and the on-wafer communication bus. It is located at the center of the hardware stack. Multiple printed circuit boards (PCBs) provide connectivity and power to the wafer; one module comprises approximately 100 PCBs. Links for external communication are provided by the PCBs at the bottom, the I/O boards (wafer I/O PCB). Associated with one wafer module are three analog readout modules for analog recordings, one power management unit and one compute node to control the system. Figure 1.3 presents the FCPs in detail. Hardware details regarding Analog Breakout PCB (AnaB) and Auxiliary Power Supply PCB (AuxPwr) and all other components can be found in HBP SP9 Specification [2014]. Image by Dan Husmann.

frequency (250 MHz) is 125 MEvent/s [Schemmel et al., 2010]. Hence, the bisection bandwidth[4] of a HICANN chip can be calculated as follows:

$$(N_{\text{horizontal}} + N_{\text{vertical}}) \cdot R_{\text{ev}} = (256 + 64) \cdot 125\,\text{MEvent/s} = 40\,\text{GEvent/s} \qquad (1.1)$$

However, a single HICANN chip can only inject events on eight horizontal buses. The sending bandwidth is given by:

$$N_{\text{sending}} \cdot R_{\text{ev}} = 8 \cdot 125\,\text{MEvent/s} = 1\,\text{GEvent/s} \qquad (1.2)$$

At the time of writing, the chip is operated at a reduced clock frequency of 100 MHz[5]; the L1 event rates are reduce by the same factor (2/5).

1.1.2. External Connectivity

To inject and extract spike stimuluses into the on-wafer neuronal network the HICANN chip provides another communication layer, the so-called Layer 2 (L2) protocol. In contrast to the time-continuous L1 bus network, the L2 link uses additional time stamps to encode spike times. Optimally packed events can be encoded using $32 + 4$ bit[6]. A full-duplex connection between every HICANN chip and the associated FCP field-programmable gate array (FPGA) operates at $R_{\text{symbol}} = 2\,\text{Gbit/s}$ [Schemmel et al., 2010].

Thus, the external event rate of a HICANN chip is:

$$R_{\text{event}} = R_{\text{symbol}} / \operatorname{sizeof}(\text{Event}) = \frac{2\,\text{Gbit/s}}{36\,\text{bit}} = 55.6\,\text{MEvent/s} \qquad (1.3)$$

The total (half-duplex) off-wafer bandwidth:

$$N_{\text{HICANN}} \cdot R_{\text{symbol}} = 384 \cdot 2\,\text{Gbit/s} = 96\,\text{GB/s} \qquad (1.4)$$

The corresponding event rate is:

$$N_{\text{HICANN}} \cdot R_{\text{event}} = 384 \cdot 55.6\,\text{MEvent/s} = 21.3\,\text{GEvent/s} \qquad (1.5)$$

The links are capable of operating full duplex. Hence, the full-duplex bandwidth and rates are twice as large.

[4]When the chip is split into two equally-sized parts, this is the bandwidth between the two parts.

[5]The reduced clock frequency provides more robust on-wafer, i.e L1, communication; higher clock speeds require further tuning of L1 parameters.

[6]A double-spike packet requires 48 bit; header and cyclic redundancy check (CRC) consume additional 16 bit. After every packet, a 8 bit pause is required. That means two spikes can be transmitted every 72 bit cycles.

The L2 protocol does not only support spike-event communication but also configuration data. A chip-internal communication bus-structure (not to be confused with the on-wafer spike network) exposes addressable memory to the FCP FPGAs. Due to the non-reliable L2 (i.e. the link does not feature a transport protocol), a dedicated transport layer protocol, the so-called HICANN-ARQ protocol, provides reliability in case of data corruption or packet loss. Configuration data is encoded using 64 bit each; hence, the packet rates are reduced by a factor of approximately two.

In the FCP FPGA, upstream data can be routed to other FCPs via the inter-wafer connections. Alternatively, it can be routed to the host. The latter introduces a bottleneck in terms of bandwidth as the host link is based on Gigabit Ethernet (1GbE). This means that eight HICANN chips share a 1 Gbit/s host connection. The Gigabit Ethernet (1GbE) links are capable of full-duplex communication, i.e. the same is true for the reverse communication path. Section 1.1.3 provides details regarding the FPGA communication PCB.

The host link uses a 32 bit encoding for a single spike. A optimally packed spike data frame contains 182 double spikes (i.e. 364 spikes per packet). Using equations (2.3) and (2.4), we can calculate the maximum spike throughput using the host link.

$$\begin{aligned} R_{\text{spike}} &= \text{Throughput}_{\text{payload}}/\mathit{sizeof}(\text{Spike}) = \frac{947.9\,\text{Mbit/s}}{32\,\text{bit}} \\ &= 29.6\,\text{MEvent/s} \end{aligned} \tag{1.6}$$

As stated in the previous section, the chip is currently operated at a reduced clock frequency; the calculated L2 speeds are reduced by the same factor. Notably, the FCP FPGA itself is not affected and the inter-FPGA or host links maintain their speeds.

1.1.3. The FPGA Communication PCB

From the users' point of view, the FPGA communication PCB (FCP) are responsible for almost all tasks related to experiments: 1. processes configuration data describing neuronal network topology and neuron model parameters; 2. recording of spike data and stimulation with spike data; 3. real-time spike data transport; 4. and runtime control. Figure 1.3 shows a FPGA communication PCB which will be used in the production-type HMF and NM-PM1 wafer modules.

The FCP carries 1.25 GiB DRAM. 256 MiB of the total memory are available for buffering Ethernet-based host communication. The remaining 1 GiB is used for buffering experiment input and output data. To a large part, this memory is occupied by input and output spike data as a single wafer configuration only requires approximately 50 MiB. Due to constraints regarding data alignment, the memory consumption in DRAM is larger, approximately 100 MiB distributed over all FCPs.

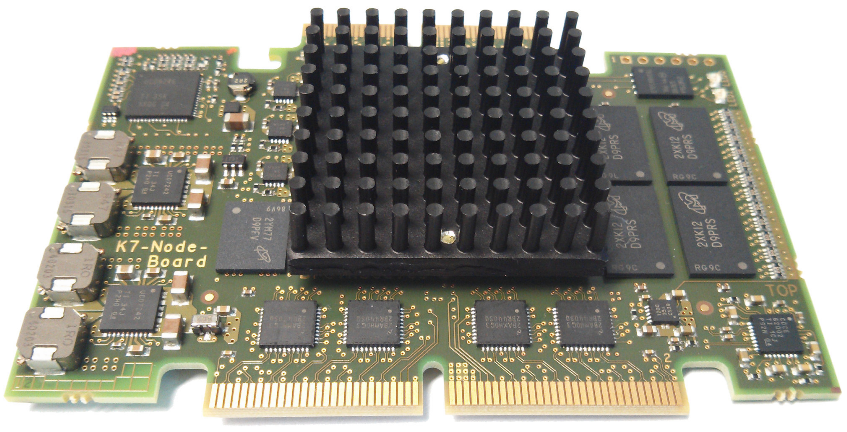


Figure 1.3.: FPGA communication PCB (FCP) used in the production-type HMF and NM-PM1 systems. The bottom connector is plugged into the MainPCB. Host connectivity is provided by four wafer I/O PCBs (two horizontal and two vertical versions) that combine 12 FCPs each. Every FCP carries 1 GiB DRAM for spike data and 256 MB for communication buffers. For details see HBP SP9 Specification [2014].

Hence, the memory consumption per FCP is only 2 MiB and the remaining memory can be used for spike data or buffering/pipelining of experiments.

In general, experiments can be categorized as being *non-interactive*/batch-style or *real-time*. The former describes experiments where external stimulus has been pre-calculated – i.e. the input does not depend on the neuronal network response – and the stimulus is *played back* to the system. This operation mode benefits from the real-time capabilities of FPGAs; the FPGA can provide fixed timings that allow for cycle-precise real-time playback and recording of data. Experiments that interact with a software in real-time are covered in chapter 3. The conventional use case, i.e. batch-style usage, is covered in chapter 2.

A complete wafer module comprises 384 HICANN chips that are grouped into 48 reticles; every reticle is governed by one dedicated FCP. These 48 FCPs are located between MainPCB and four wafer I/O PCBs (WIOs) that carry external connectors. Twelve 1GbE links and 48 inter-wafer links are located on every WIO.

Every FCP can be used individually. Networks involving multiple reticles require synchronized interaction between multiple FCP FPGAs. In particular, synchronized clocks are essential to avoid any drift which would yield desynchronized playback and recording of spike data. The NM-PM1 system implements this synchronization using a common clock source.

Gigabit Ethernet During the FACETS project, the importance of a reasonably fast and conveniently usable host interface was identified. At that time, 1GbE was state of the art and fulfilled the requirements: it supports 1 Gbit/s = 125 MB/s = 119.2 MiB/s wire speed, it is supported by standard FPGA chips, the physical connection technology (using twisted pair cat. 5 or 6 cables) is cheap and easily deployable, networking hardware for all use cases and scales is commercially available, and good software support is available. Based on the experiences of the FACETS Stage 1 system [Gutmann, 2007; Schilling, 2010], the HMF and NM-PM1 use Gigabit Ethernet as default data network technology. To ensure fast and reliable communication between FCP and host computer, a transport layer implementation has been developed by the author (see section 2.1).

1.1.4. Analog Readout

Eight HICANN chips, i.e. one reticle, share two analog output channels. This yields 96 channels per wafer module. These channels are fed to analog readout modules that can be used for recording of analog neuron membrane traces. One 19-inch rack contains four wafer modules and twelve AnaRMs connected to one analog readout module aggregator node. Each AnaRM provides a 12 bit analog-to-digital converter (ADC) sampling at 125 MHz. Recording and readout are controlled by an on-board Xilinx Spartan-6 FPGA that is connected via USB 2.0 to a host computer. The traces are buffered in local 512 MiB-DRAM memory. At full sampling rate, this provides a maximal recording time of 2.7 s real time[7]. Conversion to 1GbE is performed by one AnaRMAN per rack; i.e. twelve AnaRMs share a single 1GbE uplink. This protocol conversion is performed for a number of reasons: 1. to allow for a non-static assignment between AnaRMs and compute nodes. 2. to circumvent the maximum physical cable length of USB 2.0; 3. reduce load on control nodes – on the current compute nodes USB 2.0 transfers are computationally more expensive than 1GbE-based communication.

The author suggests to switch from the USB 2.0-based to a 1GbE-based interface as it would not only increase the throughput but also allow to omit the AnaRMAN which, in turn, reduces complexity and increases robustness.

1.1.5. Model and Parameter Domains

The neuron circuits of the HICANN chip implement the adaptive exponential integrate-and-fire (AdEx) neuron model [Millner, 2012]. Model dynamics are presented in Naud et al. [2008], an evaluation of the present hardware implementation can be found in Millner [2012]; Schwartz [2013]. Compared to the leaky integrate-and-fire model, the AdEx model adds two more variables that allow for many known electrophysiological spike patterns; the LIF model supports only a single firing mode, i.e. regular spiking.

Speed-Up The analog neuron circuit is a physical implementation of the corresponding adaptive exponential integrate-and-fire (AdEx) model. In contrast to a step-wise simulation, neuron model dynamics are emulated in continuous time. Neuron (and synapse) model parameters are translated into the hardware domain.

[7]Assuming a speed-up factor of 10^4, this corresponds to 7.6 h.

In particular, the hardware time constants of the NM-PM1 and HMF are typically 10^4 to 10^5 times faster compared to biological real time; i.e. biological time is compressed by this *speed-up* factor. The precise speed-up factor depends on the neuron and synapse parameter set. Typically, the membrane time constant is used as a reference.

Parameter Space The total parameter space is dominated by synapse data: $512 \cdot 224$ synapses per HICANN chip store 4 bit weight and 4 bit address data, 112 KiB in total. Neuron-model-specific data[8] accounts for approximately 12% of the total memory [Brüderle et al., 2011]. Thus, the total wafer parameter space is approximately 50 MiB.

1.1.6. Wafer Module Prototype System

At the time of writing, the *production-type*[9] wafer module systems were not yet available. All measurements that interacted with any part of the neuromorphic system relied on the prototype setup shown in figure 1.4.

There are some differences between the prototype wafer module and the final HMF or NM-PM1 wafer modules. The main difference is the new MainPCB which increases the number of FCPs from twelve to 48. I.e. the prototype system uses twelve Xilinx Virtex-5 FPGA-based boards (below blue fans in figure 1.4, details in figure 1.5) each controlling four reticles; the production-type systems will use 48 Xilinx Kintex-7 FPGA-based PCBs each controlling a single reticle. Hence, the maximum host bandwidth will be increased by a factor of four. Figure 1.5 shows a picture of the FCP used in the prototype system.

Changes that do not directly affect the usage of the system are: 1. the power management unit will be based on Raspberry Pi replacing an embedded ARM evaluation board; 2. the lab power supplies will be replaced by custom-built supplies based on 48 V DC power supply units (PSUs); 3. the power monitoring and control is improved.

1.2. Cluster Architecture

The properties of the previously described neuromorphic system are unique. In particular, its emulation speed (cf. section 1.1.5) allows for new types of experiments and use cases. Long-term learning experiments can easily cover multiple days in simulated time which translates to only several seconds of real time. Hence, long-running experiments directly benefit from speedup factor. In contrast, parameter sweeps are typically short but require many runs; this requires an efficient reconfiguration mechanism. Given the large emulation speedup of 10^4, the perceived or real

[8]24 analog parameter entries per neuron account for 12288 entries; the specified precision is 10 bit, i.e. 15 KiB per HICANN chip.

[9]Production-type system refers to the system as specified in HBP SP9 Specification [2014].

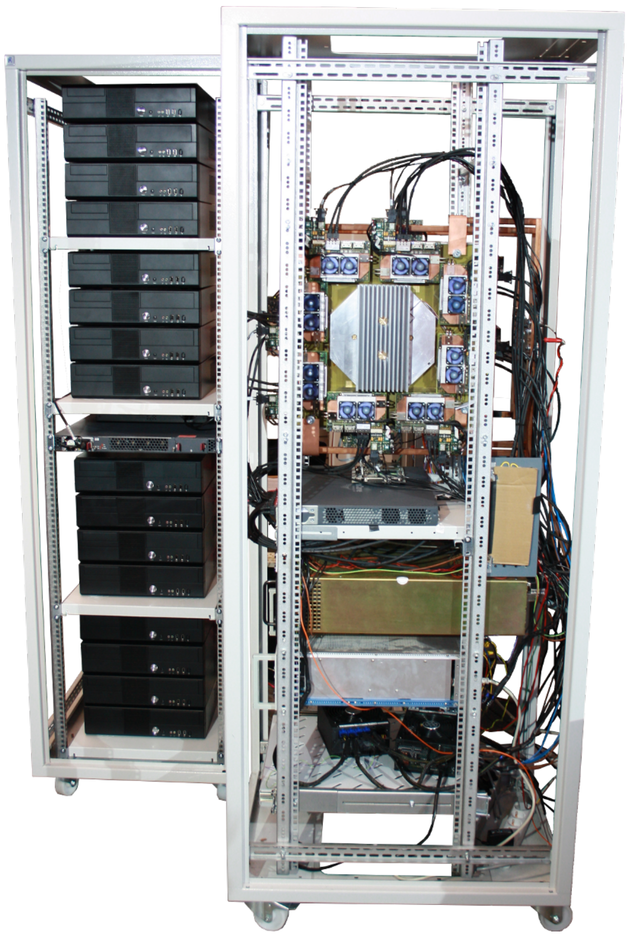


Figure 1.4.: Photograph of the first wafer prototype setup (right) together with the HMF compute cluster (left). All measurements using any part of the wafer module have been performed on this prototype. For a description of the differences between *production-type* and prototype wafer module see section 1.1.6. Details on the non-neuromorphic (or conventional) part, i.e. compute cluster and data network, can be found in sections 1.2 and 1.3.

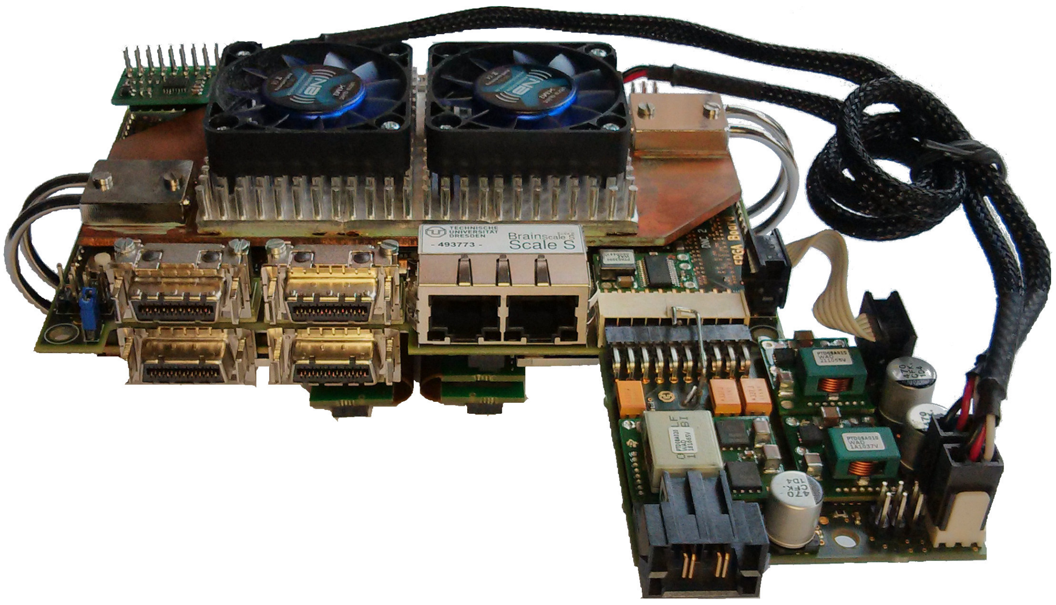


Figure 1.5.: FPGA communication PCB used in the prototype system. To provide adequate cooling, the FPGA is covered by an heat sink and two fans. Host connectivity is provided by the two centrally located 8P8C jacks (often called RJ45) using 1GbE. On the left side there are four Infiniband-type/CX4 connectors designated for inter-FPGA connectivity. The wafer *connectors* can be vaguely seen (marked by arrows) on the bottom of the PCB just below the right CX4 and the left 8P8C jack. In the lower-right a small PCB acts as a power supply.

speedup is typically not dominated by the emulation time but rather by overhead. Optimizing the over-all speedup is one main topic of this thesis.

The overhead can be examined in the different experiment execution stages: 1. processing of experiment descriptions to compute valid hardware configurations, 2. experiment preparation, 3. hardware configuration, 4. and experiment readout. All overhead categories can be optimized by improving software or hardware. The former – creating a high-performance software stack – is covered in chapters 2 and 3; the latter will be addressed for the conventional hardware parts in the following sections. In particular, the compute node and data network architectures, the conventional part, of both, the HMF and the NM-PM1 systems, was designed by the author.

1.2.1. Compute Nodes

During all experiment phases – configuration, execution, and read-out phase –, a single compute node handles a single wafer, i.e. up to 48×1GbE data streams[10]. To achieve acceptable performance within the financial budget, central processing units (CPUs) designed for server operation are avoided. At the time of writing, the Intel® Core™ i7-4771 CPU offers state-of-the-art single-thread performance at reasonable cost. Per CPU a LINPACK benchmark[11] performance of approximately 180 GFlops can be reached (see figure 1.6).

Certainly, floating point performance is not a key performance indicator for most layers in the software hierarchy of the HMF or NM-PM1: most software layers involve data intensive tasks. In particular, the computationally most expensive task is the mapping step which translates a neuronal network description to a hardware configuration. This task is more similar to the Graph 500[12] benchmark.

For large-scale mapping jobs it is planned to make use of high-performance computing (HPC) sites that are available within the Human Brain Project. Main task of the compute nodes is experiment control – a programmatically linear task. Assuming a fixed budget, high-clocked *desktop* CPUs are adequate for this job.

Another major requirement is high throughput: the wafer module provides 48 parallel 1GbE links for host communication. At the time of writing, networking interfaces providing 10-Gigabit Ethernet (10GbE) links are state of the art. Concerning the host interface, 40-Gigabit Ethernet (40GbE) is budget-wise in reach, but the remaining network equipment, especially switches, is disproportionately more expensive. See section 1.3 for details on the data network.

For a certain neuronal network experiment class low-latency links are essential. This experiment category relies on a real-time interaction between neuromorphic hardware and host computers. Due to the combination of the conventional computer architecture and the non-von Neumann, neuromorphic architecture, this is

[10]That is one full-duplex data stream per FCP.
[11]A benchmark that measures floating point performance [Dongarra, 1988].
[12]Supercomputer list based on a benchmark which focuses on data-intensive loads [Murphy et al., 2010].

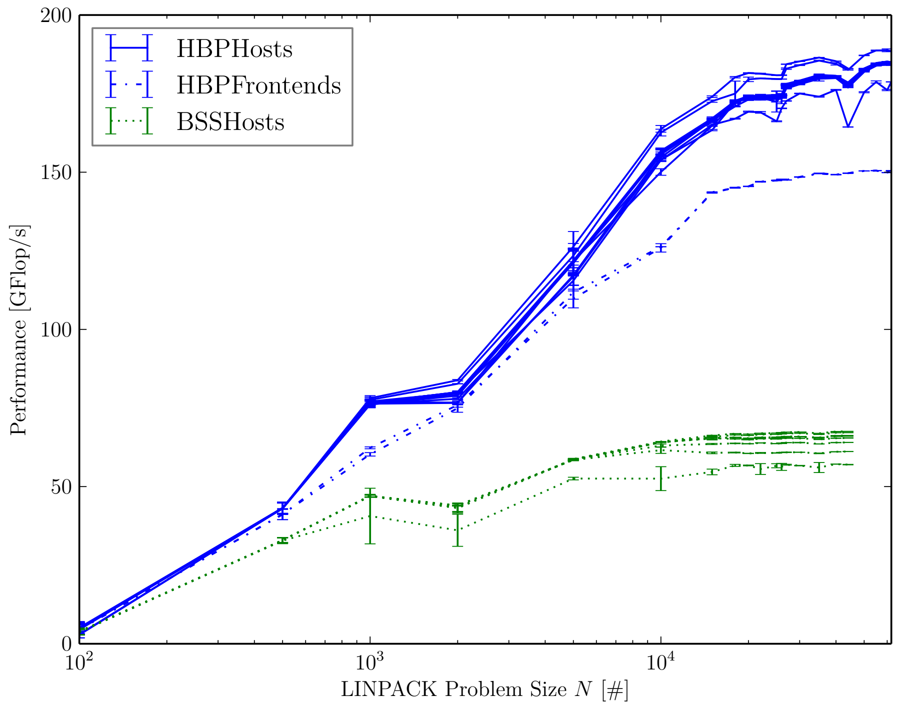


Figure 1.6.: Measurements of single-node LINPACK benchmark runs. The LINPACK problem size N is varied up to machine memory limits (i.e. 16 GiB for the HMF, 32 GiB for the NM-PM1 cluster). Floating point performance is plotted on the ordinate. The NM-PM1 cluster nodes are labeled `HBPHosts` and HMF nodes as `BSSHosts`. In addition, two NM-PM1 front-end nodes are tested; The quad-core NM-PM1 compute nodes achieve over 180 GFlop/s. In contrast to the hexa-core front-end nodes, the compute node support the AVX2 instruction set which allows for a doubled floating point peak performance per core. The HMF compute nodes are multiple CPU generations older and do not support advanced vector instructions at all. This reduces the per-core performance significantly. The errorbars denote the RMSE out of four runs. All hosts use dynamic clock rate adaptation depending on CPU's current thermal budget and power consumption. That means, large errors typically indicate temperature fluctuations. In particular, the runtime of the LINPACK benchmark is one characteristic marker to verify node installation quality. The constant offset of some compute nodes is mostly caused by differences in RAM modules; however, there could be some contributions to dynamic frequency adaption properties caused by differences between individual CPUs originating from manufacturing [Semeraro et al., 2002].

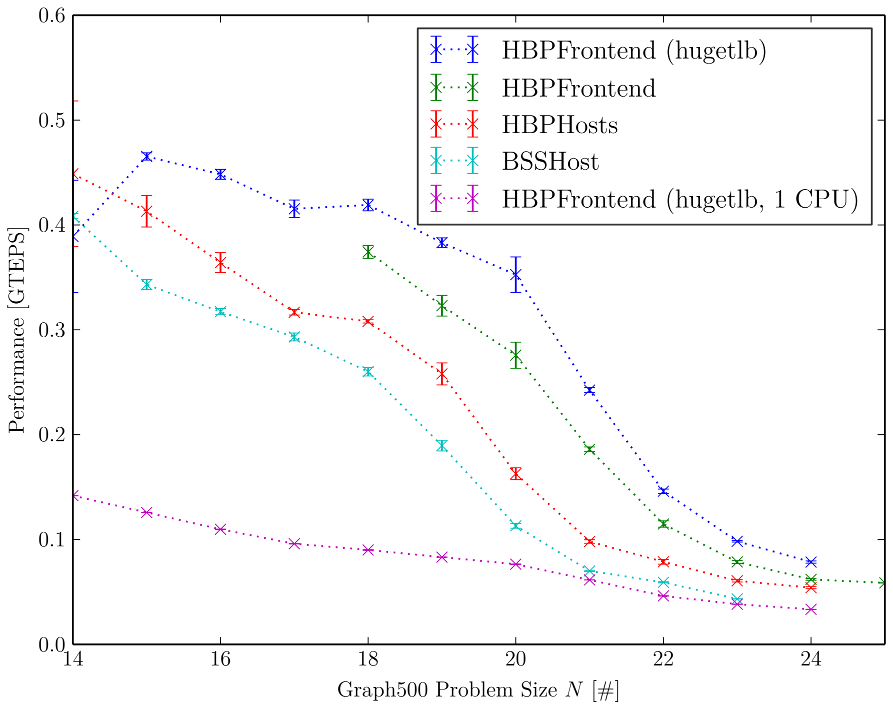


Figure 1.7.: Results of single-node Graph 500 benchmark runs [Murphy et al., 2010]. The problem size N is varied up to machine memory limits (i.e. 16 GiB for the HMF, 32 GiB for the NM-PM1 cluster). The ordinate shows the performance in terms of TEPS (traversed edges per second). Only basic optimizations were performed. On the front-end nodes the page handling of the Linux kernel was optimized for large memory allocations (`hugetlb` active in blue, disabled in green). There is a slight performance increase between HMF compute nodes and the NM-PM1 compute nodes. The larger gap between NM-PM1 compute nodes and front-end nodes is caused by the increased CPU core count. The measurement was performed in collaboration with Paul Müller.

	#	Component
CPU	1	Intel i7-4771
		Haswell, 22 nm
RAM	32 GiB	DDR3-1600
Main Board	1	Intel Q87 chipset-based
		Remote management via intel AMT
	1	PCIe ×16 Gen3 (16 lanes) slot
	4	Memory slots, DDR3-1600
NIC	1	Chelsio T520-LP-CR
	2	10GbE ports
		Low-latency
		MPI with RDMA-support
Case/PSU		1U including 2×-redundant PSU

Table 1.1.: Components of a NM-PM1 compute node. The quad-core desktop CPU runs at a base clock speed of 3.5 GHz (turbo speed 3.9 GHz). The dual-port 10GbE NIC supports low-latency operation (evaluation in section 3.3.3). The component list of the predecessor, the HMF compute node, can be found in table A.2. For details about the frontend nodes see table 1.3.

called *hybrid* operation. Details are covered in chapter 3.

The outcome of this budget optimization is presented in table 1.1. It presents a short component list of the NM-PM1 cluster nodes.

1.2.2. Storage

Experiments running on the NM-PM1 system can be data-intensive; configuration and result data of single experiments can easily reach into the GiB range. Depending on the exact data network configuration[13], the inbound and outbound data rates of a single wafer system are between 1 GiB/s and 4 GiB/s (see section 1.1.2). To support recording of experiments at full wire speed, the storage system has been split into different parts: a longer-term storage, and a fast short-term storage system. The former is a single conventional node, called *sto*, equipped with Hard disk drives (HDDs) and providing NFSv4-based network mounts (see table 1.2). Details regarding storage configuration can be found in appendix A.2.

[13]The backbone switch does not support 40 Gbit-links for all wafer modules. See section 1.3 for details.

	#	Component
CPU	1	Intel i7-4770
		Haswell, 22 nm
RAM	16 GB	DDR3-1600
Main Board	1	Intel Q87 chipset-based
		Remote management via intel AMT
	1	PCIe 3.0 ×16 slot
	1	PCIe 2.0 ×4 slot
	4	Memory Slots, DDR3-1600
NIC	1	Chelsio T580-LP-CR
	2	40GbE ports
SAS	1	LSI SAS Controller 9211-4i
	4	SATA 6G ports
Case/PSU	1	3U including 2×-redundant PSU
	16	hot-swap 3.5" SAS/SATA drive trays

Table 1.2.: Components of the NM-PM1 storage node. At the current stage, sequential reads or writes reach up to 1 GiB/s; increasing the number of HDDs could yield higher throughput. To eliminate this potential bottleneck, the node's uplink supports 40GbE. A file system benchmark is shown in figure A.1.

	#	Component
CPU	1	Intel E5-2643v2
		Ivy Bridge EP, 22 nm
RAM	64 GiB	DDR3-1600
Main Board	1	Intel C602 chipset-based
		Remote management via integrated IPMI 2.0/KVM
	2+2	PCIe ×16 Gen3 (16 lanes) slot
	24	Memory slots, DDR3-1866 non-ECC
		UDIMM ($\leq$ 128 GiB)
		RDIMM ECC ($\leq$ 768 GiB)
		LRDIMM ECC ($\leq$ 1.5 TiB)
NIC	1	Chelsio T580-LP-CR
	2	40GbE ports
SAS	1	LSI SAS Controller 9211-4i
	4	SATA 6G ports
Case/PSU	1	3U including 2×-redundant PSU
	16	hot-swap 3.5" SAS/SATA drive trays

Table 1.3.: Components of a NM-PM1 front-end node. The hexa-core server CPU runs at a base clock speed of 3.5 GHz (turbo speed 3.8 GHz). The large number of wide PCIe slots can be used for future upgrades: possibly Xeon Phi for boosting floating-point performance and/or fast PCIe-based solid state disks.

1.2.3. Frontend Nodes

Fast storage is available on the (currently) two server nodes. See table 1.3 for a list of components. The server uplinks use one 40GbE port (dual-port NIC). Each node carries 2 TiB distributed over two Solid-state disks (SSDs). Sequential read and write performance numbers are:[14]

Sequential Read	(515 ± 3) MiB
Sequential Write	(458 ± 5) MiB

Using a striped RAID array yields:

Sequential Read	(1394 ± 8) MiB
Sequential Write	(1391 ± 8) MiB

A file system benchmark is shown in figure A.2. It is planned to adopt the Ceph distributed file system [Weil et al., 2006].

[14]The benchmark numbers were acquired using `dd if=/dev/sdX of=/dev/null bs=1M` for reads, and `dd if=/dev/zero of=/dev/sdX bs=1M conv=fdatasync` for writes. Linux employs aggressive caching techniques; to reduce inter-run dependencies, before starting a run `echo 3 > /proc/sys/vm/drop_caches` was executed; after writes a final `sync` was performed.

1.2.4. Software Environment

The disk-less compute nodes boot via preboot execution environment (PXE) from a NFSv4 mount. This file system is provided by the storage node. The operating system is based on Debian Wheezy [Hertzog and Mas, 2014]. Some modifications have been performed: 1. on each node, a memory-based overlay file system is mounted to allow for non-persistent modifications of the local system (*kiosk* mode); 2. the Linux kernel version has been upgraded to 3.14 (due to hardware compatibility issues, and better support for cgroups[15]); 3. an updated OpenFabrics Alliance (OFED) software stack has been installed; 4. message passing interface (MPI)-related software packages have been recompiled due to this new stack. Software packages that are either not included in Debian Wheezy or too old, are provided using the modules environment [Modules, 2014].

Users work in their home directories which are located on another writable NFSv4 mount. All user access is mediated by the SLURM resource manager [LLNL et al., 2014]. An introduction to this resource management tool can be found in section 2.3.

1.2.5. Management and Monitoring

The cluster nodes are managed using Intel AMT. This technology allows for remote low-level hardware control. Functionality ranges from basic power control features to graphical remote access.

All nodes as well as the servers are monitored using Ganglia [Massie et al., 2012] and Nagios [Barth, 2008]. Custom plugins provide temperature sensor data and SLURM-related data.

1.3. Network Architecture

The development of scalable neuromorphic systems makes demands on the interconnection between wafer module (cf. section 1.1) and compute cluster (cf. section 1.2). Previous systems, e.g. Spikey, have been using a dedicated connection between neuromorphic chip and host computer. Another Spikey-based system provided chip interconnections by using a large PCB. In its final version, this system used 1GbE for host communication.

The analog readout system of the NM-PM1 system, AnaRM, and the latest version of the Spikey system are, except for the Spikey neuromorphic network chip, identical. Both use USB 2.0 for the host connection. See section 1.1.4 for hardware details.

When increasing the number of configurable system parts within a neuromorphic system, a scalable network topology and as well as a robust protocol stack becomes more and more important. A standard solution for data networks are Ethernet-based hardware components. The data network equipment is described in the

[15]Linux kernel-based isolation mechanism [CGroups, 2014]

following. To support all operation modes of the HMF and NM-PM1 systems, the data network architecture has to provide high throughput for the typical operation as well as low-latency operation for hybrid experiments.

Experiment Cycle In general, the experiment cycle of the NM-PM1 system involves different stages: configuration or setup phase, run phase and analysis phase. The configuration phase is dedicated to experiment preparation – the uploading of configuration data for the HICANN Wafer and input data for the FCPs. Depending on the experiment type, the host link is silent during run phase (batch-style experiments) or centered around low-latency data transfer (closed-loop experiments, see chapter 3). By experiment pipelining, the rate of batch-style experiments can be increased, i.e. the FPGA acts as a buffer for experiments – input data and output data has to be stored in FPGA memory. This operation modes is a generalization of the normal batch-style operation. It is planned to implement this mode. However, extensive modifications in the FPGA firmware are required.

Overview Every NM-PM1 or HMF wafer module provides 48 × 1GbE links, each dedicated to one FCP controlling a specific wafer part. A *wafer* data network switch aggregates these links into one 10GbE link connected to the backbone switch[16]. The compute nodes are connected via one single 10GbE to the backbone switch. Figure 1.8 presents an schematic illustration of the network topology. The BSS HMF and HBP NM-PM1 backbone switches are located at the center of the two large *circles*; compute nodes are represented as shaded rectangles, the shaded squares indicate wafer aggregation switches. Individual 1GbE links of every wafer module (yellow circles) are not shown; only the uplink to the aggregation switches is represented. Both, the HMF and the NM-PM1 systems are connected via 10GbE. The two irregularly-placed wafer (and wafer switches) represent the two lab prototype setups.

Figure 2.1 illustrates the connections within the NM-PM1 system. The central backbone switch provides 10GbE links to every compute node and to every wafer module; 40GbE links are provided to the servers (currently one storage node and two general-purpose servers). The connection to the BSS HMF system is not shown here.

Table 1.4 lists the data switches of the NM-PM1 system.

For all connections between compute node, backbone and wafer module switches Enhanced small form-factor pluggable supporting up to 10 Gbit/s (SFP+) or Quad SFP+ (QSFP) direct-attach copper cables are used. External connectivity for the NM-PM1 system is provided by one single-mode fiber (10GBASE-SR) link to the Kirchhoff-Institute for Physics (KIP); additional fibers are available for future upgrades.

[16]The number of 10GbE switch ports at the backbone switch is limited to 64; i.e. a few wafer modules can be connected using 4 × 10GbE.

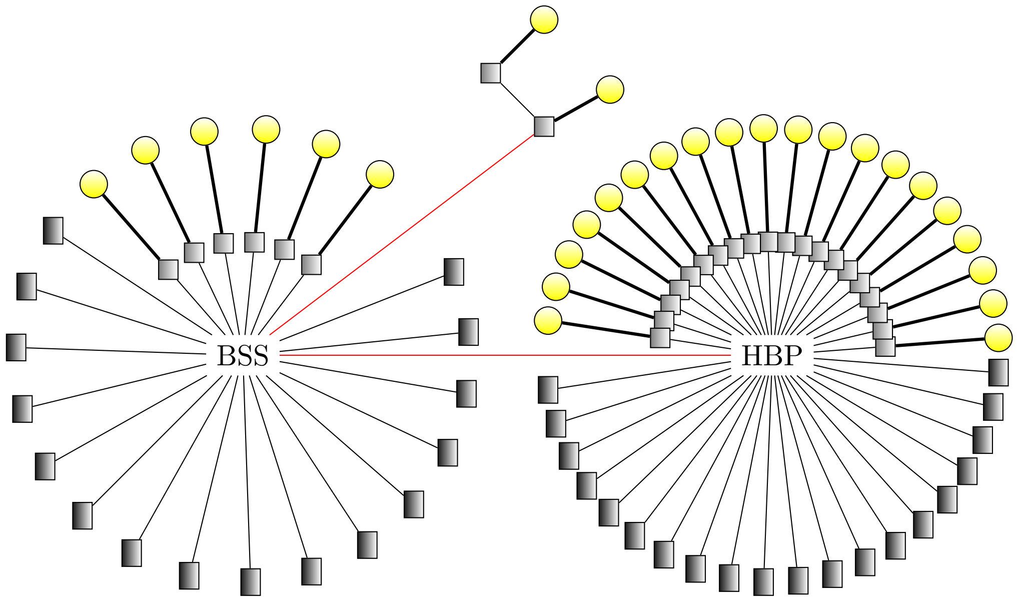


Figure 1.8.: Schematic overview of the data network architecture. Compute Nodes are plotted as rectangles, wafer module aggregation switches as squares and wafer module as yellow circles. The thick links between wafer module and wafer switches represent the 48×1GbE links between the FCPs and one wafer switch. All other lines represent 10GbE links: copper-based 10GSFP+Cu wiring within the BSS as well as HBP system, and fibre-based connections (red) between the three different locations. The centers of the two circles represent the backbone switches. The asymmetric configuration of the BSS network is due to a spatial split of the HMF: two wafer modules are located in the lab room (top) while the larger part of the BSS system is located in the server room. The NM-PM1 system is located in the ENI/HBP container building next to the KIP building (for details see figure A.3).

	#	Component
Wafer Switch	20	48-port 1GbE, $\leq$ 4-port 10GbE [Hewlett-Packard, 2014a]
Backbone switch	1	48-port 10GbE, 4-port 40GbE [Hewlett-Packard, 2014b]
Control switch	1	32-port 1GbE

Table 1.4.: Components of the NM-PM1 data network. Four experiment switches are located next to four wafer modules within one 19-inch rack. The control switch is used for remote management (e.g., power up or down, reset, remote KVM) of the compute cluster.

1.4. Summary

This chapter presented the HMF and NM-PM1 neuromorphic systems. The first section described the composition of a wafer module which has been designed by various hardware developers in Heidelberg and Dresden. Relevant hardware components, like the FCP, as well as basic hardware properties were introduced.

The second section presented the cluster architecture which was designed by the author. In particular, compute nodes, front-end and storage nodes of the NM-PM1 system were specified. Measurements of key performance indicators were presented.

The final section introduced the data network architecture which links wafer modules, compute nodes and all other network-accessible units.

The conventional part of the HMF and NM-PM1 systems has been assembled and is ready for use. In particular, the two prototype wafer modules are integrated into the dedicated data network and can be accessed from the cluster. The interconnection between the HMF and NM-PM1 installation sites is in service and the compute resources are used for prototype operation and conventional neuronal network simulations.

Regarding the AnaRMs, the author suggests to switch from the USB 2.0-based to a 1GbE-based interface. This allows to omit the AnaRMANs which, in turn, reduces complexity and increases robustness.

2. Fast Operation

Operating an accelerated neuromorphic hardware system opens up opportunities that conventional hardware cannot offer. Conveying the hardware's speed-up factor (cf. section 1.1.5) to the user is the key challenge that validates the main advantages of an accelerated neuromorphic system: power efficiency, fast parameter space exploration and long-running experiments. This chapter presents the components that are needed to accomplish this task.

The first part (section 2.1) describes the communication aspect. After an introduction, the implementation used in the Hybrid Multiscale Facility (HMF) and NM-PM1 systems is described in detail. Finally, performance measurements of the communication layers are presented.

The second part (section 2.2) presents the software environment. In the last part (section 2.3) management of system resources is investigated.

Aspects of hybrid operation, i.e. the real-time interaction between neuromorphic and conventional compute systems, are covered in chapter 3. This chapter only covers topics that are relevant to the class of self-contained experiments. That means experiments whose input data is completely pre-calculated and the *job* is not interacting with software running on the host computer. In high-performance computing (HPC) terms, this kind of setup is called *batch-style*.

Figure 2.1 presents an overview of the NM-PM1 network architecture. Once completed, the 20-wafer system will contain over 1000 network-accessible components (i.e. components carrying internet protocol (IP) addresses).

2.1. Communication Protocols

Links between user software and neuromorphic systems use varying technologies: smaller systems, like the Spikey[1] system [Pfeil et al., 2013], use USB 2.0 by now[2]. In the address-event representation (AER) community, dedicated USB 2.0 converter boards can be used for host connectivity [Berner et al., 2007]. In the pre-USB 2.0 era, due to lack of alternatives, older systems used proprietary host interfaces that caused additional development efforts [Schürmann et al., 2002]. Physically larger systems render direct links impossible as USB 2.0 limits the cable length to maximally 5 m [USB 2.0, 2000]. In general, the USB 2.0 connection technology does not support switched networks. This is especially of interest when connecting

[1]Spikey is a chip-based neuromorphic system developed during FACETS that implements 384 LIF neurons and approximately 100k synapses

[2]An evaluation of USB 2.0 from the neuromorphic point of view can be found in Merolla et al. [2005].

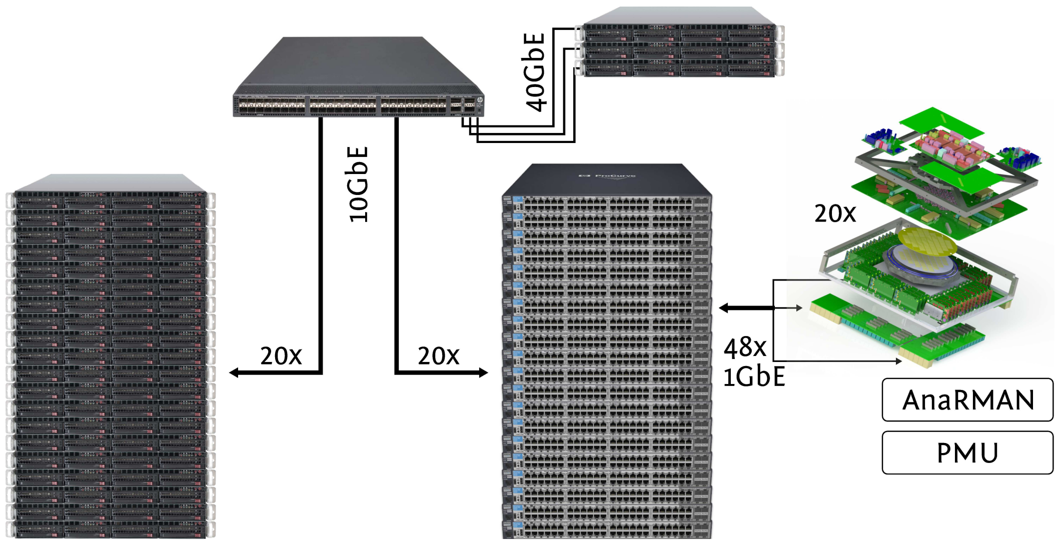


Figure 2.1.: The network topology of the NM-PM1 system: 20 compute nodes (left stack) and some servers (top right) are connected to the backbone switch (at the top; [Hewlett-Packard, 2014b]). The server links use 40GbE, the other links are based on 10GbE. Every wafer switch aggregates 48 1GbE links of a single wafer module. I.e. 20 wafer switches handle 20 wafer modules (right stack). Two more accessible units are associated with every wafer module: the power management unit (PMU) and the analog readout module aggregator node (AnaRMAN). The former is used for power control and system monitoring, the latter aggregates twelve analog readout module (AnaRM) used for analog recordings (e.g. neuron membrane traces).

links of different speeds to aggregate bandwidth. The latter is a key element when building installations consisting of multiple systems.

Hence, large systems typically resort to standard networking technologies like Ethernet (e.g., [Furber et al., 2012]). The HMF and NM-PM1 systems both use 1GbE as connection technology. The prevalence of 1GbE technology offers low costs paired with a long-range upgrade path (in terms of performance); this is the key argument for using it as host interface. Before introducing the protocol stack, the next paragraph introduces into the concepts of network protocol suites.

Protocol Suite Figure 2.2 (left) presents the famous OSI model. It is a conceptual model which groups protocol functionality into a hierarchy of layers. Interfaces are only provided to adjacent layers, a property that has been subject to discussions [RFC3439, 3].

Compared to the OSI model, the TCP/IP v4 protocol (see figure 2.2 center) or Internet protocol suite is not as structured. Information from lower layers is often used in higher layers: for example, packet fragmentation happening on the IP layer requires interaction with the TCP layer. Notably, the TCP/IP model does not specify any physical layer [RFC675; RFC791].

However, both models can be compared – some layers provide similar functionality. The OSI layers five and six are not directly represented in the TCP/IP model and their functionality is mostly covered by the TCP/IP application and transport layers.

In the following, the *transport layer* is the core interest. We use the following definition: the transport layer is responsible for end-to-end communication channels that provide an ordered data stream, provides means against data loss and corruption, and also provides flow control. The latter is needed when a slow receiver cannot cope with the inbound data rate.

IPv4 Layers At the time of writing, the internet protocol version 4 (IPv4) is still the dominant internet protocol. Due to address range limitations[3], the global network community is in a slow transition towards IPv6 which also provides other improvements[4]. Nevertheless, the current FCP FPGA[5] implementation stays with IPv4 for the meantime.

Figure 2.3 shows the basic data layout within the IPv4 frame header. In the following, only the destination and source address fields are of interest and the other fields will not be described (for further details see RFC791). The address fields are used for the communication endpoints, i.e. compute nodes and FPGA communication PCB (FCP) FPGAs. Other network-accessible units are Raspberry Pis (Raspberry Pis) used for wafer power management (cf. section 1.1), and the

[3]Internet protocol version 4 (IPv4) provides 32 bit addresses, i.e. approximately $4 \cdot 10^9$ addressable network units.

[4]For example, other improvements are: builtin-support for virtual private networks (VPNs) or, more precisely, network-layer encryption, privacy and simplified extensibility.

[5]FPGA communication PCB (FCP), field-programmable gate array (FPGA)

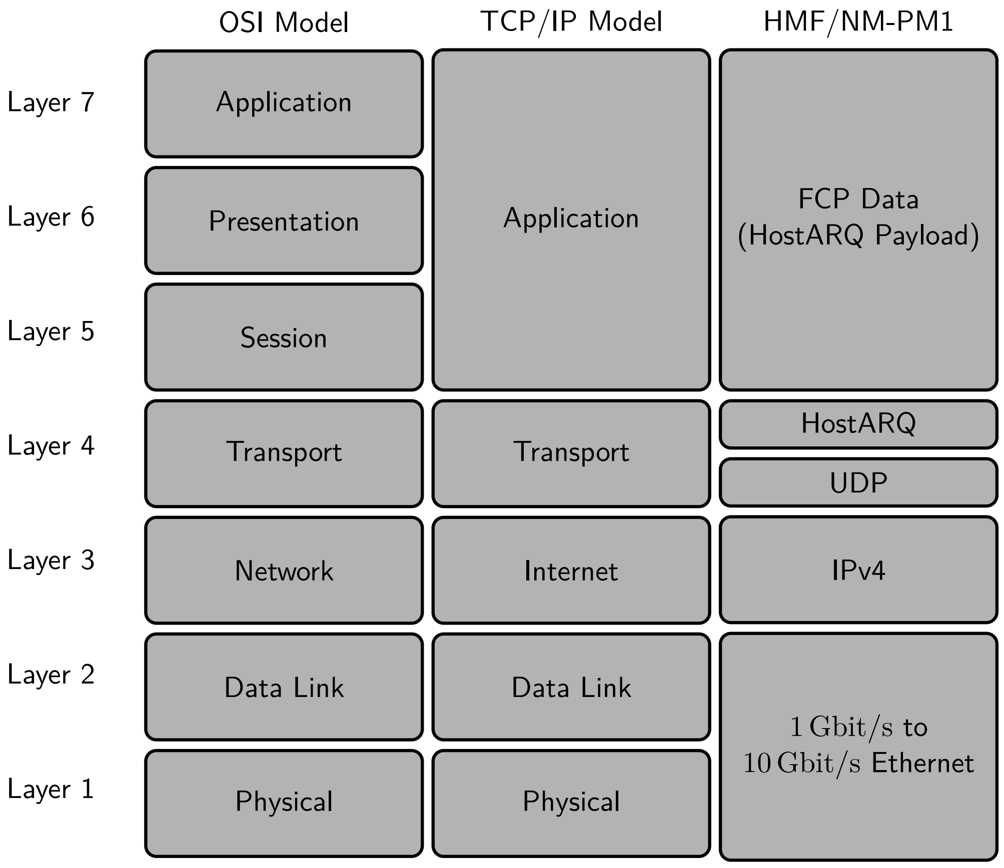


Figure 2.2.: The schematic shows communication protocol models and the implemented layers on the HMF/NM-PM1. **Left:** OSI model according to [ISO/IEC 7498-1:1994, 1994]. **Center:** TCP/IP 5-layer model according to [Tanenbaum and Wetherall, 2010]. **Right:** the protocol stack of the HMF and NM-PM1 systems. In the following sections, a custom transport layer protocol is presented – the HostARQ protocol. In the HMF and NM-PM1 systems, the layers above this layer (called application layer in the TCP/IP model) also rely on custom formats (see section 2.1.1.1). The custom HostARQ and the standard UDP protocol together form a transport layer.

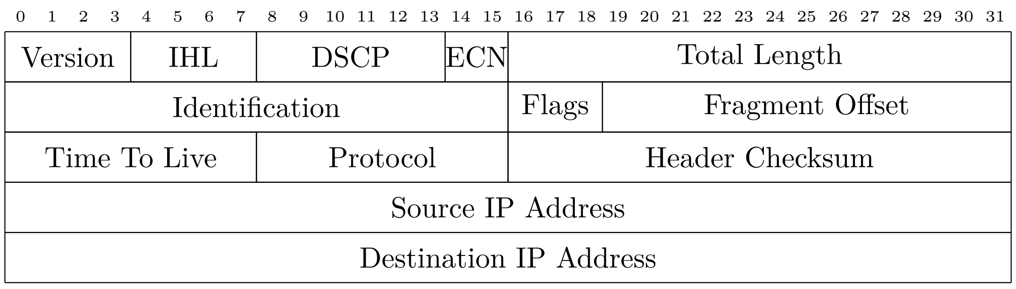


Figure 2.3.: Standard IPv4 header [RFC791] (most significant bit is marked as 0 (protocol notation)). The most important fields are *length* encoding the length, *protocol* encoding the type of the following payload, *header checksum* to verify correctness of the header, and *source* and *destination* IP addresses. All communication protocols between hosts and the neuromorphic BSS and HBP systems use this header.

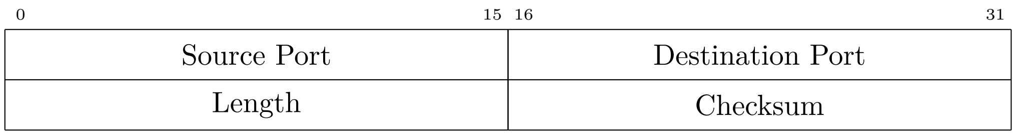


Figure 2.4.: Standard UDP header [RFC768] (most significant bit is marked as 0 (protocol notation)). As Ethernet checksumming renders the UDP checksum redundant, no *checksum* is used for the communication between neuromorphic systems and hosts. The *port* encodes the communication channel; in particular, the type of communication is encoded in the FPGA-side port (for example reliable, unreliable, or debug communication). Packet *length* is also provided by this header.

analog readout module aggregator nodes (AnaRMANs). The 20-wafer NM-PM1 setup will contain about 1000 addressable units within this network – mostly FCP FPGAs.

In general, all communication between FCP FPGAs and compute nodes employs IPv4 and UDP headers. The UDP header, shown in figure 2.4, is used for distinguishing between different sub-channels of one communication channel using different *port* numbers. For example, when communicating with the FCP FPGA different communication methods can be used: the JTAG protocol for debug purposes, one reliable communication channel that will be presented in the following sections, and an unbuffered interface for real-time interaction between the compute node and the neuromorphic system. The latter will be presented in chapter 3.

2.1.1. HostARQ

Starting from the requirements of a reliable and fast communication channel between host computer (or compute node) and HICANN chip it is obvious that for all unreliable communication channels some kind of transport-layer protocol is needed.

In particular, reliable configuration of the neuromorphic system is essential to facilitate the calibration of the inherently non-digital parts: analog neuron circuits, the so-called dendrite membrane circuits (DenMems), which always react in slightly different ways to the same input. To make this trial-to-trial variability only depend on the actual variable parts (like the analog circuits and the analog parameter storage), the communication channels have to be reliable. In the Internet ecosystem, the IPv4 protocol suite is used to cover similar requirements. For example, the transmission control protocol (TCP) provides connection-oriented, reliable communication channels; it also provides support flow control among other features. Unfortunately, the feature list comes along with a rather complex specification [RFC1122; RFC1323; RFC1379; RFC1948; RFC2018; RFC4614; RFC5681; RFC6298; RFC675; RFC6824; RFC793]. Hence, the FPGA developers [Partzsch et al., 2008–2014] decided to use a simpler transport protocol. General speaking, the main objective – reliable communication – is provided by the class of automatic repeat request protocols [Peterson and Davie, 2003; RFC3366].

The first FCP FPGA implementation utilized a simple stop-and-wait mechanism. This means, the host sends a data packet to the FPGA and waits for an response that acknowledges the successful data transmission. Figure 2.5 illustrates the protocol. For the back-channel the same mechanism could be used.

In reality, the FPGA firmware did not use any error correction mechanisms for back-channel data as it was assumed that the host buffers are large enough and data corruption is improbable on standard 1GbE hardware – at least for a first implementation. However, performance-wise this protocol implementation is unsatisfactory.

To estimate the maximum performance of this protocol we need an estimate for the round-trip time for data exchange between two compute nodes. Figure 2.6 presents 10^5 binned latency measurements plotted in a histogram. Details about the measurement can be found in section 3.3.3. The mean round-trip time (RTT) is 198.0 µs (root-mean-square error (RMSE) is 6.4 µs).

Given a typical RTT of $\sim$ 200 µs between two communication partners using 1GbE and a socket-based implementation, the maximum throughput can be calculated: maximum raw packet size of 1GbE is $S_{\mathrm{RAW}} = 1538\,\mathrm{B}$ per default (i.e. without support for *jumbo frames*[6]). Figure 2.7 illustrates an Ethernet packet. As we use IPv4 (20 B) and UDP (8 B) headers for all communication with the FCP FPGA, the effective payload (S_{UDP}) reduces from $S_{\mathrm{MTU}} = 1500\,\mathrm{B}$ to 1472 B. Now we can calculate the maximum payload throughput PT as follows:

$$
\begin{aligned}
S_{\mathrm{UDP}} &= 1472\,\mathrm{B} \text{ (maximum UDP payload size)}\\
T_{\mathrm{RTT}} &= 200\,\mu\mathrm{s} \text{ (mean round-trip time)}\\
PT = S_{\mathrm{UDP}} \cdot \frac{1}{T_{\mathrm{RTT}}} &= 1472\,\mathrm{B} \cdot \frac{1}{200\,\mu\mathrm{s}} = 7.0\,\mathrm{MiB/s}.
\end{aligned}
\tag{2.1}
$$

[6]Some 1GbE implementations also supports larger MTU sizes, i.e. $S_{\mathrm{MTU}} > 1500\,\mathrm{B}$.

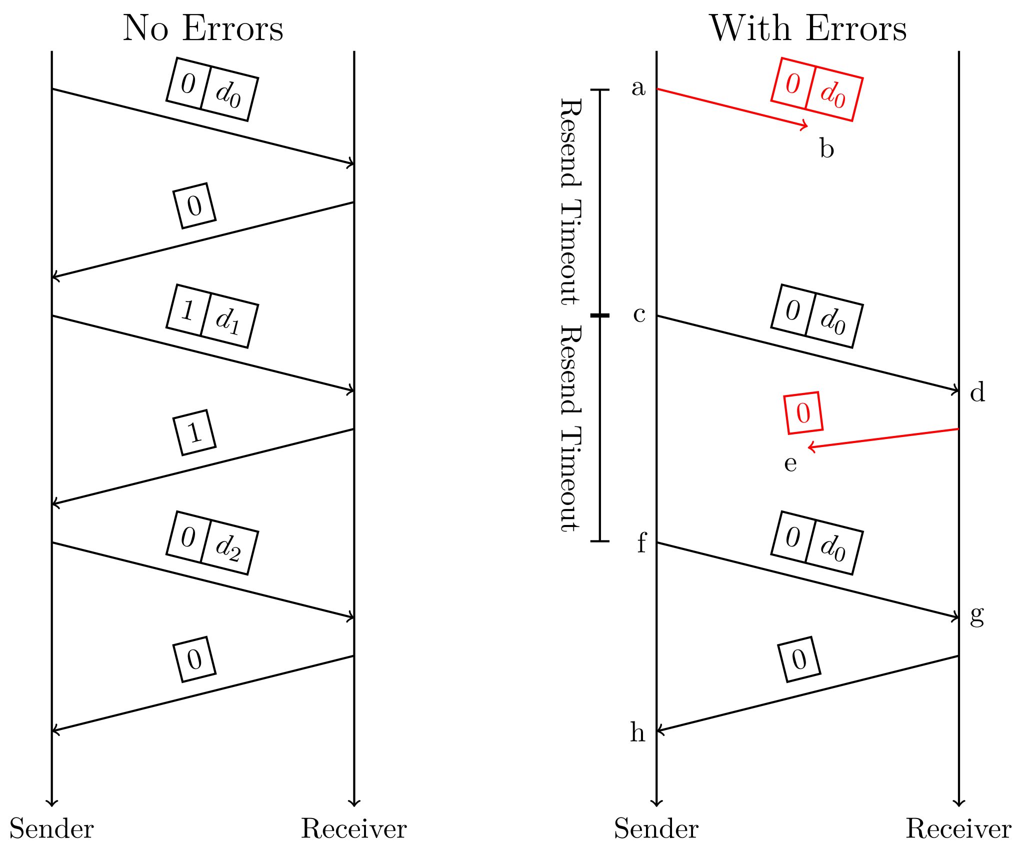


Figure 2.5.: Stop-and-wait transmission schematic. **Left:** Communication example without drops. The sender sends three packets with payloads d_0, d_1, d_2. Every packet is acknowledged by the receiver. The packets are marked for the receiver to differentiate between *new* and *resent* packets. **Right:** Communication with drops: a) the sender transmits a data packet with payload d_0 marked with `0` (1-bit marker); b) this data packet is lost; c) after the resend timeout, the sender re-transmits the packet; d) the receiver receives the packet and sends an acknowledgment carrying the packet marker 0; e) the acknowledgment packet is lost; f) after the resend timeout, the sender re-transmits the packet; g) the receiver acknowledges again; h) upon reception of the acknowledgment carrying the current marker the sender is allowed to send the next packet marked with $\mathrm{x}_{n+1} = (\mathrm{x}_n + 1) \mod 2$. If the resend timeout would be too short, for example smaller than the typical time until the ACK is received, multiple sends would occur even in cases without packet loss. If the resend timeout is too large, the resend is triggerd too late and the drop recovery time increases – the throughput is suboptimal. This schematic can be easily adapted for duplex communication: if the *remote* partner wants to send data, it can add its own marker and data. The data packet header has to be adapted to carry *remote marker*, *own marker* and *data*; acknowledgment frames remain the same.

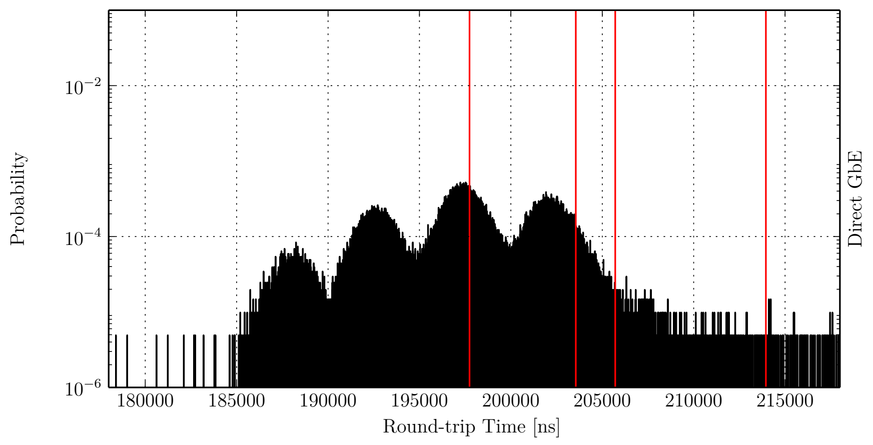


Figure 2.6.: Histogram of UDP-based *ping* latencies measured between two NM-PM1 compute nodes using directly-linked 1GbE. The ordinate has been normalized to show bin probability. The vertical red lines represent from left to right: average, 95%-percentile, 99%-percentile and 99.9%-percentile out of all measured RTTs (10^5 entries). For measurement details see section 3.3.3. Bin size was set to 1 ns – which is also the timer resolution.

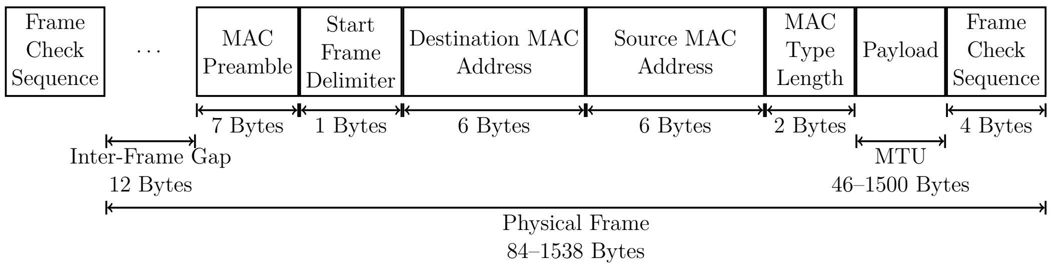


Figure 2.7.: Schematic view of an Ethernet packet. It consists a of MAC preamble (7 B), start frame preamble (1 B), source and destination MAC addresses (6 B each), MAC type or length (2 B) field, the payload (1500 B, also called MTU size), CRC checksum (4 B) and a 12 B inter-frame gap. There is another, optional field for Ethernet-level VLANs located between MAC source address and MAC type field.

1GbE has a specified bitrate of 1 Gbit/s = 125 GB/s. It translates to an payload throughput of:

$$PT_{\mathrm{max}} = \mathrm{Bitrate}_{\mathrm{GbE}} \cdot \frac{S_{\mathrm{UDP}}}{S_{\mathrm{RAW}}} = 125\,\mathrm{MB/s} \cdot \frac{1472\,\mathrm{B}}{1538\,\mathrm{B}} = 119.6\,\mathrm{MB/s} = 114.1\,\mathrm{MiB/s}$$

This means we can utilize only up to around 6% of the maximum 1GbE payload bandwidth PT_{max}. We can solve this problem by introducing a packet counter that can be used to individually identify packets and to send multiple, up to N – the window size – packets. The previous stop-and-wait automatic repeat request (ARQ) protocol is a special case using a window size of 1. On the receiving side we track the last packet that has no gaps in front, i.e. when a packet is dropped we track the last frame before the drop. The receiving side now acknowledges this tracked packet to the sender but it continues to fill its receive buffer with later packets within the window. Eventually, the sender will timeout because the acknowledgments *stall* and trigger a resend of the packet just behind the last acknowledged packet. The receiver will accept the previously dropped packet and acknowledge a much later packet in the window – the transmission can now continue. This ARQ-type is called selective-repeat-ARQ protocol [RFC3366]. Figure 2.8 presents the protocol behavior.

For illustration, we can look at the sending and receiving buffer filling levels. In this case, we represent the packet identifiers as ring buffer entries[7]. Because we want unique identifiers at all times, we use a identifier that is 1 bit longer than the number of bits we need to represent the maximum packet size. This eliminates all aliasing problems, as we cannot send more than half of the identifiers at once. Thus, in case of a window size of 4 we need 8 identifiers which are represented in 8 buffer entries. Figure 2.9 displays the filling levels of the sending and receiving side. At the beginning, both buffers are *empty*, i.e. we can send up to window-size packets. After one packet has been pushed into the sending window, the remaining buffer size is reduced by one (represented as white area in the schematic). On the other side, the receiver pushed the received packet into the receiving buffer. Eventually, it acknowledges the packet and a buffer entry is unlocked – the receiving window is empty again. The original sender receives the acknowledgment and also clears his sending window.

To allow for a more efficient back-channel, we allow data being sent *piggy-back* to the acknowledgment packets, thereby combining both communication directions into a single frame format. Figure 2.10 depicts the HostARQ protocol (HostARQ) frame format. An overview of the implemented protocol suite is shown in figure 2.2 (right column).

For faster software handling, all data fields are aligned to 32 bit words. The acknowledgment field is at the start of the frame to allow faster ACK-field updates – a negligible effect given a packet size of approximately 1500 B; the field order originates from the HICANN-ARQ protocol where the small packets (48 bit) and

[7]A real-world implementation would limit the number of buffer entries to the window size.

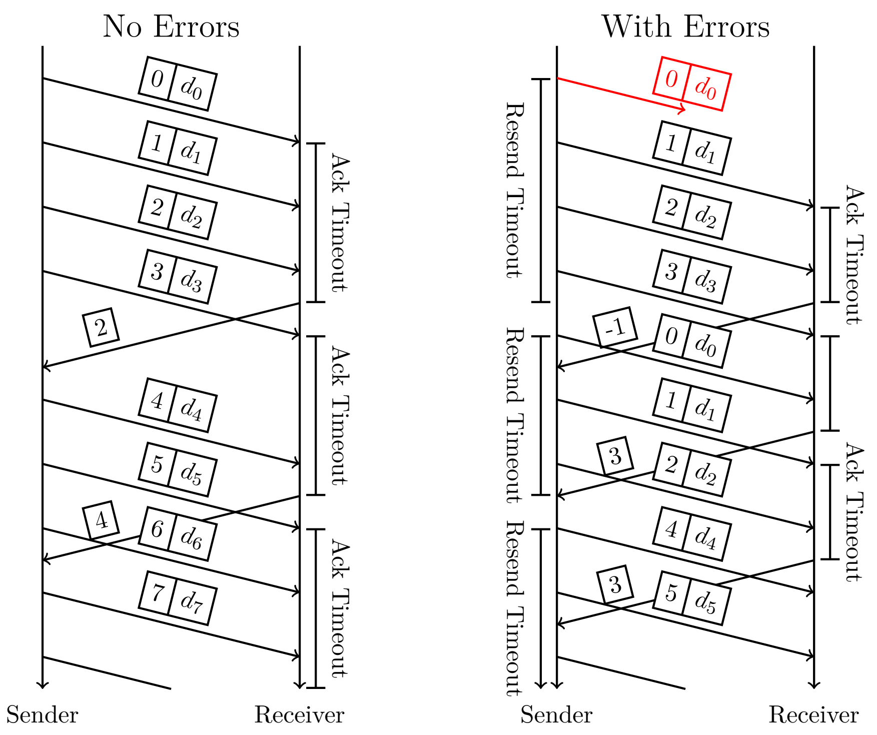


Figure 2.8.: Selective-repeat ARQ transmission schematic using window size 4. The packet *marker* uses 3 bit to unambiguously encode the sequence and acknowledgment numbers ($N_{\text{seq,ack}} > N_{\text{window size}} \cdot 2 - 1$, cf. figure 2.9). **Left:** the sender fills the 4-packet window with packets 0–3, then it waits for an acknowledgment. After the ACK timeout triggers, the receiver acknowledges up to the last contiguously received packet which is packet 2. The sender waits for the ACK and, after reception, it can now send three more packets 4–6 because three packets (0–2) have been acknowledged. A stable state has now been reached. The sender can now send new packets at minimum inter-packet time span. Hence, it achieves optimum sending performance, i.e. wire speed. **Right:** the first data packet gets lost; the sender resends the window after the resend timeout expires. Due to the delayed ACK, the sender retransmits packets 1 and 2. After reception of the next ACK for 3, the sender can shift its window and continue with new data (4 and 5). The illustration ends just before the receiver would trigger an ACK for data packet 5. Full duplex operation can be added in the same way as it can be done in figure 2.5. As described in figure 2.5, too small resend timeouts increase the number of redundant transfers whereas too high resend timeouts slow down the recovery in case of packet loss. To optimize support for back-channel/return data (i.e. data flow from *receiver* to *sender*), the ACK timeouts are introduced. If no return data is available on the receiving side, an ACK timeout waits for potential return data. Otherwise, an ACK-only frame is sent.

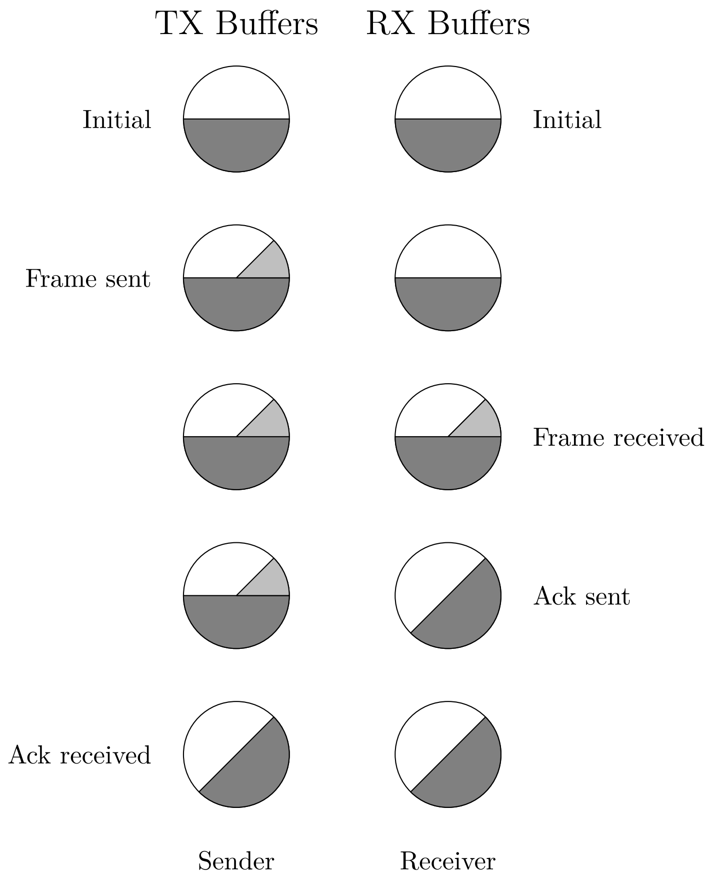


Figure 2.9.: Sliding window schematic for window size 4 and maximum sequence number 8. For simplification purposes, the total number of buffer entries equals the sequence size. A real-world protocol implementation needs only as many buffer entries as packets can be legally *in flight*, i.e. the window size. The circles represent the status of circular buffers containing data of a given sequence number (i.e. 0–7). Empty slots are plotted in white, illegal areas in gray; *active* data is plotted in light gray: on the transmitting side, this represents data that has not been acknowledged yet; on the receiving side, this represents data that has been received but not yet acknowledged. 1. in the initial state the transmission and receive windows are empty; 2. one packet is pushed by the sender which decrements the number of available `TX` slots; 3. the receiver accepts the packet, thus reducing the number of available `RX` slots – the receiver now handles/consumes the data; 4. the receiver acknowledges this packet, rendering the `RX` window empty again; 5. the sender receives the acknowledgment and restores the maximum number of transmission slots. In a typical use case, the sender pushes multiple frames until the corresponding acknowledgments are received.

0	32	48 63
Acknowledge Number	Sequence Number	
Marker Bits	Payload Type	Length

Figure 2.10.: HostARQ header (most significant bit is marked as 0 (protocol notation)). `Acknowledgment` and `sequence` number are used for the selective-repeat-type ARQ implementation (called HostARQ); The `marker` field is currently only used to flag packets that carry valid data (i.e. non-ACK-only packets). Another application would be to explicitly inform about *traffic congestion*, which is currently implicitly implemented (by detecting drops) in the software HostARQ implementation. `Type` and `Length` encodes payload data type and the number of 64-bit sized payload entries. The payload types are described in section 2.1.1.1.

short window sizes (16) urge for a maximally optimized field order. In the current FPGA implementation the packet identifiers are encoded using 16 bit; the larger field sizes have been chosen to optimize memory alignment and for even larger window sizes that are currently limited by the FPGA implementation. The *valid bit*-field currently encodes only the valid bit to mark a valid piggy-back payload; additional packet markers are under development, e.g., to mark traffic congestion. To differentiate between different payload types and payload lengths the next two 16 bit fields are used.

2.1.1.1. HostARQ Payload

All HostARQ payload is aligned to 64 bit. In the later sections the unit cmd refers to a single 64 bit payload entry. The types are encoded in the HostARQ header (figure 2.10) as well as the length given in number of entries (i.e. number of 64 bit entries). Hence, the payload type does not change within one data frame. If the data type switches, the current packet has to be closed and a new packet has to be opened.

Types At the time of writing there are four different packet types:

`0x0C5A` spike data for on-FPGA playback memory;

`0x0CA5` spike data from FPGA recording memory;

`0x0C1B` FPGA configuration data;

`0x2A1B` HICANN configuration data.

Details can be found in HBP SP9 Specification [2014][8].

[8]The specification uses an unusual bit-numbering scheme – high numbers (on the left) denote the most significant bit (MSB). The RFCs use the starts-left-with-0-and-means-MSB style.

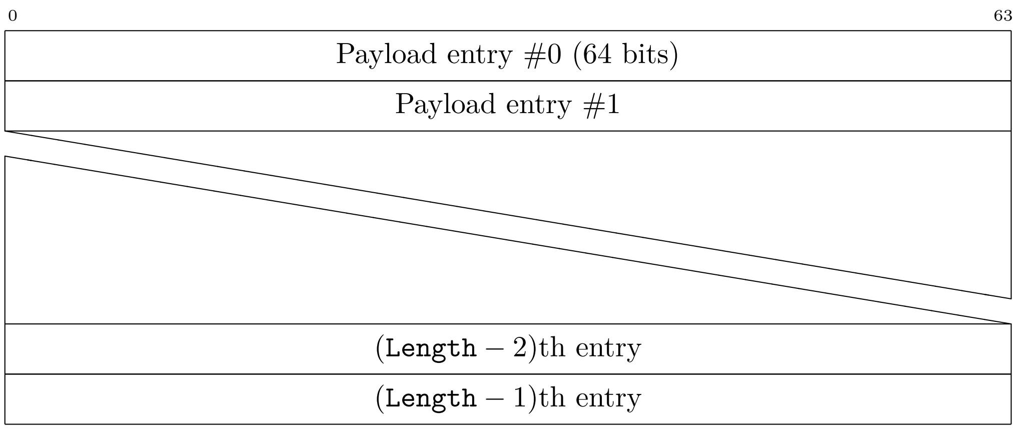


Figure 2.11.: HostARQ payload is aligned to 64 bit. The payload type and length (number of 64 bit entries) is encoded in the HostARQ header, see figure 2.10.

2.1.1.2. Software Implementation

The software implementation of the HostARQ protocol is based on the SlowControl transport protocol that was developed for the FACETS Stage 1 system by Moritz Schilling under the supervision of the author (cf. [Schilling, 2010]). It is a selective-repeat-type ARQ protocol providing some more advanced features (cf. section 2.1.1.2).

The general goal was to implement a rigorously fast transport protocol implementation even though the main target was only 1GbE [Drexler, 2009; Gutmann, 2007]. To achieve maximum performance the key components of the transport layer implementation were evaluated in detail. The implementation of a sufficiently fast application programming interface (API) required a general analysis of different inter-process communication (IPC) mechanisms. Figure 2.12 provides machine-local throughput measurements of common UNIX IPC mechanisms performed with varying block sizes. Given typical packet sizes of 1500 B, the shared-memory-based method achieves more than 1 GiB/s. Based on that evaluation, the implementation relies only on shared-memory as an IPC mechanism. However, the HostARQ software implementation makes additional efforts to further increase the throughput limit. For example, the packet handling uses a cache-line-aligned buffering scheme that allows for process-local buffering of multiple packets. This increases the transfer block size and, by looking at figure 2.12, therefore the throughput. the current implementation uses 8-packet *caches* which correspond to 12 kB.

The tasks to receive data and send data are relatively independent. Hence, the design uses three threads for sending frames, receiving frames and retransmitting of the non-acknowledged packets. However, some overlap exists in the common need for the sequence and acknowledge counters. Measures to guard against data corruption have to be taken: a fast method to provide mutual exclusion between concurrently running jobs is to use fast userspace mutexes (futexes). Franke et al.

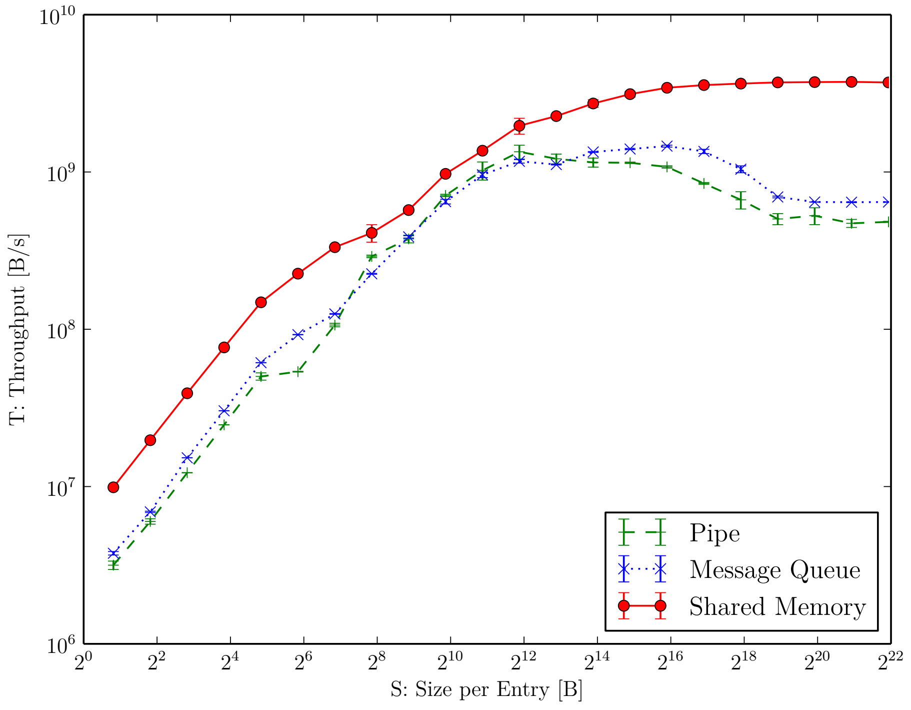


Figure 2.12.: Throughput measurement for multiple IPC methods and varying block sizes. Typical UNIX IPC mechanisms are tested: the pipe mechanism which uses unidirectional data channels, the message/packet-based communication mechanism and the shared-memory-based method. For all tested entry sizes the shared-memory-based communication mechanism outperforms all other methods. The $T \geq 1\,\mathrm{GiB/s}$ barrier is reached for entry sizes larger than $S \geq 2^{10}\,\mathrm{KiB} = 1\,\mathrm{KiB}$. Throughput saturation occurs well below entry sizes of $S = 1\,\mathrm{MiB}$. The measurement has been performed by Moritz Schilling under the supervision of the author (see Schilling [2010] for details); it was performed on Debian Wheezy, Intel i7-920 CPU, 12 GiB DDR3-1066 RAM.

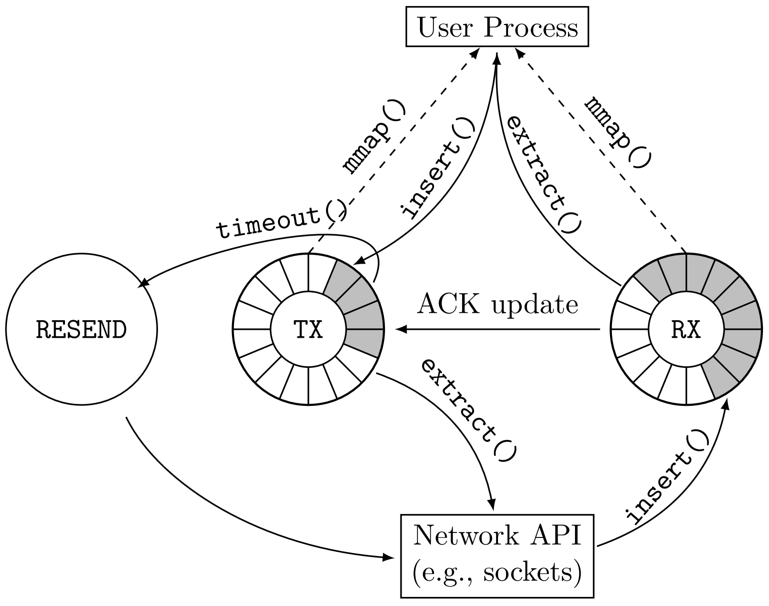


Figure 2.13.: Overview of the HostARQ software implementation. The user's process maps transmission and receive buffers into its own virtual address space (i.e. using shared memory). To transmit data the user inserts data into one or multiple `TX` buffers and marks them as ready-to-send. The receiving side can either use a non-blocking interface to check for new data or a blocking interface to wait for new data. The `TX` process handles the sending of frames after the user pushed them into the window; on the receiving side, the user can pop data from the `RX` window. A third thread handles retransmissions in case of data loss (see figure 2.8 for a schematic describing the protocol).

[2002] provides performance measurements on different mutex implementations and indicates supreme performance for the latter. Other optimizations that were considered are memory-layout optimization to eliminate false sharing of cache lines [Bolosky and Scott, 1993], and zero-copy support down to the network interface. A simplified structure of the implementation is shown in figure 2.13.

Advanced Features The software implementation provides several advanced features. First of all, there is support for adapting the RTT during protocol operation. That means, when network load increases and the communication latencies jump up, the HostARQ software adapts its resend timeout value to match the new latency. This is implemented to reduce unnecessary retransmissions when latency increases. Then again, it also decreases the timeout when the RTT decreases after a load spike. We use the RTT estimation mechanism by Ljung and Soderstrom [1983]; RFC793.

For systems having asymmetric network bandwidths like the HMF or NM-PM1, another crucial feature is to handle packet loss due to non-matching bandwidths separately. A single FCP FPGA supports only 1GbE, on the other hand the compute node supports 10GbE. This discrepancy in bandwidth can potentially lead to massive packet loss towards the FPGA if no control mechanism reduces the sending rate of the compute node. Packet loss of that kind is not trivial to handle because of the relatively long time scale it can last due to the large buffer sizes of contemporary network equipment. The sender needs some kind of smooth rate adaptation mechanism. In Internet terms this is called *congestion control* and we implemented it as given by Jacobson and Karels [1988]; RFC896.

From profiling using Valgrind-based Profiling Tool (Callgrind)[9] it became clear that the traditional UNIX socket-based access to the kernel networking layer is one bottleneck. Thus, a version was implemented that uses Linux kernel's extension to linux NIC driver framework for improving performance of high-speed networking [Kelly and Gasparakis, 2010], more precisely the `RING_TX` and `RING_RX` ring buffers that can be mapped into userspace. The HostARQ software now operates on a memory area that is partitioned into max-packet-sized elements. This fits very well into the concept of HostARQ software because of the same internal representation of data. Hence, we can replace the internal buffers by `mmap()`ed `RING_TX`/`RING_RX` buffers, i.e. remove the syscall to send or receive, and thereby eliminate additional context switches and most of the overhead which occurs at that point. For implementation details see Schilling [2010].

During implementation it also became evident that a graphical tool for debugging the data flow is beneficial. The author implemented a Wireshark packet analyzer-based packet dissector that supports the HostARQ protocol and most of the payload types. Figure 2.14 shows Wireshark dissecting a HostARQ packet that carries `FPGAPLAYBACK/0x0C5A` payload.

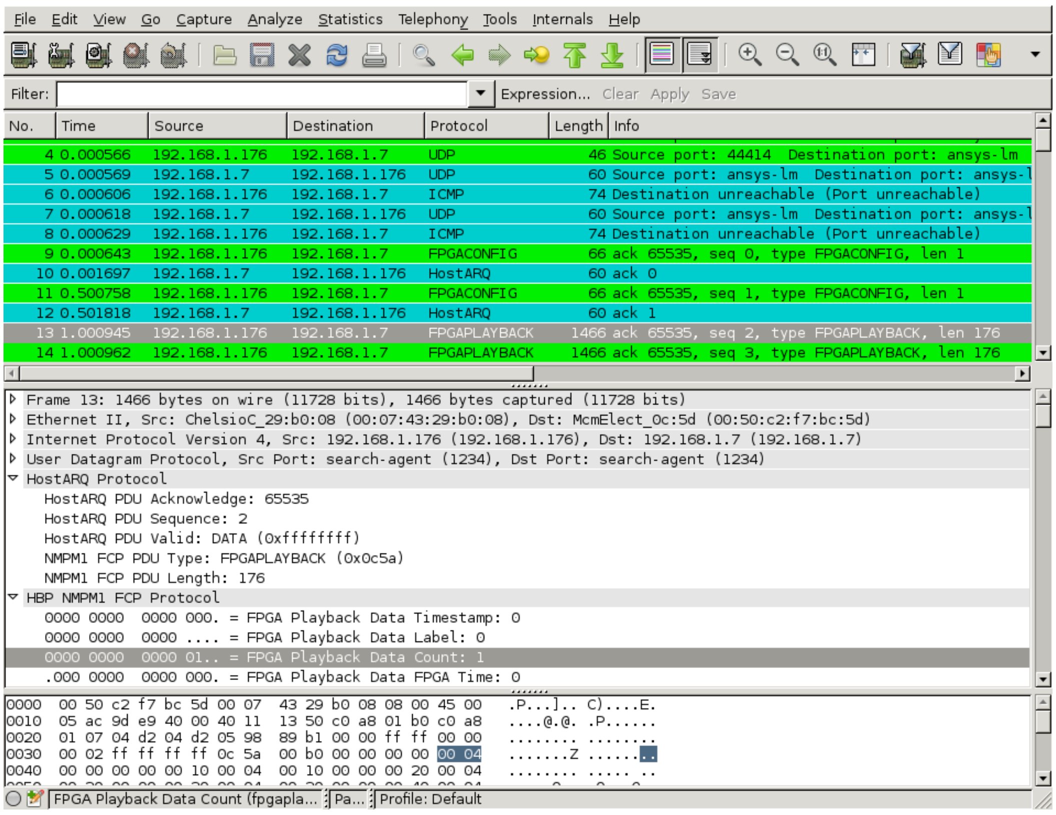
2.1.1.3. FPGA Implementation

The FPGA implementations were carried out by Vitali Karasenko under the supervision of the author. Figure 2.15 presents a simplified overview of the FCP FPGA modules. The network module (MAC/UDP), data processing and inter-FPGA modules were implemented by Partzsch et al. [2008–2014]. The figure omits the handling of the connection to the eight HICANN chips; this link uses a ARQ implementation, the so-called HICANN-ARQ, which is similar to the HostARQ link. See Karasenko [2014] for details.

2.1.2. Performance Measurements

This part of the document presents measurements that evaluate the HostARQ software and FPGA layers. The corresponding layers are described in section 2.1, esp. in sections 2.1.1 and 2.1.1.3. The effective *speed* of the NM-PM1 system is not solely defined by the acceleration factor (section 1.1.5) of the neuromorphic

[9]See [Weidendorfer, 2008] for a nice introduction.


File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Filter:
Expression... Clear Apply Save

```
No.  Time       Source         Destination    Protocol      Length Info
   4 0.000566   192.168.1.176  192.168.1.7    UDP               46 Source port: 44414  Destination port: ansys-lm
   5 0.000569   192.168.1.7    192.168.1.176  UDP               60 Source port: ansys-lm  Destination port: ansys-l
   6 0.000606   192.168.1.176  192.168.1.7    ICMP              74 Destination unreachable (Port unreachable)
   7 0.000618   192.168.1.7    192.168.1.176  UDP               60 Source port: ansys-lm  Destination port: ansys-l
   8 0.000629   192.168.1.176  192.168.1.7    ICMP              74 Destination unreachable (Port unreachable)
   9 0.000643   192.168.1.176  192.168.1.7    FPGACONFIG        66 ack 65535, seq 0, type FPGACONFIG, len 1
  10 0.001697   192.168.1.7    192.168.1.176  HostARQ           60 ack 0
  11 0.500758   192.168.1.176  192.168.1.7    FPGACONFIG        66 ack 65535, seq 1, type FPGACONFIG, len 1
  12 0.501818   192.168.1.7    192.168.1.176  HostARQ           60 ack 1
  13 1.000945   192.168.1.176  192.168.1.7    FPGAPLAYBACK    1466 ack 65535, seq 2, type FPGAPLAYBACK, len 176
  14 1.000962   192.168.1.176  192.168.1.7    FPGAPLAYBACK    1466 ack 65535, seq 3, type FPGAPLAYBACK, len 176
```

```
▷ Frame 13: 1466 bytes on wire (11728 bits), 1466 bytes captured (11728 bits)
▷ Ethernet II, Src: ChelsioC_29:b0:08 (00:07:43:29:b0:08), Dst: McmElect_0c:5d (00:50:c2:f7:bc:5d)
▷ Internet Protocol Version 4, Src: 192.168.1.176 (192.168.1.176), Dst: 192.168.1.7 (192.168.1.7)
▷ User Datagram Protocol, Src Port: search-agent (1234), Dst Port: search-agent (1234)
▽ HostARQ Protocol
    HostARQ PDU Acknowledge: 65535
    HostARQ PDU Sequence: 2
    HostARQ PDU Valid: DATA (0xffffffff)
    NMPM1 FCP PDU Type: FPGAPLAYBACK (0x0c5a)
    NMPM1 FCP PDU Length: 176
▽ HBP NMPM1 FCP Protocol
    0000 0000  0000 000. = FPGA Playback Data Timestamp: 0
    0000 0000  0000 .... = FPGA Playback Data Label: 0
    0000 0000  0000 01.. = FPGA Playback Data Count: 1
    .000 0000  0000 000. = FPGA Playback Data FPGA Time: 0
```

```
0000  00 50 c2 f7 bc 5d 00 07  43 29 b0 08 08 00 45 00   .P...].. C)....E.
0010  05 ac 9d e9 40 00 40 11  13 50 c0 a8 01 b0 c0 a8   ....@.@. .P......
0020  01 07 04 d2 04 d2 05 98  89 b1 00 00 ff ff 00 00   ........ ........
0030  00 02 ff ff ff ff 0c 5a  00 b0 00 00 00 00 00 04   .......Z ........
0040  00 00 00 00 00 10 00 04  00 10 00 00 00 20 00 04   ........ ..... ..
```

```
FPGA Playback Data Count (fpgapla... | Pa... | Profile: Default
```

Figure 2.14.: Screenshot showing Wireshark, an open-source network packet analyzer. The upper part shows a scrollable list of packets that can be filtered and sorted. In the lower half, the currently selected packet is shown in detail. In this case, the custom HostARQ dissector plugin shows a packet that carries `FPGAPLAYBACK` data.

circuits. It rather depends on everything that contributes to the total execution time of an experiment: preparation of the hardware configuration which involves multiple software layers, configuration of the neuromorphic hardware, the mostly negligible execution time on the hardware, and finally the retrieval of result data. As a first step, the connection between wafer modules and compute nodes will be evaluated. Later sections cover the other mentioned topics.

Figure 2.15 presents an overview of the components that are involved in this measurement.

***Wire* Speed** The maximum theoretical throughput of each FCP FPGA is limited by the protocol implementation. IPv4 addresses, UDP ports, checksums and the HostARQ header (data fields for *type* and *length* of the payload data) contribute to the overhead.

Using standard MTU (1500 B) and the protocol headers sizes ($\text{sizeof}(\text{IPv4}) = 20\,\text{B}$, $\text{sizeof}(\text{UDP}) = 8\,\text{B}$, $\text{sizeof}(\text{HostARQ}) = 16\,\text{B}$) the maximum payload size per frame is:

$$\begin{aligned}\text{sizeof}(\text{Frame}_{\text{payload,max}}) &= \text{sizeof}(\text{MTU}) - \text{sizeof}(\text{headers})\\ &= \text{sizeof}(\text{MTU}) - \text{sizeof}(\text{IPv4} + \text{UDP} + \text{HostARQ})\\ &= 1456\,\text{B}\\ &= 182\,\text{Cmd}.\end{aligned} \tag{2.2}$$

The number of commands (Cmd) refers to number of HostARQ payload entries (see section 2.1.1.1); all payload types are 64 bit large.

Given 1GbE symbol rate (1 Gbit/s), standard Maximum Transfer Unit (1500 B) and the corresponding maximum physical frame size[10] = 1538 B, the maximum throughput can be calculated:

$$\begin{aligned}\text{Throughput}_{\text{payload}} &= \text{Bitrate}_{\text{1GbE}} \cdot \frac{\text{sizeof}(\text{Frame}_{\text{payload,max}})}{\text{sizeof}(\text{Frame}_{\text{physical,max}})}\\ &= 947.9\,\text{Mbit/s}\\ &= 112.9\,\text{MiB/s}.\end{aligned} \tag{2.3}$$

HICANN configuration commands (cmd) consume 64 bit each. Hence, the maximal HICANN cmd-rate is:

$$\text{Throughput}_{\text{payload}} = 14.8\,\text{MCmd/s}. \tag{2.4}$$

These numbers represent the theoretical, maximal throughput. Later sections refer to these numbers when calculating efficiency.

[10] This includes MAC preamble, start frame delimiter, source and destination MAC addresses, type field, CRC and inter-frame gap; cf. IEEE 802.3ab [1999].

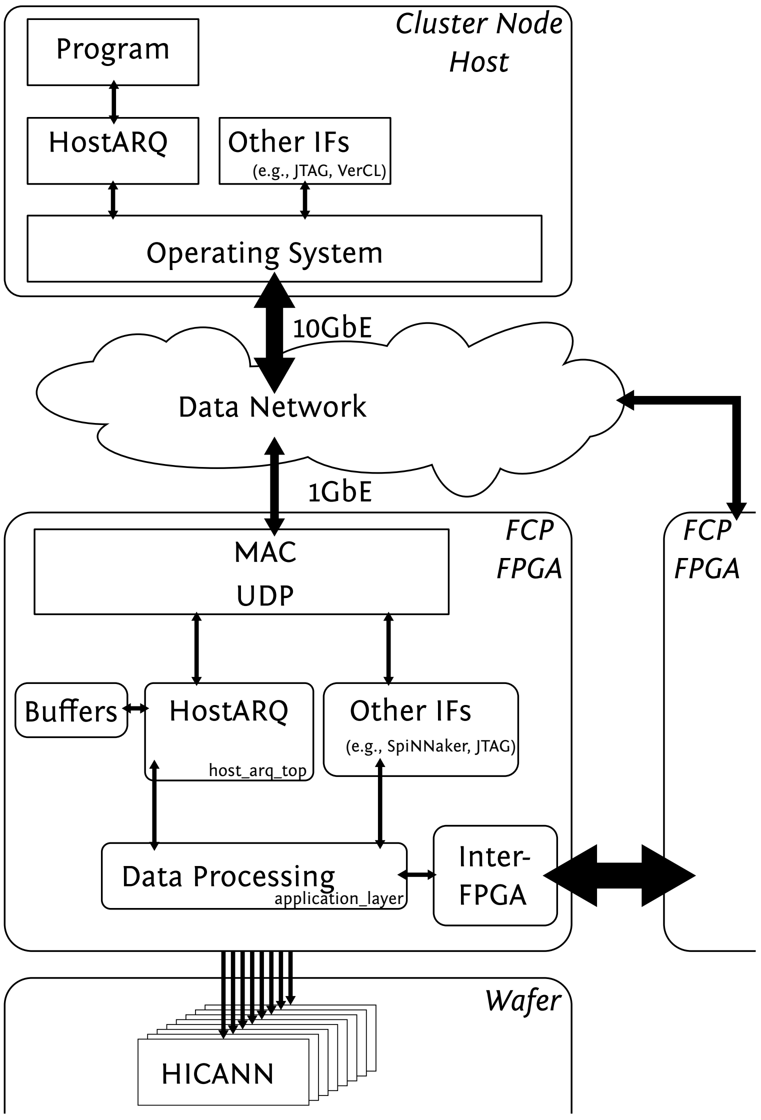


Figure 2.15.: The schematic provides an overview of the HostARQ tests. A test *program* communicates with varying remote endpoints: 1. in section 2.1.2.2 the connection between a test *program* and the *HostARQ* module in the FCP FPGA is tested; 2. in section 2.1.2.3 links from one compute node to multiple FCP FPGAs are used; 3. section 2.1.2.4 presents throughput measurements of data flowing down to one or more HICANN chips; 4. section 2.1.2.6 presents measurements between multiple hosts. On the compute node the test program interacts with the HostARQ software implementation which, in turn, uses the operating system for communication; within the FCP FPGA, modules for link-layer, HostARQ protocol, and buffer handling provide similar functionality. The data network is described in figure 1.8.

The Bandwidth-Delay Product The bandwidth-delay product can be used to calculate the link capacity between two protocol processing endpoints.

$$\text{Bandwidth} \times \text{One-way Delay} = \text{Link Capacity} \tag{2.5}$$

or:

$$\text{One-way Delay} = \frac{\text{Link Capacity}}{\text{Bandwidth}} \tag{2.6}$$

This means that given the raw bandwidth and one-way delay of a connection the link can only be used at optimal throughput if the protocol can efficiently fill up the link capacity.

In the case of HostARQ, the design goals are 1GbE bandwidth using delays in the order of 5 ms. Higher maximum latencies would further relax the software timing requirements, i.e. the protocol handlers could sleep longer. Performance-wise this would increase throughput stability in case of heavy-loaded systems. The mentioned number was selected as a trade-off between FPGA memory consumption and software timing constraints.

The current FPGA implementation uses sizeof(window) = 512 Frames. Using equation (2.6), this yields:

$$\text{One-way Delay} = \frac{512 \cdot \text{sizeof}(\text{Frame}_{\text{physical,max}})}{125\,\text{MB/s}} \approx 6.3\,\text{ms}. \tag{2.7}$$

Given the current FPGA implementation, wire-speed can only be reached if the ACK timing is faster than 6.3 ms.

2.1.2.1. Virtex-5 1GbE/UDP Core

At the time of writing all available wafer modules are based on the prototype FCPs that has been developed for the FACETS and BSS projects. Each prototype FCPs is equipped with a Xilinx Virtex-5 FPGA [Xilinx, Inc., 2009]. Core FPGA development was performed by a partner group located in Dresden [Partzsch et al., 2008–2014]; basically, all the non-HostARQ FPGA modules have been developed there.

During HostARQ integration testing it became evident that some modules are not capable of running at full 1GbE speed. Notably, almost all bugs that were found during HostARQ integration are related to problems that only occur when processing data at high speeds (issue #1286 collects all current problems). Some of which occurred after long test times; those bugs were hard to find when simulating the FPGA code base as it the simulation speed is very slow[11]. That is why it was decided for the upcoming Xilinx Kintex-7 FPGA-based design to shift development style to a more test-driven approach (see Beck [2002] for an introduction to test-

[11]The current simulation setup requires approximately 1 h real-time for 20 ms simulation time.

driven development). The author created multiple tests that verify long-term[12] stability and correctness of the FPGA firmware in cooperation with the HostARQ software. The tests[13] – basically acting as regression tests – can be used for both, FPGA simulation-based, and real-world testing.

Due to time constraints and the parallel development efforts for the new FCP, it was decided that for the old Xilinx Virtex-5 FPGA design non-critical issues are not addressed anymore [Grübl et al., 2014; Partzsch et al., 2008–2014]. This does not apply to the HostARQ and other non-FPGA-specific code as the code does not depend on any hardware specifics. The hardware-specific code parts are basically in maintenance mode as the development effort has been shifted to the upcoming Kintex-7 design of the new FCPs modules. In particular, one remaining issue that will not be fixed in the Virtex-5-design is the limited outbound bandwidth of the 1GbE FPGA module. Using the NCSim simulator, the back-to-back FPGA transmit performance was acquired by measuring the duty cycle[14]: data was constantly applied to the so-called UDP core and the outbound network port was monitored. On average, the network module implementation can transmit one byte every 12.15 ns, compared to wire speed[15] this only yields 66% of the theoretical throughput.

At the time of writing, tests that verify wire-speed performance (and correctness) in the Kintex-7 design have not yet been performed as the FPGA code base has not yet been completed [Partzsch et al., 2008–2014].

2.1.2.2. Between Host and FCP FPGA

This section analyzes the HostARQ protocol performance between one compute node and one FCP FPGA. We concentrate on the evaluation of the protocol itself; therefore, we do not use HICANN chips as communication partners but only the HostARQ transport layer within the FCP FPGA. All FPGA-based measurements were performed using FCP FPGA firmware SVN revision 1056 on wafer module #0. One dedicated HMF compute node served as the host computer. The data network uses one of the BSS wafer switches [Hewlett-Packard, 2013] where all wafer modules #0 FCP are connected via 1GbE. Host connectivity is provided by one 10GbE link. Section 2.1.2.4 presents measurements that cover the complete communication chain. That means that we communicate between compute node and HICANN chip by writing to and reading from the on-chip static random access memory (SRAM).

Half-Duplex As a first step, the downwards connection starting from the compute node and connecting to a single FCP FPGA is tested. In the FPGA, the data is

[12]The HostARQ FPGA and software implementations were verified using a week-long stability test running at maximum speed.

[13]HostARQ-based tests are `tmecm_hicannreads`, `tmecm_hostarqloopback`, `tmecm_pbmemtraceloop` and `tmecm_switchramviahostarq`.

[14]The time the network port was active in relation to the total time.

[15]1GbE supports 1 Gbit/s, i.e. one byte every 8.0 ns.

discarded as soon as it has been successfully received. On the upstream connection only ACK packets are transported. Figure 2.15 shows an overview of the setup. The flow control feature of the data network switch is activated to prevent packet loss due to bandwidth mismatch or network congestion. Thus, the resend timers can be set to very relaxed values as we do not expect packet loss to occur. Section 2.1.2 states the important tuning parameters: minimum delay and maximum link capacity. The former can be minimized by reacting as fast as possible, i.e. by acknowledging incoming data frames as fast as possible – this is called the ACK timeout. The latter is essentially the number of bytes that are allowed to be *in flight*. In terms of ARQ-type protocols this is called the window size (cf. figure 2.8).

When minimizing the ACK timeout the proportion of packets that only acknowledge received data increases. In addition, the total response time is given by the RTT (approximately a hardware constant) plus the ACK timeout and a delay due to the finite software reaction latency. The second tuning parameter, the window size, has a relative hard limit due to the properties of the FPGA implementation[16] – the maximum window size we achieved to fit into the FPGA was about 2048 packets, see [Karasenko, 2014] for details.

A sweep of tuning parameters is shown in figure 2.16. Each plot shows payload throughput versus ACK timeout – the latter is plotted logarithmically to cover three orders of magnitude. The half-duplex throughput is given by the test data size and the protocol handling time. The start is trivially defined as the point in time when the first data entry is pushed into the HostARQ software API. The end is given by the point in time when the last sent packet is acknowledged by the communication partner and this ACK frame is received by the HostARQ software running on the host computer. To accomplish a clean end timing, the API was enhanced to check for *protocol idle* state which is equal to empty transmission buffers. The test data size is approximately 1.4 GB. Theoretical throughput (cf. equation (2.3)) is plotted as a dashed horizontal line; the maximum delay that yields theoretically optimal throughput (cf. equation (2.6)) is plotted as vertical line.

For small window sizes ($\leq$ 4 packets) the theoretical peak performance is not reached. The HostARQ software reacts too slow and the delay is above the optimum value. For window size = 8 packets, we see wire-speed for the first time. The maximum delay given this link capacity is ≈ 1 ms to still achieve wire-speed. Larger window sizes yield more relaxed maximum delays. The self-defined goal of ≥ 5 ms (for the blue measurement points, not for the theoretical delay) is reached for window size 512 packets.

A rewrite of the FPGA HostARQ module is currently performed by Vitali Karasenko to allow for larger window sizes. After this modification, window sizes up to the total size of FCP FPGA dynamic random access memory (DRAM) will be possible. On both FCPs, the Virtex-5-based and the Kintex-7-based designs, up to 512 MiB DRAM can be allocated to the host interfaces [HBP SP9 Specification, 2014]. This allows for a window size of approximately $1.8 \cdot 10^5$ packets – which

[16] A bit vector as large as the window size is needed; the routing failed above 2048 packets.

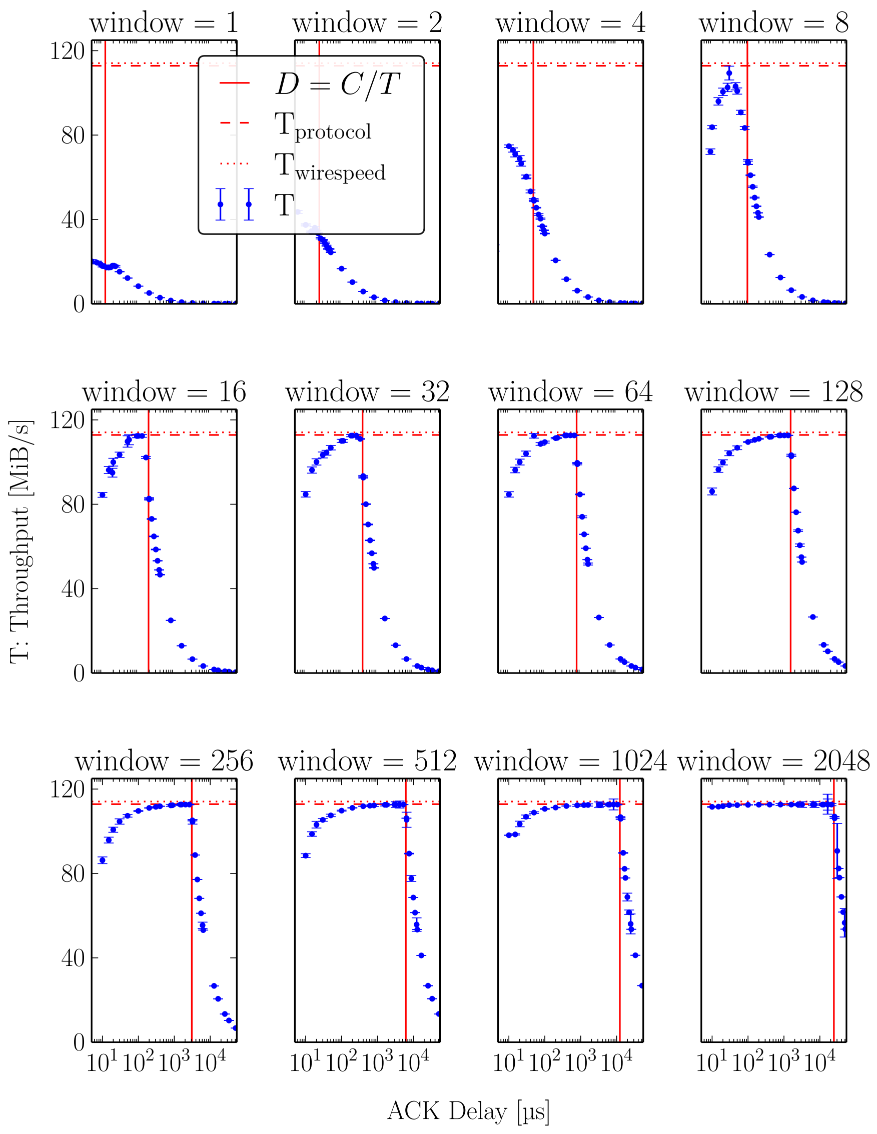


Figure 2.16.: Host to FPGA throughput (T) measurement utilizing only the down connection (half-duplex). The window size is varied between 1 (top left) and 2048 (bottom right). For each window size, the ACK delay timing is swept from 10 µs to 10 ms. The dashed horizontal line illustrates optimal throughput T (cf. equation (2.3)), the vertical line marks the bandwidth-delay product (cf. equation (2.6)). In contrast to the following measurement, figure 2.17, the packet size was constant in this measurement, i.e. the total protocol capacity increased with growing window size. The measurement was performed in collaboration with Vitali Karasenko [Karasenko, 2014].

corresponds to more than 4 s for maximum ACK timeout. As the ACK timeout determines the call frequency of the sending threads, large maximum delays relax software timing constraints and allow for a more efficient data handling, i.e. more packets are handled in one call.

Importance of Large Packet Size Figure 2.17 presents protocol throughput as a function of packet size. The total protocol capacity is kept constant, i.e. doubling the packet size reduces the window size by the same factor:

$$\text{sizeof}(\text{packet payload}) \cdot \text{sizeof}(\text{window size}) = \text{const}$$

This measurement has been performed to evaluate the importance of larger packet sizes in comparison to larger window sizes.

The solid lines in figure 2.17 display theoretical behavior which can be calculated using 1GbE symbol rate and the packet size (cf. equation (2.3) and substituting the maximal payload size by the current one). Due to packet overhead (headers and protocol overhead like inter-frame gap or the checksum), the maximum throughput can only be reached for standard-sized packets. Packets that are larger than standard MTU can cross the dashed line as it was calculated for the standard MTU. In particular, it asymptotically approaches the topmost dotted line as this corresponds to the physical symbol rate of the link – 125 MB/s and the relative overhead decreases for larger packets. The data network architecture supports frame sizes of up to 9 kB. Kintex-7 supports up to 16 kB [Xilinx, Inc., 2012]. Regarding the old Virtex-5-based FCP, no change is expected because development of FPGA-specific code is in maintenance mode (cf. section 2.1.2.1).

For each data point the optimum latency was measured by sweeping the ACK timeout and finding the maximum delay where optimum performance is still reached (for details see figure A.4). The measured throughput matches the theoretical throughput for packet sizes above 50 entries, i.e. 400 B. In the range below that point, the host computer is too loaded[17] to interact fast enough. For both network interface controllers (NICs) (BSS and NM-PM1 clusters) interrupt behavior is largely configurable. For example, a timeout can be used to accumulate multiple packets into a single interrupt. Additionally, a count-based rule can be configured. In this measurement, the *auto* setting was used. Using this setting the NIC tries to maximize throughput automatically. In comparison with disabled interrupt coalescence maximum interrupt rates of about 150k could be monitored. Hence, the HostARQ software is too busy with reacting to incoming ACK frames and the sending window is not updated fast enough, the communication stalls and the throughput drops.

Full-Duplex Based on the test setup used for the half-duplex tests an independent back-channel was added. As soon as the FPGA's HostARQ module successfully re-

[17]The load originates from interrupts that are caused by the high packet rate

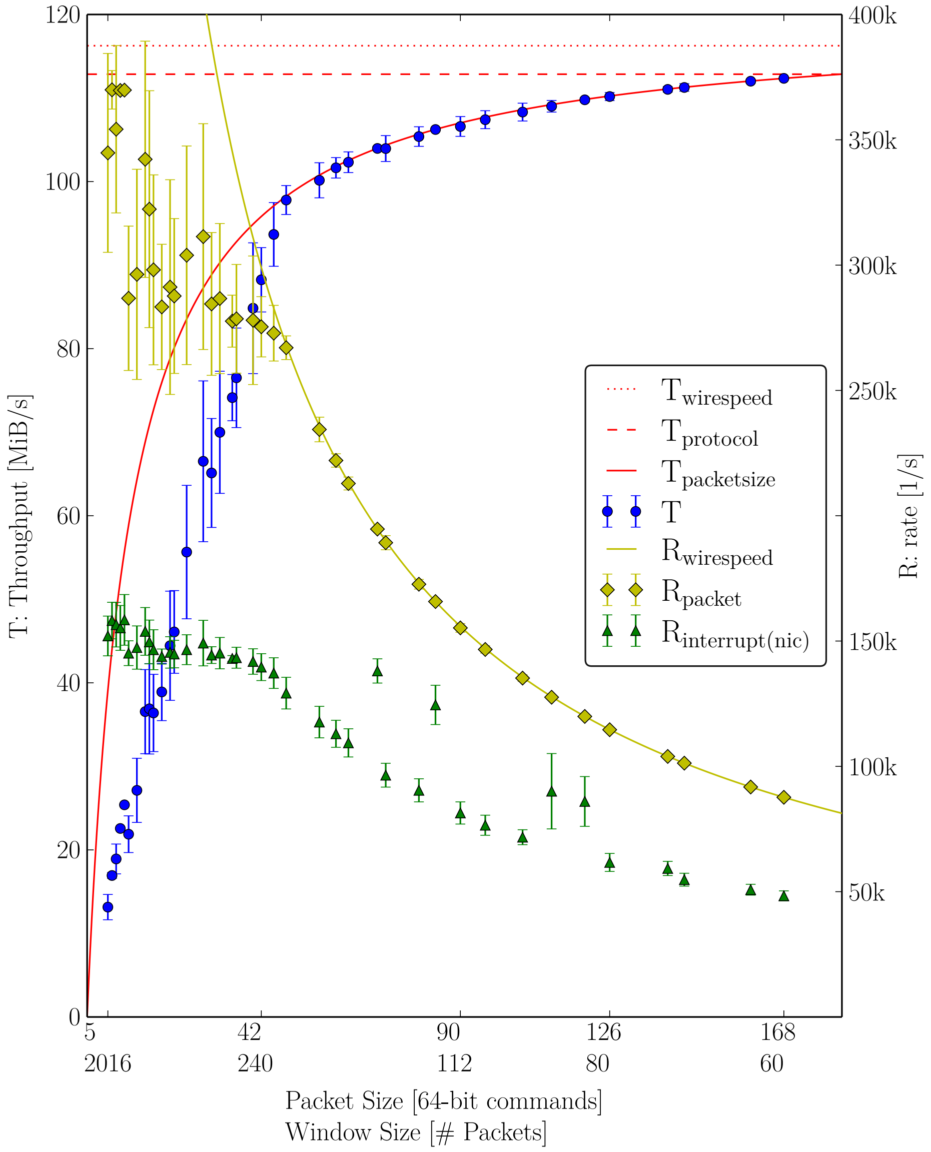


Figure 2.17.: Host to FPGA throughput (T) measurement utilizing only the down connection (half-duplex). The packet size is varied keeping sizeof(packet size) $\cdot$ sizeof(window) $=$ const, i.e. the window size is decreased by the same factor as the packet size is increased. Hence, the total protocol buffer capacity is constant. Throughput T is plotted on the left axis, packet and host interrupt rate R on the right axis. For packet sizes larger than $\approx 50\,$Cmd the measured points are in good agreement with the theoretical calculation (solid lines, see equation (2.3)). For lower packet sizes, the interrupt rate saturates because of software overhead; this limits the throughput. See *large packet size* paragraph on page 60 for details. The measurement was performed in collaboration with Vitali Karasenko [Karasenko, 2014].

ceives a data packet, a loop-back mechanism takes the data out of the *receive* buffer and pushes the same data into the *transmit* buffer. On the upstream connection both, data and – depending on the timeout values – ACK-only packets are transported. Figure 2.15 shows an overview of the setup. In a loop-back setup a stall in the backward direction will eventually stall the downward chain because of the limited buffer space in the endpoint that implements the loop-back. In contrast to a loop-back setup protocol the case of independent communication channels can stall independently in both directions.

After tuning the ACK timings (in software and FPGA) to be in the range of 100 µs to 500 µs and the resend timers to be larger than the RTT of the connection, we can now test protocol stability in terms of throughput over time. This is shown in figure 2.18.

The throughput was measured on the host computer and recalculated every 100 ms. Based on the individual measurements the average and RMSE of the throughput can be calculated:

$$\begin{aligned} T_{\text{inbound}} &= (74.80 \pm 0.02)\,\text{MiB/s} \approx 66.2\% \cdot \mathrm{T}_{\text{TUD UDP core}} \\ T_{\text{outbound}} &= (112.36 \pm 2.99)\,\text{MiB/s} \approx 99.5\% \cdot \mathrm{T}_{\text{protocol}}. \end{aligned}$$

To summarize, the host-to-FPGA troughput reaches nearly wire-speed, the reverse channel is limited by the current FPGA implementation. As explained in section 2.1.2.1, it is expected that this bug is only related to the current FPGA firmware of the prototype system; however, as the new systems rely on new FPGA hardware and – at least regarding this bug – new firmware source code, this bug will not be fixed by the responsible developer [Hartmann, 2014].

The next sections focus on the typical use case; multiple FCP FPGAs of a single wafer module are accessed by a single host.

2.1.2.3. Between Host and Multiple FCP FPGAs

This section covers the typical operation mode of one wafer module. One control host interacts with multiple FPGAs on a single wafer module. In figure 2.15 only a single FCP FPGA is shown in detail. However, another connection to a second FPGA is indicated on the right. For the prototype setups it was planned that each wafer has 12 FCP FPGAs. Due to design changes[18] this changed to 48 FCPs per wafer for the BSS and NM-PM1 production systems. At the time of writing only 8 FPGAs were installed on the prototype wafer module #0. Hence, only tests up to 8 FPGA communication partners were performed.

As in the previous section, the host's communication endpoint was the HostARQ implementation within the FCP FPGA. This means that most of the data flow path within the FPGA was not used. The latter is covered in section section 2.1.2.4.

[18]A larger number of smaller FCP FPGAs allows for lower total costs.

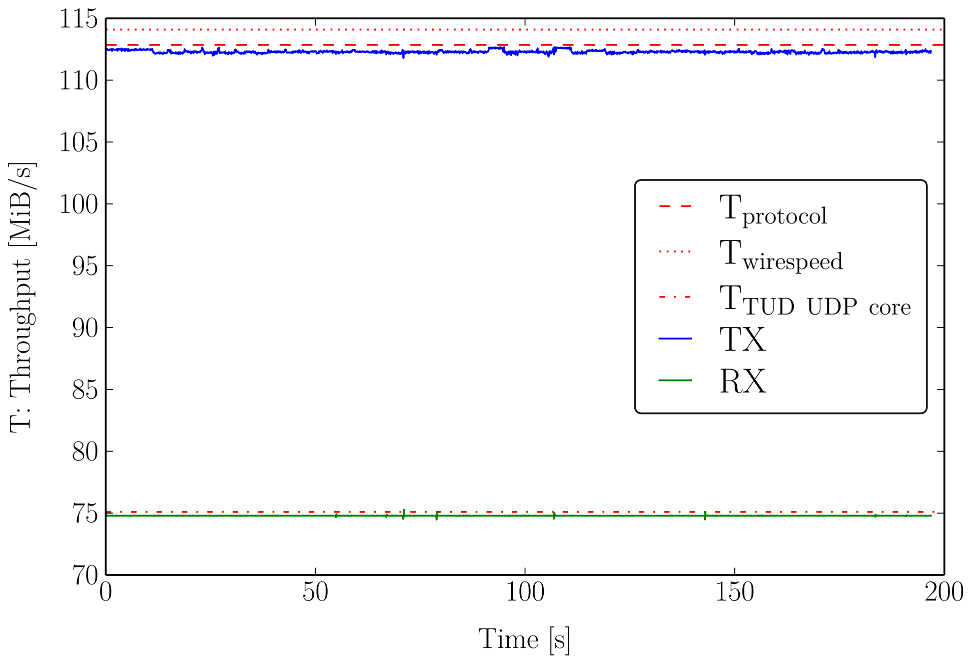


Figure 2.18.: Host to FPGA throughput stability measurement utilizing up and down connection (full-duplex). Outbound traffic is plotted in blue (TX), inbound traffic is plotted in green (RX). Raw wire speed ($T_{\text{wirespeed}}$) is plotted as red dotted line, the maximum payload throughput (T_{protocol}) is plotted as red dashed line. Due to a sub-optimal 1GbE/UDP core implementation, the FPGA is not capable of sending at full speed (cf. section 2.1.2.1), plotted as red dash-dotted line $T_{\text{TUD UDP core}}$. The next FPGA implementation for the upcoming FCP is expected to correct this misbehavior. Both, outbound and inbound traffic is stable over the complete test period of 200 s. The measurement was performed in collaboration with Vitali Karasenko [Karasenko, 2014].

Loop-back Measurement Figure 2.19 presents a scaling test of the HostARQ software in the loop-back case. The communication partners, the FCP FPGAs, are connected via 1GbE to the wafer switch which aggregates to a 10GbE link. This link is connected to the host computer. From the raw numbers (10:1), the host could handle up to 10 FPGAs in parallel at full speed.

On the host side, for each of the FPGAs individual random data streams are generated. Afterwards, the received data from each endpoint is compared against the sent data to rule out erroneous data handling. A total of 1342.77 MiB[19] is transferred in every test. This means that the transfer size per FPGA is reduced when multiple FPGAs are used. Figure 2.19 plots throughput against the number of parallel connections. The maximum throughput per FPGA is limited to approximately 66%, see section 2.1.2.1 for details. In the top part of the figure individual numbers for throughput are given for every FPGA. Given the resolution of the plot the bars match each other – i.e. all streams have balanced throughput even in the case of 8 FPGAs where the test only takes about 2 s. In the lower panel the aggregated value is plotted; the RMSE is calculated by averaging 10 runs. The measured throughput is in perfect agreement with the calculated maximum. Further measurements that extend the number of FPGAs are planned as soon as the new FCPs are available – the setup of the old wafer, i.e. being equipped with 8 FPGAs, will presumably stay the same.

Non-loop-back Measurement Due to the functional test overlap – i.e. throughput vs. number of communication partners, the non-loop-back measurements are only shown for HICANN-based tests in section 2.1.2.4. The performance in the FPGA-based case will be at least as high as for the HICANN-based case because the communication channel between host and FPGA stays the same and the minor change in buffer depth[20] will not change the performance.

Multiple FPGAs At the time of writing, the FCP firmware had at least two known problems when using multiple FPGAs in parallel (single-user or multi-user).

Firstly, during performance testing, the author experienced sudden drops in throughput when increasing the number of FPGAs in parallel. After recording experiment network traffic, a quick look into Wireshark revealed that the FPGAs resort to sending broadcast frames. This happens because of address resolution protocol (ARP) table overflows and it is normal behavior. However, due to implementation insufficiencies, the current firmware is limited to a small number of ARP table entries. The precise number is configurable, but for reasonable counts[21] the firmware build process fails due to excessive resource consumption.

Secondly, when receiving frames addressed to the Ethernet broadcast address the current FPGA UDP implementation has a certain probability that it starts to

[19]That corresponds to one million full packets, each carrying 176 entries that are 8 B words long.

[20]The effective protocol window size is not defined by the buffer size of the HostARQ implementation in the FPGA but rather by the total buffer size measured up to the HICANN interface.

[21]48 FCPs and multiple nodes per wafer thus, 64 entries seems reasonable.

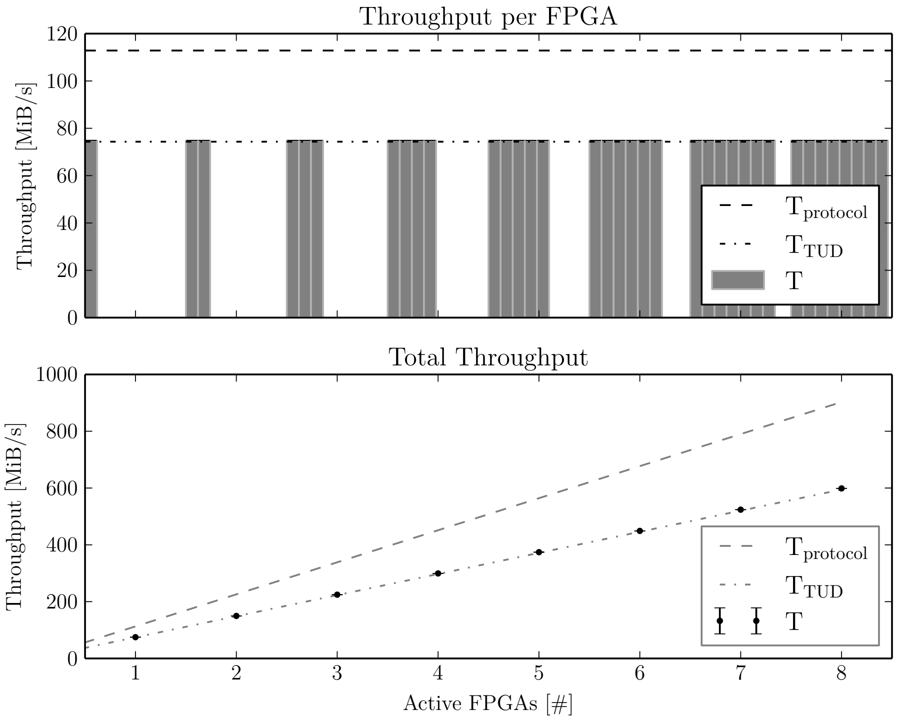


Figure 2.19.: Throughput measurement using the HostARQ loopback-mode in the FPGA. The FCP FPGAs are connected via 1GbE to the wafer switch. One 10GbE link is used to connect the switch with the host computer. The host sends individual random data streams to every FCP FPGA. A HostARQ loopback module within the FPGA returns the data to the host. Correct data handling is verified on the host by comparing sent and received data. On the x-axis, the number of parallel connections, i.e. FPGAs, is varied up to the maximum number of available FCPs on the first wafer prototype system. In the upper panel, individual connections/FPGAs are plotted as bars (i.e. a maximum of eight bars for 8 FPGAs). The height of the bars denote throughput, the error (RMSE) is too small to be visible. The lower panel provides aggregate values for the throughput; the error (RMSE) is too small to be visible. The dashed lines shows the maximum payload throughput ($T_{\text{wirespeed}}$). Due to a suboptimal 1GbE/UDP core implementation, the FPGA is not capable of sending at full speed (cf. section 2.1.2.1); the reduced rate is plotted as dash-dotted line ($T_{\text{TUD UDP core}}$). Given the current FPGA firmware bandwidth limitations, the plot shows perfectly linear scaling.

ignore other filtering mechanisms (e.g. IPv4 address or port number). At the time of writing, this bug is not reliably reproducible and further debugging efforts are needed.

For both problems one feasible workaround is to use VLANs [IEEE 802.1Q, 2003]. Every FCP is assigned to a unique VLAN. The FPGA switch port is operated in *untagged* mode, which filters out all unselected VLANs. On the host-side switch port, all VLANs are activated in *tagged* mode – this adds an additional VLAN header to all Ethernet frames. The host itself does not join any VLAN until an experiment job starts. This is why the maximum number of ARP table entries is now limited to 2 (if the data network switches are operated in transparent mode, otherwise the number increases by the number of hops between FPGA and Host). The experiment job specifies the set of utilized FPGAs – using this list, the resource management software now enables the matching VLAN network interface. As a consequence, Host-FPGA connections are now essentially point-to-point links.

Both bugs (collected in issue #1325) have been filed in the project management software and the FPGA developers are working on a solution. Nevertheless, the present workaround scales up to 85 wafer modules[22].

2.1.2.4. Between Host and HICANN

This section evaluates HostARQ performance between one compute node and multiple HICANN chips located on multiple FCP FPGAs. Figure 2.15 shows the eight HICANN chips that are linked to a single FCP FPGA[23]. The same wafer module setup is used as in section 2.1.2.2 with eight FCPs and at least one active reticle per FPGA. That means that up to $8 \cdot 8 = 64$ HICANNs can be accessed in this setup. Hence, the 64 HICANNs can receive at most 8 Gbit from the eight 1GbE links. The reverse connection is slower due to a bug in the FPGA firmware (described in section 2.1.2.1) – at most 66% wirespeed can be achieved. In contrast to the previous measurements, the effective buffer sizes are slightly larger due to the pipelined structure of the data flow within the FPGA. The exact buffer depth can be easily measured by sending illegal read commands to a HICANN chip that cannot respond; the protocol stalls after all buffer space has been filled up. At the time of writing, the total buffer size corresponds to approximately 650 packets. This means that the window size – currently 512 packets – dominates the total buffer space.

Half-Duplex This test measures the performance of the HostARQ protocol in the HICANN write case. The test software generates write commands to a fast memory location[24] on the HICANN chip. If multiple HICANNs are used, each chip is ad-

[22]12-bit VLAN *colors* $\Rightarrow$ 4096 VLANs, each carrying one FCP make up for 85 wafer modules.

[23]In the prototype system 32 HICANN chips are linked to a single FPGA.

[24]The top-left synapse switch crossbar is used. It connects vertical Layer 1 (L1) buses to synapse drivers which feed the signal into the synapse rows.

dressed in a round-robin fashion[25] The maximum payload size of a single HostARQ frame is 1456 B, which translates to 182 × 64-bit commands (for details see section 2.1.2). For optimal DDR SDRAM[26] burst alignment, the current FPGA implementation uses only 176 commands by default (see [Karasenko, 2014] for details). Therefore, a single HostARQ frame is filled with 176 commands and repeatedly sent to the FPGAs. After 1342.77 MiB[27] have been written to the system the measurement stops. In contrast to the loop-back-based measurements before, the exact point in time when the last write has been executed is difficult to determine. However, the total buffer capacity of the system, as mentioned above, is dominated by the HostARQ window size which is 512 Frames ($512 \cdot 1456\,\mathrm{B} = 728\,\mathrm{KiB}$). Compared to the total size of the test data set this contribution can be ignored.

Figure 2.20 shows the throughput measurements for 4 and 8 FPGAs/reticles. As shown in Karasenko [2014], single HICANN links cannot saturate 1GbE. Hence, the number of target HICANNs per FPGA is also varied; this is plotted on the x-axis.

Due to hardware stability issues on the wafer module[28], measurements using more than 48 HICANNs in parallel were skipped. For sufficiently large numbers of target HICANNs (i.e. three or more) figure 2.20 shows >= 94% of the maximum protocol throughput.

Full-Duplex Reading from the HICANN chip SRAM requires a single *read* command for every date to return – both are packed into 8 bit entries. This means that the full-duplex operation corresponds to reading from the HICANN chip; input data size equals output data size. However, the same issue as in the previous loop-back measurements affects the maximum throughput: the limited outbound bandwidth of the FPGA implementation; see section 2.1.2.1 for details. The outbound throughput is limited to 66% of 1GbE performance which is 112.9 MiB/s (see equation (2.3)). Thus, using the complete first prototype setup, we can expect the maximum throughput to be

$$8 \times 0.66 \cdot 112.9\,\mathrm{MiB/s} = 596.1\,\mathrm{MiB/s}$$

Unfortunately, at the time of writing, a bug in the core of the data handling code path within the FPGA is triggered early; this has been reproduced in simulation by the FCP FPGA firmware developers [Partzsch, 2014]. Hence, the stability of the current FPGA firmware does not permit to produce enough data to acquire robust results. However, due to the promising results that are shown in figure 2.19 and figure 2.20 the author expects rather good results. The FPGA developers that are responsible for this code part ([Partzsch et al., 2008–2014]) are actively working to resolve this issue.

[25]That means the target HICANN rotates.

[26]Double data rate synchronous dynamic random-access memory.

[27]See footnote 19 for details.

[28]The high-speed connection between FPGA and HICANN gets disrupted and a reinitialization is needed.

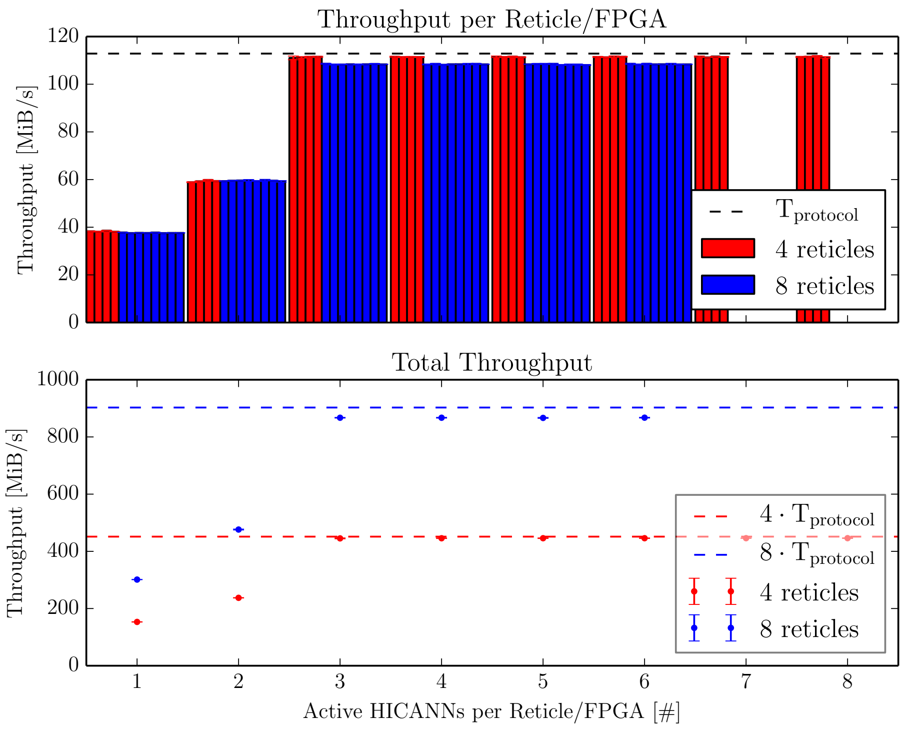


Figure 2.20.: The plot shows host to HICANN write performance. The test produces write commands directed to one or more HICANN chips. All data packets contain 176 write commands. Each write command is directed to a fast chip memory location (top-left synapse switch crossbar) on the HICANN chip. As writing to a single HICANN chip cannot saturate the 1GbE connection between the host and the FCP FPGA, the number of addressed HICANNs per FCP is increased from 1 to 8 (x-axis) which is the maximum number of HICANNs per reticle (and for the new FCP). A total of 1342.77 MiB are written per FPGA (see footnote 19 for details). Due to hardware stability issues the measurements involving more than 48 (i.e. using more than eight FCPs and six HICANN chips per FPGA in parallel) were skipped. The eight 1GbE connections from the FCPs are aggregated by the wafer switch into a single 10GbE connection to the host computer. The dashed lines in the lower panel show the maximum bandwidth available in total (four or eight times 1GbE, see equation (2.3)). In the setup using four FCPs and more than two HICANN chips per FCP, the total bandwidth is in perfect agreement with the calculated optimum. The eight FCPs case shows a minor reduction in performance compared to the calculated optimum. No further optimization was performed as the overall performance is over 94%.

2.1.2.5. Transport-layer Latency

The worst case latency when communicating between host computer and neuromorphic system is largely given by the total buffer capacity of the system and can be calculated using equation (2.5). Depending on the amount of other data transfers the RTT varies due to differences in buffer usage. Therefore, for applications with constraints regarding communication latency, the transport is suboptimal. An alternative communication method is presented in section 3.2.1 and latency measurements are presented in section 3.3.3.

2.1.2.6. Inter-host Performance

To complete the throughput measurements, the production-type setup has to be tested. Due to the lack of sufficient numbers of FCP FPGA installed on wafer modules, the measurements have been performed between two compute nodes. The 48 links between a single compute node and the wafer module are implemented as independent HostARQ communication channels; just as it is the case for the wafer module setup.

Figure 2.21 presents the measurement between two NM-PM1 compute nodes connected via 10GbE. Compared to the FPGA-based measurements, the settings used in this setup are unchanged. Assuming correct FPGA behavior, the results acquired using this setup should give a good estimate for later hardware-based measurements. However, to achieve 10GbE at least 10 1GbE links (FCP FPGAs) are required. In this measurement, a single communication channel is already capable of sustaining over 700 MiB/s. Using four channels, the maximum performance was found to be:

$$(1.070 \pm 0.009)\,\mathrm{GiB/s}.$$

This corresponds to over 99% wire-speed throughput (see equation (2.3)).

To explore the capabilities of the current HostARQ software implementation, the test has been repeated on the NM-PM1 frontend nodes which are equipped with 40GbE NICs. Figure 2.22 presents the measurement. When using the standard MTU of 1500 B the throughout is still limited to less than 2×10GbE. This is caused by high interrupt rates that increase software overhead. Figure 2.17 presents a measurement to evaluate the importance of larger packet sizes. To estimate the effect of increased packet sizes, four more measurements have been performed using 3000 B, 4500 B and 9000 B. The latter has been chosen as the current network infrastructure is limited to approximately this packet size. Using this MTU, the throughput can be increased up to the theoretical maximum (using eight parallel channels). Using eight channels and a MTU of 9 kB, the maximum performance was found to be:

$$(4.534 \pm 0.062)\,\mathrm{GiB/s}.$$

This corresponds to over 99% wire-speed throughput (see equation (2.3)).

In summary, both the HostARQ software implementation and the compute nodes

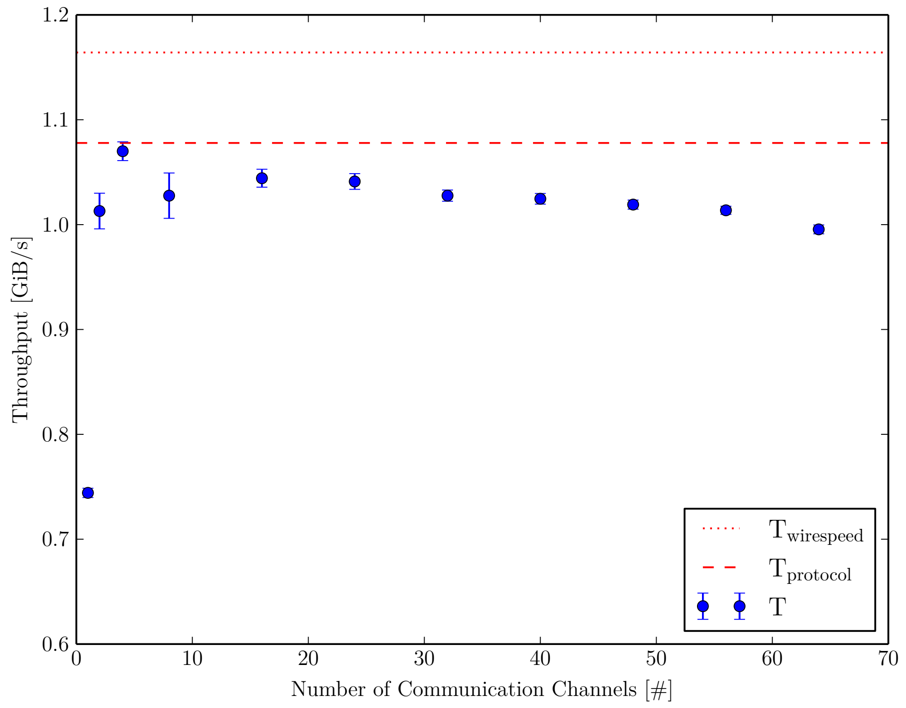


Figure 2.21.: Inter-host throughput measurement utilizing up and down connection (full-duplex). The hosts (two NM-PM1 compute nodes) are connected via 10GbE. On the x-axis, the number of parallel HostARQ communication channels is plotted; the ordinate denotes payload throughput (T). The errors show the RMSE of all communication channels. That means the errors are a measure for inhomogeneity between the communication channels. Raw wire speed ($T_{\text{wirespeed}}$) is plotted as red dotted line, the maximum payload throughput (T_{protocol}) is plotted as red dashed line. For more than one communication partner, the throughput reaches more than 90% of the theoretical maximum. This rise is caused by an increased effective window size as all communication channels use independent buffers, i.e. the timing constraints are more relaxed. 48 communication channels are used in the production-type wafer systems. With increasing communication channels there is a slight decrease in efficiency. This is caused by protocol handling overhead.

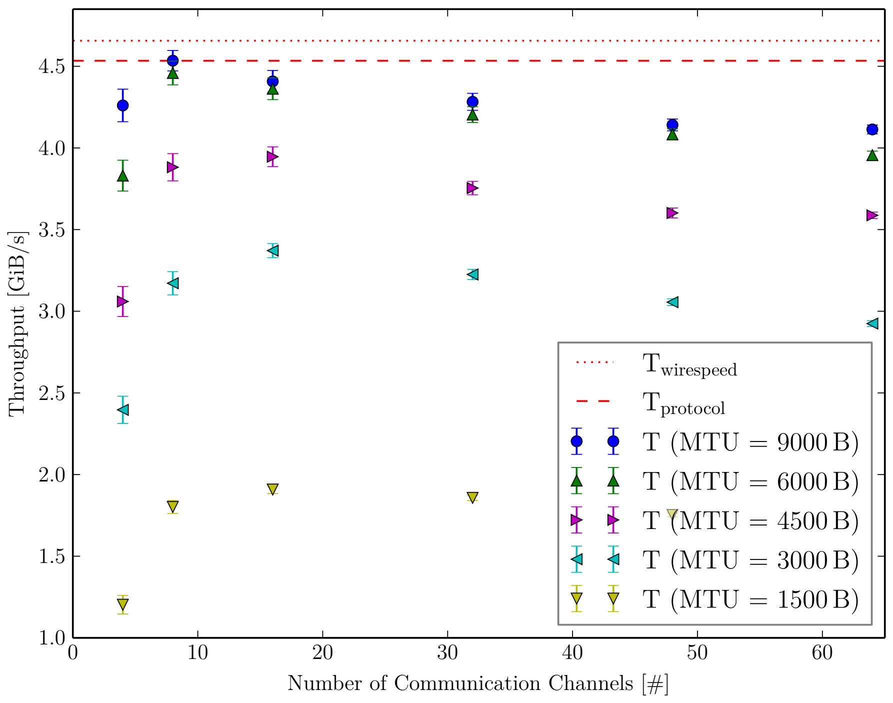


Figure 2.22.: Inter-host throughput measurement utilizing up and down connection (full-duplex). The hosts (two NM-PM1 frontend nodes) are connected via 40GbE. This setup is a reference measurement to explore the capabilities of the current HostARQ software implementation. On the x-axis, the number of parallel HostARQ communication channels is plotted; the ordinate denotes payload throughput (T). Results for the maximum MTU supported by the data network, approximately 9000 B, are plotted in blue (circles). A selection of other MTUs is plotted. The errors show the RMSE of all communication channels (cf. figure 2.21). Raw wire speed ($T_{\text{wirespeed}}$) is plotted as red dotted line, the maximum payload throughput (T_{protocol}) is plotted as red dashed line. For four or more communication partners, the throughput reaches more than 90% of the theoretical maximum for MTUs between 6 kB to 9 kB. No further tuning has been performed, except for the increased MTU, all settings match the FCP FPGA setup. Packet sizes of 4.5 kB and below achieve suboptimal performance. This is caused by software overhead and high interrupt rates; the situation is similar to the setup shown in figure 2.17 (see *large packet size* paragraph on page 60). For the production-type FCP FPGA firmware it is planned to support larger, *jumbo* frames; the FPGA supports frame sizes larger than the switches, i.e. the data network constrains the maximum size.

perform adequately, the specified wire speed is reached. Looking further ahead, with a slight modification[29] of the FCP FPGA firmware the software will scale up to 40GbE. The NM-PM1 frontend nodes can be used for special cases where maximum throughput is required; individual wafer modules have to be linked via four 10GbE links to the backbone switch. However, the port count is too low to allow for a fully 40GbE-connected NM-PM1[30].

[29] The FCP FPGA supports arbitrary packet sizes. However, the data network is limited to the maximum sizes allowed by the switches, i.e. approximately 9 kB.

[30] As stated in section 1.2.1, a homogeneous 40GbE topology was too expensive for the NM-PM1.

2.2. Operation Software

Compared to neuronal network simulators, the operation of a neuromorphic hardware system is difficult; the hardware realization of neurons, parameter storage, topology; every component adds additional constraints to the system. Some constraints can be hidden from the user in a sensible way, others are difficult to handle. Nevertheless, a software interface has to take every effort to simplify system usage, to make the system available for as many users as possible. One welcome side effect is that non-hardware-exports from outside can use the system and provide feedback to the hardware developers. Future hardware revisions can then be improved based on data that was acquired while using the system – in addition to other scientific data sources.

However, one lesson learned from the operation of the single-chip-based system, Spikey, is to make it also possible to expose hardware details. The experimenter should have the ability to choose the level of hardware abstraction he wants to adopt. For this purpose, all software layers now have an API that can be used to access layer-specific data structures and to manipulate behavior.

From the users' point of view, the main advantage of accelerated neuromorphic hardware has to be conveyed: the speed of the system. Operating the system at an speedup factor (cf. section 1.1.5) of 10^4 while providing inadequate software performance at the same time invalidates the eligibility of the whole system. A high-performance system can be used for classes of neuronal network experiments that are hard to simulate conventionally: in particular, long-term learning experiments, long-lasting in general and experiments that are repeated very often – e.g., to acquire statistics – are good points. Another argument has been already stated for the Spikey system: interactive exploration of parameters [Brüderle and Müller et al., 2009]. The latter implies the use of the acceleration factor by re-configuring the system and re-running the experiment as soon as the experimenter adjusts a parameter; due to the speedup factor the result is available to the user after a short timespan. For this reason, this is called *interactive* operation of the system.

A last application relies on real-time interaction between software running conventional hardware and a neuronal network running on neuromorphic hardware. This operation modes has differing demands and it is therefore not treated in this chapter. However, chapter 3 concentrates on this application.

Software Requirements User-friendly operation of such a complex system is crucial. The interface should be convenient and as familiar to experimentalists as possible. That is why the software system implements the PyNN API – a cross-simulator API for the description of neuronal network experiments. Section 2.2.4 introduces the API and mentions implementation details that differ from the usual implementations, e.g., `pyNN.nest` or `pyNN.neuron`. The second facet is the non-expert usability. To a large fraction, this means hiding of non-essential hardware-specifics like topological constraints, variations of the neuron circuits and defects of any component. This is covered by a translation method that processes hardware properties, calibration along with defect data and determines a valid representation of the user's network description. We call this step the *mapping* of a neuronal network. The mapping's output is a hardware configuration that is as isomorphous to the input network description as possible[31] This mapping step is described in section 2.2.3. Acquiring calibration and defect data as well as storage of the data is covered in section 2.2.6.

Another requirement is resource management. Both, the HMF and the NM-PM1 system will consist of multiple wafer modules and compute nodes. Traditionally, experiments on neuromorphic systems have been using a single-user-single-system approach. When scaling up to multiple users this approach does not scale – at least if larger networks are needed and experiments want support for fair resource sharing and support for multi-wafer experiments. In general, this problem is similar to the situation on typical HPC systems. While the inter-user fairness aspect and the specification of job size (i.e. the amount of hardware resources) are equivalent to that of traditional HPC systems, the aspect of non-identical hardware resources[32] and short runtimes are not. This aspect is covered in section 2.3.

An important point is testing. When operating large systems that are essentially always being optimized and modified, it is important to be able to automatically test and verify functionality. This aspect is covered in section 2.2.7.

Finally, the integration of the systems into larger frameworks, like the unified portal is handled in section 2.3.2.

Previous Efforts The author contributed to the development of the PyHAL software stack that was developed for the FACETS project [Brüderle, 2009; Müller, 2008]. Python-based hardware abstraction layer (PyHAL) provides a high-level user interface; it is an implementation of the PyNN API. Most of the requirements that were stated in the previous paragraph are implemented in this layer: the mapping, most of the calibration and the resource management. Compared to the HMF and NM-PM1 systems, each task is much simpler. For example, the PyHAL mapping has no notion of a bus structure, because the spike transport on the Spikey chip uses direct connections between neurons. The PyHAL software stack employs a *call-return*-based execution scheme[33] using a two-layered framework. However, the Python-based hardware abstraction layer (PyHAL) does not provide any support for binary serialization. That means, after invoking a PyNN script, the execution will continue until an error occurs or the experiment finishes.

The lower-level API provides access to the hardware units, e.g., the synapse

[31]Neuronal networks running on a wafer module have certain connectivity constraints; to put it simply, small network can be mapped without topological distortions, but larger networks exhibit synapse loss and spike loss.

[32]Every wafer has different analog and defect characteristics. Therefore, the map of defect components is different. Depending on the experiment this can be an issue. Support for explicit wafer requirements is crucial.

[33]That means that components provide a set of services that are invoked by other components [Clements, 2011].

weight matrix or neuron parameters. Internally it creates an in-memory representation of the control FPGA's experiment program, the so-called *playback memory program*. This is a precomputed FPGA-cycle-accurate sequence of instructions and events to send to the Spikey chip [Grübl, 2007]. The coding style is object-oriented with hardware units represented as C++ classes.

For a new software framework, some aspects of the old software stack proved worthwhile: 1. the user interface, PyNN, has been adopted by other neuromorphic systems [Galluppi et al., 2010], one graphics processing unit (GPU)-based system [Nageswaran et al., 2009] and is actively[34] used for conventional software simulations [Kaplan et al., 2013; Schmuker et al., 2014]; 2. the integration of analog readout capabilities into the software stack makes it easy to record neuron membrane traces; 3. the integrated usage of calibration data to mitigate the effect of fixed-pattern variation between neuromorphic circuits; 4. moving of data-intense computations from the `Python` to the C++ layer.

However, some aspects required improvement. The rather limited number of abstraction layers and their inaccessibility by the users resulted in *parameter tunneling*, i.e. the top-level interface took parameters for all lower-level functionality. Hence, it seemed beneficial to provide APIs for all the software layers. Together with the PyNN interface, this automatically requires Python-based wrappings for all the software components. For the existing PyHAL, the Python wrapping was done using the `boost::python` library and handwritten code. The author tested the automated wrapper code generator Py++ for this purpose and it proved to be adequate [Müller, 2011a].

Another item is the suboptimal API definition of the old internal interfaces. This generated a situation where, for example, the calibration of some parameter is PyNN-based and uses the PyHAL layer, and another parameter that is calibrated using the low-level layer. Generally speaking, a specification of components, APIs and architectural guidelines were missing.

The old execution model is strictly call-return-based, which makes resource utilization difficult as the part of runtime that really requires hardware access is only a fraction of the typical runtime. A more suitable execution scheme for non-interactive/batch-style experiments is to use a pipeline data flow model [Clements, 2011]. Using this model, we can introduce binary representations at the border between each layer and thereby suspend execution at this point. Later steps can re-load the binary representation and continue processing the data.

Although the old software stack already provided some methods for automatic testing [Brüderle, 2009, 3.2.5], the missing automation caused extra effort which hindered the early detection of problems.

Over the lifetime of the PyHAL stack – it is still in use for the USB 2.0 Spikey systems – software performance improved steadily. When PyHAL was introduced the Spikey system bandwidth was limited by an old proprietary host interface that

[34]Adapting PyNN to support large-scale HPC hardware has been the topic of an dedicated BSS workshop mid 2013, e.g., [Kaplan, 2013; Müller, 2013].

dominated experiment execution times. With the introduction of the USB 2.0-based interfaces the situation changed. Now the software part dominated the execution times because the host interface is more than a magnitude faster than before [USB 2.0, 2000; Schilling, 2010]. Moving computationally expensive code from Python to C++ improved the situation to a point where experiment update rates $\gg 10\,\mathrm{Hz}$ are possible [Pfeil et al., 2014].

Architecture of the New Software Stack When comparing Spikey and the new wafer-scale architecture, some properties increase the complexity of configuring and operating the systems [Schemmel et al., 2006, 2008]: 1. a bus structure was introduced to support on-wafer spike communication; 2. the analog parameter storage now uses floating gates; 3. neurons feature a more complex neuron model and are interconnectable; 4. off-chip links are configurable in terms of direction, spike sorting properties and time-stamping. The main issue arises from the wafer-scale property as many chips have to be configured and controlled in parallel. To programmatically support the developers and users as much as possible strong type-safety was added to the list of requirements.

To help users of the HICANN-based systems, support for simulating the system as a whole at a coarse level or support for detailed single-circuit simulations are required.

Based on experiences acquired while using and developing the PyHAL software stack[35] of the Spikey single-chip system during the FACETS project, the author envisioned an improved software architecture [Müller, 2011b] for the then upcoming wafer system. The architectural main difference compared to the PyHAL data flow model is the transformation from being call-return-based into pipe-line-based. The essential software components of this model are shown in figure 2.23.

Due to the fact that most experiments run in non-interactive mode, i.e. batch-style, this is a valid transformation. Additional refinements are: 1. keep PyNN as user API but move all data handling into C++ layer; 2. introduce API definitions for every functional component (i.e. split PyHAL into user API, mapping and calibration layers) – also called structured programming; 3. use automated Python wrapping of every (C++) layer; 4. deeply-integrated support for testing.

The stronger dependency on abstraction layers made other changes possible. For example, the new hardware abstraction layer (HALbe) supports multiple targets. The ESS back end can be used as a preparatory step when transforming neuronal network models from software simulation to neuromorphic hardware emulation. Another important back end is the link to neuron circuit simulations that run on dedicated hardware developer servers.

Text Structure In the following sections different levels of abstraction in the software stack will be described. In general, the text follows a bottom-up approach.

[35]See [Brüderle and Müller et al., 2009] for details on the implementation.

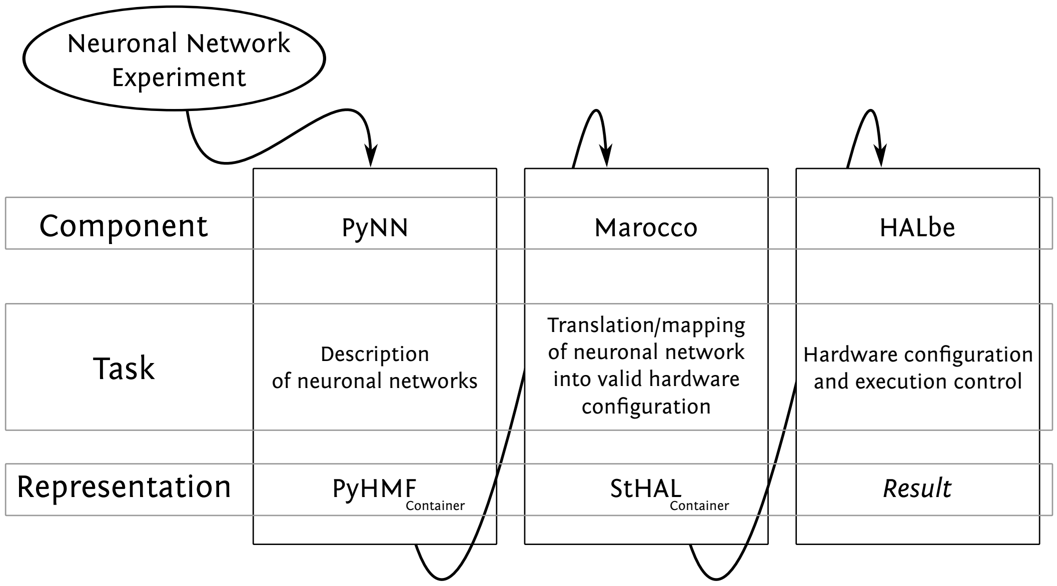

Neuronal Network Experiment
Component
PyNN
Marocco
HALbe
Task
Description of neuronal networks
Translation/mapping of neuronal network into valid hardware configuration
Hardware configuration and execution control
Representation
PyHMF Container
StHAL Container
Result

Figure 2.23.: Data-flow-centric view of the user software stack of the BSS and NM-PM1 systems. A neuronal network experiment is described using the PyNN API. Its binary representation, the PyHMF container, is transfered to Marocco which computes a hardware representation of the input network description and generates binary representation, the StHAL container. Finally, the hardware configuration is used by HALbe to configure the system (or to simulate hardware parts). The schematic indicates that representation layers are used to store intermediate results; this process is often called *serialization* or *marshalling*. That means, that the three columns can run in a pipelined fashion; for example, the mapping can prepare many hardware configurations which can be executed at a later point in time. This includes spatial separation – the last execution *stage* requires hardware access which enforces execution on the HMF-CP or NM-PM1 compute nodes. Notably, the mapping stage can be dispatched to a large-scale cluster system to provide a high rate of experiment configurations which can than be scheduled to run on the hardware system.

Communication layers aside[36], we start with the bit-formatting layer that exposes configuration and readout methods for hardware units. This layer is internally referred to as HALbe. Section 2.2.1.4 presents the SimDenMem component – an interface to a transistor-level hardware neuron simulator – as an example for a HALbe back end. Subsequently, the container layer (StHAL) will be described. This layer aggregates individual hardware unit data structures into chip or wafer data collections. The third layer covers the translation – or *mapping* – between high-level neuronal network descriptions and hardware-specific realizations. Extensive documentation of the mapping module, called Marocco, can be found in Jeltsch [2014]. Adjoint to this mapping are data sources that provide calibration (Calibtic) and defect data (ReDMan); the sources of calibration and defect data are described in an additional section, 2.2.6, because of their multi-layer dependencies. On top of these layers sits the main user interface – PyNN; to be precise, a PyNN API implementation[37]. Following these descriptions a quick performance evaluation is performed to assess eligibility of the software stack for the HMF and NM-PM1 systems. Finally, the resource management and test infrastructure complete this chapter.

Figure 2.24 presents an overview of the software modules and the typical flow of execution.

Collaborative Work The implementation of this software architecture was a collaborate effort. In the following table the main contributors are mentioned:

HALbe	Alexander Kononov, Christoph Koke, Eric Müller, Sebastian Jeltsch
StHAL	Christoph Koke, Eric Müller
Marocco	Sebastian Jeltsch
RPC Layers	Eric Müller, Sebastian Jeltsch
PyHMF	Eric Müller, Sebastian Billaudelle, Sebastian Jeltsch
SimDenMem	Eric Müller, David Hinrichs
ESS	Bernhard Vogginger, Constantin Pape, Paul Müller
Calibtic, ReDMan	Johann Klähn, Sebastian Jeltsch
Resource Management	Eric Müller, Paul Müller
Test Infrastructure	Eric Müller, Christoph Koke, Kai Husmann

2.2.1. HALbe

The common interface for all the user configurable hardware components is called HALbe. This layer not only provides an API, but also a coordinate system and data containers. During hardware development, components are often mirrored or

[36]They are covered in section 2.1.

[37]The current implementation covers the majority of PyNN version 0.7.

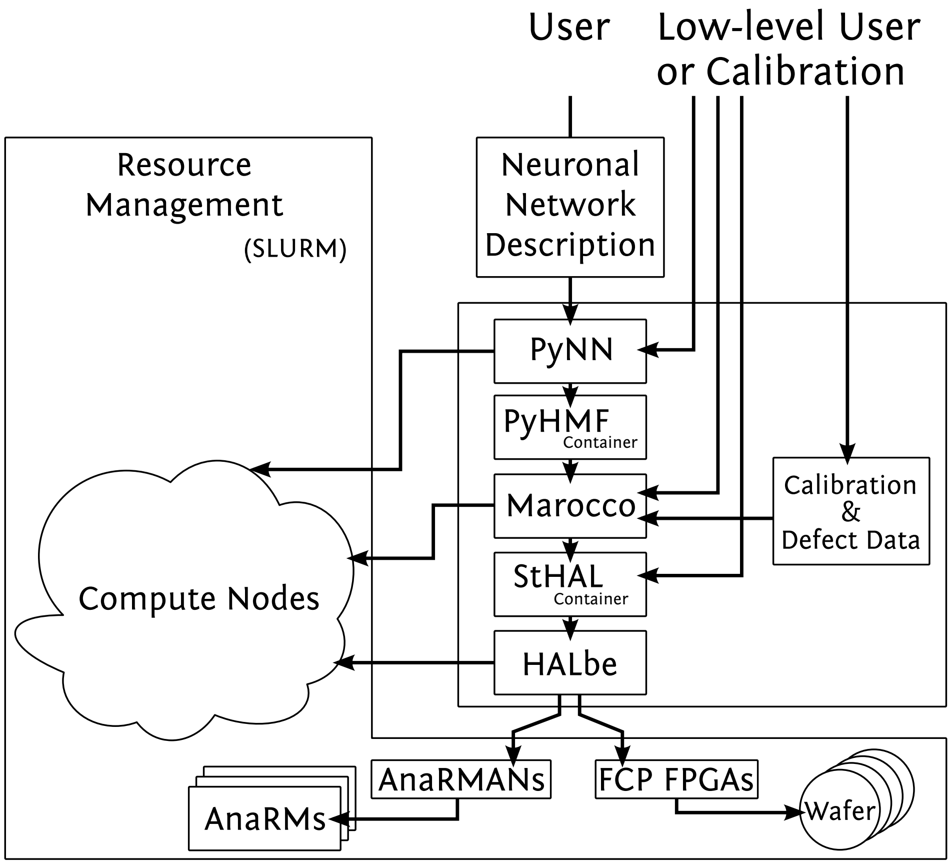

User
Low-level User or Calibration
Resource Management (SLURM)
Neuronal Network Description
PyNN
PyHMF Container
Marocco
StHAL Container
HALbe
Calibration & Defect Data
Compute Nodes
AnaRMANs
FCP FPGAs
AnaRMs
Wafer

Figure 2.24.: Execution flow of the HMF and NM-PM1 systems' software stack. The non-expert user starts a translation job to translate his PyNN-based neuronal network description into a binary representation (PyHMF container). This container is used as input for the mapping (Marocco) step – which can run as another batch job. Its output is again a binary representation, the StHAL container. Eventually, the StHAL container is transferred to hardware and the experiment can then be executed. Afterwards, the result data is stored in binary format (also called StHAL container). The last step, called HALbe, accesses the hardware and needs to be scheduled by the resource management system; SLURM guarantees that no hardware unit is used in parallel at the same time. The managed hardware resources are the analog readout modules (AnaRMs) which are connected to the analog readout module aggregator nodes (AnaRMANs) and the FCP FPGAs. In addition to this flow, the expert, low-level user or calibrator can use the lower software layers directly to generate a certain hardware configuration more easily. For example, this is necessary to test all bus structures on the wafer as the mapping will usually deterministically choose a certain connection between two neurons.

packed into larger groups; therefore, hardware-specific coordinates often look irregular from the high-level point of view. That is why user-side coordinates and data structures are used. In general, the addressing scheme adheres the C programming language array layout (left-to-right, top-to-bottom). A detailed description of the coordinate system can be found in [Jeltsch, 2014, 4.2.3]. Generally speaking, HALbe handles the conversion of configuration data from software-specific formatting into hardware-specific formatting.

The interface is based on free, *stateless*[38] functions taking a *handle* that identifies the communication channel to the corresponding hardware unit, and coordinates to identify a unit within the hardware entity. Functions *writing* to the hardware additionally take a third argument that contains the data to write; functions *reading* from the hardware return unit-specific data containers.

In the following code listing examples for write access (i.e. a *setter*) and read access (*getter*) are shown. The first example configures a row of *crossbar* switches that connect horizontal and vertical buses on the wafer.

```
// namespace HICANN
void set_crossbar_switch_row(
        Handle::HICANN& h,                 // communication channel
        Coordinate::HLineOnHICANN const& y, // coordinate 1st part
        Coordinate::Side const& s,         // coordinate 2nd part
        CrossbarRow const& switches        // data container
);
```

The second example reads out the recording memory of an analog readout module (AnaRM). Due to the integration of the analog readout module aggregator node into the NM-PM1 system[39], all analog-to-digital converter (ADC) functionality is currently modified to support remote procedure call (RPC)-based operation when accessing a remote AnaRM. The underlying RPC mechanism is presented and evaluated in section 2.2.5.

```
// namespace ADC
raw_data_type get_trace(
        Handle::ADC & h
);
```

The implementation of the API supports several *back ends*: accessing the real Neuromorphic physical model system (NM-PM), the ESS (ESS) (cf. section 2.2.1.3) and multiple debugging modes (e.g., to visualize low-level configuration data, or to assess neuron behavior, cf. section 2.2.1.4). In case of the NM-PM back end, the main objective is the translation between user-friendly coordinates and data

[38]The first parameter, the handle, is modified during the call, but its state is hidden from the user.

[39]The additional host was added to support larger distances between compute nodes and wafer modules.

containers on the one hand and low-level hardware commands accessing hardware entities on the other hand.

Some functionality of HALbe triggers remote operations. This involves control and monitoring of external voltages and accessing the AnaRM modules for analog recording. The remote operations are handled by RCF, a RPC framework that is evaluated in section 2.2.5

Power Control The power management units are Raspberry Pi-based hosts that are assigned to one single wafer module each. The Raspberry Pis access dedicated monitoring and control hardware using micro-controllers for deterministic and fast response behavior. At the time of writing, HALbe provides remote functions for powering reticles, reading external voltages as well as setting some external voltages. For example, the latter is used to evaluate and tune Layer 1 bus reliability by measuring error rates in relation to the voltages set.

Analog Readout The analog readout module aggregator nodes are hosts that aggregate twelve analog readout modules for four wafer modules each (i.e. a single rack). Due to the limited wiring length of the USB 2.0 protocol that is used for the host link of the AnaRMs, a direct connection between compute node and AnaRM is not possible. Furthermore, the static assignment of wafer modules to compute nodes is suboptimal, as it adds constraints on resource management and introduces additional points of failure[40]. For details see section 1.1.4.

2.2.1.1. Wafer-global Operations

When sharing a prototype[41] wafer module between multiple users some global operations like the *design reset* have to be guarded. This reset is needed to ensure a clean starting state of the HICANN chips on the wafer. To provide maximal robust operation, the current software stack triggers this reset at the beginning of every configuration cycle.

Figure 2.25 sketches the orchestration of the global reset between multiple processes. The implementation of this scheme uses traditional UNIX lockfiles that have to be *exclusively* locked for the reset operation. The exclusive lock is granted when no other process holds any lock on the lockfile. During hardware access, all processes have to hold the lock in *shared* mode to prevent the exclusive lock from being granted. After finishing hardware access, every process relinquishes its lock and enters a loop that tries to acquire an exclusive lock on the lockfile. As soon as all processes stopped accessing the hardware, i.e. releasing their shared lock, one of the waiting processes is selected[42] and acquires the exclusive lock. After the reset

[40]If a compute node breaks, the assigned wafer would have to be reassigned to another compute node – a process that would involve rewiring.
[41]The production setup provides FCP FPGA-wise reset capability. No reset orchestration will be needed anymore.
[42]The `flock()` manpage does not specify any grant order.

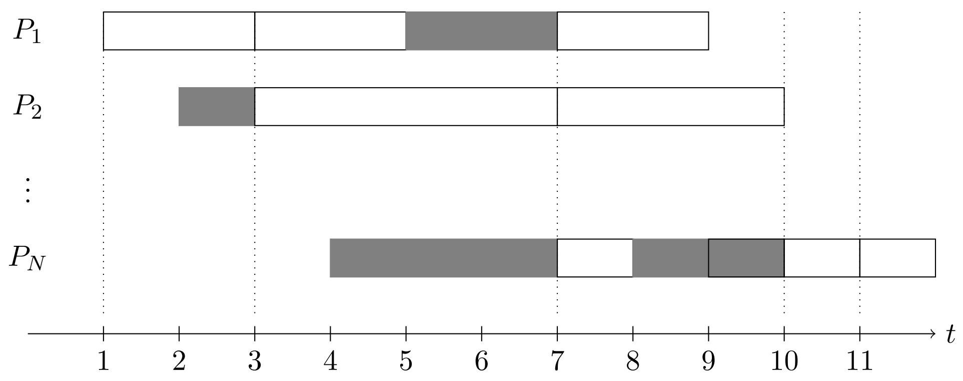


Figure 2.25.: Wafer-global operations have to be orchestrated. This operation scheme is used for multiple processes (and users) accessing the prototype wafer modules for the *chip* reset which is required after power-up and when the chip does not respond anymore. The current software state triggers this reset for each configuration/execution cycle to ensure robust operation. In this example, process P_1 starts at time 1 and is the single process that accesses the wafer; the reset can be pulled without interference of others. Process P_2 starts at time 2 and has to wait for P_1 to come to the next configuration cycle. Now, at $t = 3$ the reset is performed and both processes can access (non-overlapping) hardware resources on the same wafer in parallel. At $t = 4$ process P_N starts and has to wait for all other processes to end their current execution cycle. At $t = 7$, all processes want to start the next configuration cycle and the reset can be performed. At $t = 10$, only process P_N is running as the other processes terminated. Now the reset can be triggered without the need to wait for the other processes.

operation has been performed, the lock is relinquished and a new shared lock is acquired. The last lock *conversion* introduces a race that could result in multiple resets being performed by multiple processes – which is a legal operation because the shared lock that marks the *hardware access* phase has not yet been acquired. However, these redundancy takes time and therefore this problem has been solved by monitoring the modification timestamp of the lockfile. The exclusive lock operation is mapped to a *write* access in the Linux operating system. That means that every time the write timestamp changes, the reset has been performed. This check is done before and after acquiring the exclusive lock to exclude the race that is also possible here. This locking scheme is deadlock-free as no lock is held when the exclusive lock is acquired. When a process terminates its locks are automatically relinquished by the operating system; in particular, this robustness against abnormal program termination was the main reason for the lockfile-based implementation.

2.2.1.2. Scheriff

The HALbe interface and the hardware itself do not impose a specific configuration order. However, not every access pattern is legal. For example, after a HICANN chip reset (often called *wafer reset*) it is necessary to re-initialize the high-speed links between FCP FPGA. To tackle this problem, all API functions are annotated with *configuration states*. The order of state transitions is then checked by a finite state machine, called State Checking and Error Identification Framework (Scheriff), to identify illegal transitions. The list of legal transitions has been defined by the corresponding hardware developers [Grübl, 2013; Schemmel, 2014]. The author constructed a transition table and integrated the FSM into the HALbe framework.

2.2.1.3. ESS

The ESS is a simulation of the wafer module. Not all parts of the system are covered. For instance, the FCP FPGA and analog readout is only partly integrated. That means that not all effects arising from these components are correctly simulated. In general, the ESS is a rather coarse simulator. This simulation was initially developed as a simulation environment during HICANN-chip development. Later, missing parts were filled to reach a state where neuronal networks could be simulated on the ESS [Vogginger, 2010]. Subsequently, the ESS has been integrated into the HALbe framework [Pape, 2013]. Now the user can select the neuronal network execution back end to switch between hardware emulation and ESS simulation. Details concerning the ESS can be found in Vogginger [2010], the HALbe integration is described in Pape [2013]. Another operation mode is *dump* mode which generates file output that can be used for visualization of the configuration and the SimDenMem transistor-level simulation. The latter is described in the next section.

2.2.1.4. SimDenMem

One example of an application that is interfaced using HALbe is the transistor-level circuit simulation. Due to software licensing issues analog simulations are typically only available for chip developers. Furthermore, the simulation interfaces supplied by the analog circuit simulators are very generic and not optimized for neuronal network modelers.

Hence, an user-friendly interface to such an analog simulation is important. This is addressed by the SimDenMem. It provides a link to the analog simulation that covers essential parts of the neuron and synapse circuits. The list includes synapse driver, synapse and neuron membrane circuit.

The HALbe back end for simulation of analog circuits (SimDenMem) is a HALbe (see section 2.2.1) API implementation targeting an IPC-based simulation back end. Coordinates and data containers are appropriately converted. For example, boolean values enabling or disabling transistors have to be converted into analog voltage levels (e.g., 1.8 V digital power supply voltage for the Wafer (1.8 V) (VDD) or 0 V) and a relevant subset (e.g., the neuron circuit parameters) of all analog parameters has to be extracted. A client-server-based software using IPC transfers the simulation job onto a simulation server. The simulation server uses a proprietary analog circuit simulator to obtain results and returns the data to the IPC client. In the last step, result data is returned to the user and can now be visualized. As long as users utilize only one DenMem the experiments can be executed on both, the NM-PM system or the HALbe back end for simulation of analog circuits (SimDenMem) back end.

The author implemented the HALbe-specific software parts and remote call framework (RCF)-based communication client-server application that provides input data to the analog simulation. The link between this data and the analog simulator was implemented by Andreas Hartel. Improvements regarding the synaptic input and synapse configuration were performed by David Hinrichs under the supervision of the author [Hinrichs, 2014].

2.2.1.5. Real-time Access

Performing experiments on the HMF (or NM-PM1) that make use of real-time interaction between neuromorphic systems and compute nodes is another use case. To support this operation mode a thin software layer, called virtual environment for closed-loop experiments (VerCL), has been implemented by the author. This API provides methods to communicate spikes between FCP and compute node at low latency.

From the user's perspective, the software part of an experiment running in real-time requires additional precautions to eliminate unpredictable latency sources like page faults or call overhead. This makes it difficult to use convenience functions that are available for batch-style experiments; typically, the user code has to operate directly in the hardware value and time domain. During runtime, extended permissions are also needed to control real-time behavior of the operating system environment, and the custom network hardware (cf. section 1.2.1). Details are presented in section 3.4. To be precise, chapter 3 as a whole focuses on the real-time aspect of the HMF.

2.2.2. StHAL

At the time of writing, the stateful hardware abstraction layer (StHAL) consists of two parts: a collection of HALbe-based data containers that represent the configuration of one or multiple HICANN chips, and a configuration routine that calls the HALbe functions in a canonical sequence. The latter and HALbe's Scheriff have been introduced because different users applied different configuration sequences that produced unstable results. Hence, it was decided to provide *one single* configuration routine for all users. This decision was a trade between configuration speed, when only the absolute necessary parts are configured, and robustness. At the time of writing, the latter is still preferred. However, re-configurations which, for example, only switch between analog output channels have been implemented to speed up measurement routines that sweep over multiple neurons. The latter is used in the calibration framework. Later optimizations may implement robust support for partial (re-)configurations and thereby increase speed.

2.2.3. Marocco

Given the user's neuronal network description[43], a valid hardware configuration has to be determined. This process, called *mapping*, is not trivial as the existing neuromorphic system offers many tunable yet limited parameters. In particular, there are topological constraints that add dependencies on connections between neuron circuits (see Jeltsch [2014, 1.5] for a detailed introduction to the on-wafer event network). Other constraints come from limited parameter precision, limited bandwidths, crosstalk/noise, and variation between circuits[44].

Some transformations can be performed in a straight forward way: for example, the translation of neuron parameters can be performed individually and for a subset of those parameters it can be reduced to a linear mapping. However, most transformations are much more complex. The *quality* of a mapping is largely model-dependent as, for example, some neuronal network models are more sensitive to *distortions* of the topology than others. The main goal is to convey model functionality and dynamics from the biological description to the hardware emulation. In general, there can be multiple mappings that resemble a given biological network equally well. However, when increasing the network size, i.e. increasing the level of hardware utilization, it may get impossible to find good representations of a neuronal network description. Given a neuronal network description, Petrovici et al. [2014] present methods to cope with such distortions.

On the other hand, it may be possible for smaller networks to rely on manual hardware configuration. However, due to hardware complexity this process is slow and error-prone. Therefore, automated mapping is crucial to enable the user to map larger networks on the neuromorphic systems. For the automated mapping process a set of heuristic algorithms perform the transformation of the individual network properties. During the FACETS project, a first mapping method, MappingTool, has been implemented [Ehrlich et al., 2010]. In contrast to the MappingTool, the current implementation, called Marocco, aims for better scaling behavior and modularity. Details on Marocco can be found in Jeltsch [2014].

2.2.4. PyNN

PyNN is a simulator-independent API for specifying neuronal network models. Neurons and connections can be grouped into higher-level constructs, statistical measures can be used to describe parameters. It emerged from the computational

[43]The PyNN API which is used to describe such networks is presented in the next section.

[44]Software simulations exhibit deterministic distortions which are, for example, caused by limited floating point precision or quantization of spike event times.

neuroscience community because every simulation tool provides its own, proprietary input and output language. To overcome this issue, common input formats were developed which provide a single user interface to neuronal network simulators. There are alternative input formats; the most interesting alternatives are presented in a later paragraph.

Compared to a hardware-specific programming interface, most neuronal network modelers prefer the ease of use a common input format. Therefore, the HMF/NM-PM1 software stack implements the PyNN API[45] Figure 2.26 sketches a selection of PyNN's supported simulation or, in case of neuromorphic hardware, emulation back ends.

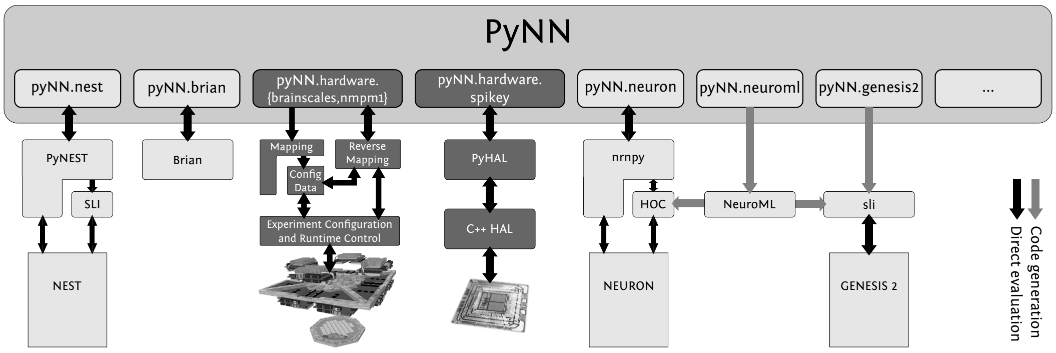


Figure 2.26.: The PyNN modeling API supports multiple simulation and emulation back ends: the neuromorphic hardware systems developed in Heidelberg are shaded darkly. On the left is the wafer-scale-specific back end which is presented in the text. NEST and NEURON are the standard PyNN simulator back ends. The declarative neuronal network description language NeuroML is covered in a later paragraph; PyNN supports NeuroML as an output format. For information regarding the other back ends see PyNN [2014].

In order to demonstrate the simplicity of the PyNN user interface, the following code snippet describes a simple network with a Poisson spike source projecting to a pair of `IF_curr_alpha` (leaky integrate-and-fire model with current-based synapses producing alpha-shaped postsynaptic potentials (PSPs)) neurons[46]:

```python
import pyNN.SIMULATOR as sim
import numpy

sim.setup(timestep=0.1, min_delay=0.2, max_delay=1.0)

cell_params = {
```

[45]For modeling, the PyNN interface is the preferred input format. However, lower-level access is possible using the Marocco mapping tool or the hardware access layers.
[46]A complete API documentation can be found on the PyNN homepage, cf. [PyNN, 2014].

```
7          'tau_refrac': 2.0,
8          'v_thresh': [-50.0, -48.0],
9          'tau_syn_E': 2.0,
10         'tau_syn_I': 2.0
11 }
12
13 output_pop= sim.Population(2, IF_curr_alpha(**cell_params))
14
15 tstop = 1000.0
16 rate = 100.0
17 number = int(2*tstop*rate/1000.0)
18 spike_times = numpy.add.accumulate(
19         numpy.random.exponential(1000.0/rate, size=number))
20
21 input_pop = sim.Population(1,
22         SpikeSourceArray(spike_times=spike_times))
23
24 projection = sim.Projection(input_pop,
25                             output_pop,
26                             sim.AllToAllConnector(),
27                             sim.StaticSynapse(weight=1.0)
28 )
29
30 input_pop.record('spikes')
31 output_pop.record(('spikes', 'v'))
32
33 sim.run(tstop)
34 sim.end()
```

The simulation back end is initialized in line 4 Two populations are created in line 13 and 21: the *output* population contains two neurons with identical model parameters which are defined from row 7 to 10. Lines 15 to 19 create a spike train out of exponentially distributed inter-spike intervals, i.e. a Poisson spike source – this is used as the *input* population. A connection between the two populations is created in line 24 to 28. Line 31 and 32 activate recording of spikes on both populations as well as neuron membrane voltage recording for the output population[47]. Finally, the simulations runs for a defined time interval. To execute this script on different simulators or hardware platforms, only the first line has to be modified. However, precisely matching results are not guaranteed as, for example, the NM-PM1 is an analog system exhibiting variations of different kinds (for details see section 2.2.3).

[47]The spike source has no membrane to record from.

The HMF/NM-PM1[48] PyNN back end uses the client-server approach to split the software stack into the PyNN interface implementation and the back-end specific part. On the user side, the client implements the PyNN API and uses an IPC mechanism to trigger experiment execution and retrival of experiment data. Additionally to the PyNN functionality, the IPC layer provides user authorization and authentication.

Data Format A suitable storage format for the PyNN user interface is currently developed by the NeuralEnsemble community [Neural Ensemble, 2008]: Neo. Its data model is hierarchical. Data series of varying sampling rate, start and end time can be grouped together. These groups represent *trials* or *runs* that share certain properties, e.g. the same set of parameters. With respect to the neuromorphic systems, the *AnalogSignal* and *SpikeTrain* types are most important ones. In Neo, these groups are called *segments*. The segments itself can be grouped into *blocks* which represent the file contents but contain extra meta data to uniquely identify the contents. To put it more simply, Neo is a collection of annotated NumPy arrays and convenience functionality. Details can be found in Neo [2014].

For experiments running on the HMF or NM-PM1 the existing Neo implementation can already be used as it supports the input and output data formats that are defined in the PyNN API[49].

Alternatives A different approach to describe neuronal network models is to replace the procedural description by a declarative description. One widely used example for this approach is the markup language NeuroML [Gleeson et al., 2010]. It defines model properties of varying levels of detail. The top-most layer, called *NetworkML*, is similar to `connect()` and `class Projection` in conjunction with the `class Connector` functionality of PyNN. The underlying layers, called *ChannelML* and *MorphML*, describe cell properties that both are handled by `create()` and `class Population` in PyNN. The first language revision focuses on compartmental neuron models using Hodgkin-Huxley-type synapses.

An initial implementation of the PyNN-to-NeuroML converter is included in the upstream PyNN sources. The inverse is more difficult, as the current PyNN API does not support a specification of arbitrary cell layouts but is fixed to a set of standard cell topologies. Users may add new cell and synapse types, but the link to a simulation back end has to be created as well[50].

Conversely, the NineML language focuses on the description of spiking point neurons with activity-dependent plasticity. The PyNN community plans to integrate import and export of NineML-formatted models in later revisions.

[48]At the current stage, the BSS and HBP systems' hardware is equal. Thus, and PyNN.hardware.nmpm link to the same PyNN implementation.

[49]The `PyNN.hardware.nmpm` back end adheres to the PyNN API version 0.7.

[50]For some PyNN back ends, e.g. Brian spiking neural network simulator (Brian) and NEURON, there has been some effort to support the runtime creation of new cell types.

Higher-level Abstractions In the computational neuro-scientific community, higher-level abstractions of neuronal network descriptions as well as high-level experiment handling are contemporary research topics. Regarding the network topology, tools like the connection set algebra (CSA) provide means to use a general formalism for describing connectivity in neuronal networks [Djurfeldt, 2012]. Other tools address the problem of high-level management of sophisticated neuronal network models [Antolík and Davison, 2013; Stevens et al., 2013].

2.2.5. Connecting Software Pipe-line Components

The software architecture described up to this point bases on a data flow model that follows the pipe-line scheme. Using this scheme, the individual components communicate via defined set of data structures. These serialized[51] data structures are produced by the preceding pipe-line step and read by the subsequent component. This means that the pipe-line architecture allows for introducing communication layers between the components.

One main goal of the software architecture is to enable multi-site operation. For example, based on the flow shown in figure 2.23 the PyNN API user generates the serialized biological network description on his local machine. Afterwards, this data is transfered to a compute cluster that allows for fast and parallel mapping from neuronal network descriptions to hardware configuration data. In the third step, the configuration data is transfered to the compute cluster associated with the neuromorphic hardware devices. The resource management (see section 2.3) mechanism schedules the experiment for execution and thereafter provides result data back to the user.

Given these requirements, the author evaluated existing remote procedure call (RPC) technologies; the chosen method, called RCF, will be presented in the following sections. Remote call framework (RCF) is a C++ library that provides standard C++ interfaces for remote procedure calls. The evaluation of this method has been performed by Kai Husmann under the supervision of the author [Husmann, 2012].

Figure 2.27 presents a measurement which evaluates RCF. Throughput is measured using varying call policies, for example asynchronous or synchronous, and data types. To summarize, using large vectors of identical elements yields optimal performance; RCF is able to sustain a throughput of approximately 1 GiB/s. This and other measurements performed by Kai Husmann (supervised and coordinated by the author, see Husmann [2012]) laid foundations for the application of RCF in the pipelined software architecture (see figure 2.23).

Figure 2.28 presents throughput measurements between the PyNN-based network description and the mapping input layer. In figure 2.23 this point is indicated by the second arrow going from *PyHMF Container* to *Marocco*. The throughput saturates approximately at 2 GB/s for networks larger than 500 neurons. In summary, the

[51]Serializing or marshalling means translating memory data structures into a binary format that can be stored or transmitted and can later be used to reconstruct the original data structure.

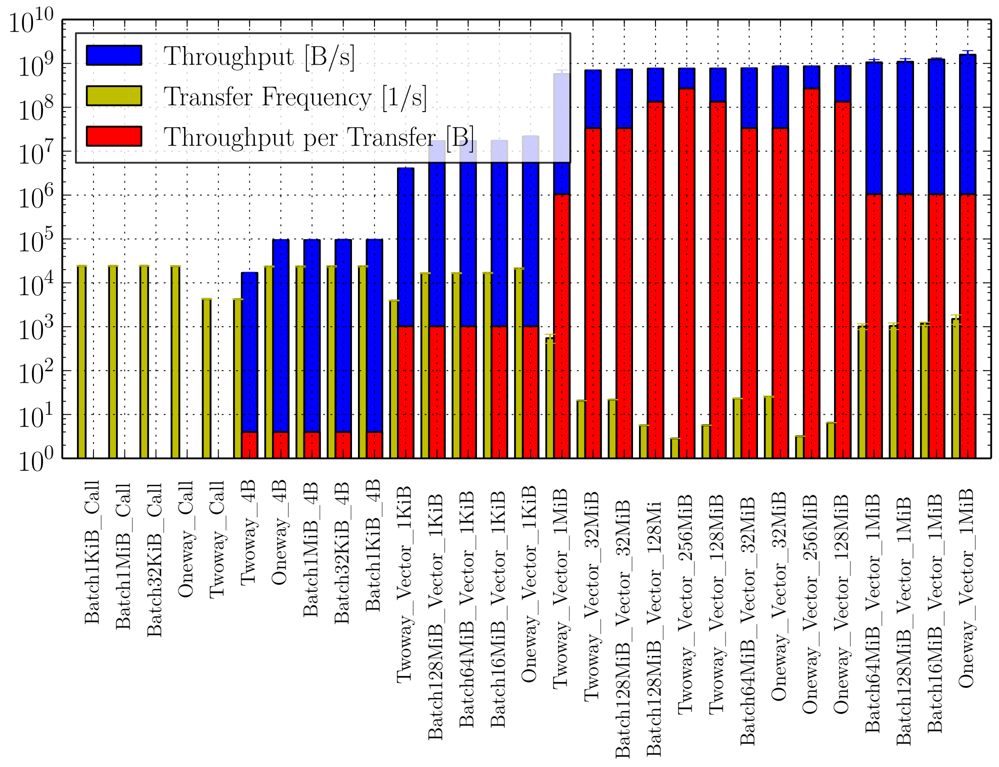


Figure 2.27.: Throughput measurement for the remote procedure call framework RCF using boost serialization and different call semantics as well as transfer objects types. The measurement was performed locally on a HMF compute node (i.e. it uses the kernel network stack on the loopback device).For every parameter setting on the x-axis, the large, blue bars indicate throughput measured in B/s. Red bars indicate individual transfer size that was tested. The narrow bars indicate the frequency of transfers measured in Hz. For tests that contain the *Vector* identifier, 2 GiB were transferred; the other tests used only 2 MiB to reduce the measurement time. On the left, the measurements named `*_Call` represent empty calls (i.e. zero transfer size) to acquire the maximum call frequency possible using this setup. In RCF terms, *oneway* calls denote asynchronous transfers and *twoway* calls represent synchronous transfers (i.e. the next transfer can only start after the previous has been completed). *Batch* transfers use oneway calls and an RCF-internal grouping strategy to reduce the number of transfers; the maximum batch size is given after the *Batch* identifier. Transfers of arrays of identical elements are marked with `Vector`; the individual element size is given at the end of the identifier. In a nutshell, using large vectorized transfers, i.e. > 1 MiB, allows for throughputs above 1 GiB/s. Figure A.5 shows the same measurement using the RCF-internal `SF` mechanism. The plot is based on measurements performed by Kai Husmann under the supervision of the author; for details see Husmann [2012].

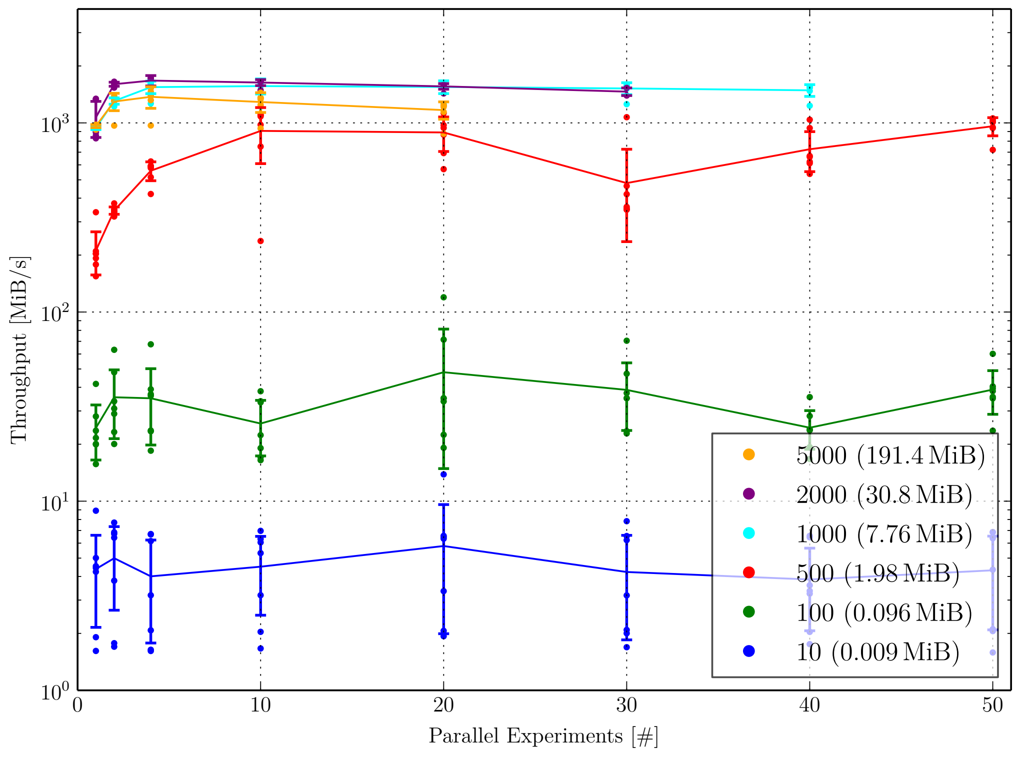


Figure 2.28.: Throughput measurement of user-defined neuronal network descriptions. Each color encodes a specific network size; for example, red encodes experiments containing 500 neurons. To evaluate the influence of parallelism, the number of parallel experiments, or jobs, is varied. Same-colored data points with equal x represent varying number of experiment receivers, i.e. the parallelism of the mapping software input stage. Every combination of these three variables represents a data point that has been repeatedly measured (10000 measurements for the two smallest networks, 800 for the largest); however, each data point resembles only the total runtime over all repetitions of this setup. Solid lines depict averages over varied receiver counts; the error bars show their RMSE. The neuronal network contains one neuron population and random (5%) connections of fixed weight between the neurons. In memory, the connections are represented as a dense/full matrix. After creating the experiment, it is converted to a PyHMF container and transfered to the input stage of the Marocco mapping component (see figure 2.23 for an illustration of the software components). This step employs the serialization technique of the remote procedure framework RCF presented in figure 2.27. The plot is based on measurements performed by Sebastian Jeltsch (for details see [Jeltsch, 2014]). See table A.3 for measurement details.

performance is adequate as it does not pose a limitation for a fast operation of the neuromorphic system.

2.2.6. Calibration

Neuromorphic systems that use analog circuits to model cell dynamics exhibit variations between individual units of the same circuits. This statistical effect arises during hardware production [Lovett et al., 1998; Pelgrom et al., 1998]. These fixed pattern fluctuations are characteristic for every hardware entity. Thinking of the neurons or synapses, these effects modify the individual parameters of the model equations. Consequently, instances of the same circuit behave different for the same input stimulus. Inhomogeneity which is due to fixed-pattern variations can be reduced by applying a calibration data set that adapts per-circuit parameters to achieve a matching behavior between instances of the same circuit. The possibility to calibrate neuron or synapse-wise is one key feature of the neuromorphic systems developed in the author's group [Fieres et al., 2008; Schemmel et al., 2006, 2007, 2008].

In addition, effects which are related to activity in neighboring circuits contribute to time-dependent variability as noise. Any analog implementation can potentially be influenced by noise. However, the amount of acceptable noise is model-specific. For new hardware generations the amount of shielding or effort to avoid noise is a tunable variable during development.

2.2.7. Automatic Testing

Hardware and software development efforts often focus on solving individual problems while neglecting functionality as a whole. In practice, this often leads to regression-type bugs – i.e. bugs that break previously working features – causing extra debugging effort and integration problems. In software engineering, the *continuous integration* method addresses this problem by automatically applying a test suite verifying as much functionality as possible at the earliest possible time. Test results provide feedback to the developers, side effects can be identified.

The author introduced continuous-integration-based development for the software developers in the Electronic Vision(s) group (Visions) [Müller, 2012]. Previously existing tests (standalone binaries) have been integrated into the common CI framework *Jenkins* [Jenkins, 2014]. By now, the development has been changed to a more test-driven approach where new code is directly tested by the CI test framework. On the programming side, the Google C++ testing framework (GTest) suite is used; its integration was a collaboration between Kai Husmann, Christoph Koke, Sebastian Jeltsch and the author.

The Jenkins server triggers pure software tests at every new software revision that is pushed to the group's central repository server; however, in order to keep all neuromorphic setups available for interactive use, tests that require access to neuromorphic systems are scheduled to run only during the night. In particular, hardware tests are executed in a specific order to minimize the number of redundant fails: low-level tests that verify basic functionality are executed before the high-level tests. Presently, high-level tests involving the PyNN API verify single neuron circuits by stimulating with on-wafer current stimulus, on-wafer random spike sources or external spike input and subsequently evaluating the experiment output. A *synfire*-chain-based test is currently under development by the calibration team and will be executed on a regular basis as soon as the success rate is high enough to be able to draw any conclusion from the result.

2.2.8. BSS Live System

Providing potential users of the neuromorphic system with the ESS is a preparatory step towards hardware. To relieve users from building the software stack, the author started to modify an *Ubuntu Live CD* to include the complete software stack to run PyNN-based experiments on the ESS. This work has been started for the FACETS project. Under supervision of the author, Kai Husmann migrated the build flow to a continuously integrated, Jenkins-based system. I.e. when any of the dependent components is changed, the Jenkins build server triggers an updated build of the live system.

Based on the *Live CD*-based distribution, a Docker-container-based distribution was created in collaboration with Kai Husmann. The Docker framework is an increasingly popular method to deploy complex applications due to the ability to include all software dependencies in the container. Deployment is as simple as downloading the container and executing the application. Another key feature of the Docker framework is support for container versioning and updating of containers. For documentation see [Docker, 2014].

2.3. Resource Management

The operation of the HMF and NM-PM systems involves many resource management tasks. From the user's point of view the situation is comparable to computing resources on a conventional HPC system. On the front-end side, the user issues experiment jobs, on the back-end side many hardware entities have to be orchestrated. In particular, this includes user authentication as well as authorization, assigning fractions of the system to jobs, job preparation, keeping track of hardware usage and failures, post-job clean-up, and maintaining fairness between users.

Prototype Operation The typical prototyping conditions for the BSS system are as follows: One wafer module is partially equipped with FCPs. The FCPs and a couple of AnaRMs are connected to a dedicated lab host computer by 1GbE and USB 2.0, respectively. The users login onto the lab computer and start programs that access the FCP units on a first-come-first-served basis. Overlapping hardware accesses of multiple programs or users terminates the program with an error message. Access control is not enforced. Running jobs are not preempted while different jobs can interfere with each other. One typical example which causes interference between users are global operations like resetting the digital part of the complete wafer or error conditions like exceeding the power limit of one or more power supply units (PSUs). Manual interactions happen frequently; power errors have to be acknowledged, power limits adjusted, firmware updated or reset buttons pressed. In general, the global system state is unknown to the single experiment jobs. This explains why a full reset is performed when starting an experiment. Basic system health monitoring is available in-place: a LabVIEW-based GUI reports analog power status; power limits can be set and verified by manual interaction with the PSUs. Test software can verify hardware behavior on the higher-level basis, for example, testing the communication link to the HICANN chip or verifying neuron behavior.

Scaling Up Looking at the specified sizes of the BSS and HBP systems it becomes clear that the previously described *prototype operation* mode will not scale to larger hardware installations. To get an impression of the installation size, we focus on the components[52] of the 20-wafer NM-PM1 system which is in the process of being set up:

960	FCP FPGAs
20	Power management units
20	Compute Nodes
20	Analog readout module aggregator nodes

This yields approximately 1000 IP addresses in total.

[52]See chapter 1 for hardware details.

However, this is only one part of increased operation complexity. Another part is the prospected user base; for the lab setups the operation relied on manual assignments of hardware components to users. This operation mode does neither scale to larger experiments nor to large user counts. In particular, some shared parts of the system – for example the analog readout modules are shared across several reticles and FCP FPGAs, respectively – require arbitration between experiment.

Requirements Starting from these prospects a set of requirements can be formulated. The first set of requirements addresses hardware features: for example, all hardware operation has to be sufficiently robust to allow for remote operation; user interaction with the wafer module should not require the manual interaction with the system. At the time of writing, this mostly applies to lab power supplies used in the prototype setup that currently cannot be remotely controlled. In addition, the status of this hardware support infrastructure has to be monitored automatically and this data should be accessible by the user. Similarly, the user should be able to acquire all operating parameters and get feedback when any monitored component failed during the experiment.

As stated in the previous paragraph, the need for arbitration of shared hardware units is another requirement. This arbitration implies the need for a queuing system to allow for batch-style execution. The latter directly leads to fairness between users as another requirement.

The next set of requirements deals with automatic handling of user's experiments and controlling hardware usage in general. At this point, operating the NM-PM1 system can be well compared to operating an HPC cluster: user management, job management and fairness are the main objectives here. However, there are some differences between the two target architectures. HPC clusters consist of sets of identical hardware entities (e.g., compute nodes that contain identical CPUs or GPUs) — analog neuromorphic hardware is imperfect and calibration as well as defect data is specific to individual hardware entities [Müller, 2008]. That means that neuronal network models are often tuned to a specific neuromorphic system, e.g., to a specific set of HICANN chips. Switching to other HICANN chips is possible, but will require additional adaptations. For reproducibility it is crucial for the resource management to support the assignment of experiments to a specific set of hardware.

Finally, the prospected sweet spot for accelerated neuromorphic hardware adds constants regarding scheduling speed and possibly scalability towards much larger installations.

In general, one can distinguish between the following types of resources:

- System-global resources (FCP FPGAs)
- Node-local resources (analog readout modules linked to analog readout module aggregator nodes)
- Exchangeable resources (Compute Nodes)

Requirements regarding experiment execution and scheduling:

- Reservation of system components for specific time span
- Robust resource assignment (i.e. prevention of illegal access)
- Robust clean-up
- Fairness between users and jobs
- User-friendly interface

Further requirements that are essential to convey the advantages of accelerated neuromorphic systems to the users:

- Experiment or job throughput $\gg 10\,\text{Hz} \Rightarrow$ fast parameter sweeps and interactive usage
- Scalability towards larger installations (i.e. containing more wafer modules and compute nodes)

Based on the list above, one can realize a largely overlapping set of requirements between conventional HPC systems and the NM-PM1 system. This is why the author decided to test traditional cluster resource management software packages. One of the most common tools is SLURM[53], a job scheduler for Linux-based systems. At the time of writing, SLURM was one of the most popular cluster management systems of the TOP500 supercomputer list [TOP 500, 2014]; in particular, five out of the top 10 supercomputers use SLURM[54]. Many Blue Gene/P-based or Blue Gene/Q-based systems rely on the default batch submission system, the IBM LoadLeveler.

SLURM The SLURM software architecture relies on multiple services. One service, called `slurmd`, handles job execution and runs on every compute node. The scheduler, called `slurmctld`, runs on one or more dedicated servers. Optionally, job and user accounting is performed by `slurmdbd`; this database service relies on an underlying SQL service[55]. Fault-tolerance is achieved by supporting fail-over to a backup system which provides secondary control and database services [LLNL et al., 2014, cf. *Failure Management Support*]. Figure 2.29 presents an overview of this software architecture.

There is only sparse benchmark data on SLURM installations [Georgiou and Hautreux, 2013]; however, installations on the largest supercomputer systems indicate that scalability towards high queue lengths and higher user counts will not be an issue [TOP 500, 2014].

[53]Simple Linux utility for resource management.
[54]2014, June: Tianhe-2, Sequoia, Piz Daint, Stampede, Vulcan
[55]Storage is also possible using text files; however, the scaling properties of plain text files in terms of searching and updating entries are too limiting.

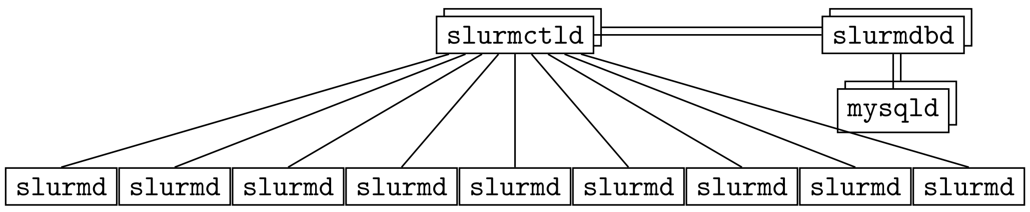


Figure 2.29.: SLURM is a cluster resource management tool. A central scheduler, `slurmctld`, controls the individual compute node and interacts with an job and user accounting database (`slurmdbd`). The `slurmd`s run on every compute node and handle job execution. The central services can be duplicated to provide a fault-tolerant environment.

Based on the requirements stated in the last paragraph, we can now evaluate SLURM's adequacy for scheduling on neuromorphic systems. Some properties of the neuromorphic systems cannot be directly mapped to cluster resources. In particular, the global but unique wafer modules as well as the analog readout modules connected to specific wafer modules are not typical resources on HPC systems.

When looking at the SLURM feature list, there are two striking items: *consumable licenses* and *local resources*. The former is typically used for restricting the number of parallel software licenses in use – this matches the requirement of global resources. The latter is used for the integration of GPU-type and many integrated core (MIC)-type resources into the resource management system – this matches the requirement of node-local resources. A first test of the features proved adequate functionality. However, for production operation an improved user interface will be required[56].

Other requirements that were stated in the previous section concern job execution. The user-friendly interface (including interactive operation), support for reservations, and fairness come for free. Notably, the latter is tunable and extensible via a software plugin interface. However, robust clean-up after job termination – especially after aborts and crashes – is compute node-specific. After thorough testing of the existing software stack, the author decided to wrap jobs into Linux *control groups* (cgroups)[57] which is also supported by SLURM. This is one step away from operating-system-level virtualization but already provides a strong isolation of processes. The main advantages are reliable memory and CPU restriction as well as forceful process termination [CGroups, 2014]. Considering FCP FPGA and software bugs in a production environment, it could become important to include restrictions on connectivity by setting up firewall-based rules for the cgroups. However, the latter has not been tested.

Finally, the next section covers neuromorphic-hardware-specific issues regarding fast job throughput.

[56]The built-in interface requires explicit resource specification on the command line. This duplicates the specification of the experiment resources.

[57]Essentially, LXC combines cgroups and namespacing to provide an userspace virtualization.

2.3.1. Measurements using SLURM

To evaluate SLURM's performance potential the author tested some aspects that are of importance to the HMF and NM-PM1 systems. Compared to conventional HPC machines accelerated neuromorphic systems have partially different use cases. Some applications and use cases are overlapping with conventional machines. This includes long-running batch jobs producing moderate amounts of data[58]. Sharing of wafer resources is certainly possible and needed for current prototype setups but for the later *production* systems it will be more robust to disable wafer sharing and resort to a purely time-sharing system. Both usage patterns can be found in HPC systems and are completely supported.

Other use cases or job properties are not completely congruent with typical HPC usage. The HMF and NM-PM1 systems allow for high job execution rates as the accelerated emulation reduces total runtime. Typical neuronal network experiments only require some minutes of simulation time which translates to some milliseconds of wall-clock time. One application are parameter sweeps including interactively scanning and optimizing properties of the network and thereby triggering re-execution of whole experiments[59]. Other candidates are long-term learning experiments and gathering of data for statistical purposes.

Hence, given these requirements the performance of SLURM was evaluated using multiple micro-benchmarks: the first benchmark uses the existing prototype wafer to execute test transmissions to HICANN chips (i.e. the whole communication chain) on four FCPs:

```
for spike train size in 1 MiB – 512 MiB (approximately 128 MEvent; random data) do
    submit 1000 jobs, store wall-clock time;
    trigger execution;
    while jobs in queue do
        for resources (i.e. Wafer/FPGA licenses) do
            acquire resources
        end
        prepare experiment of given size;
        execute experiment;
        release resources;
    end
    calculate difference between now and earlier wall-clock time;
end
```

Algorithm 1: Micro-benchmark to test SLURM performance using the HMF prototype system. The corresponding measurements are shown in figure 2.30.

Other benchmarks that are performed test scaling properties, i.e. job throughput,

[58]The wafer module system is able to produce 1 GB to 4 GB per second of output data. That is not typical to single nodes within small-scale/20-node HPC clusters.

[59]This was already stated in [Brüderle and Müller et al., 2009] as one sweet spot of accelerated neuromorphic systems.

in relation to increasing number of execution targets. Results for the benchmarks are shown in figures 2.30 to 2.32. All tests were performed on the BSS cluster (i.e. HMF-CP).

Figure 2.30 measures the scheduling overhead of SLURM. The number of communication endpoints, i.e. FCPs, was chosen arbitrarily. At the time of writing, up to eight endpoints are available on the first wafer prototype system. The minimum execution time was found to be 300 s for 1000 queued experiments, i.e. 300 ms per job on average which corresponds to an maximum experiment rate of 3 Hz per wafer module. In summary, a single wafer module managed by SLURM supports experiment rate of up to approximately 3 Hz; this is good enough if SLURM scales well to multiple wafer modules.

Hence, the scaling of execution time was evaluated depending on the number of compute nodes. Figure 2.31 presents super-linear scaling which is better than the expected linear scaling behavior.

Due to the speedup factor, experiments performed on accelerated neuromorphic system are typically short-lived. Figure 2.32 measures the minimum time which is required per job to terminate depending on the number of queued jobs. The plot shows a low plateau from approximately 16 to 10^4 queued jobs. The average execution and submission time is approximately 20 ms which corresponds to 50 Hz job rate.

2.3.2. HBP Unified Platform

The HBP project makes a massive effort to comprehensively integrate information resources and enable collaboration between all neuroscientific fields. This effort converges on the *unified platform* which will provide the technological means for scientists to collaborate on large-scale projects. Figure 2.33 sketches its structure from a control-flow point of view. The unified platform will make it possible for researchers to design experiments based on biological data, simulate or emulate the experiment on neuromorphic systems, and evaluate the result using analysis tools potentially running on HPC systems.

To succeed, the integration of all HBP *subprojects* is essential. With respect to the HBP neuromorphic subproject, the integration requirements are similar to those of conventional HPC sites. In particular, the integration of local resources into a global system is challenging. This concerns locally managed hardware resources, users and jobs as well as a method for global data access. The other tasks are addressed by grid computing technologies [Foster and Kesselman, 1998]. At the time of writing, the topics of user authentication, resource management, and data access are currently discussed. For example, solutions could be based on OpenID (OpenID), integrated rule-oriented data system (iRODS) and uniform interface to computing resources (UNICORE), respectively.

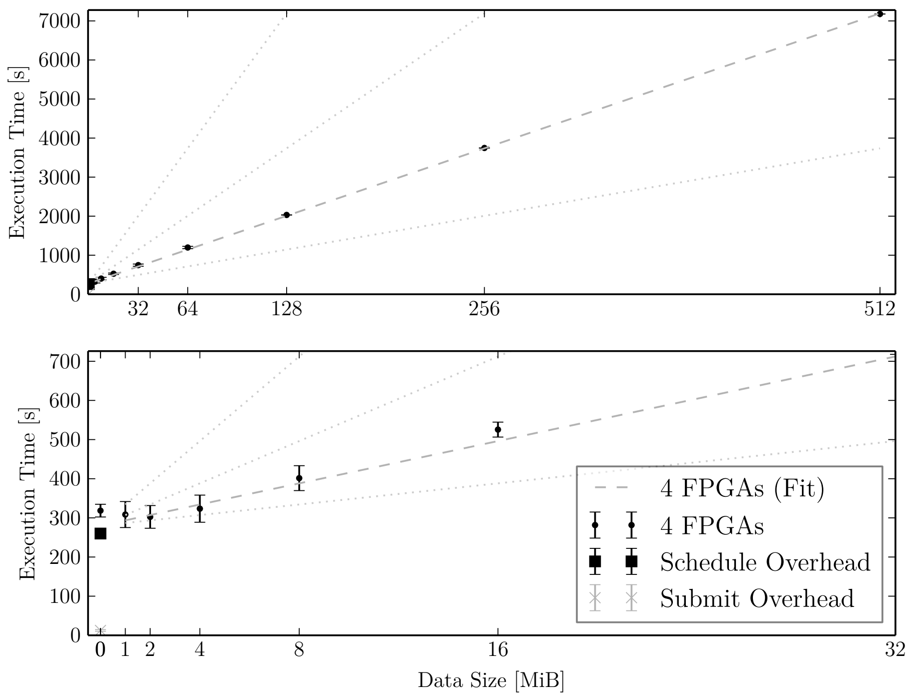


Figure 2.30.: Evaluation of SLURM execution overhead. The data points were acquired according to algorithm 1. The execution run times are given for a set of 1000 experiment jobs (errors are RMSE out of 3 averaging runs). Experiments of varying spike train sizes (x-axis) are transmitted in parallel to one third of the HMF prototype setup FCPs. A linear fit to the data is plotted as dashed gray line. The behavior is almost perfectly linear down to small experiments of a few MB. In the lower plot a zoom-in of the region 0 MiB to 32 MiB is plotted. The constant overhead is approximately 300 s for 1000 jobs, i.e. 300 ms per job on average. The dotted lines indicate from top-to-bottom the expected scaling for 1, 2 or 8 FPGA communication partners. An evaluation of the scaling properties for multiple parallel execution units is presented in figure 2.31.

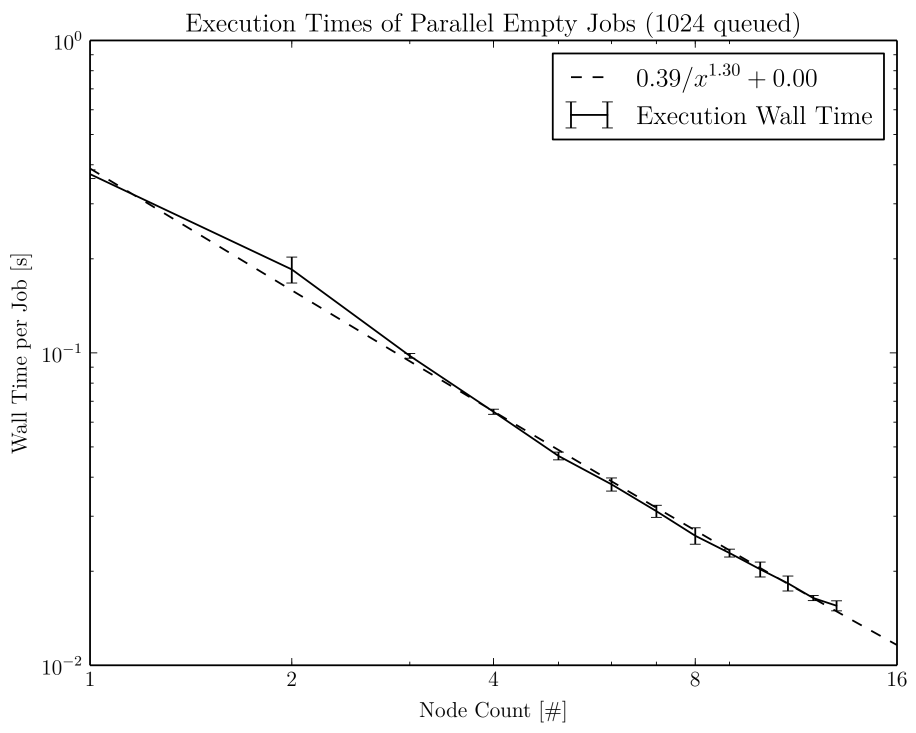


Figure 2.31.: Throughput of the SLURM setup on the HMF-CP in relation to node count. Scaling is super-linear (see fitted dashed line in double-log plot) which is better than the expected linear scaling, i.e. the setup is adequate for HMF and NM-PM1 operation. A quick evaluation of the log files indicated a relation to the execution preparation steps. In particular, the standard UNIX login process was performed faster for larger node counts. Further tests that evaluate this effect are needed.

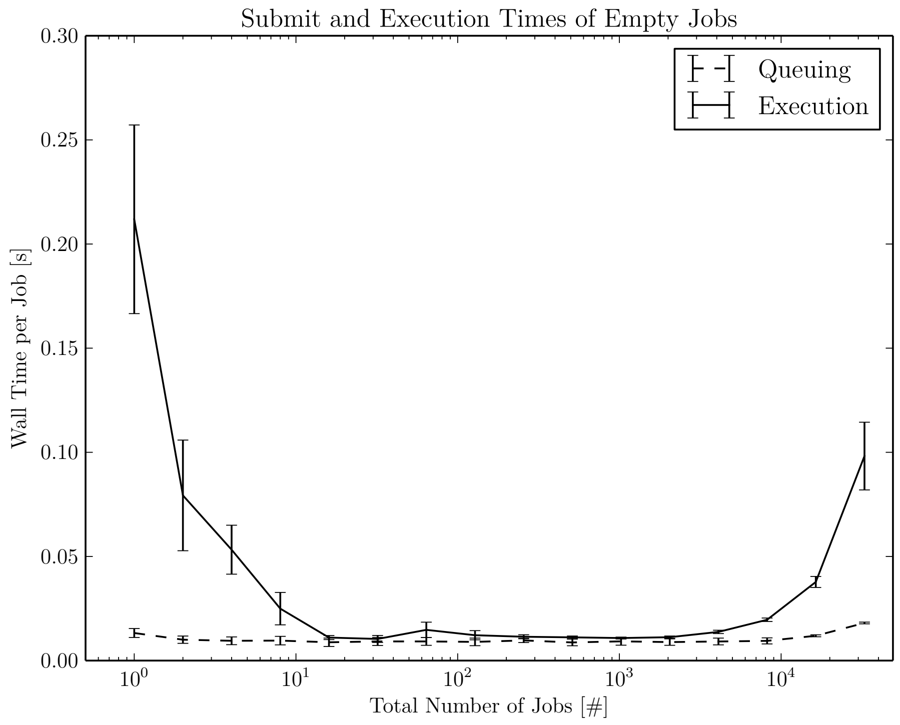


Figure 2.32.: Throughput of the SLURM setup on the HMF-CP in relation to the number of queued jobs. Notably, this does not relate to already executed jobs; i.e. there was no measurable effect on the number of already processed jobs. This measurement was performed on 14 compute nodes of the HMF. The jobs are *empty* and terminate immediately after being started. On the ordinate the average per-job runtime is plotted. Inserting a job into the queue is a constant time operation, the dashed line, i.e. queuing time, stays almost constant. Regarding the execution time, a plateau is reached between 16 and approximately 10^4 queued jobs. Smaller queue lengths suffer from non-pipelined job execution, i.e. the pre- and post-execution job phases cannot be hidden from the total runtime. Above 16 entries, the following jobs can be prepared for execution and this overhead is hidden from the total runtime. Large queue lengths ($> 10^4$ jobs) suffer from SLURM management overhead. Certainly, queue lengths up to 10^4 are already sufficient for the current system. However, further tuning is required when scaling to larger installations (e.g., 1000-wafer modules).

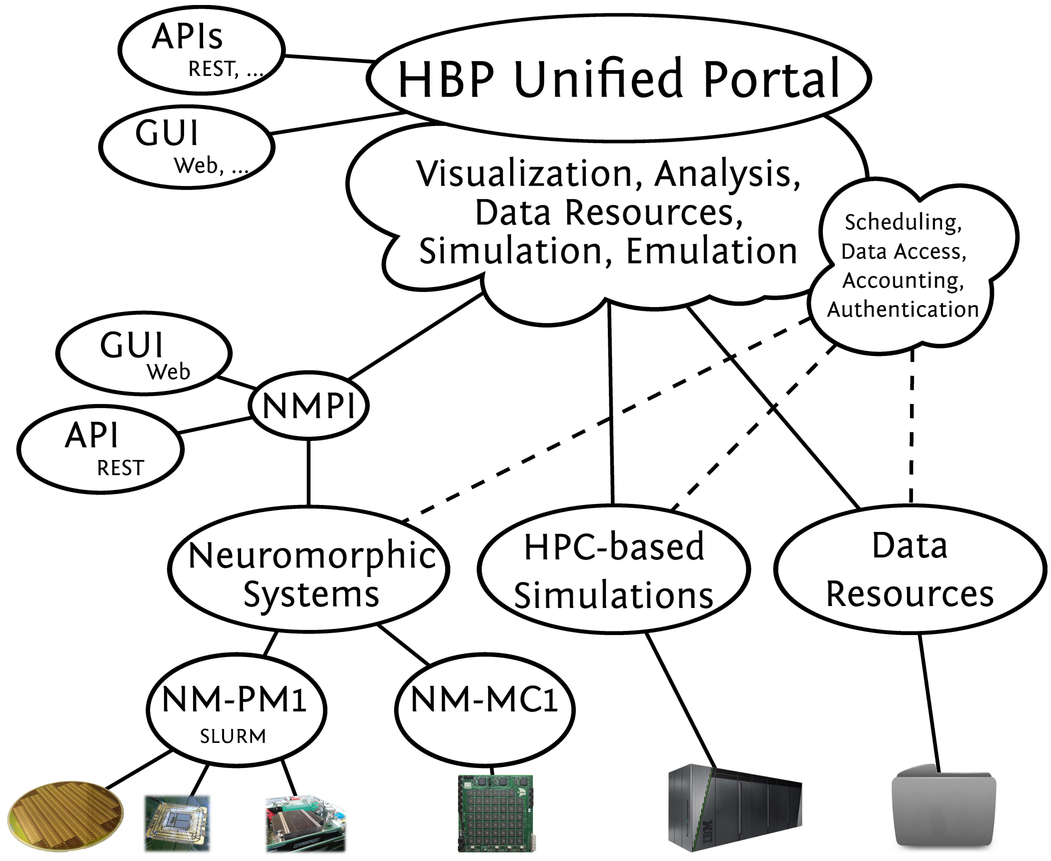


Figure 2.33.: Within the HBP project, a unified interface is currently under development. HBP envisions this *unified portal* as a common interface for accessing data resources, visualizing and analyzing data, and simulation building as well as simulation on conventional HPC hardware or neuromorphic hardware. For this purpose, a tight integration of the neuromorphic emulation platforms, i.e. the NM-PM1 and NM-MC1 is needed. The neuromorphic platform introduced a middleware, called NMPI, that provides an REST-based interface for the UP as well as a job queuing system for the NM-PM1 and NM-MC1 sites [Davison et al., 2014]. A link between the NMPI and the SLURM-based resource management system for the HMF/NM-PM1 is already operational. However, global data access and accounting/authentication using the HBP user database has still to be integrated.

2.4. Summary

This chapter presented the implementation of software and FPGA firmware which allows for a fast operation of the HMF and NM-PM1 systems. The first section worked out the details of the communication chain between host computer and the neuromorphic system. Theoretical peak performance has been reached and adequate scalability properties have been shown. However, the author identified insufficiencies in the current FCP FPGA firmware implementation; the responsible FPGA developers are currently working on fixes to resolve these problems.

The second section covered the higher software layers. A new software architecture based on the pipeline data flow model (cf. Clements [2011]) has been presented. The inter-operation performance between the pipeline stages has been measured and the results show sufficient performance. However, the optimization of neuronal network experiment throughput has just begun. At the time of writing, the configuration of analog neuron parameters[60] dominates the total configuration time. Hence, optimization efforts have to concentrate on this hardware-specific property first. Later optimizations, like pipelining of experiments, will become worthwhile as soon as the duration of the configuration stage surpasses the experiment transmission time.

The third section covered efforts to enable a larger user base to work on a multi-wafer neuromorphic installation. The SLURM resource manager has been evaluated; it provides adequate performance for the HMF and NM-PM1 systems; even larger installations are already possible. However, when reaching 1000 wafer modules another evaluation[61] will become necessary. Within the HBP project, the integration of the NM-PM1 system into the UP is a current topic; the current state was discussed.

[60]The analog parameters are stored in *floating gate blocks* which are accessed by an on-chip controller. Up to the present, optimization was focused on parameter precision.

[61]The resource bookkeeping increases with the number of resources. Thus, dedicated databases will be needed to provide adequate performance.

3. Hybrid Operation

One of BrainScaleS (BSS)'s main foci is bridging scales – temporal and spatial ones. Building experiments of varying network size, detail level and run times challenges conventional simulators. Without thorough optimization, general purpose high-performance computing (HPC) machines are not very efficient when it comes to neuronal network simulations. Spike data communication methodology is an ongoing research topic [Helias et al., 2012; Hines et al., 2011; Thibeault et al., 2013]. Memory consumption is another limitation on the maximum network size that can be simulated [Kunkel et al., 2012]. On a more fundamental level, advances based on the von Neumann architecture suffer from increasing difficulties in the feature size reduction [Thompson and Parthasarathy, 2006], power consumption [Esmaeilzadeh et al., 2011; Shafique et al., 2014] and communication [Perrin, 2011]. Large scale simulations running on HPC hardware already carry a price tag due to the energy consumption. Usually, power consumption is one area where neuromorphic systems are considerably better than conventional simulations. Another crucial feature of *accelerated* neuromorphic systems is the speed-up factor (cf. section 1.1.5) that allows for fast parameter space exploration and long-running experiments. Studies of learning and development are typical applications for large-time-scale experiments. Especially the interaction of such experiments with a simulated environment is a promising field of application for neuromorphic systems.

The BSS project envisions the combination of an accelerated neuromorphic system and a conventional HPC system, two fundamentally different computational concepts combined into a hybrid system. For non-accelerated, or real-time, neuromorphic systems, robotics are a typical application. However, in case of accelerated systems, the real world is too slow for most applications that depend on interaction. Simulated environments provide the possibility to benefit from the speedup and to perform closed-loop experiments.

The basic idea is to combine flexible software simulations running on conventional compute hardware and neuronal networks being emulated on the neuromorphic part. One vision is to simulate virtual environments exchanging sensory data and motor commands with the neuromorphic hardware. It is important to note, that the inherent speedup factor, see section 1.1.5, of the accelerated neuromorphic hardware gives a constraint on minimum simulation speed. Sparse coding schemes and fast computability are key aspects of the software implementation. When running such experiments on a hybrid system, a broad range of temporal scales can be explored.

In BSS terms, this system is called the Hybrid Multiscale Facility (HMF). Building this system is a major goal of the BSS project. The HMF comprises the accelerated neuromorphic hardware and a small HPC cluster. A description of the

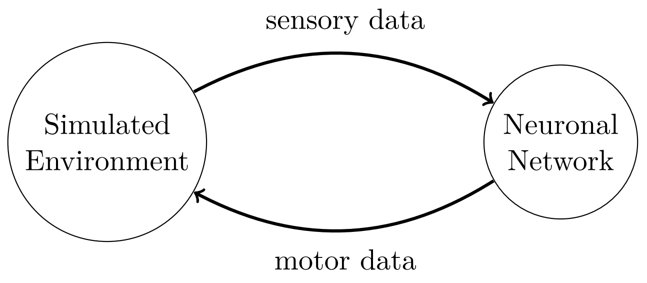


Figure 3.1.: Closed-loop (as defined in section 3.1) schematic. A simulation environment provides sensory data for the neuronal network to compute an answer, for example, the motor data. In the chapter at hand, it is assumed that the timing of sensory and motor data has a rather fixed relation with defined maximum connection latencies – this operation mode is called *hybrid* or *real-time* closed-loop mode.

neuromorphic part can be found in section 1.1, for details (hardware implementation) see Millner [2012]; Millner et al. [2010]. The cluster architecture was developed by the author and the design is presented in section 1.2.

3.1. Closed-loop Experiments

Tagging a specific setup to be a *closed-loop* experiment is ambiguous. However, in the field of neuroscience there is basic consensus for some properties of the term (cf. [Potter et al., 2014]): 1. there is some kind of interaction between one or more components; 2. the interaction consists of two elements, one forward data flow and an associated feedback connection closing the *loop*.

In the following, we focus on the interaction between a spiking neuronal network and another component that interacts with the neuronal network by means of processing sensory information and providing feedback to the network. This component is called *simulated environment*. Figure 3.1 shows a sketch of this setup.

3.2. Real-time Closed-loop Operation

Operating the HMF in closed-loop mode is challenging. The speed-up factor (cf. section 1.1.5 – typically 10^4 for the current hardware revision) and communication latencies impose an upper bound for communication properties (i.e. throughput and latency) as well as simulation detail. Specifically, for synchronized operation of a software simulation and the time-continuous neuromorphic substrate it is essential to provide predictable and minimal communication latencies.

For software simulations it is also demanding to process incoming sensory data quickly enough to provide motor output before the maximal latencies are reached.

Missing the deadline[1] by a large margin typically disables correct network behavior – however, this depends on the properties of the neuronal network model. To a certain extent it might be possible to tune models to cope with increased communication latencies. The current neuromorphic hardware implementation uses typical speedup factors in the order of 10^4 – providing a similar speed using software simulators is difficult, see section 3.5.5. For later hardware revisions it is planned [Electronic Vision(s), 2014] to support lower speed-up factors and lower-latency data network technologies. The latter would reduce the real communication latency, and the former would reduce the latency in the *biological*, or simulation, time domain of the neuronal network. Basic communication schemes between HMF neuromorphic part and HMF conventional part are presented in the next section.

In the following paragraph typical properties of current data network technologies are stated. State-of-the-art commercial off-the-shelf (COTS) technologies like 10-Gigabit Ethernet (10GbE) or 40-Gigabit Ethernet (40GbE) support hop latencies of down to 270 ns (10GbE) and 220 ns (40GbE) [Mellanox, 2013a] and approximately 1 µs message passing interface (MPI) one-way ping latency [HP and Mellanox, 2012; Mellanox, 2013b] while providing up to $40 \cdot 10^6$ messages per second [Tolly, 2012]; power consumption is around 2 W to 4 W per port. Mellanox [2014] reports down to 100 ns hop latency for some InfiniBand description switches. The EXTOLL High Performance Interconnection Technology supports even higher throughput and reduced latencies at lower power consumption: For the Tourmalet network interface controller (NIC) EXTOLL [2014] reports 400 ns to 600 ns MPI one-way ping latency, 60 ns port-to-port latency while providing up to $100 \cdot 10^6$ messages per second. Typical power consumption is 1 W per optical port.

3.2.1. Data Exchange

Conventional closed-loop setups combine software simulators running in simulated time. This communication scheme is shown in figure 3.2. The simulations can be paused and resumed by the operating system and in principle[2] it is possible to provide means of data injection and extraction during the simulation. Communication time steps (i.e. the time interval between data injection and extraction points – possibly but not necessarily equivalent to a calculation step) provide synchronization points for data exchange. The simulations are paused until all communication partners complete their time step and provide data for the data exchange.

Classical distributed neuronal network simulators, like the NEURON Simulator [Hines and Carnevale, 2006], use a very similar communication scheme for normal operation. Some simulators, e.g. NEURON Simulator [Hines and Carnevale, 2006], support a partially interleaved communication and simulation execution scheme: Hines et al. [2011] reports on an evaluation of this scheme using the proprietary communication technology of a IBM Blue Gene/P supercomputer; for large clusters

[1]A term often used in real-time systems: deadline scheduler try to guarantee a specific start time for a request. See, for example, Stankovic [1998].

[2]Modifications of software source code are needed.

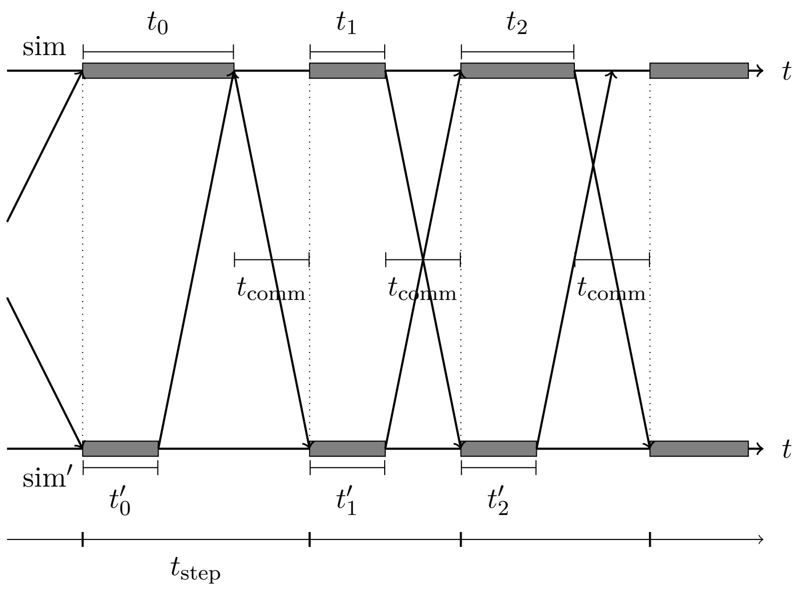


Figure 3.2.: Schematic showing the interaction between two software simulations running with the same simulation time step but varying execution speeds, i.e. *conventional closed-loop* experiments. On the horizontal axes the wall clock times are plotted. Differing simulation run times per *simulation time step* are represented by varying run times $t_{0..n}$. Exchanging data between the simulators consumes the time t_{comm}. This overhead time adds up to the effective time step $t_{\mathrm{step}} = \max(t_n, t'_n) + t_{\mathrm{comm}}$, that is the effective simulation speed of the combined simulation.

(i.e. more communication partners in the cluster than the typical neuron fan-in count) and large network sizes this provides improved scaling properties. Eppler et al. [2007], for instance, report improved performance on a small cluster using `MPI_Allgather()` compared to the Complete Pairwise Exchange [Tam and Wang, 2000] algorithm that then was the default for NEural Simulation Tool [Gewaltig and Diesmann, 2007] (NEST); in the current version [NEST Initiative, 2014] it is based on shared-memory for pure thread-based simulations and for MPI-based simulations it is configurable to be a CPEX or `MPI_Allgather()` (default).

MUSIC Combining different, already existing large-scale simulations into a larger-scale simulation is a common task. One interesting approach to simplify this task is the Multi-Simulation Coordinator (MUSIC) application programming interface (API) [Ekeberg and Djurfeldt, 2009]. It provides an interface for software simulators for inter-simulator data exchange. A thin (i.e. typically one extra function call) MPI wrapper binary splits the computational resources into configurable chunks that are subsequently used by the individual simulators. The communication channels

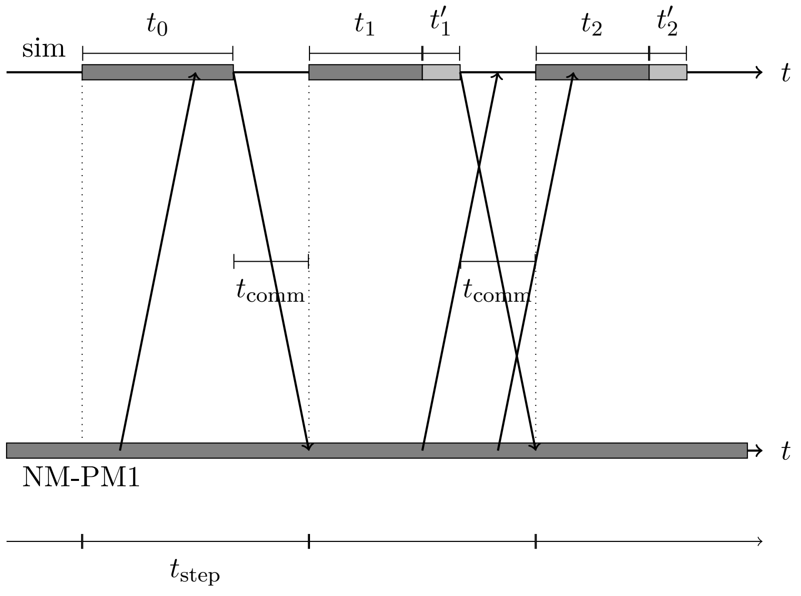


Figure 3.3.: Schematic showing the interaction between software simulator and neuromorphic hardware. This setup is used for *hybrid*//time-continuous closed-loop experiments. In contrast to the conventional setup (see figure 3.2), the lower communication partner operates in continuous time. The minimal effective time step for the environment simulator update has an lower bound of $t_{\text{step}} = max(t_{0..n}) + t_{\text{comm}}$. If the time step is decreased further, the simulated environment will at least once fail to send response data *in time*.

between the simulators are managed by MUSIC which employs a look-up table for translating between simulator-local identifiers and the global, MUSIC identifiers[3]. Within the simulation environments dedicated *proxy* units – neuronal connection endpoints, similar to neurons – encapsulate the non-local communication channels [Djurfeldt et al., 2010].

HMF The previous, simple communication scheme relies on the fact that every communication partner can block until a certain datum is available. This blocking behavior does not affect the simulated time domain but only the *wall-clock*[4] time. In the light of time-continuous simulation partners this is not possible as *pausing* is not an option. Hence, for communication with the HMF neuromorphic part – where neuronal network state evolution cannot be paused –, the communication scheme has to be adapted. Figure 3.3 presents this communication mechanism.

After powering up and configuring the HMF neuromorphic part (HMF-NP) the

[3]In the MUSIC documentation they are called *global indices*.

[4]The wall-clock time is the (real) time that passed between starting and finishing a task

neuronal network can be already *active*[5], e.g. by setting up a network that allows for self-sustained activity. The simulated environment generates input for the on-wafer network, while the on-wafer network sends information back to the environment. We can certainly define experiments that do not rely on any time relation between the communication partners – but in the context of this chapter we assume that the setup has real-time timing constraints. This means that if any communication partner reacts too fast or too slow, the experiment will fail.

3.3. Hardware Platform

Based on the experiences acquired during the FACETS project (details about the communication protocols in section 2.1) the BrainScaleS and Human Brain Project NM-PM1 use Gigabit Ethernet (1GbE) as the FCP FPGAs[6] host interface. In particular, the evaluation of the transport layer for the Spikey-based[7] neuromorphic systems looked promising and it was decided to use standard 1GbE technology for the host interface. For details of the host interface see section 2.1.1.

The wafer-scale system comprises multiple FPGA communication PCBs (FCPs). Hence, using link aggregation and 10GbE was a simple solution for horizontal scaling. The NM-PM1 utilizes 48 individual FPGAs per wafer.

For the BSS project, the aspect of real-time interaction between software simulations and neuromorphic hardware emerged. The next sections cover neuronal model requirements, the method to measure latency and results from the HMF and the upcoming NM-PM1 systems.

3.3.1. Latency Requirements

The latency requirements for closed-loop operation mode yields additional design constraints for the control hosts. If we assume a speed-up factor of $S = 10^4$, in the biological domain all latencies are increased by the same number – i.e. the wall-clock time interval is scaled by the same factor. This means that the interconnection between state-of-the-art 10GbE NICs [Mellanox, 2013b] (approx. 1 µs MPI one-way latency) yields:

$$\begin{aligned} t_{\text{latency}_\text{wall}} &= 1\,\mu\text{s} \\ t_{\text{latency}_\text{biological}} &= \text{S} \cdot t_{\text{latency}_\text{wall}} = 10\,\text{ms} \end{aligned} \tag{3.1}$$

We also have to take into account that the current wafer module, more precisely the vertical wafer I/O PCBs, do not directly support 10GbE but rather 1GbE which increases the minimum latency because of the lower bit rate and because of

[5]That means that one can measure spike activity on the wafer.
[6]FPGA communication PCB field-programmable gate array
[7]Spikey is a chip-based neuromorphic system developed during FACETS that implements 384 LIF neurons and approximately 100k synapses

the media conversion that is performed by a switch. For the BSS setup the switch specification states below 2.9 µs for 1GbE and 1.3 µs for 10GbE [Hewlett-Packard, 2013]. Section 3.3.3 provides results for measurements on the HMF system.

For the current speed-up and hardware setup it seems difficult to replace arbitrary neuronal network parts by some software part as the typical delays within the network are significantly below that range. However, constructing experiments that replace whole sensory input pathways representing biological latencies of around 100 ms could be possible if the computational complexity can be kept low.

3.3.2. Communication Interface

The FCP FPGA uses a dedicated, logical interface for real-time spike communication based on a collaboration between the Spiking Neural Network Architecture (SpiNNaker) project and the developers of the FCP firmware at Technische Universität Dresden (TUD) (cf. Rast et al. [2013]). The current implementation is limited to the essential requirements of real-time communication: it avoids any buffering and is a code path that only uses a small look-up table (supporting 1024 entries) to translate addresses and forward spikes between HICANN chip and host computer[8]. Figure 3.4 depicts the SpiNNaker frame format. Internet protocol version 4 (IPv4) and user datagram protocol (UDP) headers are still used but are not shown here. An overview of the standard protocol layers can be found in section 2.1. The split into communication channels (for example, real-time spike data and non-real-time configuration data) is based on the UDP port number.

To configure the FPGA SpiNNaker interface UDP port 1850 is used. At the time of writing, the packet specifications are not yet available in the system HBP SP9 Specification [2014] but only in source code (lowest-level API for hicann access (hicann-system):`units/stage2_hal/source/spinn_controller.cpp`).

During Capo Caccia Cognitive Neuromorphic Engineering Workshop 2013, the spike data specification was developed further. Quoting the wiki page[9], the formatting is now:

> *The payload will consist of two 32 bit containers. The first 32 bit contain in the lower bits 24 bit pulse source ID, the upper 8 bit are reserved (i.e. for additional, application-specific payload). The second 32 are similarly split, with the lower 24 bit encoding source timestamp, with the upper 8 bit again reserved for custom payloads. These custom payloads could be e.g., amplitude values in an interface to MEAs [multi-electrode arrays].*

At the time of writing, there is no comment on longer frames (i.e. multi-spike packets). However, the current implementation supports multiple spikes per frame according to figure 3.4. Further details on the source timestamp:

[8]Currently, the FPGA implementation only support a single remote communication partner per FCP.

[9]`https://capocaccia.ethz.ch/capo/wiki/2013/immns13#UDPStandard`

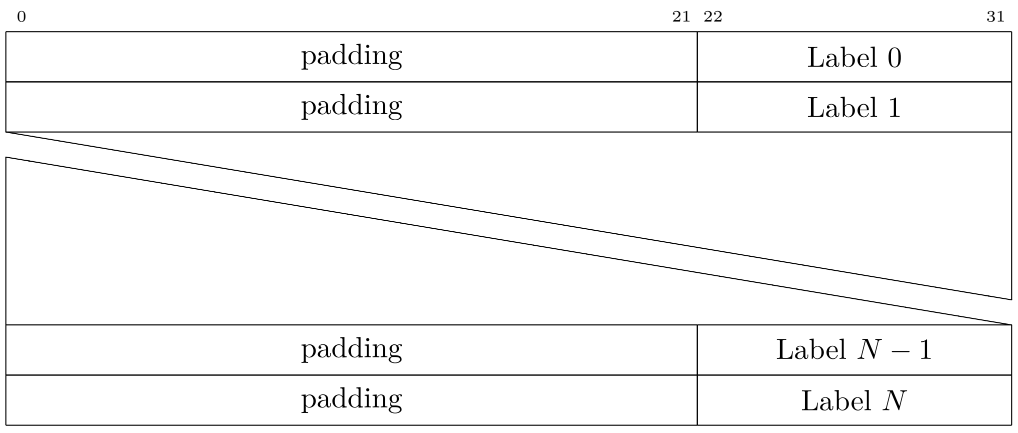


Figure 3.4.: SpiNNaker input frame format. Entries are aligned to 32-bit *words*. The lower 10 bit within each word encode an event identifier (the *label*) that is used by the FCP FPGA as a look-up index to translate between SpiNNaker-specific and HICANN-chip-specific L1 addresses. IPv4 and UDP headers are still used to specify the communication channel between compute node and FCP FPGA, see section 2.1 for details. For example, SpiNNaker spike data is expected to use UDP port 1851.

> *The default equivalent of the least significant bit (LSB) timestamp would be* 50 µs*. This value was chosen since we cannot go much lower due to latency of Ethernet switches (even for a local network). On the other hand, much larger values could lead to timing precision problems with respect to MEA interfaces. With a 24 bit timestamp, we can thus have a 14 min experiment before the timestamp wraps. If track is kept of the wraparounds, this should be sufficient for unambiguously identifying every pulse for any reasonable experiment (i.e. where the max distance between two consecutive pulses from the same source is lower than 14 min).*

The specification seems to be rather focused on real-time systems, as the definition of timestamps is too coarse for accelerated models and the comment on Ethernet latency does not apply to newer Ethernet specifications like 1GbE or 10GbE.

Based on these specified protocol properties, it is not adequate for the HMF and NM-PM1 systems to implement this specification. Nevertheless, the current firmware adheres at least the spike payload format which makes it possible to connect HMF and SpiNNaker systems and possibly other neuromorphic devices – at least technically; however, the speed-up mismatch makes closed-loop experiments difficult or even impossible.

A technical demonstration[10] of spike exchange between chip-based HICANN setup and SpiNNaker is presented in Rast et al. [2013]. However, this setup uses

[10]It demonstrates a closed-loop interaction using a speed-up conversion that scales spike rates up and down between the systems.

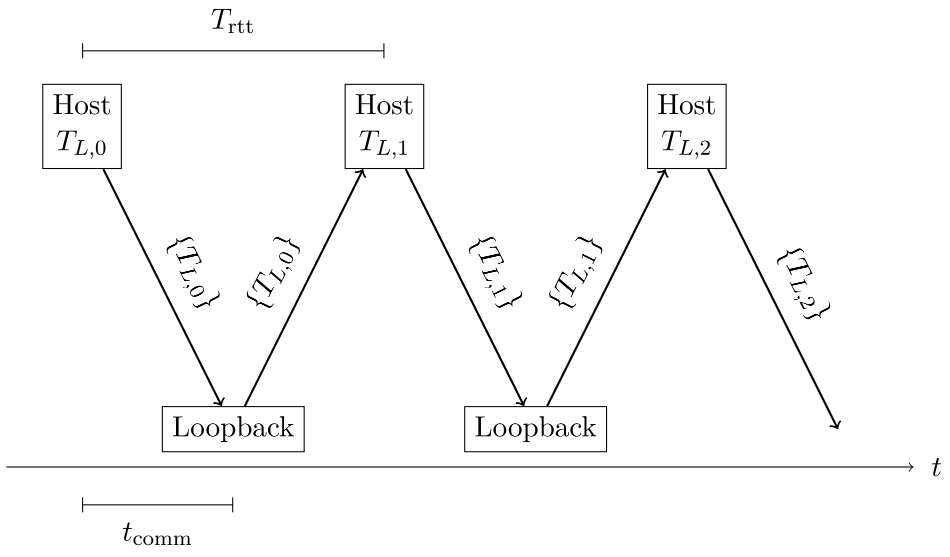


Figure 3.5.: Measurement of the round-trip time. The host sends local *timestamps* T_L to a communication partner that loops the incoming data back to the host.

a *spike rate translation* module to adapt the spike rate to the individual system. This solution works for exactly the presented experiment but is not universally applicable.

3.3.3. Latency Measurement

Figure 3.5 shows the measurement procedure for communication latency measurements – basically the same method that the UNIX utility `ping` uses. One communication partner sends machine-local timestamps T_{L_i} to a remote node which loops back the timestamps. Thus, the communication round-trip time is given by:

$$T_{\text{rtt}} := T_{\mathrm{L}i+1} - T_{\mathrm{L}i} \tag{3.2}$$

Assuming symmetrical communication partners, the one-way latency is:

$$T_{\text{latency}} = \frac{1}{2} \cdot T_{\text{rtt}} \tag{3.3}$$

If we include remote timestamps to the returned data (i.e. replacing $\{T_{L,n}\}$ by $\{T_{L,n}, T_{R,n}\}$), the two communication partners can now synchronize their clocks to each other by applying equations (3.2) and (3.3). To estimate the remote time T_{R}

at time $i + 1$:

$$T_{\mathrm{R}i+1} = T_{\mathrm{R}i} + \frac{1}{2} \cdot T_{\mathrm{rtt}} \tag{3.4}$$

Because of timing jitter, the implementation uses window-based averaging to ensure smooth evolution of time estimates:

$$T_{\mathrm{rtt}} = \frac{1}{N} \cdot \sum_{j=i-N}^{i-1} T_{\mathrm{L}j+1} - T_{\mathrm{L}j} \tag{3.5}$$

with $N \approx 100$.

Measurement Software The latencies measurements in the following sections use a custom ping implementation that uses the virtual environment for closed-loop experiments (VerCL) software package developed by the author (for details see section 3.4). For the compute node-based measurements, both communication partners start the measurement tool. The *slave* only replies to incoming packets with copies of the inbound data, the *master* sends local timestamps and evaluates the time difference for responses. That means, the communication partners execute the ping-pong scheme presented in figure 3.5 10^5 times. The individual round-trip times (RTTs) acquired are binned into 1 ns bins – that is the resolution for time measurements using the UNIX API (i.e. `clock_gettime()`).

3.3.4. HMF

If not stated otherwise, all measurements in this chapter have been performed using the first BSS prototype wafer system. A detailed description of the system, as well as the differences from later NM-PM1-based systems can be found in section 1.1.6.

Differences in the wafer module exist due to the reorganization of the FCPs. The early prototype systems use 12 FCPs (based on Xilinx Virtex-5 FPGA) per wafer module, the later BSS and NM-PM1 systems use 48 FCPs (based on Xilinx Kintex-7 FPGA) per wafer module.

Figure 3.6 shows individual latency measurements and the time synchronization mechanism using the methods described in equations (3.3) and (3.4). The progress of the clock synchronization mechanism is shown in the upper plot. After the initial synchronization phase, ranging up to approximately 2000 iterations (i.e. $2000 \cdot T_{\mathrm{rtt}} \approx 50\,\mathrm{ms}$), the time difference stabilizes. Using HMF compute nodes that are directly connected, we obtain for one-way latency (figure 3.6):

$$T_{\mathrm{latency}_{\mathrm{hmf}}} = 12.1\,\mu\mathrm{s} \pm 0.2\,\mu\mathrm{s}.$$

The error distribution is slightly skewed towards higher latencies as the lowest possible latency is a hard barrier but higher values can occur due to suboptimal software timing. Details can be found, for example, in Brown and Martin [2010].

Error ranges are given as root-mean-square error (RMSE) values. For the new NM-PM1 compute node architecture, histograms are plotted in figure 3.7.

3.3.5. NM-PM1

Section 1.1.6 explains the differences between the production-type HMF/NM-PM1 systems and the existing prototypes. From this chapter's perspective, the main difference is the compute node setup as the NM-PM1 system uses a newer CPU generation and updated NICs.

The decrease in latency is important for closed-loop operation. Compared to the latency measured for the BSS cluster, the latency improves 5-fold:

$$T_{\text{latency}_{\text{mix}}} = 2.4\,\mu\text{s} \pm 0.2\,\mu\text{s}. \tag{3.6}$$

As the software environment and network topology stayed the same, this improvement is caused by the improved node hardware specification (cf. section 1.2.1 and [Chelsio, 2013a]). The updated NIC supports a proprietary communication API [Chelsio, 2011] that could further reduce the latency – that option has not yet been evaluated. For Chelsio's top-of-the-range NIC T520-LL-CR[11] one-way UDP latencies down to 1.687 µs are given by Chelsio [2013b].

A histogram of the distribution of individual measurements is shown in figure 3.7. From this data we can derive estimates for the hop latencies of different network components. Because of the two-fold crossing of every component when using the ping-pong protocol (figure 3.5) to measure the round-trip time T_{rtt} we estimate the hop latency t_{latency_i} of component i to be:

$$T_{\text{latency}_i} = \frac{1}{2} \cdot T_{\text{rtt}_i} = \frac{1}{2} \cdot (T_{\text{rtt}} - \sum_{j \neq i} T_{\text{rtt}_j})$$

For the NM-PM1 backbone switch [Hewlett-Packard, 2014b] we obtain approximately 0.6 µs and for an aggregation or wafer switch [Hewlett-Packard, 2014a] we obtain 1.0 µs. Both values are well below the specified worst case latencies [Hewlett-Packard, 2014a,b] – which is what we expect due to low load, i.e. the network infrastructure was only used by this connection test.

3.3.6. HMF vs. HICANN Latency

Figure 3.8 presents a latency measurement between HICANN chip and compute node according to section 3.3.3. The HICANN chip configuration is basically the same as for the real-time closed-loop experiment setup presented in section 3.5.6 which utilizes the chip-based loop-back mode. At first sight, the author decided to test switch behavior when converting from 10GbE to 1GbE and used an artificial setup that contained two aggregation switches connected via 1GbE and the compute

[11]That is the *ultra-low-latency* version of the NM-PM1 compute node NIC.

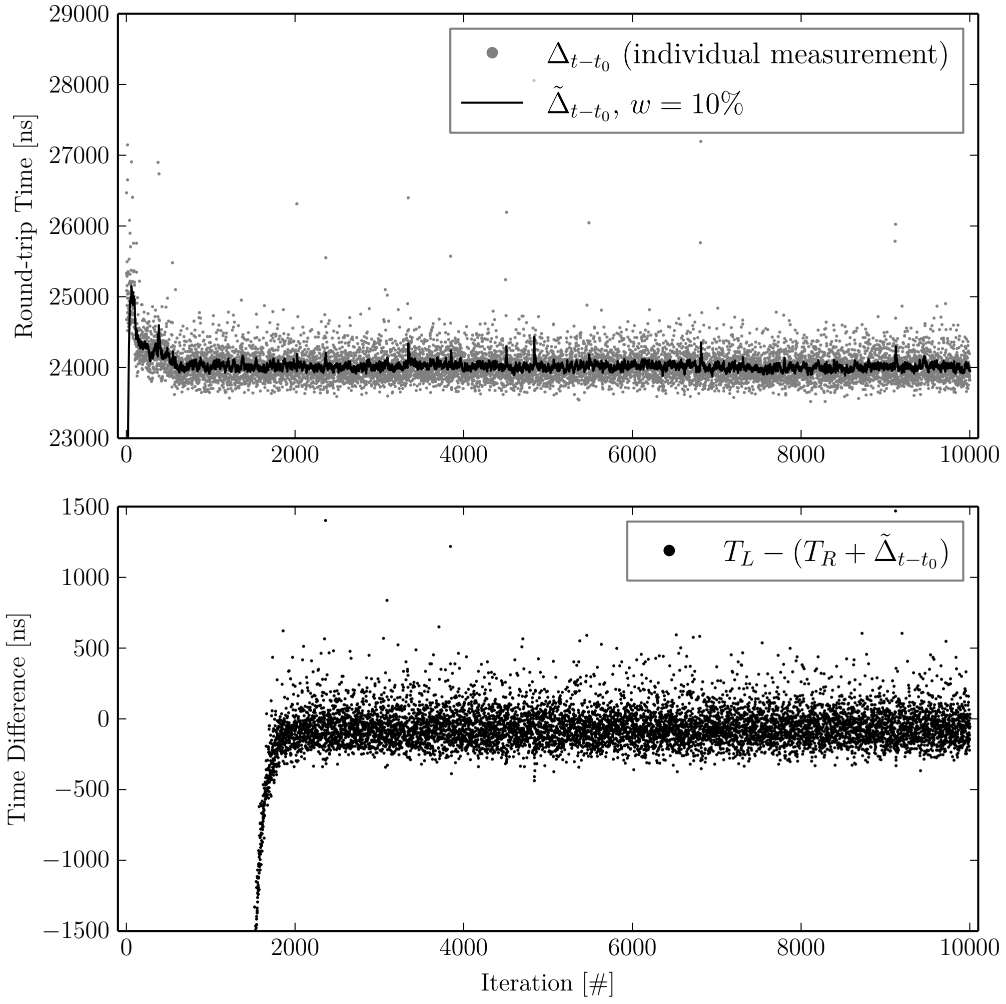


Figure 3.6.: The upper panel depicts a RTT (i.e. two times the end-to-end latency) measurement between two HMF compute nodes. The solid black line shows the windowed average of the gray data points which represent the RTT of individual measurements. The RTT converges to $24.2\,\mu s \pm 0.4\,\mu s$. In the lower panel, the time synchronization between two compute nodes is shown. After the initial synchronization phase ($2000 \cdot T_{\mathrm{rtt}} \approx 50\,\mathrm{ms}$), the calculated time difference converges to $0.0 \pm 0.3\,\mu s$. This plot uses the methods described in equations (3.3) and (3.4).

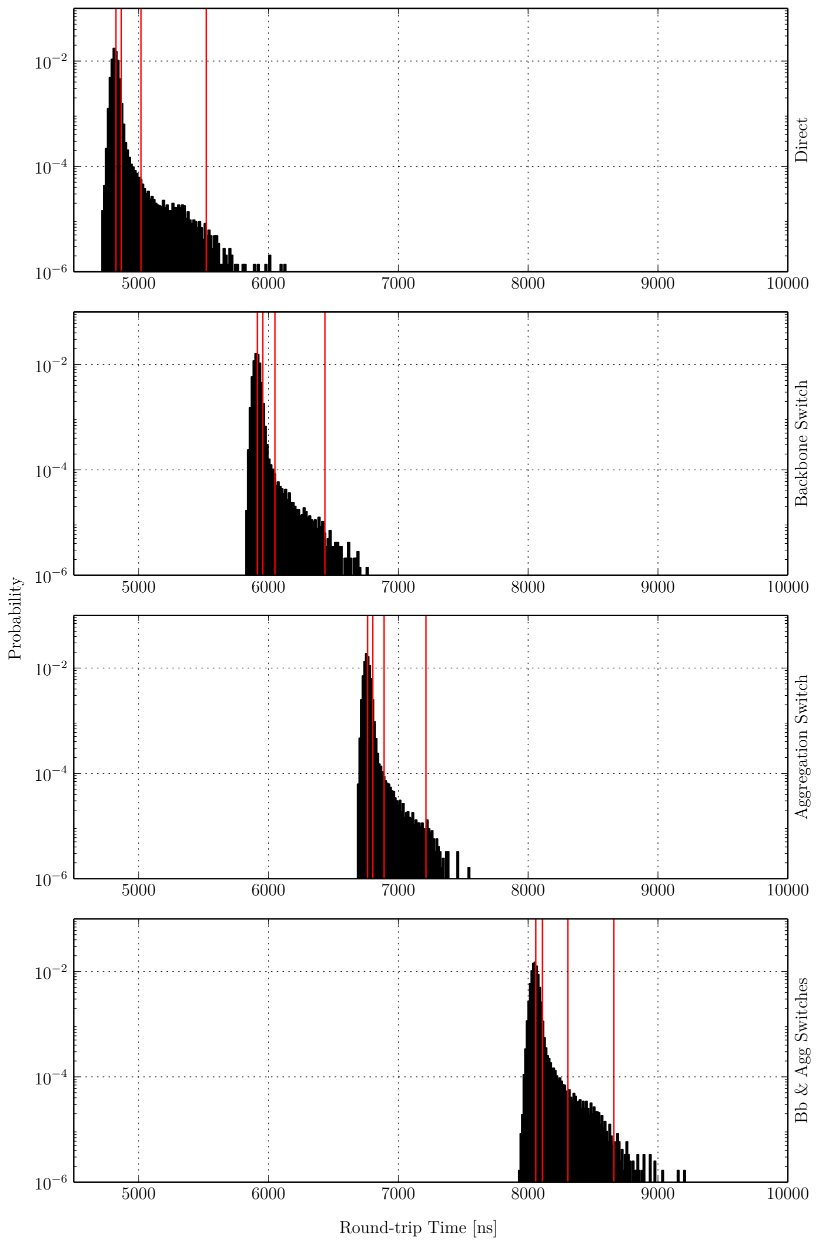


Figure 3.7.: Round-trip time (i.e. two times the end-to-end latency) measured between multiple NM-PM1 compute nodes: 1. directly connected (4.8 µs ± 0.3 µs); 2. connected via the backbone switch (5.9 µs ± 0.3 µs); 3. connected via one aggregation/wafer (agg) switch (6.8 µs ± 0.3 µs); 4. connected via backbone (bb) and one aggregation (agg) switch (8.1 µs ± 0.4 µs). The vertical red lines represent from left to right: average, 95%-percentile, 99%-percentile and 99.9%-percentile out of all measured RTTs (i.e. 10^5 *ping*s). Bin size was set to 1 ns – which is also the timer resolution.

nodes connected to each switch via 10GbE. This setup should give a rough estimate of the expected latency between FCP FPGA and compute node as for the HMF – as well as for the NM-PM1 – network topology we use one wafer switch and one backbone switch to connect the components (cf. section 1.3).

Due to the separated locations of prototype setups and cluster hardware[12], it was not easily possible to use this setup for latency measurement but the author resorted to a single-aggregation switch setup which yields latencies that are slightly below the expected values – see section 3.3.5 for different measurements using one or two switches.

For the HICANN-chip-based test, the measurement software was adapted: we *reset* the FCP FPGA before the measurement and send the spikes to the SpiNNaker network interface (cf. section 3.3.2). The packet length stays constant as we used the same packet layout for the previous measurements. Results are shown in the lower panel of figure 3.8. Both, the increased latency[13], and the massive timing jitter compared to the purely software-based measurements – in figure 3.7 and in the upper panel – are not expected. Partially, the timing jitter can be explained by an unoptimized FCP FPGA SpiNNaker interface implementation as some lookups are sequentially scanning the list of possible targets [Partzsch, 2014]. It is planned to investigate this effect in more detail.

3.4. Software Infrastructure

Based on the experiences of developing the HostARQ protocol (HostARQ) software, the author implemented a thin software layer, VerCL, that uses non-blocking access from and to the operating system to communicate data over the network. A small wrapper for the spike data format described in section 3.3.2 is included as well as small tests to measure latency and ensure functionality. These tests act as regression tests to report performance degradation. In section 3.4.1, the performance of a VerCL-based *ping* implementation is compared to a conventional socket-based implementation.

The interaction between the simulated environment and the NIC is depicted in figure 3.9. The VerCL layer handles the shared-memory communication with the kernel-space NIC driver. For transmission (`TX`) and reception (`RX`) independent memory regions are memory-mapped (using `mmap()`) into the user process space. Packets transmission works as follows:

1. hand over data to VerCL layer using one of the send routines;
2. VerCL waits for empty buffer entries in the circular `TX` buffer by checking a *ready/empty* flag;

[12]At the time of writing the prototype wafer setups were located in the lab, the BSS cluster in the server room, and Human Brain Project (HBP) cluster was located in the container building.

[13]Two intermediate hops between two compute nodes are faster than one intermediate hop for the HICANN-chip-based measurement.

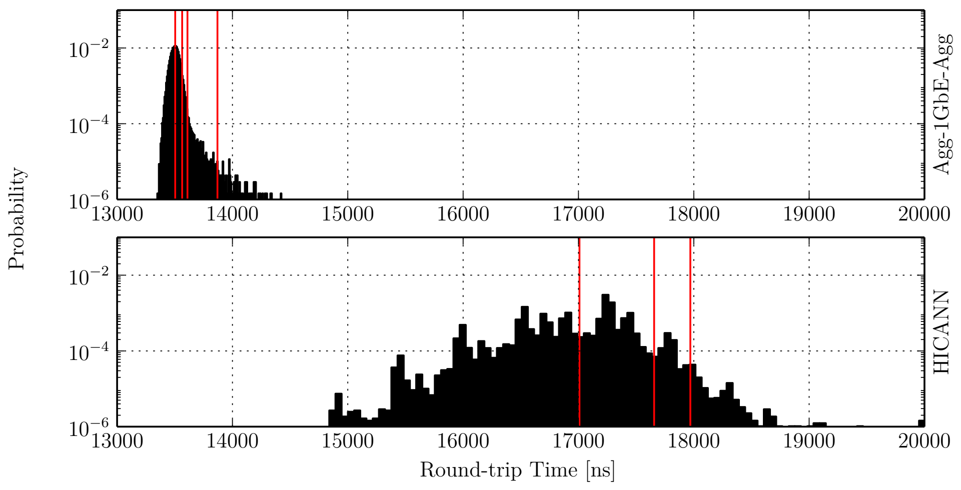


Figure 3.8.: Round-trip time (i.e. two times the end-to-end latency) measured between NM-PM1 compute node and HICANN chip. In the upper panel a setup was constructed to emulate the presence of 1GbE media between two compute nodes – this includes two aggregation/wafer (agg) switches and a 1GbE link in between: $(13.5 \pm 0.4)\,\mu\text{s}$. The lower panel depicts the measurement between a compute node and one HICANN chip which is connected via 1GbE of the FCP: $(17.0 \pm 0.8)\,\mu\text{s}$. The vertical red lines represent from left to right: average, 95%-percentile, 99%-percentile and 99.9%-percentile out of all measured RTTs (i.e. 10^5 *ping*s); the 99.9%-percentile of the HICANN-chip-based measurement is out of plot range ($P_{99.9\%} = 30.8\,\mu\text{s}$). Bin size was set to 1 ns – which is also the timer resolution. The ordinate is plotted in log scale to emphasize the tail towards higher latencies which would not be easily visible in linear scale.

3. VerCL writes the data into the buffer and marks it as *ready*;

4. the NIC driver uses the NAPI, a polling-based interface, [Kelly and Gasparakis, 2010] to wait for new entries (by checking the *ready* flag);

5. after handing over the data to the NIC the driver marks the buffer entry as *empty*.

This lock-free[14] (and syscall-free) mechanism is typically used by packet capture (e.g., `tcpdump`) and injection tools. These properties minimize the amount of context switches needed for communication. Zero-copy versions of the write routines exist; the VerCL layer only checks for the *empty* flag and returns a pointer to the memory location. The client program can now access the memory and, after completion, the VerCL layer marks the location as *ready*.

Receiving side data flow is inverted: the NIC driver marks buffer entries as ready, the user space checks for new entries by checking the *ready* flag.

The source code is a single header that uses compiler macros to enforce inlining of crucial functions for sending and receiving packets. During the setup phase some optimizations are employed to reduce the timing jitter during runtime. This includes pre-allocating and pre-faulting of all used memory regions, forcing memory to physical memory (`mlock()`), setting real-time priority (`sched_setscheduler()`) of the processes, setting CPU affinity masks (`sched_setaffinity()`) to force sending and receiving threads to the corresponding kernelspace worker threads/interrupt handlers of the NIC driver. For typical sources of timing jitter on conventional Linux/x86 hardware see Brown and Martin [2010]; Duval [2009]; McKenney [2009].

3.4.1. Comparison to a Standard `socket()`-based Implementation

To motivate the custom implementation presented in the previous section, we compare it with a standard UNIX socket implementation. The same hardware platform (NM-PM1 compute node and direct 10GbE wiring), the same software environment (i.e. operating system, compiler, etc.) and compiler options were used. Source code for the socket-based implementation is listed in appendix A.4. It essentially opens the socket, and uses `sendto()` and `recvfrom()` to exchange 8-byte timestamps (in ns) between two compute nodes.

Figure 3.10 depicts the latency distribution of both, the VerCL-based and the standard socket-based, implementations. The timing jitter as well as the average latency of the VerCL-based version is significantly lower: the average latency is approximately a factor of 36.4 lower, the timing jitter is reduced by a factor of 8.0.

[14]The Intel x86 CPU architecture (x86) memory model, i.e. the rules for reordering loads to and stores from memory, allows for a barrier-free *empty*/*ready* flag update after writing the associated packet data. For a basic introduction see Hennessy and Patterson [2007, section 4.6]; the x86 memory model is described in IntelArch [2014, section 3.3.1].

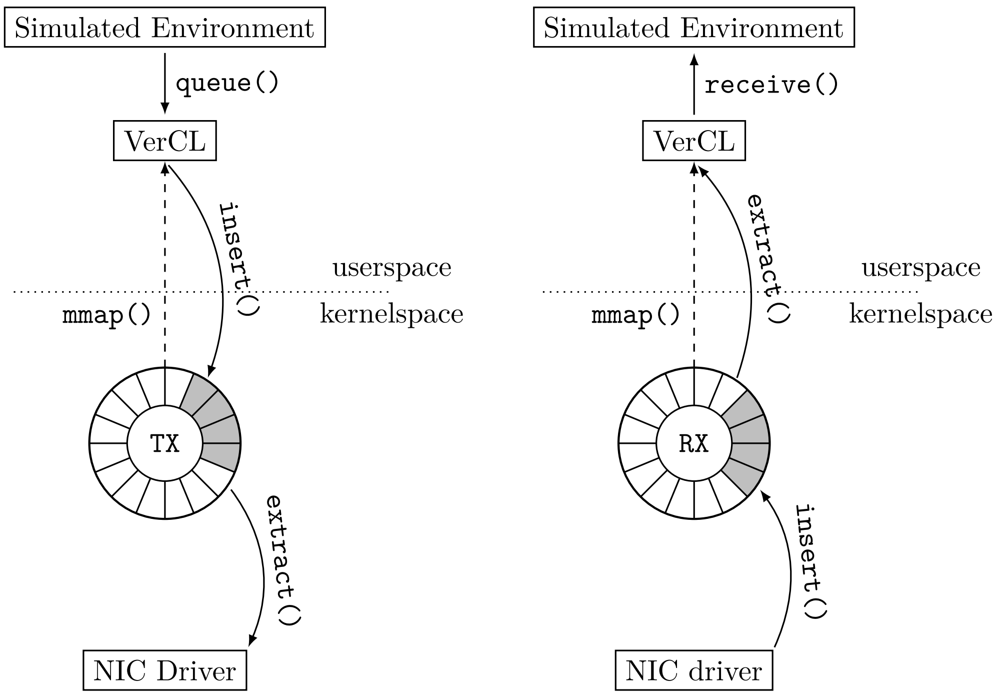


Figure 3.9.: VerCL software overview. The environment simulator runs in userspace and uses the VerCL layer to send data to and receive data from the network. Ring buffers in kernelspace are memory-mapped to the process-space (i.e., into the virtual memory of the simulated environment) and can now be used for `syscall()`-free communication; no context switches are needed for packet reception or transmission.

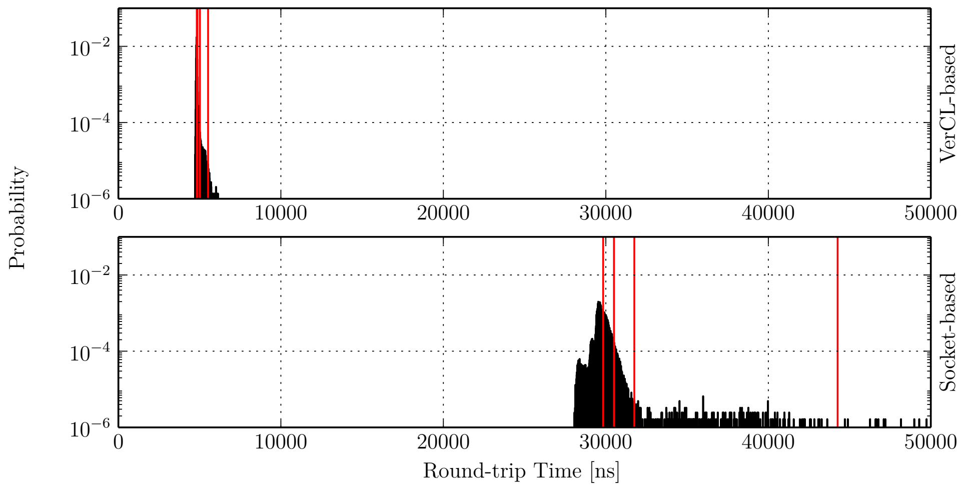


Figure 3.10.: This figure demonstrates the importance of a carefully optimized low-latency software communication framework. The panels show histograms of the relative occurrence probabilities of measured RTTs (on x-axis). Results using the author's software implementation, the VerCL framework, are shown in the upper panel. In the lower panel, results using a traditional `socket()`-based implementation are shown. Both measurements use the same hardware (HMF compute node) and software environment, the nodes are directly connected. The upper plot displays the same data set as already shown in figure 3.7 (uppermost plot). Compared to the latter, the x-axis was zoomed out to make both plots fit into the same range. RTTs are 4.8 µs ± 0.3 µs for VerCL and 174.9 µs ± 2.4 µs for the socket-based implementation. The vertical red lines represent from left to right: average, 95%-percentile, 99%-percentile and 99.9%-percentile out of all measured RTTs (i.e. 10^5 *ping*s). Bin size was set to 1 ns – which is also the timer resolution.

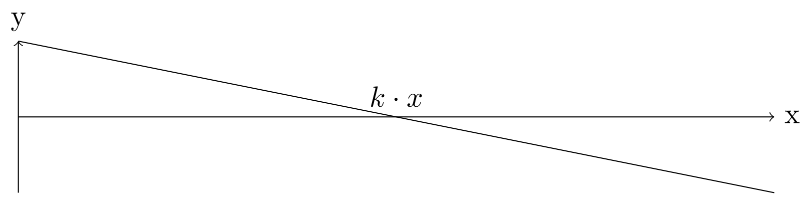
3.5. Experiment

To demonstrate the capabilities of the HMF conventional part (HMF-CP) and the software framework developed by the author, a demonstration experiment is developed and tested in the following sections. There have been significant contributions by Paul Müller and Nils Fischer. In particular, the initial software-simulated implementation of the experiment has been developed by Paul Müller. Nils worked on the blacklisting and calibration of the neuron circuits.

Before demonstrating the final setup which demonstrates the interaction between a conventional compute node and the neuromorphic system, intermediate steps are presented: 1. a non-real-time interaction with a NEST-based neuronal network simulation; 2. a non-real-time interaction with the HICANN chip operating in loop-back mode; 3. a real-time interaction with custom-implemented current-based leaky integrate-and-fire model neurons; 4. a real-time interaction with the HICANN chip operating in loop-back mode; 5. and finally, the real-time interaction with neurons the HICANN chip.

3.5.1. Virtual Environment

The virtual environment models a 1-dimensional space containing a movable object. A force[15] k acts on the object and pulls it to $x_{\text{center}} = 0.5$.



In this model, the object's position is externally updated, i.e. the detected object position is shifted by a time interval τ. This yields for the equation of motion a delay differential equation [Richard, 2003]:

$$\dot{x}(t) = -k \cdot x(t - \tau) \tag{3.7}$$

One solution of this partial differential equation can be found using the ansatz:

$$x(t) = e^{at} \tag{3.8}$$

[15]This is the first time derivative not the second as in $f = m\ddot{x}$.

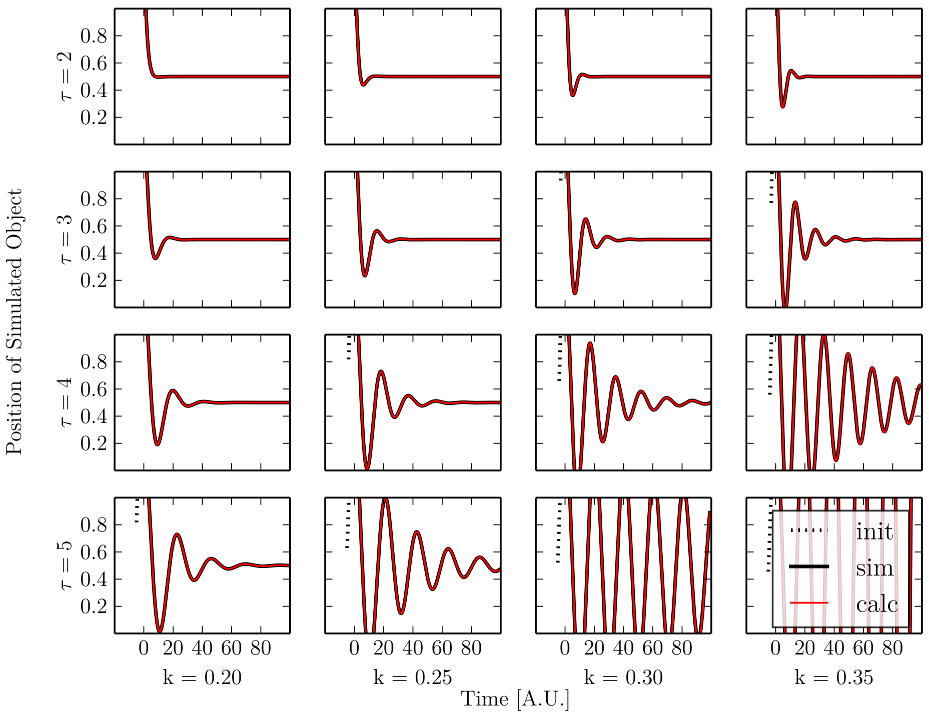


Figure 3.11.: Simulated environment results for varying delays d and forces k. The dotted start of the curves are *past time* values for $t < 0$ obtained from the analytical solution. The black curves show simulated behavior using a step-wise update with $\Delta t = 0.01$, the red lines depict the analytical solution. Due to the small Δt, the curves for simulated and calculated results overlay completely.

After applying equation (3.8) to equation (3.7):

$$\begin{aligned} ae^{at} &= -k \cdot e^{at}e^{-a\tau} \\ ae^{a\tau} &= -k \\ a\tau e^{a\tau} &= -k\tau \\ a\tau &= W(-k\tau) \\ a &= \frac{W(-k\tau)}{\tau} \end{aligned}$$

with the Lambert W function. Hence,

$$x(t) = e^{W(-k\tau)/\tau \cdot t}. \tag{3.9}$$

We can now plot the solution in figure 3.11 using a range of values for k, τ.

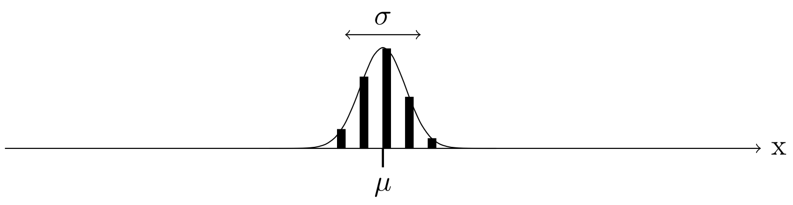
3.5.2. Virtual Environment Implementation

The software implementation of the *environment* is a real-time-triggered loop that updates the simulation in time steps t_{step}. Input of the update loop is the object's position provided by the *detector*, output is the updated object position x_{i+1}. Based on equation (3.7):

$$x_{i+1} = x_i - k \cdot (x_{i+1-d} - x_{\text{center}}) \tag{3.10}$$

with d being the *delay* of the detector response, i.e. the number of discretized time steps to look into the past. The minimum delay is $d = 1$, which corresponds to the previous time step.

Every source and target of a spike encodes a discrete location p_i in space. A histogram of a Gaussian distribution is used to translate between scalar value $x = \mu$ and a spike-based position encoding. σ is a tunable simulation parameter which is kept constant.



For outgoing spike rates ν targeting neuron i (out of N), this yields:

$$\nu_i(t) = \nu_{\max} e^{-\frac{(x_i-\mu)^2}{2\sigma^2}} \tag{3.11}$$

$\nu_{\max}$ is normalized to an average spike-count s per update cycle t_{cycle}:

$$\nu_{\max} = \frac{s}{t_{\text{cycle}} \cdot N \cdot \sigma \cdot \sqrt{2 \cdot \pi}}$$

Input spikes are accumulated for t_{cycle} and extracted at the start of every update cycle. The detected object position is calculated as follows:

$$x_i = \sum_j \nu_j \cdot p_j - x_{\text{center}} \tag{3.12}$$

Using equation (3.10) the updated position is calculated, and the output is transmitted as described above. The experiment continues for a fixed number of cycle iterations.

The complete interaction cycle between the virtual environment simulation (section 3.5.2) and some *detector* is shown in figure 3.12. In the following sections different *detector* implementations will be presented: section 3.5.5 describes cLIF software implementation, section 3.5.6 uses the HICANN chip in *loopback* mode which is normally used for communication tests, and finally using calibrated neurons and synapses of the HICANN chip.

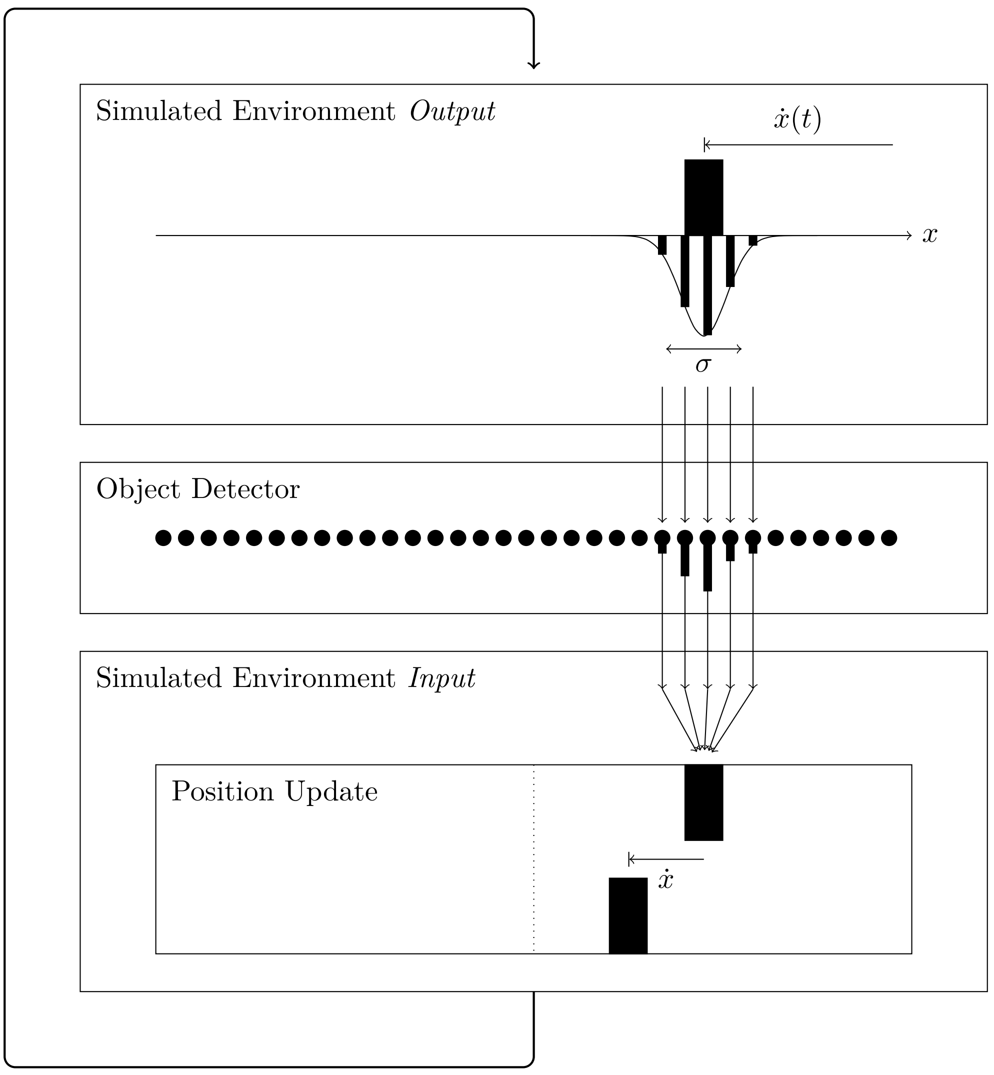


Figure 3.12.: This schematic depicts the time-course of the closed-loop experiment setup. Starting on top, the object is located at x on a 1-dim axis. Its position is updated using the distance to $x_{\text{center}} = 0.5$ and a *force* k (see equation (3.7) for details). The new position is encoded using spikes that encode different locations on the x-axis – a Gaussian profile is used with $\mu = x$ and σ being a constant *projection width*. After the spikes leave the environment simulation, they are transported to a communication partner that is either another simulation (section 3.5.5) or the neuromorphic HICANN chip (sections 3.5.6 and 3.5.7). This partner acts as a simple *detector* for the object position and generates output spikes that encode the detected position. Finally, the detector's output spikes are transmitted to the environment simulation.

3.5.3. Software-based Setup

As a first step, the presented closed-loop experiment has been realized as a pure software implementation. This has been done in collaboration with Nils Fischer and Paul Müller.

Detector neurons are simulated in a software simulator, NEST. The environment simulation is implemented in Python and uses the PyNN API to interface with NEST. After injecting spikes into NEST, the simulation is executed for a simulation time interval t_{step}. Upon completion, the environment simulation extracts the output spikes and updates its state based on the steps presented in section 3.5.2. The conversion between scalar and spike-encoded object location happens at the start (equation (3.12)) and end of the environment update (equation (3.11)).

Resetting (i.e. removing all spikes while keeping neuron states) a simulation in NEST is possible, but in the implementation at hand another execution scheme is used: a sketch of this scheme is depicted in figure 3.13. Based on the reproducibility of the input to output relation of the detector neurons we can implement an execution scheme that increases the total simulation time in a step-like manner. After the n-th simulation time interval t_{step}, the environment simulation produces a set of output spikes that are appended to the neuron simulator's input spike train. When restarting the neuronal simulation and using this ever extending spike train as input, we can simulate now for $n + 1$ time intervals t_{step} and reproducing the spike output of the n earlier time intervals. The only *new* part is the output of the last t_{step} which is the detector response to the input of the last time interval. The main advantage of this scheme is the re-usability for a later neuromorphic hardware-based setup (cf. section 3.5.4). The simulation time t is plotted on the horizontal axis, the simulation iteration i is plotted on the vertical axis. New output is generated in the black boxes that represent the last time step in every iteration. Obviously the total runtime scales quadratically with the number of iterations – the only purpose of this execution scheme is to facilitate debugging.

Figure 3.14 displays simulations using this setup. The *force* k and the *delay* d of the virtual environment are varied (for the update rule see equation (3.10)). Quantitatively, the behavior of the analytical solution (cf. figure 3.11) matches the one at hand. For increasing k and d, the system starts to oscillate.

3.5.4. Setup based on HICANN Loopback

Based on the pure software implementation it is now possible to use debug functionality of the HICANN chip to implement the detector neurons on neuromorphic hardware. To be precise, we do not use any neuronal functionality but rather forward the input spikes back to the sender. This intermediate step was done to solve software, firmware and hardware problems independently.

This so-called loopback mode is typically used for testing communication performance and reliability. The HICANN chip supports eight links with configurable directions for off-wafer spike input and output (see HBP SP9 Specification [2014]

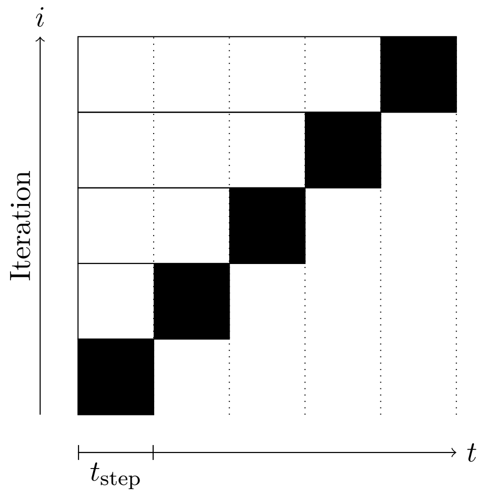


Figure 3.13.: Non-real-time closed-loop simulation setup using a step-like execution scheme. Simulation time is plotted on the horizontal axis. Every simulation iteration runs for the time interval t_{step}, i.e. the i-th iteration (counted from 1) runs until $t_{\text{end}} = i \cdot t_{\text{step}}$. After every time interval t_{step} the spike output of this last time interval is processed by the environment simulation to update its state and thereby new input for the neuronal simulation is generated. This spike data is provided to the neuronal simulation for the next time interval t_{step}. The environment simulator runs only at the *dotted* times, i.e. not taking up any simulation time t. For the next simulation time interval $i + 1$, the neuronal simulation is restarted, losing the complete simulation state. Due to the reproducibility of the neuronal simulation, we can prepend the input spikes of the i previous runs and re-acquire a matching simulation state at the end of the i-th time interval. The simulation can now progress until $(i + 1) \cdot t_{\text{step}}$ and, in the last time interval t_{step}, produce new spike output for the environment simulation. The time intervals that generate new data are marked as black boxes.

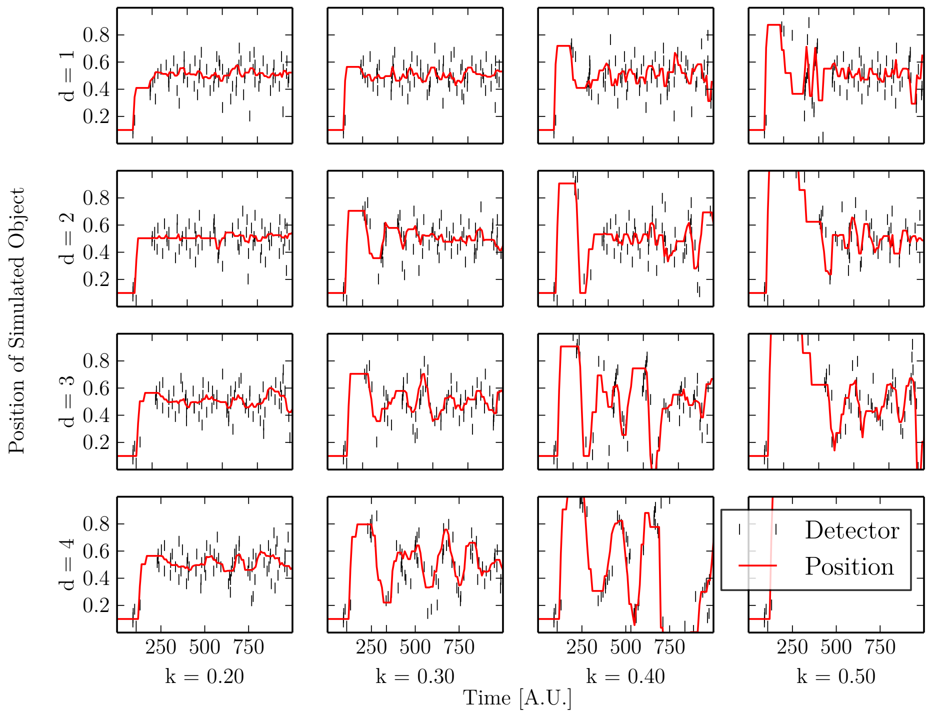


Figure 3.14.: Closed-loop experiment using NEST and a non-real-time setup according to figure 3.13. The object location between $[0, 1)$ is evenly distributed to 32 neurons using $\sigma = 0.1$ (ν_{max} is normalized to 1 spike per update cycle). The neuron model is `IF_cond_exp` using default parameters (cf. listing 2). Spike input weights are set to 0.1 µS – roughly tuned to release one spike per pre-synaptic spike if the inter-spike distance is large compared to the synaptic and membrane time constants ($\tau_{\text{synE}} = 5\,\text{ms}$ and $\tau_m = 20\,\text{ms}$, respectively). The time step of the simulation is $t_{\text{step}} = 10\,\text{ms}$. A total of 100 iterations are executed, i.e. 1 s total simulation time.

for hardware details). We configure links 0, 2, 4, 6 for spike input and 1, 3, 5, 7 for spike output. For the loopback mode we now connect link 0 to 1, 2 to 3 and so on.

Compared to the NEST-based setup (cf. section 3.5.3) the experiment sequence stays the same. The call to the NEST simulator is replaced by a call to the API that executes single experiments on the wafer system (cf. sections 2.2.1 and 2.2.2). This means that there is no real-time interaction between the neuromorphic hardware and the simulated environment. We take advantage of the execution scheme presented in figure 3.13.

A plot of this operation mode is not shown here because of the similarity of results to the NEST-based setup shown in the previous section (section 3.5.3). As stated above, this operation mode was primarily used to debug the real-time closed-loop experiment presented in section 3.5.6. However, parts of this setup are still useful for the real implementation; in particular, the FCP configuration for this setup can remain the same for the closed-loop and later hardware-neuron-based versions.

3.5.5. Real-time Software-based Setup

As a next step, the neuronal simulator is converted to a simulator running in real time. In the previous software-based setup, we used NEST for simulation of the neuronal network. However, the NEST simulator is not capable of operating in real time. On the other hand, at least one other common neuronal network simulator – NEURON – has real-time capabilities [Destexhe and Bal, 2009]. Nevertheless, quick tests indicated simulation speeds that are not high enough to be a valid replacement for accelerated neuromorphic hardware.

Hence, we re-implemented neuron simulator that models leaky integrate-and-fire model neurons and current-based synapses – the design goal was simulation speed. This work has been accomplished in collaboration with Paul Müller. The LIF model is given by:

$$C_\mathrm{m}\frac{dV(t)}{dt} = -g_L(V(t) - E_l) + I(t) \tag{3.13}$$

with synaptic input current $I(t)$, the leakage potential E_l, the conductance towards the leakage potential g_L, and membrane capacity C_m [Burkitt, 2006]. In case of the leaky integrate-and-fire model with current-based synapses, the synaptic current $I(t)$ adheres:

$$I(t) = \begin{cases} 0 & t < t_1 \\ I(t_{i-1}) + w & t = t_i \\ I(t_i) \cdot e^{-\tau_\mathrm{syn}(t-t_i)} & t_i < t < t_{i+1} \end{cases} \tag{3.14}$$

with the first input spike at $t = t_1$, other input spikes at t_i, the weight w, and the synaptic time constant τ_syn. This describes an exponentially shaped current input decaying with τ_syn and increasing by the synaptic weight w when a pre-synaptic spike arrives.

Now, to minimize the number of floating point operations per time step (cf. Izhikevich [2004]) we substitute:

$$\begin{aligned} W &= V - E_\mathrm{l} \\ J &= \frac{\Delta t}{\tau g_l} I \\ w_j &= \frac{\Delta t}{\tau g_l} w \end{aligned} \tag{3.15}$$

For the update step $i \rightarrow i+1$ progressing time by Δt, a forward Euler integration scheme is used. This scheme provides a fast integration and sufficient precision (see figure 3.15). The time update is implemented as follows:

$$\begin{aligned} W_{i+1} &= (1 - \frac{\Delta t}{\tau}) W_i + J_i \\ J_{i+1} &= (1 - \frac{\Delta t}{\tau_\mathrm{syn}}) J_i + \sum_{k=0}^{N_\mathrm{spikes}} \delta_{k, \lfloor t_k / \Delta t \rfloor} w_j \end{aligned} \tag{3.16}$$

with the Kronecker symbol δ and pre-synaptic spike times t_k with synaptic weights w. As in the case of the conventional cLIF model, the membrane voltage is clamped to a reset voltage for the refractory period after reaching the threshold voltage. All these variables are rescaled according to equation (3.15).

The implementation at hand requires two comparisons, two multiplications and one addition per neuron and time step; if the neuron is in the refractory period, one multiplications, one addition and one comparison are skipped and one integer subtraction, the refractory period update, is performed. An assembler dump is provided in listing 2.

To verify implementation correctness, figure 3.15 compares the behavior for varying time-step resolutions and the NEST reference implementation.

Based on this neuron implementation we can implement a real-time closed-loop experiment that interacts between the simulated environment running on one compute node and the custom cLIF implementation running on another compute node. Figure 3.16 presents a sweep of varying time steps t_step and delays d. In the left column the sweep over multiple delays d is shown (the number of steps the simulated environment looks into the past, cf. equation (3.10)). For growing delays the object position starts to oscillate. The right column displays varying update time steps t_step, i.e. the update frequency of the simulated environment and of the neuron simulator. When reducing t_step the relative impact of the constant communication latency t_comm (cf. figure 3.3) increases. Effectively, this introduces (non-integer) delays that can be compared to the delay d mentioned before. Quantitativly matching behavior is plotted side by side.

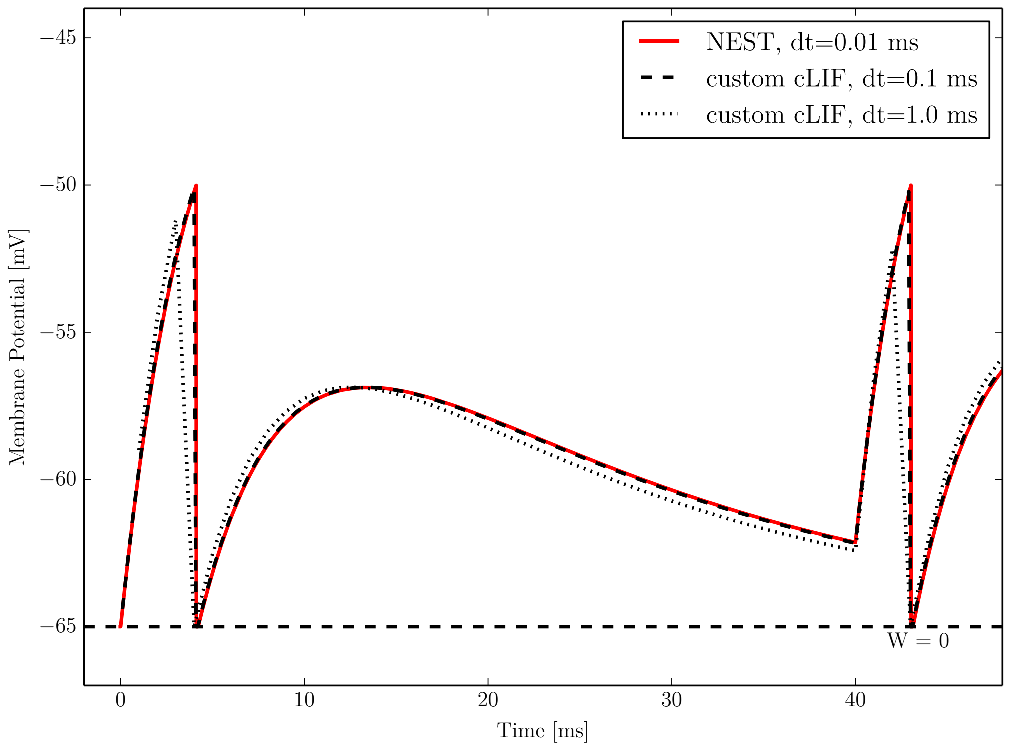


Figure 3.15.: Time course of simulated neuron membrane traces. The cLIF implementation is shown for time steps $dt = 0.1\,\text{ms}$ and $1.0\,\text{ms}$, the reference is based on the NEST cLIF implementation using a very small of timestep (0.01 ms). Neuron parameters are set to PyNN default parameters (cf. listing 2), synaptic weights are 6 nA, which is strong. For the PyNN default time step of 0.1 ms the custom cLIF implementation matches very closely to the reference. However, due to performance considerations the experiment uses the coarse time step.

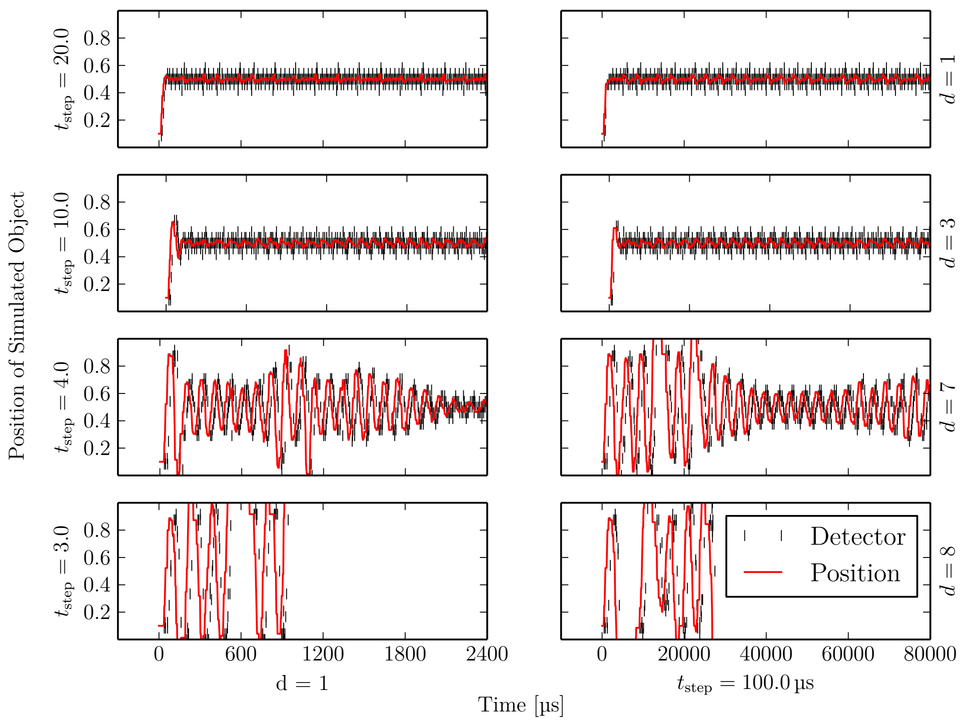


Figure 3.16.: Software-based closed-loop experiment. The *detector* neurons are implemented in software and run on a second cluster node. To be able to simulate a high speed-up factor, the implementation was optimized and uses simple cLIF dynamics (cf. section 3.5.5). The time-axis was zoomed in to provide similar range of *simulated* time compared to figure 3.14 – the complete data set can be found in figure A.7. In contrast to the previous plots, the parameters d and t_{step} are varied in the left and right columns. On the left, the software *delay* parameter d is fixed and the update (or execution) time step t_{step} is increased to enlarge the relative impact of communication latency – the detector response is delayed. The behavior is similar to the right column, where t_{step} is large and constant, i.e. the relative communication latency is negligible, and the software *delay* parameter d (the number of steps the update loop looks into the past) is increased. The plots were selected to show typical behavior – i.e. the effective delays do not necessarily match between left and right plots. In particular, the *force* parameter was lowered to allow for sensible behavior in a large d-range. A complete list of parameters can be found in table A.4.

Update of Simulated Environment	1150.0± 10.0	ns
without communication	667.0± 5.0	ns
Update of CLIF	370.0± 10.0	ns
without communication	57.7± 1.6	ns
`rand()`	12.3±<0.1	ns
`xorshf96()`	3.6±<0.1	ns
25× `sqrt()`	207.5±<0.1	ns
25× `exp()`	732.5±<0.1	ns
25× `gaussian()`	865.0±<0.1	ns
25× `lorentz()`	91.0±<0.1	ns

Table 3.1.: Run times for the different code paths. The total runtimes in the upper half are averages of individual measurements during operation. In the lower half micro benchmark results are listed. This means that the mentioned functions are repeatedly called and only the total runtime of the benchmark is recorded – this might alienate the real impact compared to a function call that is embedded within other code. The cLIF runtimes are only applicable to the purely software-based real-time setup presented in section 3.5.5. All numbers have been acquired on a HMF compute node using 25 neurons. Compiler options are stated in appendix A.3.

Critical Timings The evaluation of the code path in Valgrind-based Profiling Tool (Callgrind) shows last-level cache misses, i.e. forced access to physical memory, in the `receive()` and the `queue()` functions to be the major issue. However, the runtime overhead for the synchronous `send()` can be hidden by using a dedicated sender thread. Additionally, some micro benchmarks were performed to estimate the influence of different code parts. In particular, the cost of spike transmission and reception is of general interest as it will roughly stay the same when using the presented software framework. Figure 3.17 shows the cost of spike transmission and reception for the HMF system. Details about the micro benchmarks can be found in appendix A.3.

3.5.6. Real-time Setup based on HICANN Loopback

Starting from the setup described in section 3.5.4 we adapt the setup for real-time operation. The only required modification is using another interface to the FCP that forwards all data directly to the HICANN chip(s) instead of repeatedly loading a growing part of the experiment and afterwards reading back the result data. This means that spike playback and recording facilities of the FCP chip are not used anymore and the execution scheme changes from the step-like scheme shown in

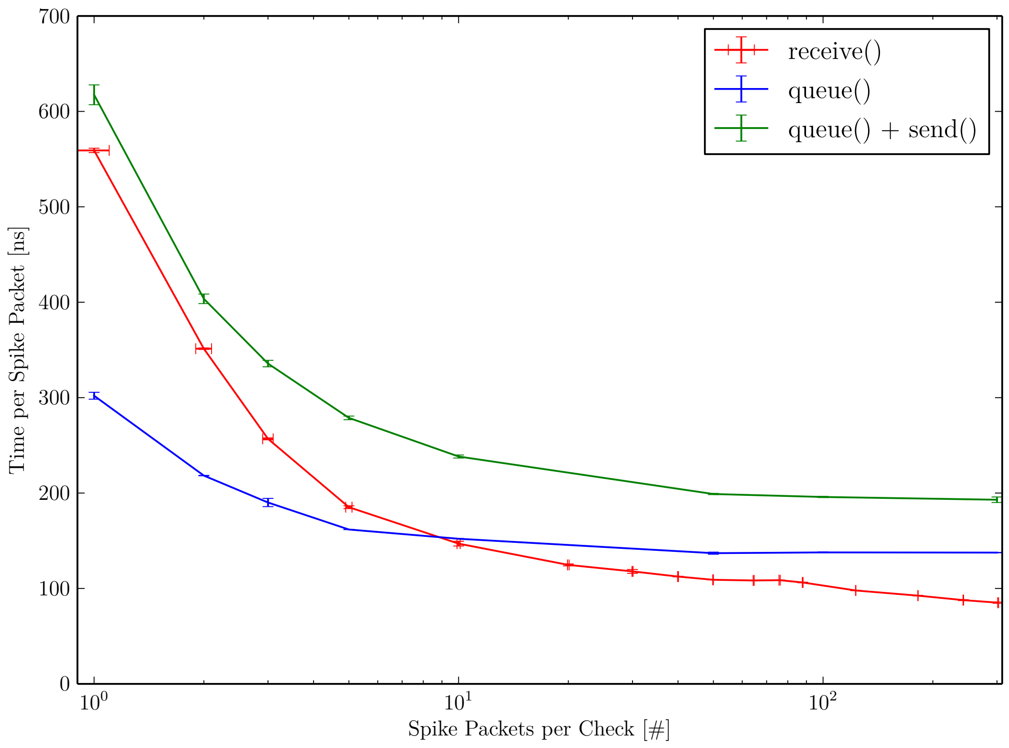


Figure 3.17.: Measured processing time per spike packet for varying batch sizes. The measurement was performed on two HMF compute nodes, one sender and one receiver. The Callgrind-based evaluation showed high probability for last-level cache misses in the single packet case. For queuing it seems reasonable that we experience a first-access overhead because that memory region was not used before (and it takes a long time to reach the same buffer entry the second time; even in that case, the cache entry would be invalid, because of DMA accesses by the NIC). On the receiving side, the DMA operation transferring data from the NIC to the kernel data structures also invalidates the cache. Obviously, handling multiple packets at once improves throughput because the cache (and prefetching on receiving side) can be used for entries following after the first one.

figure 3.13 to a time-continuous experiment.

The real-time interface of the FCP is described in section 3.3.2. Additionally, using this operation mode of the HICANN chip yields the lowest latency possible for communication between the host computer and the wafer system. In particular, there are no contributions by the synaptic and neuron model time constants.

Figure 3.18 presents the measurement performed on the HMF using the HICANN chip-based loop-back mode as *detector*. For this particular setup a single HICANN chip (#276) on wafer 0, i.e. the first wafer setup, was used. Starting from a relaxed time step of 100 µs the update cycle time is reduced down to 5 µs. Real time is plotted on the x-axis. This, together with the slow update time step, yields a large spike distance for the first row experiment using 100 µs. The spike density increases towards the bottom because of the decreasing update time step – the simulated environment is called more frequently and thus produces more spikes. At approximately 10 µs the simulated object starts to overshoot because of the increasing communication delay. More precisely, the relative portion of communication latency out of the update cycle time t_{step} increases. Fully oscillating behavior is shown in the last row at 5 µs update time step.

3.5.7. Setup using the HMF

Finally, the actual analog neuron circuits are directly used as detectors. This final step, i.e. migrating from the HICANN chip-based loop-back mode to using the neuron circuits, requires the configuration of the complete HICANN chip: L1 buses, synapse drivers, synapses and neurons have to be configured. From the hardware perspective the configuration is split into multiple stages (cf. sections 2.2.1 and 2.2.1.2). The configuration order represents dependencies between hardware entities. For example, after setting the clock frequency, the digital parts of the HICANN chip have to be resetted to start from a defined state.

Jeltsch [2014] presents algorithms to automatically transform a neuronal network description into a hardware configuration – for non-real-time experiments this *mapping* flow is already in place. The neuronal network can be described using the PyNN API, the mapping processes this data and create a corresponding hardware configuration (see section 2.2.3 for details).

However, at the time of writing it is not yet possible to describe real-time experiments using the PyNN-based flow. In particular, the description of interactions between neuronal networks described in PyNN and virtual environments is not yet possible. The next PyNN API version (0.8) will provide basic support for MUSIC which will at least allow for inter-neuronal-network communication. The author collaborates with the PyNN and MUSIC developers to enable a comprehensive solution. Regarding the mapping flow, it is not yet possible to acquire *reverse* mapping information, i.e. the assignment of hardware neurons and synapses to the corresponding objects in the PyNN layer. Hence, a manual configuration is used and the key elements are described in the following.

Compared to the previous loop-back mode experiments described in section 3.5.6,

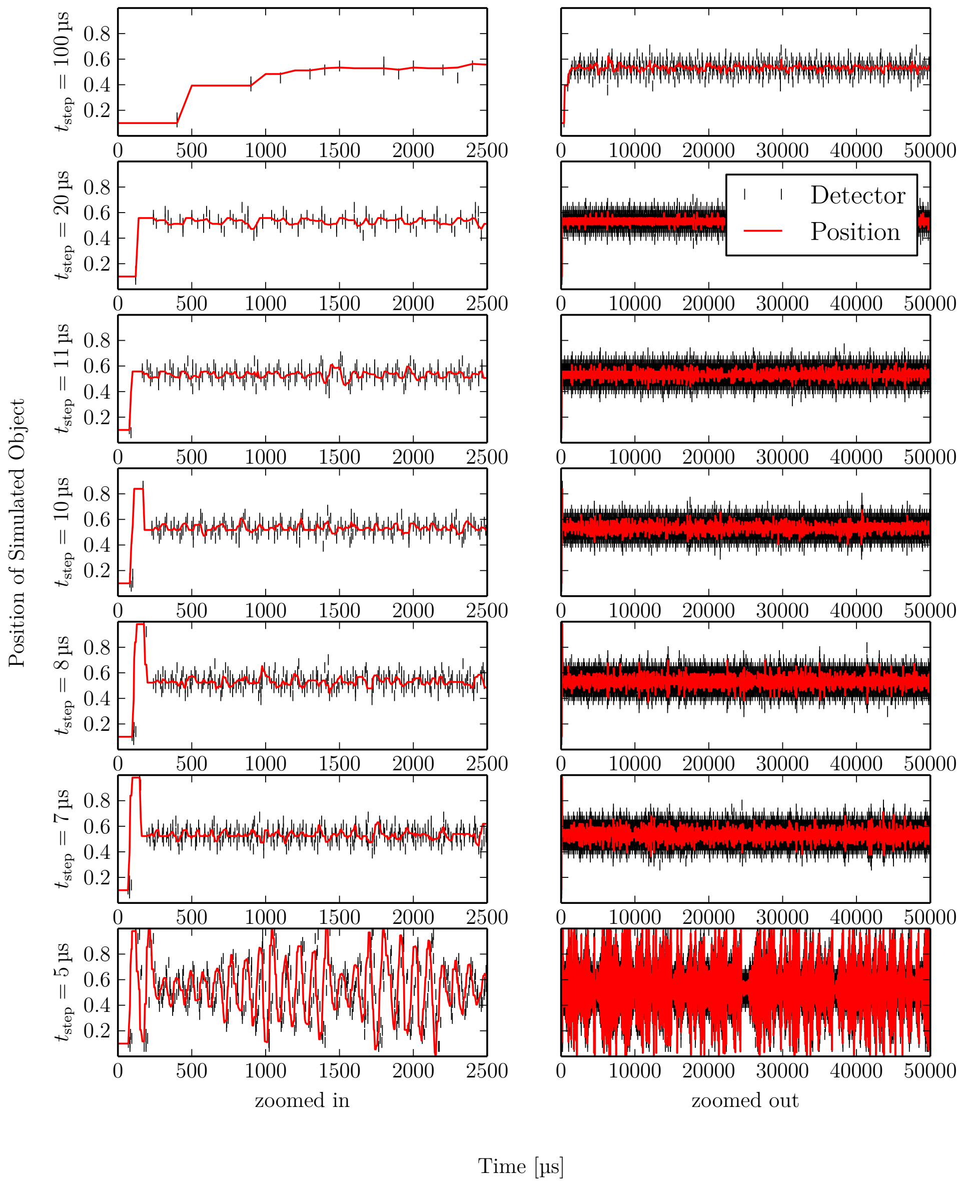


Figure 3.18.: Hybrid closed-loop experiment runs using HICANN loop-back mode. The update time step t_{step} is varied from top to bottom. Simulated object position on the y-axis, wall-clock time on the x-axis. For decreasing t_{step} the impact of the fixed communication latency increases, i.e. the detector response is delayed. Compared to the latency obtained in section 3.3.4 ($(12.1 \pm 0.2)\,\mu\text{s}$), the third row ($t_{\text{step}} = 11\,\mu\text{s}$) approximately corresponds to $d = 1$. The last row is similar to the $d = 2$ case. A complete list of parameters can be found in table A.5.

the FPGA setup stays the same. Deviating from the previous HICANN configuration, we disable the loop-back mode and configure the *neuromorphic* parts of the chip. Four digital network chip (DNC) channels are configured as input and the other 4 channels are used as output – as in the loop-back setup. The loop-back mode is disabled and the input spikes are routed to four synapse rows instead. The other four DNC channels are used to extract spikes from 4 output buffers[16] where $4 \cdot 32$ neuron circuits, the so-called DenMems, are connected to. This means that only the top-half of one HICANN chips is used for this experiment. The synapses are configured to listen to unique L1 address LSBs[17]. Additionally, addresses 0 and 15 are not used for input traffic. The former is used for *background* spikes[18], the latter is used to *disable* synapses in this setup. Because of these limitations, only 14 out of 16 addresses can be received per synapse in this setup. The four synapse rows are independent and we can finally address up to $4 \cdot 14 = 56$ individual synapses. Thus, by using one synapse per neuron we can address the same number of neurons, i.e. 56, using this setup. Figure 3.19 sketches the hardware units that are involved in this configuration.

To obtain homogeneous neuron and synapse behavior, individual calibration data has been provided by the calibration team [Kleider et al., 2014]. In addition to the calibration, a blacklisting of misbehaving neurons is performed. The quality criteria is the *gain* function – the neuron spike response to pre-synaptic spikes. In the present calibration data set, the synapses are already tuned towards maximum efficacy which allows for a similar setup as in the previous section. Under biologically more realistic constraints, the synaptic weights are weaker and multiple concurrent input spikes are needed to trigger a post-synaptic spike. To minimize the number of blacklisted neurons, a single parameter is tuned: the neuron membrane voltage threshold v_{thresh}. The parameter is optimized to maximize the number of neurons that fire with the input firing rate multiplied by a certain factor (see below). This tuning is performed on the calibration input parameters; we take advantage of the existing calibration data to achieve a homogeneous neuron behavior.

The spike train for *gain* testing is constructed as follows:

Figure 3.20 plots the corresponding spike train. To evaluate robustness of neuron response, this spike train is repeated 10 times and multiple chip re-configurations are performed. The latter is needed because of trial-to-trial variation in analog parameters [Kleider et al., 2014; Kononov, 2011; Millner, 2012] that can, for example, change the voltage difference between resting and threshold potential.

We define the *gain* as:

$$G := \sum_i \frac{N_{\text{out}_i}}{N_{\text{in}_i}} \tag{3.17}$$

[16]Neuron circuits, so-called dendrite membrane circuits (DenMems), are grouped into output buffers which handle the output of 64 neurons, 32 of the top half and 32 of the bottom half.

[17]The upper two bits are handled by the synapse driver; i.e. the synapse listening address comprises four bits.

[18]The L1 buses need a minimum of activity to operate (for details see Hock [2009, 2.1.2 and 2.4.1]). This activity is generated on-chip, uses address 0 and is typically applied to all buses.

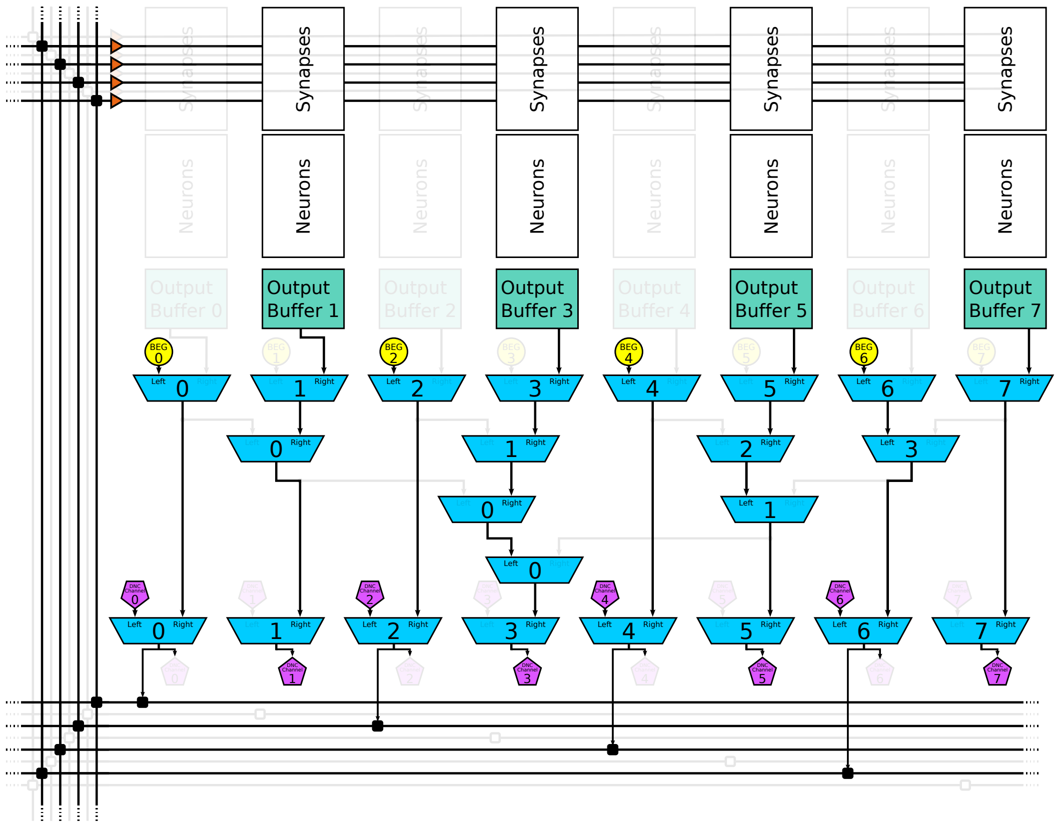


Figure 3.19.: Schematic showing the hardware configuration for the closed-loop experiment setup. The external input is injected at the upper row of DNC channels (pentagon-shaped, purple), whereas the output is extracted from the bottom row. Four L1 (horizontal and vertical black lines) buses are used to transport spike data to four synapse drivers (triangles) in the upper half of the chip (the lower half is not used in this setup). The same buses are also fed by the BEGs to ensure proper *locking* of the asynchronous bus connections (for details see Hock [2009, 2.1.2 and 2.4.1]). To simplify the schematic, the remaining L1 configuration is not shown. The synapse drivers forward the signal to the synapse rows where every neuron uses one synapse per row to listen for matching events. Section 3.5.7 explains the neuron and synapse configuration details. For details of the hardware topology see section 1.1.1. The drawing is based on Alexander Kononov's `full_hicann.svg` drawing.

```
Data: intervals = 1000 µs, 500 µs, 100 µs, 50 µs, 10 µs
Data: counts = 2, 5, 20, 50, 100
Data: lasttime = 100 µs
Data: waittime = 3 ms
Data: spike_train
for count, interval in zip(counts, intervals) do
    for i in range(count) do
        spike_train.append(lasttime + i * interval)
    end
    lasttime += waittime
end
```

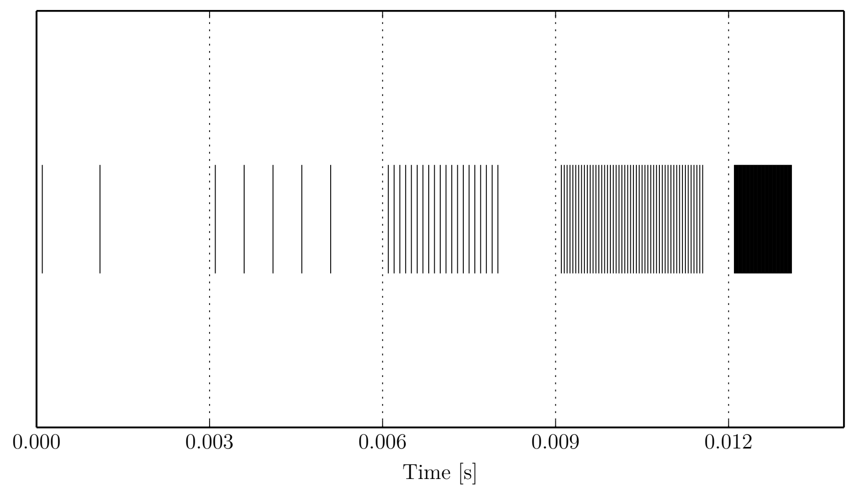


Figure 3.20.: Visualization of the input spike train used for *gain* evaluation (cf. equation (3.17)). The spikes are spaced regularly, the hardware inter-spike interval is reduced from 1 ms down to 10 µs in multiple steps.

with i being the *waittime* interval as defined in the algorithm above and $N_{\text{in,out}}$ being the number of input/output spikes in this time interval. Finally, we blacklist all neurons that do not adhere:

$$MIN = 0.5 < G < MAX = 4.0 \tag{3.18}$$

This blacklisting takes place at the end of all re-configuration trials – the intersection of all *stable* neurons is computed. The remaining set of neurons is sorted by $1/abs(1.0 - G)$ and minimum variance.

One test trial is shown in figure 3.21.

Using this configuration, 128 neuron circuits out of 512 can be stimulated and read out. For the set of calibration data, a maximum of approximately 30 neurons meet the stated criteria. Results from sweeping v_{thresh} and applying the filtering from above can be seen in Figure 3.22.

The robustness of the setup is unsatisfactory, as after a small number of tests (approximately 10) every neuron fails to satisfy equation (3.18) eventually. Due to the robust behavior of the previous experiment setup (section 3.5.6), this robustness issue seems to be related to the neuromorphic parts that are now being used. One known source of re-configuration variability is the limited precision of the analog parameter storage, the floating gate cells [Kononov, 2011; Millner, 2012]. Increasing robustness of calibration and analog behavior in general is a current goal of the calibration team [Kleider et al., 2014]. The same holds for the communication chain and software behavior; the author reported on efforts to achieve adequate reliability in these areas in section 2.1.

Figure 3.23 presents the measurement on the HMF using hardware neurons as *detector*. The same chip was used as in the previous setup. In comparison to the previous loop-back-based measurement, the resolution of object positions is reduced. The current calibration data and the particular chip configuration limits the number of usable neurons to approximately 16 (see figure 3.22 for details) when applying the criteria stated in equation (3.18). However, in this particular measurement the number is reduced even further to obtain even more robust response behavior. The result plots are very similar to the loop-back case, especially the phase transition to the overshooting behavior when moving from 11 µs to 10 µs. The complete oscillation for the 5 µs-case was not stable. From trial to trial the object marker either *left* the mapped interval $[0, 1]$ or leveled out as shown in the present plot.

3.6. Summary

This chapter focused on the real-time closed-loop operation of the HMF system. After defining the task at hand in sections 3.1 and 3.2, the hardware environment was introduced. Sections 3.3.4 and 3.3.5 presented latency measurements of the BSS and of the upcoming HBP system. An evaluation of the communication latency between compute node and HICANN chip was shown in section 3.3.6. The

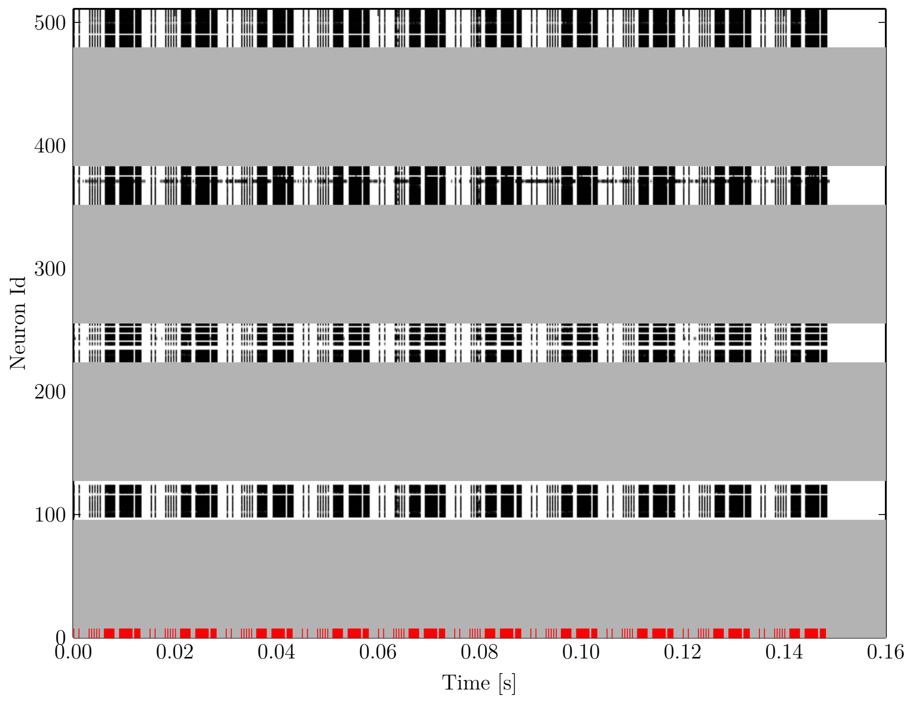


Figure 3.21.: Raster plot that shows neuron response to external stimulus (see figure 3.20). Most neurons respond to a single input spike (plotted in red at the bottom) with one or multiple output spikes. Some neurons do not respond, e.g. the large gap from neuron 234 to neuron 238. After chip reconfiguration the set of responding neurons changes, i.e. the set of robustly working neurons is smaller (see figure 3.22). The gray region marks unused chip parts. The LIF calibration data set (details in table A.6) is used and v_{thresh} is tuned to maximize the number of *working* neurons in this setup.

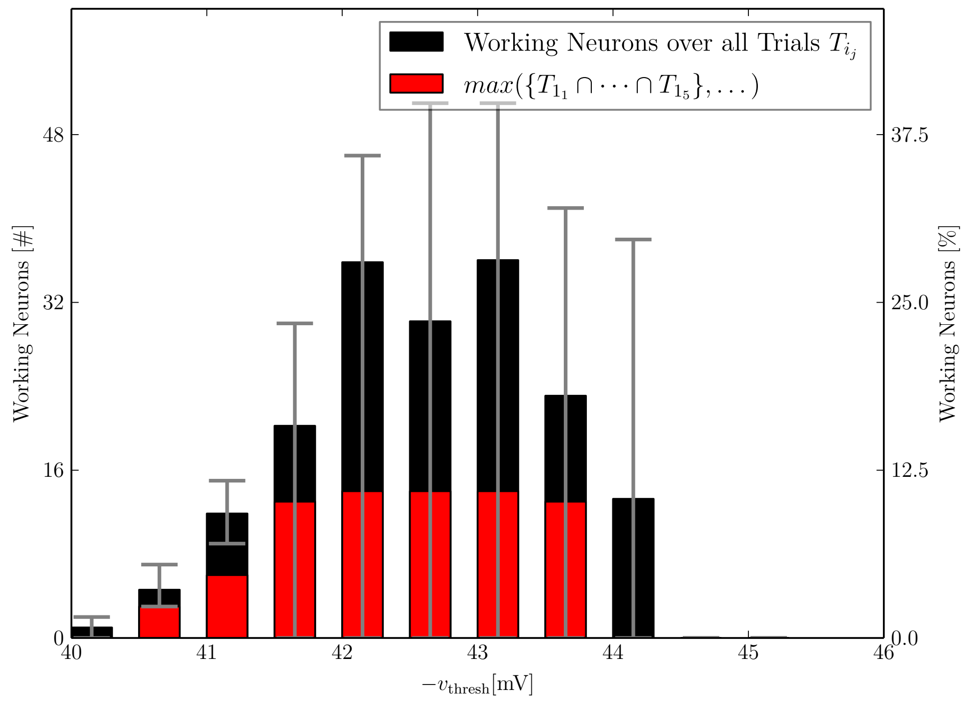


Figure 3.22.: Sweep over neuron threshold voltage (v_{thresh}). The y-axis represents the number of *working* neurons according to the criteria stated in equation (3.18). The black bars show the average number per trial and the error denotes the min/max number within five runs i consisting of five re-configuration trials j each – i.e. the average and min/max over 25 re-configuration trials T_{i_j}. At the end of each run the set of neurons that worked in all re-configuration trials is calculated, i.e. the set of neurons which fulfilled the criteria in all five re-configuration trials. The red bars mark the largest set of working neurons in one of these five runs (j).

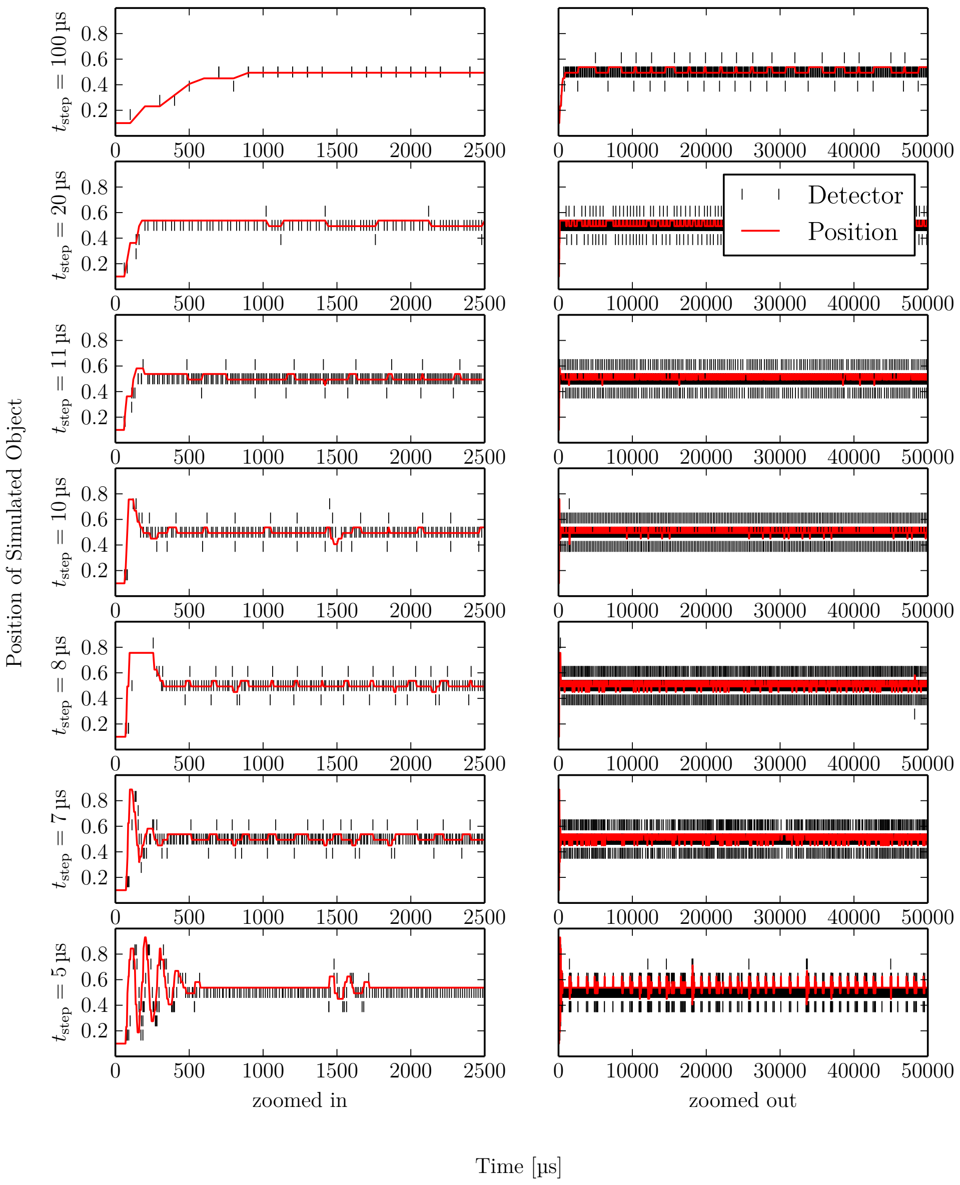


Figure 3.23.: Hybrid closed-loop experiment using a conventional compute node to simulate the virtual environment and nine hardware neurons on a HICANN chip located on the first prototype wafer. The update time step t_{step} is varied from top to bottom. The simulated object position is plotted on the y-axis, the wall-clock time on the x-axis. The parameters match the setup shown in figure 3.18 except for the number of detectors. To obtain robust behavior only the 9 most robust neurons acquired in figure 3.22 were used. For decreasing t_{step} the impact of the fixed communication latency increases, i.e. the detector response is delayed. A complete list of parameters can be found in table A.6.

communication software stack was presented in section 3.4.

Subsequently, an experiment was formulated concentrating on a demonstration of real-time interaction between conventional compute hardware and the neuromorphic system. It was deliberately designed to react to varying communication latencies: a simulated environment interacts with a *detector* over a communication network in real time (cf. sections 3.5.1 and 3.5.2). The influence of communication latency on experiment behavior was explored in a conventional simulation (cf. section 3.5.3). Sections 3.5.4 to 3.5.6 presented intermediate steps towards the implementation on hardware: 1. a pure round-based software implementation that shows basic experiment characteristics and allows for parameter tuning in a fully deterministic environment; 2. a version based on an execution scheme that allows for round-based execution on the HMF; 3. a real-time version of the pure software implementation was implemented when the FCP firmware was not yet real-time capable; 4. a real-time version interacting with the HMF in loopback mode.

Finally, section 3.5.7 shows the complete experiment. The simulated environment runs on a compute node and communicates over the data network with the neuronal network which is implemented on the wafer module. However, there's still a lot to be done: on the one hand, the robustness of the neuronal network setup has to be improved to allow for larger neuron counts. Furthermore, the combination of limited analog neuron parameter precision and the current calibration data which is optimized towards maximal synaptic weight leads to a high trial-to-trial variation. On the other hand, both the compute cluster as well as the real-time software interface can compete with state-of-the-art communication mechanisms using vendor-optimized MPI software stacks. Regarding the FPGA real-time interface, further improvement is needed to decrease the latency jitter caused by suboptimal FPGA firmware behavior. Nevertheless, this is the first real-time closed-loop experiment ever that has been performed on accelerated neuromorphic hardware.

On the software side, future efforts will focus on a user-friendly integration of the MUSIC framework as a higher-level spike interface. Similar to PyNN, this will simplify switching between purely software-based closed-loop experiments and hybrid operation on the neuromorphic systems.

Discussion

This thesis concentrates on the exploitation of the unique speed provided by the HMF and NM-PM1 systems, i.e. the speedup factor. It can thereby open up the possibility for novel applications in computational neuroscience. In conventional neuronal network software simulations, the batch mode is prevalent. All external experiment stimulus is predetermined prior to the start of the simulation. Executing such experiments on an accelerated neuromorphic system provides much higher experiment *throughput*. Consequently, the analysis of result data can be performed earlier which speeds up the test-refine cycle. When thinking out the idea of this iterative workflow, a fast non-pipelined operation mode would allow for *interactive* experimenting with the system. At high update rates, the user can perform intuition-guided exploration of parameter spaces; a technique which often leads more quickly to success than sweeping over a large set of parameters. In contrast to these standalone-type experiments, another setup involves the modification of the external stimulus which is applied to the neuronal network. Due to the involvement of both, conventional and neuromorphic hardware systems, this mode is called *hybrid* operation. Closed-loop experiments that imply a motor-feedback control loop, are a typical example for hybrid experiments. In general, such experiments make demands on the maximum reaction delay of the conventional compute hardware. This delay can be split into two parts, communicational and computational latencies. The former is a model-independent property of the communication infrastructure.

To achieve these goals, all parts of the hierarchy of software and conventional hardware were evaluated and optimized. The compute cluster nodes as well as the network architecture were designed to achieve high data throughput while providing the option to operate at low latencies; both features are key requirements of the prospected operation modes. In particular, the stimulus and output data rates of the NM-PM1 are typically three to four orders of magnitude higher than typical large-scale software simulations. For example, the NEST-based simulation mentioned in the introduction produced 1 MEvent/s (in real or wall-clock time) of (neuron) output spikes on average. Assuming a relaxed 64 bit spike encoding, the bandwidth of the NM-PM1 host interface supports from 125 MEvent/s to 500 MEvent/s per wafer. Indeed, this data network is only used for host communication and another network provides connectivity between the FCP FPGAs. This is achieved by one dedicated data network switch per wafer module which aggregates the 48 individual 1GbE-based links into up to four 10GbE links. A backbone switch provides inter-connectivity between all the individual wafer switches and the compute cluster. The current setup provides a backbone consisting of logically

64×10GbE links. For the NM-PM1 40 links are assigned to the 20 wafer modules and 20 compute nodes, i.e. a single 10GbE link each; the remaining links are used for servers and as uplink to the institute; approximately 10 links can be used to boost links to some wafer modules or to increase the uplink bandwidth. Assuming the minimum of 20×10GbE links to the wafer modules[19], a total spike rate of up to 2.5 GEvent/s can be produced or consumed.

Consequently, to exploit the fast communication infrastructure a reliable high-performance transport-layer protocol was implemented for the communication between FCP FPGAs and compute nodes. The protocol correctness and performance were tested to verify adequate behavior. A throughput benchmark using all currently installed FCP FPGAs achieved 94% wire speed, i.e. (846.7 ± 1.2) MiB/s payload throughput between one compute node and eight 1GbE communication endpoints. To extrapolate to the full system which will contain enough communication endpoints to fill all four upstream 10GbE links, the throughput was benchmarked between two compute nodes using 10GbE as well as 40GbE. For both setups it was possible to achieve over 99% wire speed ((1.070 ± 0.009) GiB/s and (4.534 ± 0.062) GiB/s, respectively). In case of the 10GbE link test, the software was unmodified with respect to the version used for communication with FCP FPGAs. However, the 40GbE-based setup required an enlarged packet size to reduce the packet rate, and thereby interrupt rate, at the host computer. The same modification is under development for the FPGA protocol implementation.

To convey the speed of the NM-PM1 to the user of the system, a high-performance software stack is just as important as the communication structures mentioned before. Accordingly, a pipelined software architecture was designed and the individual components were collaboratively implemented. The individual pipeline stages provide serialization layers to suspend and resume execution. This allows for dispatching of computationally expensive tasks to dedicated HPC machines whereas the experiment execution stage resumes operation on the NM-PM1 compute cluster. Adequate performance reaching beyond 10GbE for the experiment input stage as well as the remote procedure call (RPC) mechanism in general was shown.

For large-scale neuromorphic systems like the HMF and NM-PM1 systems, resource management is essential. The NM-PM1 will contain approximately 1000 remotely accessible components, mostly FCP FPGAs and devices for analog readout, which have to be scheduled to individual experiment *jobs*. Fairness between users is another requirement which comes into play when the user count increases. A common tool for resource management in HPC systems, simple Linux utility for resource management (SLURM), was examined in terms of performance and characteristic features. For the HMF/NM-PM1 system the performance is sufficient and it has been integrated into the work flow for the first two prototype systems which are currently available.

Regarding the hybrid operation, a communication software framework was developed to support low-latency communication. For directly linked compute nodes,

[19]This assumes a single 10GbE link per wafer module.

the one-way latency was measured to be (2.4 ± 0.2) µs. Compared to the standard UNIX API, i.e. the `socket()` interface, a 36-fold decrease in latency was achieved. Measurements including the backbone switch, aggregation/wafer switch or both switches were performed and yielded (3.0 ± 0.2) µs, (3.4 ± 0.2) µs and (4.1 ± 0.2) µs, respectively. The latency down to the HICANN chip was found to be (8.5 ± 0.4) µs. A comparison to measurements performed between FCP FPGA and compute node discovered deficiencies in the current FPGA firmware implementation. In particular, the timing jitter, or spread of the individually measured latencies, and the increased average latency were not expected as the FPGA is a inherently real-time capable device. The issue is under investigation by the FPGA developers as well as by the author.

The software framework developed for real-time communication has been provided by the author to MUSIC API developers. The MUSIC API provides a common interface for describing inter-simulator communication in the same way as the PyNN API provides a common interface for neuronal network descriptions. A first draft version of a real-time-capable MUSIC was created as presented at a HBP meeting mid-2014. By adding support for real-time communication into MUSIC, the original target audience, i.e. software simulators, can be extended towards robotics and other systems requiring real-time communication. Converging to a single API at this point provides portability of closed-loop experiments between different execution platforms which would otherwise require a rewrite of the interfaces handling external communication. This applies in particular to the transition from round-based software simulations to real-time setups.

To demonstrate the capabilities of the NM-PM1 system, a hybrid, closed-loop experiment was designed and successfully evaluated. The experiment is based on the interaction of a simulated environment and a layer of neurons acting as simple *sensors*. An object or marker can move on an one-dimensional axis within a force field which pulls towards the center. The object position is translated into a spike train which is transmitted to the neurons located on the wafer module. The neurons act as a forwarding stage relaying a similar spike train back to the simulated environment. In turn, this spike train is translated into an apparent object position and a position update of the real position is performed based on the force exhibited at the apparent location, i.e. the sensory-motor loop is closed. This simple model is already capable of interesting behavior depending on the communication and computation latencies. The path from a purely software-based experiment to an implementation using real neurons on the wafer module was successfully followed and key components were analyzed. This hybrid, closed-loop setup is the first experiment ever which was performed on an accelerated neuromorphic system. It will act as a template for future hybrid experiments.

Scope The input and output data rates of a single NM-PM1 wafer module are already several orders of magnitude faster than in any other system in the neuromorphic community. Neuromorphic systems based on photonics could consume and produce even higher data rates due to the larger speedup factor of 10^8 [Kravtsov et al., 2011]. However, in science typical data-intense tasks are performed in high-energy physics. In particular, the lowest software-based trigger processes and filters the highest inbound data rates. In the case of the LHCb experiment[20], the typical inbound data rate from the detector, or last hardware-based trigger stage, to the first software-based trigger stage is in the order of Tibit/s [Alessio et al., 2014]. Subsequently, the data is dispatched to a set of 1500 compute nodes which process the data to partially reconstruct the individual events and filter the data thereby reducing the output rate to 250 MiB/s. In the perspective of computational neuroscience, the analysis step is very model-dependent and no universal comparison can be performed. However, assuming a similar level of complexity it is clear that the compute cluster cannot conduct an online, or non-stop, data analysis when the HMF/NM-PM1 operates uninterruptedly. In terms of high-energy physics, the data acquisition can only run during beam time and the read-out as well as trigger stages are optimized towards maximum recorded luminosity. On the other hand, the *beam time* of the NM-PM1 corresponds to the simulation or emulation time of neuronal network experiments. Hence, the main purpose of the compute cluster is handling the experiment flow, be it pipelined or real-time operation; computationally expensive tasks have to be performed on dedicated HPC hardware to maximize the usage of the neuromorphic system.

The standard transport protocol of the IPv4 suite, the transmission control protocol (TCP), is the default solution for reliable connections in the Internet as well as local-area networks. However, it is a complex protocol that covers areas which are not important for locally accessing FPGA-based systems. In particular, basic features such as segmentation of packets, selective acknowledgments, window scaling, timestamps and side-band data are unnecessary and would complicate an FPGA implementation. Additionally, existing implementations for FPGAs mostly rely on a software-based protocol handling running on a soft-core central processing unit (CPU) which is implemented on the FPGA. Hence, this thesis provides a custom alternative to TCP, the HostARQ protocol. This solution scales well up to state-of-the-art network technologies like 40GbE. In particular, techniques like ring-buffer-based communication between kernelspace and the userspace HostARQ protocol implementation provides a similarly low overhead as a pure kernel implementation. One drawback of a userspace implementation, the increase in communication latency, is not relevant for a reliable transport protocol which utilizes large buffer areas in all communication endpoints.

The process of building an FPGA configuration can be compared to the task the software stack handles during experiment preparation. Indeed, the input stage translates a higher-level description into a lower-level data format; the mapping and routing step produces a configuration of the hardware substrate which can in turn be loaded onto the FPGA device. This sequence is very similar to the workflow of the NM-PM1 software stack. In particular, the individual stages produce intermediate

[20]One of the four large experiments at the Large Hadron Collider at CERN, Geneva.

data structures, which can subsequently be processes by the next stage.

On the other hand, the neuronal network emulation phase resembles tasks of data-intense or low-latency applications. One example mentioned before are high-level/software-based triggers in high-energy physics which also include soft[21] real-time constraints. However, the individual events are dispatched to a server farm which increases the time budget for computations; for example, at LHCb the typical processing time is in the order of 10 ms per 60 KiB-sized event.

In terms of using the NM-PM1 system, the task is similar to resource management on general-purpose HPC machines on first sight. When going down to the details, there are some differences: in contrast to typical HPC jobs, the run times of individual experiments are very short due to the speedup factor. Another difference are inhomogeneities between neuromorphic hardware resources which require users to specify a list of individual hardware components needed for an experiment. Nevertheless, the common SLURM tool is capable of performing these tasks.

Outlook Within the HBP project, a massive effort is pursued towards an integration platform, called the unified portal (UP). It will act as a common interface for data access, data visualization, data analysis, simulation building and simulation execution. Both, the HPC as well as neuromorphic platforms, including the NM-PM1, will be integrated. For users this will allow for maximally simplified access to the NM-PM1 system as well as a toolchain for input and output data processing. Regarding the technical challenges, the most important aspect is global data access. The NM-PM1 system is capable of running many experiments in short time spans while consuming and generating data at very high rates. A deep integration of data access into the UP will provide the possibility to perform experiments on the NM-PM1 while the analysis can run on HPC hardware. Similarities to the field of high-energy physics are present. In particular, the distributed concept of grid computing requires global access to common large-scale data resources, a common accounting and authentication system as well as job scheduling. Other areas addressed by the UP cover exchange data formats, building and abstract description of neuronal network experiments.

After completion, the NM-PM1 system will provide up to 4 million neurons and almost 900 million synapses. Compared to large-scale software simulations, this is four orders of magnitude less. The next version, called the, called the NM-PM2, will provide enhanced synaptic precision and a plasticity processor for flexible learning rules.

For data-intense applications the implementation of higher-level functions into the FCP FPGAs would reduce the load on the compute nodes. Examples for functions operating down to the wafer module are spike train replay, rate-modulated spike train generators and merging of multiple spike train sources. For the upstream connection, spike rate counters, spike source filters and sophisticated recording rules

[21]The experiment does not fail if a small fraction of data is lost; however, the amount of data loss has to be as small as possible to maximize the recorded luminosity.

can reduce the data volume transfered to the host computer.

A continuation plan exists for the NM-PM1 system which envisions up to $5 \cdot 10^9$ neurons and $1.3 \cdot 10^{12}$ synapses distributed on 5000 wafers. In terms of the compute cluster and network architecture, the current approach is capable to scale up into thousands of wafer modules or compute nodes. State-of-the-art data networking equipment supports up to 576×10GbE per switch which helps to keep the communication latency low; hardware configuration, experiment runtime control as well as handling of data streams are tasks that can be intrinsically parallelized. However, the mapping stage within the software flow is not trivially scalable; efforts to parallelize this task and dispatch the computation to large-scale HPC hardware are needed.

Later hardware revisions, i.e. after the NM-PM1, will reduce the amount of external components supporting the wafer module. In particular, the FPGAs are main contributors to the total power consumption; the integration of all communication infrastructures onto the wafer will provide a more scalable system by minimizing the power consumption as well as the assembly work. Technologies like EXTOLL can provide connectivity for both, inter-wafer communication and host connectivity. It supports low-overhead messaging at high packet rates which facilitates spike data communication. The remote memory access allows for direct access from the host to RAM located on the wafer module.

Compared to software simulations, large-scale neuromorphic systems in general provide high simulation speed as well as power efficiency. The slow simulation speed of large-scale neuronal network simulations is often prohibitive in terms of energy and time consumption. Especially, the emulation speed provided by accelerated neuromorphic systems can enable new experiments requiring long simulation durations to capture learning effects which slowly evolve over large time scales [Zenke and Gerstner, 2014]. In the end, the route to large-time-scale experiments as well as high-throughput and interactive modeling leads to neuromorphic systems.

Bibliography

F. Alessio, L. Brarda, E. Bonaccorsi, D. H. Campora Perez, M. Chebbi, M. Frank, C. Gaspar, L. Granado Cardoso, C. Haen, E. Van Herwijnen, R. Jacobsson, B. Jost, N. Neufeld, R. Schwemmer, V. K. Subbiah, and A. Zvyagin. The LHCb data acquisition during LHC run 1. *Journal of Physics: Conference Series*, 513 (1), 2014. doi: 10.1088/1742-6596/513/1/012033.

J. Antolík and A. P. Davison. Integrated workflows for spiking neuronal network simulations. *Frontiers in Neuroinformatics*, 7(34), 2013. doi: 10.3389/fninf.2013.00034.

W. Barth. *Nagios: System and Network Monitoring.* No Starch Press Series. No Starch Press, 2008.

K. Beck. *Test Driven Development: By Example.* Kent Beck signature book. Addison-Wesley, 2002.

R. Berner, T. Delbrück, A. C. Balcells, and A. Linares-Barranco. A 5 Meps $100 USB2.0 address-event monitor-sequencer interface. In *ISCAS*, pages 2451–2454. IEEE, 2007. doi: 10.1109/ISCAS.2007.378616.

W. J. Bolosky and M. L. Scott. False sharing and its effect on shared memory performance. In *4th USENIX Symposium on Experiences with Distributed and Multiprocessor Systems*, San Diego, California, Sept. 1993. USENIX Association.

R. Brette and W. Gerstner. Adaptive exponential integrate-and-fire model as an effective description of neuronal activity. *J. Neurophysiol.*, 94(5):3637 – 3642, 2005. doi: 10.1152/jn.00686.2005.

J. Brown and B. Martin. How fast is fast enough? choosing between Xenomai and Linux for real-time applications. In *Twelfth Real-Time Linux Workshop*, Nairobi, Kenya, Oct. 2010.

D. Brüderle. *Neuroscientific Modeling with a Mixed-Signal VLSI Hardware System.* PhD thesis, Ruprecht-Karls-Universität Heidelberg, 2009.

D. Brüderle, E. Müller, A. Davison, E. Muller, J. Schemmel, and K. Meier. Establishing a novel modeling tool: A python-based interface for a neuromorphic hardware system. *Front. Neuroinform.*, 3(17), 2009. doi: 10.3389/neuro.11.017.2009.

D. Brüderle, M. Petrovici, B. Vogginger, M. Ehrlich, T. Pfeil, S. Millner, A. Grübl, K. Wendt, E. Müller, M.-O. Schwartz, D. Husmann de Oliveira, S. Jeltsch, J. Fieres, M. Schilling, P. Müller, O. Breitwieser, V. Petkov, L. Muller, A. Davison, P. Krishnamurthy, J. Kremkow, M. Lundqvist, E. Muller, J. Partzsch, S. Scholze, L. Zühl, C. Mayr, A. Destexhe, M. Diesmann, T. Potjans, A. Lansner, R. Schüffny, J. Schemmel, and K. Meier. A comprehensive workflow for general-purpose neural modeling with highly configurable neuromorphic hardware systems. *Biological Cybernetics*, 104:263–296, 2011. doi: 10.1007/s00422-011-0435-9.

A. N. Burkitt. A review of the integrate-and-fire neuron model: II. inhomogeneous synaptic input and network properties. *Biological cybernetics*, 95(2):97–112, 2006. doi: 10.1007/s00422-006-0082-8.

Chelsio. Wiredirect, 2011. URL `http://www.chelsio.com/nic/wire-direct/`.

Chelsio. T520-lp-cr – product brief, 2013a. URL `http://www.chelsio.com/wp-content/uploads/2013/10/T520-CR-PB.pdf`.

Chelsio. Preliminary ultra low latency report, 2013b. URL `http://www.chelsio.com/wp-content/uploads/2011/05/Ultra-Low-Latency-Report-040813.pdf`.

P. Clements. *Documenting Software Architectures: Views and Beyond*. SEI series in software engineering. Addison-Wesley, 2 edition, 2011.

USB 2.0. *Universal Serial Bus Revision 2.0 specification*. Compaq, Hewlett-Packard, Intel, Lucent, Microsoft, NEC, and Philips, 2.0 edition, Apr. 2000. URL `http://www.usb.org/developers/docs/`.

A. P. Davison, D. Guarino, and J. Chavas. HBP neuromorphic platform interface (NMPI), 2014.

A. Destexhe and T. Bal. *Dynamic-Clamp: From Principles to Applications*. Springer Series in Computational Neuroscience. Springer, 2009.

M. Djurfeldt. The connection-set algebra—a novel formalism for the representation of connectivity structure in neuronal network models. *Neuroinformatics*, 10(3): 287–304, 2012. doi: 10.1186/1471-2202-12-S1-P80.

M. Djurfeldt, J. Hjorth, J. M. Eppler, N. Dudani, M. Helias, T. C. Potjans, U. S. Bhalla, M. Diesmann, J. H. Kotaleski, and Ö. Ekeberg. Run-time interoperability between neuronal network simulators based on the MUSIC framework. *Neuroinformatics*, 8(1):43–60, 2010. doi: 10.1007/s12021-010-9064-z.

Docker. *An open platform for distributed applications for developers and sysadmins*. Docker, Inc., 2014. URL `http://www.docker.com`.

J. J. Dongarra. The LINPACK benchmark: An explanation. In *Proceedings of the 1st International Conference on Supercomputing*, pages 456–474. Springer Berlin / Heidelberg, 1988. doi: 10.1007/3-540-18991-2_27.

U. Drepper. What every programmer should know about memory, Nov. 2007. Red Hat, Inc.

J. Drexler. Entwurf und Implementierung einer parallelen Netzwerkschnittstelle zum Betrieb Künstlicher Neuronaler Netze. Diploma thesis (german), Ruprecht-Karls-Universität Heidelberg, 2009. HD-KIP-09-05.

D. Duval. From fast to predictably fast. In *Proceedings of the Ottawa Linux Symposium*, OLS '09, pages 79–86, 2009. Red Hat, Inc.

M. Ehrlich, K. Wendt, L. Zühl, R. Schüffny, D. Brüderle, E. Müller, and B. Vogginger. A software framework for mapping neural networks to a wafer-scale neuromorphic hardware system. In *Proceedings of the Artificial Neural Networks and Intelligent Information Processing Conference (ANNIIP) 2010*, pages 43–52, 2010.

Ö. Ekeberg and M. Djurfeldt. *MUSIC – Multi-Simulation Coordinator Users Manual*, 2009.

Electronic Vision(s). Specification of the HICANN-DLS ASIC, 2014. URL `https://gitviz.kip.uni-heidelberg.de/projects/hicann-dls`.

Modules. *The Environment Modules package provides for the dynamic modification of a user's environment via modulefiles.* Environment Modules Project, 2014. URL `http://modules.sourceforge.net`.

J. Eppler. Personal communication, 2014.

J. M. Eppler, H. E. Plesser, A. Morrison, M. Diesmann, and M.-O. Gewaltig. Multithreaded and distributed simulation of large biological neuronal networks. In F. Cappello, T. Herault, and J. Dongarra, editors, *Recent Advances in Parallel Virtual Machine and Message Passing Interface*, volume 4757 of *Lecture Notes in Computer Science*, pages 391–392. Springer Berlin Heidelberg, 2007. doi: 10.1007/978-3-540-75416-9_55.

H. Esmaeilzadeh, E. Blem, R. St. Amant, K. Sankaralingam, and D. Burger. Dark silicon and the end of multicore scaling. In *Proceedings of the 38th Annual International Symposium on Computer Architecture*, ISCA '11, pages 365–376, New York, NY, USA, 2011. ACM. doi: 10.1145/2024723.2000108.

EXTOLL. Introducing EXTOLL Tourmalet, 2014. URL `http://www.extoll.de/images/pdf/EXTOLL_Tourmalet_2014_05.pdf`.

J. Fieres, J. Schemmel, and K. Meier. Realizing biological spiking network models in a configurable wafer-scale hardware system. In *Proceedings of the 2008 International Joint Conference on Neural Networks (IJCNN)*, 2008. doi: 10.1109/IJCNN.2008.4633916.

A. Fog. *Instruction Tables – List of Instruction Latencies, Throughputs and Micro-operation Breakdowns for Intel, AMD and VIA CPUs*, 2014a. URL `http://www.agner.org/optimize/instruction_tables.pdf`.

A. Fog. *The Microarchitecture of Intel, AMD and VIA CPUs – An Optimization Guide for Assembly Programmers and Compiler Makers*, 2014b. URL `http://www.agner.org/optimize/microarchitecture.pdf`.

I. Foster and C. Kesselman, editors. *The Grid: Blueprint for a New Computing Infrastructure.* Morgan Kaufmann, 1st edition, 1998.

H. Franke, R. Russell, and M. Kirkwood. Fuss, futexes and furwocks: Fast userlevel locking in linux. In *Proceedings of the Ottawa Linux Symposium*, OLS '02, pages 479–595, 2002.

S. B. Furber, D. R. Lester, L. A. Plana, J. D. Garside, E. Painkras, S. Temple, and A. D. Brown. Overview of the SpiNNaker system architecture. *IEEE Transactions on Computers*, 99(PrePrints), 2012. doi: 10.1109/TC.2012.142.

F. Galluppi, A. Rast, S. Davies, and S. Furber. A general-purpose model translation system for a universal neural chip. In K. Wong, B. Mendis, and A. Bouzerdoum, editors, *Neural Information Processing. Theory and Algorithms*, volume 6443 of *Lecture Notes in Computer Science*, pages 58–65. Springer Berlin / Heidelberg, 2010. doi: 10.1007/978-3-642-17537-4_8.

Y. Georgiou and M. Hautreux. Evaluating scalability and efficiency of the resource and job management system on large HPC clusters. In W. Cirne, N. Desai, E. Frachtenberg, and U. Schwiegelshohn, editors, *Job Scheduling Strategies for Parallel Processing*, volume 7698 of *Lecture Notes in Computer Science*, pages 134–156. Springer Berlin / Heidelberg, 2013. doi: 10.1007/978-3-642-35867-8_8.

M.-O. Gewaltig and M. Diesmann. Nest (neural simulation tool). *Scholarpedia*, 2 (4):1430, 2007. URL `http://www.scholarpedia.org/article/NEST_(NEural_Simulation_Tool)`.

P. Gleeson, S. Crook, R. C. Cannon, M. L. Hines, G. O. Billings, M. Farinella, T. M. Morse, A. P. Davison, S. Ray, U. S. Bhalla, S. R. Barnes, Y. D. Dimitrova, and R. A. Silver. NeuroML: A language for describing data driven models of neurons and networks with a high degree of biological detail. *PLoS Comput Biol*, 6(6):e1000815, June 2010. doi: 10.1371/journal.pcbi.1000815.

A. Grübl. *VLSI Implementation of a Spiking Neural Network.* PhD thesis, Ruprecht-Karls-Universität, Heidelberg, 2007. HD-KIP 07-10.

A. Grübl. Personal communication, 2013.

A. Grübl, V. Karasenko, and E. Müller. Electronic vision(s) FPGA and low-level performance team – personal communication, 2014.

C. Gutmann. Implementation einer Gigabit-Ethernet-Schnittstelle zum Betrieb eines Künstlichen Neuronalen Netzwerkes. Diploma thesis (german), Ruprecht-Karls-Universität Heidelberg, 2007. HD-KIP-07-08.

S. Hartmann. Personal communication, 2014.

M. Helias, S. Kunkel, G. Masumoto, J. Igarashi, J. M. Eppler, S. Ishii, T. Fukai, A. Morrison, and M. Diesmann. Supercomputers ready for use as discovery machines for neuroscience. *Frontiers in Neuroinformatics*, 6(26), 2012. doi: 10.3389/fninf.2012.00026.

J. L. Hennessy and D. A. Patterson. *Computer Architecture: A Quantitative Approach.* Morgan Kaufmann, Amsterdam, 4th edition, 2007.

R. Hertzog and R. Mas. *The Debian Administrator's Handbook, Debian Wheezy from Discovery to Mastery.* Galerie Perrotin, 2014.

Hewlett-Packard. HP 2910 al switch series – quickspecs, Dec. 2013.

Hewlett-Packard. HP 5120-48G EI switch with 2 interface slots (JE069A) – datasheet, Feb. 2014a.

Hewlett-Packard. HP 5900AF-48XG-4QSFP+ switch (JC772A) – datasheet, Feb. 2014b.

M. Hines, S. Kumar, and F. Schürmann. Comparison of neuronal spike exchange methods on a Blue Gene/P supercomputer. *Frontiers in Computational Neuroscience*, 5(49), 2011. doi: 10.3389/fncom.2011.00049.

M. L. Hines and N. T. Carnevale. *The NEURON Book.* Cambridge University Press, Cambridge, UK, 2006.

D. Hinrichs. Software development in the context of dendrite membrane simulation. Bachelor thesis, Ruprecht-Karls-Universität Heidelberg, 2014.

M. Hock. Test of components for a wafer-scale neuromorphic hardware system. Diploma thesis, Ruprecht-Karls-Universität Heidelberg, 2009. HD-KIP-09-37.

HP and Mellanox. HP Mellanox low latency benchmark report, 2012. URL `http://www.mellanox.com/related-docs/whitepapers/HP_Mellanox_Low%20Latency_Benchmark%20Report%202012.pdf`.

HBP SP9 Specification. *Neuromorphic Platform Specification.* Human Brain Project, Mar. 2014. URL `https://gitviz.kip.uni-heidelberg.de/projects/hbp-sp9-specification--d9-7-1`.

K.-H. Husmann. Handling spike data in an accelerated neuromorphic system. Bachelor thesis, Ruprecht-Karls-Universität Heidelberg, 2012.

IEEE 802.1Q. Virtual bridged local area networks. IEEE Standard 802.1Q-2003, IEEE 802.1 Working Group, 2003.

IEEE 802.3. Ethernet. IEEE Standard 802.3-2012, IEEE 802.3 Ethernet Working Group, 1999.

IEEE 802.3ab. 1000Base-T. IEEE Standard 802.3ab-1999, IEEE 802.3 Ethernet Working Group, 1999.

IEEE 802.3ak. 10GBASE-CX4. IEEE Standard 802.3ak-2004, IEEE 802.3 Ethernet Working Group, 2004.

IEEE 802.3ba. 40GBASE-CR4. IEEE Sbandard 802.3ba-2010, IEEE 802.3 Ethernet Working Group, 2010.

IntelArch. *Intel 64 and IA-32 Architectures Software Developer's Manual*. Intel, June 2014.

IOzone3. *IOzone Filesystem Benchmark*, 2014. URL `http://www.iozone.org`.

ISO/IEC 7498-1:1994. Information technology — open systems interconnection — basic reference model: The basic model. ISO/IEC Standard 7498-1:1994, ISO, Geneva, Switzerland, Nov. 1994.

E. M. Izhikevich. Which model to use for cortical spiking neurons? *IEEE Transactions on Neural Networks*, 15(5):1063–1070, Sept. 2004. doi: 10.1109/TNN.2004.832719.

V. Jacobson and M. J. Karels. Congestion avoidance and control. In *Symposium Proceedings on Communications Architectures and Protocols*, SIGCOMM '88, pages 314–329, Nov. 1988.

S. Jeltsch. *A Scalable Workflow for a Configurable Neuromorphic Platform*. PhD thesis, Ruprecht-Karls-Universität Heidelberg, 2014. HD-KIP 14-51.

Jenkins. *Jenkins CI Server*, 2014. URL `http://jenkins-ci.org`.

B. Kaplan. Presentation from KTH. BrainScaleS WP4 workshop: PyNN for supercomputers and neuromorphic hardware, 2013.

B. A. Kaplan, A. Lansner, G. S. Masson, and L. U. Perrinet. Anisotropic connectivity implements motion-based prediction in a spiking neural network. *Frontiers in Computational Neuroscience*, 7(112), 2013. doi: 10.3389/fncom.2013.00112.

V. Karasenko. A communication infrastructure for a neuromorphic system. Master's thesis, Ruprecht-Karls-Universität Heidelberg, 2014.

R. Kelly and J. Gasparakis. Common functionality in the 2.6 linux network stack. Whitepaper, Intel, Apr. 2010.

M. Kleider, C. Koke, D. Schmidt, and S. Schmitt. Electronic vision(s) calibration team – personal communication, 2014.

A. Kononov. Testing of an analog neuromorphic network chip. Diploma thesis, Ruprecht-Karls-Universität Heidelberg, 2011. HD-KIP-11-83.

K. S. Kravtsov, M. P. Fok, P. R. Prucnal, and D. Rosenbluth. Ultrafast all-optical implementation of a leaky integrate-and-fire neuron. *Optics express*, 19(3):2133–2147, 2011. doi: 10.1364/OE.19.002133.

S. Kunkel, T. C. Potjans, J. M. Eppler, H. E. Plesser, A. Morrison, and M. Diesmann. Meeting the memory challenges of brain-scale network simulation. *Frontiers in Neuroinformatics*, 5(35), 2012. doi: 10.3389/fninf.2011.00035.

O. Lawlor, H. Govind, I. Dooley, M. Breitenfeld, and L. Kale. Performance degradation in the presence of subnormal floating-point values. In *Workshop on Operating System Interference in High Performance Applications*, St. Louis, Missouri, Sept. 2005.

CGroups. *Control Groups – Kernel Documentation*. Linux, 2014. URL `https://www.kernel.org/doc/Documentation/cgroups/`.

L. Ljung and T. Soderstrom. *Theory and Practice of Recursive Identification (Signal Processing, Optimization, and Control)*. The MIT Press, October 1983.

LLNL, SchedMD, et al. *Simple Linux Utility for Resource Management*, 2014. URL `http://slurm.schedmd.com`.

S. J. Lovett, M. Welten, A. Mathewson, and B. Mason. Optimizing MOS transistor mismatch. *IEEE Journal of Solid-State Circuits*, 33(1):147–150, Jan. 1998. doi: 10.1109/4.654947.

M. Massie, B. Li, B. Nicholes, and V. Vuksan. *Monitoring with Ganglia*. O'Reilly and Associate Series. O'Reilly Media, Incorporated, 2012.

P. E. McKenney. 'real time' vs. 'real fast': How to choose? In *Eleventh Real-Time Linux Workshop*, Dresden, Germany, Sept. 2009. OSADL.

Mellanox. SX1036 – product brief, 2013a. URL `http://www.mellanox.com/related-docs/prod_eth_switches/SX1036_Product_Brief.pdf`.

Mellanox. ConnectX-3 Pro – product brief, 2013b. URL `http://www.mellanox.com/related-docs/prod_adapter_cards/PB_ConnectX-3_Pro_Card_EN.pdf`.

Mellanox. IS5024 – product brief, 2014. URL `http://www.mellanox.com/related-docs/prod_ib_switch_systems/IS5024.pdf`.

P. Merolla, J. Arthur, and J. Wittig, Jr. The USB revolution. *The Neuromorphic Engineer*, 2(2):10–11, 2005.

S. Millner. *Development of a Multi-Compartment Neuron Model Emulation.* PhD thesis, Ruprecht-Karls-Universitä Heidelberg, November 2012. HD-KIP 12-83.

S. Millner, A. Grübl, K. Meier, J. Schemmel, and M.-O. Schwartz. A VLSI implementation of the adaptive exponential integrate-and-fire neuron model. In J. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R. Zemel, and A. Culotta, editors, *Advances in Neural Information Processing Systems 23*, pages 1642–1650, 2010.

E. Müller. Operation of an imperfect neuromorphic hardware device. Diploma thesis, Ruprecht-Karls-Universität Heidelberg, 2008. HD-KIP-08-43.

E. Müller. Code-generator-assisted wrapping: from C++ to Python. Electronic Vision's Group Meeting – Presentation, Apr. 2011a.

E. Müller. HMF OS – overview, command and data flow. Electronic Vision's Group Meeting – Presentation, July 2011b.

E. Müller. BSS – HW/SW release 1: Automatic and continuous testing. Electronic Vision's Group Meeting – Presentation, June 2012.

E. Müller. Presentation from UHEI. BrainScaleS WP4 workshop: PyNN for supercomputers and neuromorphic hardware, 2013.

R. C. Murphy, K. B. Wheeler, B. W. Barrett, and J. A. Ang. Introducing the graph 500. Technical report, Sandia National Laboratories, 2010.

J. M. Nageswaran, N. Dutt, J. L. Krichmar, A. Nicolau, and A. V. Veidenbaum. A configurable simulation environment for the efficient simulation of large-scale spiking neural networks on graphics processors. *Neural Networks*, 22(5–6):791–800, 2009. doi: 10.1016/j.neunet.2009.06.028.

R. Naud, N. Marcille, C. Clopath, and W. Gerstner. Firing patterns in the adaptive exponential integrate-and-fire model. *Biological Cybernetics*, 99(4):335–347, Nov. 2008. doi: 10.1007/s00422-008-0264-7.

NEST Initiative. NEST 2.4.2, 2014. URL `http://www.nest-simulator.org/download/gplreleases/nest-2.4.2.tar.gz`.

Neural Ensemble. Website. `http://www.neuralensemble.org`, 2008.

Neo. *A base library for handling electrophysiology data in Python.* The NeuralEnsemble Initiative, 2014. URL `http://www.neuralensemble.org/neo`.

PyNN. *A Python package for simulator-independent specification of neuronal network models.* The NeuralEnsemble Initiative, 2014. URL `http://www.neuralensemble.org/PyNN`.

C. Pape. Vergleich der ESS mit neuromorpher Hardware über eine gemeinsame Bedienungsschnittstelle. Bachelor thesis (german), Ruprecht-Karls-Universität Heidelberg, 2013. HD-KIP 13-93.

J. Partzsch. Personal communication, 2014.

J. Partzsch, V. Thanasoulis, S. Hartmann, and S. Scholze. TU Dresden collaboration partners, FPGA team – personal communication, 2008–2014.

M. J. M. Pelgrom, H. P. Tuinhout, and M. Vertregt. Transistor matching in analog CMOS applications. In *IEEE International Electron Devices Meeting*, pages 915–918, Dec. 1998. doi: 10.1109/IEDM.1998.746503.

D. Perrin. Complexity and high-end computing in biology and medicine. In H. R. Arabnia and Q.-N. Tran, editors, *Software Tools and Algorithms for Biological Systems*, volume 696 of *Advances in Experimental Medicine and Biology*, pages 377–384. Springer New York, 2011. doi: 10.1007/978-1-4419-7046-6_38.

L. L. Peterson and B. S. Davie. *Computer Networks: A Systems Approach, 3rd Edition*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2003.

M. A. Petrovici, B. Vogginger, P. Müller, O. Breitwieser, M. Lundqvist, L. Muller, M. Ehrlich, A. Destexhe, A. Lansner, R. Schüffny, J. Schemmel, and K. Meier. Characterization and compensation of network-level anomalies in mixed-signal neuromorphic modeling platforms. *PLoS ONE*, 2014. URL `http://arxiv.org/abs/1404.7514`. pending publication.

T. Pfeil, A. Grübl, S. Jeltsch, E. Müller, P. Müller, M. A. Petrovici, M. Schmuker, D. Brüderle, J. Schemmel, and K. Meier. Six networks on a universal neuromorphic computing substrate. *Frontiers in Neuroscience*, 7:11, 2013. doi: 10.3389/fnins.2013.00011.

T. Pfeil, J. Jordan, T. Tetzlaff, A. Grübl, J. Schemmel, and K. Meier. The effect of heterogeneity on decorrelation mechanisms in spiking neural networks: a neuromorphic-hardware study. publication in preparation, 2014.

S. M. Potter, A. El Hady, and E. E. Fetz. Closed-loop neuroscience and neuroengineering. *Frontiers in Neural Circuits*, 8(115), 2014. doi: 10.3389/fncir.2014.00115.

R. Prahov, A. Graupner, and E. Müller. Configuration management from the perspective of integrated circuit design. In *Electrical Electronics Engineers in Israel (IEEEI), 2012 IEEE 27th Convention of*, pages 1–5, Nov. 2012. doi: 10.1109/EEEI.2012.6376940.

A. D. Rast, J. Partzsch, C. Mayr, J. Schemmel, S. Hartmann, L. A. Plana, S. Temple, D. R. Lester, R. Schüffny, and S. Furber. A location-independent direct link neuromorphic interface. In *IJCNN*. IEEE, 2013. doi: 10.1109/IJCNN.2013.6706887.

RFC1122, R. T. Braden. *Requirements for Internet Hosts — Communication Layers*, Oct. 1989. URL `http://tools.ietf.org/html/rfc1122`.

RFC1323, V. Jacobson, R. Braden, and D. Borman. *TCP Extensions for High Performance*, May 1992. URL `http://tools.ietf.org/html/rfc1323`.

RFC1379, R. Braden. *Extending TCP for Transactions – Concepts*, Nov. 1992. URL `http://tools.ietf.org/html/rfc1379`.

RFC1948, S. Bellovin. *Defending Against Sequence Number Attacks*, May 1996. URL `http://tools.ietf.org/html/rfc1948`.

RFC2018, J. Mahdavi, S. Floyd, and A. Romanow. *TCP Selective Acknowledgment Options*, Oct. 1996. URL `http://tools.ietf.org/html/rfc2018`.

RFC3366, G. Fairhurst. *Advice to link designers on link Automatic Repeat reQuest (ARQ)*, Aug. 2002. URL `http://tools.ietf.org/html/rfc3366`.

RFC3439, R. Bush and D. Meyer. *Some Internet Architectural Guidelines and Philosophy*, Dec. 2002. URL `http://tools.ietf.org/html/rfc3439`.

RFC4614, M. Duke, R. Braden, W. Eddy, and E. Blanton. *A Roadmap for Transmission Control Protocol (TCP) Specification Documents*, Sept. 2006. URL `http://tools.ietf.org/html/RFC4614`.

RFC5681, M. Allman, V. Paxson, and E. Blanton. *TCP Congestion Control*, Sept. 2009. URL `http://tools.ietf.org/html/RFC5681`.

RFC6298, V. Paxson, M. Allman, J. Chu, and M. Sargent. *Computing TCP's Retransmission Timer*, June 2011. URL `http://tools.ietf.org/html/RFC6298`.

RFC675, V. Cerf, Y. Dalal, and C. Sunshine. *User Datagram Protocol*, Dec. 1974. URL `http://tools.ietf.org/html/rfc675`.

RFC6824, A. Ford, C. Raiciu, M. Handley, and O. Bonaventure. *TCP Extensions for Multipath Operation with Multiple Addresses*, Mar. 2011. URL `http://tools.ietf.org/html/RFC6824`.

RFC768, J. Postel. *User Datagram Protocol*, Aug. 1980. URL `http://tools.ietf.org/html/rfc768`.

RFC791. *Internet Protocol*. Information Sciences Institute, University of Southern California, Sept. 1981. URL `http://tools.ietf.org/html/rfc791`.

RFC793. *Transmission Transport Protocol*. Information Sciences Institute, University of Southern California, Sept. 1981. URL `http://tools.ietf.org/html/rfc793`.

RFC826, D. C. Plummer. *An Ethernet Address Resolution Protocol*, Nov. 1982. URL `http://tools.ietf.org/html/rfc826`.

RFC896, J. Nagle. *Congestion Control in IP/TCP Internetworks*, Jan. 1984. URL `http://tools.ietf.org/html/rfc896`.

J.-P. Richard. Time-delay systems: an overview of some recent advances and open problems. *Automatica*, 39(10):1667–1694, 2003. doi: 10.1016/S0005-1098(03)00167-5.

RIKEN. Largest neuronal network simulation achieved using k computer. Press Release, Aug. 2013. URL `http://www.riken.jp/en/pr/press/2013/20130802_1`.

J. Schemmel. Personal communication, 2014.

J. Schemmel, A. Grübl, K. Meier, and E. Muller. Implementing synaptic plasticity in a VLSI spiking neural network model. In *Proceedings of the 2006 International Joint Conference on Neural Networks (IJCNN)*. IEEE Press, 2006. doi: 10.1109/IJCNN.2006.246651.

J. Schemmel, D. Brüderle, K. Meier, and B. Ostendorf. Modeling synaptic plasticity within networks of highly accelerated I&F neurons. In *Proceedings of the 2007 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 3367–3370. IEEE Press, 2007. doi: 10.1109/ISCAS.2007.378289.

J. Schemmel, J. Fieres, and K. Meier. Wafer-scale integration of analog neural networks. In *Proceedings of the 2008 International Joint Conference on Neural Networks (IJCNN)*, 2008. doi: 10.1109/IJCNN.2008.4633828.

J. Schemmel, D. Brüderle, A. Grübl, M. Hock, K. Meier, and S. Millner. A wafer-scale neuromorphic hardware system for large-scale neural modeling. In *Proceedings of the 2010 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1947–1950, 2010. doi: 10.1109/ISCAS.2010.5536970.

M. Schilling. A highly efficient transport layer for the connection of neuromorphic hardware systems. Diploma thesis, Ruprecht-Karls-Universität Heidelberg, 2010. HD-KIP-10-09.

M. Schmuker, T. Pfeil, and M. P. Nawrot. A neuromorphic network for generic multivariate data classification. *Proceedings of the National Academy of Sciences*, 111(6):2081–2086, 2014. doi: 10.1073/pnas.1303053111.

F. Schürmann, S. Hohmann, J. Schemmel, and K. Meier. Towards an artificial neural network framework. In A. Stoica, J. Lohn, R. Katz, D. Keymeulen, and R. Zebulum, editors, *Proceedings of the 2002 NASA/DoD Conference on Evolvable Hardware*, pages 266–273. IEEE Computer Society, 2002. doi: 10.1109/EH.2002.1029893.

M.-O. Schwartz. *Reproducing Biologically Realistic Regimes on a Highly-Accelerated Neuromorphic Hardware System.* PhD thesis, Ruprecht-Karls-Universität Heidelberg, 2013.

G. Semeraro, G. Magklis, R. Balasubramonian, D. H. Albonesi, S. Dwarkadas, and M. L. Scott. Energy-efficient processor design using multiple clock domains with dynamic voltage and frequency scaling. In *Proceedings of the Eighth International Symposium on High-Performance Computer Architecture*, pages 29–40, Feb. 2002. doi: 10.1109/HPCA.2002.995696.

M. Shafique, S. Garg, J. Henkel, and D. Marculescu. The eda challenges in the dark silicon era: Temperature, reliability, and variability perspectives. In *Proceedings of the The 51st Annual Design Automation Conference on Design Automation Conference*, DAC '14, pages 1–6, New York, NY, USA, 2014. ACM. doi: 10.1145/2593069.2593229.

J. A. Stankovic. *Deadline Scheduling for Real-Time Systems: Edf and Related Algorithms.* Real-time systems series. Springer, 1998.

J.-L. R. Stevens, M. Elver, and J. A. Bednar. An automated and reproducible workflow for running and analyzing neural simulations using lancet and ipython notebook. *Frontiers in Neuroinformatics*, 7(44), 2013. doi: 10.3389/fninf.2013.00044.

A. T. C. Tam and C.-L. Wang. Efficient scheduling of complete exchange on clusters. In *13th International Conference on Parallel and Distributed Computing Systems (PDCS 2000)*, Las Vegas, Aug. 2000.

A. S. Tanenbaum and D. J. Wetherall. *Computer Networks.* Prentice Hall, 5th edition, 2010.

C. M. Thibeault, K. Minkovich, M. J. O'Brien, F. C. Harris, and N. Srinivasa. Efficiently passing messages in distributed spiking neural network simulation. *Frontiers in computational neuroscience*, 7, 2013. doi: 10.3389/fncom.2013.00077.

S. Thompson and S. Parthasarathy. Moore's law: the future of si microelectronics. *Materials Today*, 9(6):20–25, June 2006. doi: 10.1016/s1369-7021(06)71539-5.

Tolly. Mellanox SX1016 & SX1036 10/40GbE switches performance and power consumption evaluation, 2012. URL `http://tolly.com/DocDetail.aspx?DocNumber=212113`.

TOP 500. TOP500 supercomputer sites, 2014. URL `http://www.top500.org`.

B. Vogginger. Testing the operation workflow of a neuromorphic hardware system with a functionally accurate model. Diploma thesis, Ruprecht-Karls-Universität Heidelberg, 2010. HD-KIP-10-12.

XML 1.0. *Extensible Markup Language (XML) 1.0.* W3C, 5.0 edition, Nov. 2008. URL `http://www.w3.org/TR/REC-xml`.

J. Weidendorfer. Sequential performance analysis with callgrind and kcachegrind. In M. Resch, R. Keller, V. Himmler, B. Krammer, and A. Schulz, editors, *Tools for High Performance Computing*, pages 93–113. Springer Berlin / Heidelberg, 2008.

S. A. Weil, S. A. Brandt, E. L. Miller, D. D. E. Long, and C. Maltzahn. Ceph: A scalable, high-performance distributed file system. In *Proceedings of the 7th Symposium on Operating Systems Design and Implementation*, OSDI '06, pages 307–320. USENIX Association, 2006.

Xilinx, Inc. Virtex-5 family overview, Feb. 2009. URL `http://www.xilinx.com/support/documentation/data_sheets/ds100.pdf`.

Xilinx, Inc. 7 series FPGAs overview, Oct. 2011. URL `http://www.xilinx.com/support/documentation/data_sheets/ds160.pdf`.

Xilinx, Inc. *Kintex-7 FPGA Connectivity Targeted Reference Design: User Guide*, Nov. 2012. URL `http://www.xilinx.com/support/documentation/boards_and_kits/ug927-K7-Connectivity-TRD.pdf`.

Xilinx, Inc. 7 series FPGAs overview, Feb. 2014. URL `http://www.xilinx.com/support/documentation/data_sheets/ds180.pdf`.

F. Zenke and W. Gerstner. Limits to high-speed simulations of spiking neural networks using general-purpose computers. *Frontiers in Neuroinformatics*, 8(76), 2014. doi: 10.3389/fninf.2014.00076.

Glossary

10GbE
: 10-Gigabit Ethernet [IEEE 802.3ak, 2004]. 5, 17, 27, 30, 34, 35, 38, 52, 57, 64, 65, 68–70, 72, 107, 110–112, 115, 118, 120, 147, 148, 152, 180, 181, 183, 184

  10GBASE-SR. 34

  SFP+ 10G Twinax Direct-Attach Copper. 35

1GbE
: Gigabit Ethernet [IEEE 802.3ab, 1999]. 17, 21–23, 26, 27, 33–36, 38, 39, 42, 44, 45, 49, 52, 54, 56, 57, 60, 63–69, 94, 110–112, 115, 119, 147, 148, 180

40GbE
: 40-Gigabit Ethernet [IEEE 802.3ba, 2010]. 5, 27, 31, 32, 34, 35, 38, 69, 71, 72, 107, 148, 150

8P8C
: 8 position 8 contact connector. Modular connectors have been originally used in telephone wiring. 26

ACK
: Acknowledge successful data reception. 43, 45, 46, 48, 56, 58–60, 62, 191

ADC
: Analog-to-digital converters transform continuous signals, e.g. voltages, into digital numbers. 23, 80

AdEx
: Adaptive exponential integrate-and-fire model [Brette and Gerstner, 2005]. 18, 23

AER
: Address-event representation describes a neuronal event, i.e. spike, using only an identifier. The spike time is implicitly encoded in the events presence. It often connotes a handshake protocol for data exchange. 37

AnaB
: Analog Breakout PCB. 17, 19

AnaRM
: Analog readout module. 18, 19, 23, 33, 36, 38, 79–81, 94, 95, 97

AnaRMAN
Analog readout module aggregator node. 18, 23, 36, 38, 41, 79–81, 94, 95

API
An application programming interface provides a specified interface to a software component. Functions, parameters and return values are covered. 49, 58, 73–78, 80, 83–86, 88, 89, 93, 108, 111, 114, 115, 127, 130, 136, 149, 172, 176

ARP
Address resolution protocol [RFC826]. 64, 66

ARQ
Automatic repeat request protocols provide reliable data transfer. 42, 45, 46, 48, 49, 52, 58

AuxPwr
Auxiliary Power Supply PCB. 17, 19

Backbone switch
The backbone switch interconnects compute nodes and wafer module. 17, 34, 35, 117, 118, 149

BEG
Background event generator. 139

Blue Gene/P
IBM Blue Gene/P supercomputer. 96, 107

Blue Gene/Q
IBM Blue Gene/Q supercomputer. 96

Boost
Boost C++ libraries. 90

boost::serialization
A C++ library providing capabilities. Binary and text-based output formats are supported. 90

Brian
Brian spiking neural network simulator. 88

BSS
BrainScaleS project. 15, 34, 35, 41, 56, 57, 60, 62, 75, 77, 88, 94, 99, 105, 110, 111, 114, 115, 118, 141, 172, 182

C
The C programming language. 80, 186

C++
: The C++ programming language. 75, 76, 89, 92, 168, 171

Calibtic
: Database for calibration data. 78

Callgrind
: Valgrind-based Profiling Tool. 52, 134, 135

CephFS
: The Ceph file system is based on the open-source Ceph distributed object storage system [Weil et al., 2006]. 32

cgroups
: Linux kernel-based isolation mechanism [CGroups, 2014]. 33, 97, 174

CI
: Technique in software development which emphasizes early integration of changes and testing. 92, 181

cLIF
: Leaky integrate-and-fire neuron model with current-based synapses. 123, 125, 130–134, 187

clock_gettime
: syscall to retrieve the time of the specified clock. 114, 183, 185

CMOS
: Complementary metal-oxide-semiconductor. 18, 172

Compute cluster
: A collection of computers interconnected by a dedicated network. 17, 25, 33, 35, 89, 145, 147, 148, 150, 152

Compute node
: A single compute node as part of a Cluster. 17, 19, 23, 27–30, 33, 34, 36, 38, 39, 41, 42, 44, 52, 54, 55, 57, 66, 69, 70, 74, 77, 80, 81, 84, 90, 94–97, 99, 102, 112, 114–120, 122, 123, 131, 134, 135, 141, 144, 145, 148–152, 168, 183, 186, 193

COTS
: Commercially available products. 107

CPEX
: Complete Pairwise Exchange [Tam and Wang, 2000]. 108

CPU
: Central processing unit. 27–30, 32, 50, 95, 97, 115, 120, 150, 172, 180–182

CRC
Cyclic redundancy check. 20, 44

DDR
double data rate. 67

Debian Wheezy
Debian Wheezy. 33, 186

DenMem
The dendrite membrane circuit is the basic building block of the analog neurons on the HICANN. Multiple units can be connected to increase the maximum number of inbound or pre-synaptic connections. 42, 84, 138

DMA
direct memory access. 135

DNC
digital network chip. 138, 139

Docker
Tool for automated deployment of applications inside software containers (using LXC, Linux containers). 93

ENI
Container building located next to the KIP building. 35, 190

ESS
Executable System Specification. 76, 78, 80, 83, 93

Ethernet
Ethernet is a family of data networking technologies. The original version provided 10 Mbit/s [IEEE 802.3, 1999]. 21, 33, 39, 41, 42, 44, 64, 66, 112

EXTOLL
High-performance interconnection technology which provides ultra-low latencies and high message rates [EXTOLL, 2014]. 107, 152

FACETS
The FACETS project (fast analog computing with emergent transient states). 18, 22, 37, 56, 74, 76, 85, 93, 110, 178

FACETS Stage 1
Accelerated neuromorphic system based on Spikey chips. 23, 49

FCP
FPGA communication PCB. 17–24, 26, 27, 34–36, 39, 41, 42, 52, 54–58, 60, 62–69, 71, 72, 79, 81, 83, 84, 94, 95, 97–100, 104, 110–112, 114, 118, 119, 130, 134, 136, 145, 147–149, 151, 180, 182, 184

flock
syscall to acquire locks on an open file. 81

Flow control
Flow control provides means to throttle the sending rate within a communication channel to not overwhelm the receiving side. 42

FPGA
Field-programmable gate array. 5, 10, 16, 18, 20–24, 26, 34, 39, 41, 42, 48, 52, 54–69, 71, 72, 75, 79, 81, 83, 94, 95, 97, 100, 104, 110–112, 114, 118, 138, 145, 147–152, 171, 173, 178–180, 182, 184, 191

FSM
finite state machine. 83, 178

futex
fast userspace mutex. 49

GCC
GNU Compiler Collection. 183, 186

GPU
graphics processing unit. 75, 95, 97

Graph 500
Supercomputer list based on a benchmark which focuses on data-intensive loads [Murphy et al., 2010]. 27, 29

GTest
Google C++ testing framework. 92

GUI
Graphical User Interface. 94

HALbe
hardware abstraction layer back end. 76–81, 83, 84, 181, 184

HBP
Human Brain Project. 13, 15–17, 27, 34, 35, 41, 88, 94, 99, 103, 104, 110, 118, 141, 149, 151, 179, 182

HDD
Hard disk drive. 30, 31, 182

HICANN
The high-input count analog neuronal network chip (version 2) is the current design of the NM-PM1 and BSS systems. It is a mixed-signal implementation of a highly configurable neuronal network produced in 180 nm CMOS technolgy. 18, 20–24, 41, 48, 52, 54, 55, 57, 64, 66–68, 76, 81, 83, 84, 94, 95, 98, 111, 112, 115, 118, 119, 123, 125–127, 134, 136–138, 141, 144, 149, 170, 184

HICANN Wafer
A 20 cm silicon wafer with 384 HICANN ASICs interconnected by wafer-scale postprocessing. 34

HICANN-ARQ
HICANN-ARQ protocol. 16, 21, 45, 52

hicann-system
lowest-level API for hicann access. 111, 184

HMF
Hybrid Multiscale Facility. 12, 15, 17, 18, 21–25, 27–30, 34–37, 39, 40, 51, 57, 74, 78, 79, 84, 86, 88, 90, 94, 98, 100–106, 110–112, 114–116, 118, 122, 134–136, 141, 145, 147, 148, 150, 176, 177, 181, 183, 186

HMF conventional part. 77, 99, 101, 102, 107, 123, 182

HMF neuromorphic part. 107, 109

HostARQ
HostARQ protocol. 16, 40, 45, 48, 49, 51–58, 60, 62, 64–67, 69–71, 118, 150, 181

HPC
high-performance computing. 27, 37, 74, 75, 94–99, 103, 105, 148, 150–152

InfiniBand
InfiniBand. 107

Intel Core
Brand name used by Intel for high-end consumer central processing units (CPUs). 27

IOzone3
IOzone is a filesystem and disk benchmarking tool. 188, 189

IP
internet protocol. 37, 39, 41, 94

IPC
inter-process communication. 49, 50, 83, 84, 88

IPv4
internet protocol version 4. 39, 41, 42, 54, 66, 111, 112, 150

IPv6
internet protocol version 6. 39

iRODS
integrated rule-oriented data system. 99

Jenkins
Open-source continuous integration tool. 92, 93

JTAG
Joint Test Action Group (JTAG) is colloquial name for IEEE 1149.1 which is a standard for a test access port and boundary-scan architecture. 41

Kintex-7
Xilinx Kintex-7 FPGA [Xilinx, Inc., 2014]. 24, 56–58, 60, 114

KIP
Kirchhoff-Institute for Physics. 35, 170, 190

KIP
Kirchhoff-Institute for Physics. 34

KVM
keyboard video mouse. 35

L1
Layer 1. 18, 20, 66, 81, 112, 136, 138, 139

L2
Layer 2. 20, 21

LabVIEW
National Instruments LabVIEW. 94

LIF
leaky integrate-and-fire model. 23, 37, 86, 110, 130, 142, 178, 186

LINPACK
A benchmark that measures floating point performance [Dongarra, 1988]. 13, 27, 28

Linux
Linux Operating System. 29, 32, 33, 52, 82, 96, 120

LSB
least significant bit. 112, 138

LXC
Linux kernel-based containment system (cgroups plus namespacing). 97, 170

MAC
media access controller. 44, 52

MainPCB
wafer module main PCB. 17, 22, 24

MappingTool
Software package to transform a biological network description into a hardware configuration. Superseded by Marocco. 85

Marocco
Marocco. 77–79, 85, 86, 89, 91, 174, 181

MIC
many integrated core. 97

mlock
syscall to force memory of calling process' virtual address space into RAM, prevent paging out. 120

mmap
syscall to map files or devices into the virtual address space of the calling process memory. 51, 52, 118, 121

MPI
message passing interface. 33, 107, 108, 110, 145

MSB
most significant bit. 41, 48

MTU
Maximum Transfer Unit. 42, 44, 54, 60, 69, 71, 191

MUSIC
Multi-Simulation Coordinator. 108, 109, 136, 145, 149

mutex
mutual exclusion. 49, 51

NAPI
extension to linux NIC driver framework for improving performance of high-speed networking [Kelly and Gasparakis, 2010]. 52, 120

NCSim
Cadance NCSim. 57

Neo
Python package for representing electrophysiology data. 88

NEST
NEural Simulation Tool [Gewaltig and Diesmann, 2007]. 13, 86, 108, 123, 127, 129–132, 147

NeuroML
Declarative, XML-based specification language for neuronal network models Gleeson et al. [2010]; XML 1.0 [2008]. It support 3 levels of detail: anatomical, biophysical and topological description. 86, 88

NEURON
NEURON Simulator [Hines and Carnevale, 2006]. 86, 88, 107, 130

NFSv4
NFSv4. 30, 33

NIC
network interface controller. 30–32, 52, 60, 69, 107, 110, 115, 118, 120, 135, 174, 181

NineML
Declarative, XML-based specification language for neuronal network models. 88

NM-MC
Neuromorphic many-core system.

neuromorphic multi-core architecture version 1. 103

NM-PM
Neuromorphic physical model system. 80, 84, 94

Neuromorphic physical model version 1 including all system components and the control cluster. 12–15, 17, 18, 21–24, 27–41, 44, 51, 52, 60, 62, 69–72, 74, 77–80, 84, 86–88, 94–96, 98, 101, 103, 104, 110, 112, 114, 115, 117–120, 147–152, 172, 177, 183, 189, 193

Neuromorphic physical model version 2 including all system components and the control cluster. 151

NMPI
HBP neuromorphic platform interface. 103

NumPy
Python package for scientific computing. Its core is the N-dimensional array data structure. The array provides strided views on memory that is kept in the underlying C implementation. 88

OFED
OpenFabrics Alliance. 33

OpenID
OpenID, or OID, is an open standard and decentralized protocol for user authentication. 99

OSI
Open Systems Interconnection. 40

PCB
printed circuit board. 10, 16, 18, 19, 21, 22, 24, 26, 33, 39, 110, 167, 168, 171, 174, 180, 182

PCIe
Peripheral Component Interconnect Express. 30, 32, 181

PMU
power management unit. 18, 19, 24, 38, 81, 94

PSP
postsynaptic potential. 86

PSU
power supply unit. 24, 30–32, 94, 181

PXE
preboot execution environment. 33

Py++
Python package that generates code for automated wrapping to Python. 75

PyHAL
Python-based hardware abstraction layer. 74–76

PyHMF
PyNN for the HMF. 77–79, 89, 91, 181

PyNN
API for procedural specification of neuronal network models. 73–79, 85, 86, 88, 89, 93, 127, 132, 136, 145, 149, 176, 177, 186

PyNN.hardware.nmpm
PyNN back end for NM-PM1. 88

Currently, the HMF and NM-PM1 use the same interface. 88

Python
Python Programming Language. 74–76, 176

QSFP
Quad SFP+. 34

RAM
random access memory. 28, 152, 174

dynamic random access memory. 21–23, 58

synchronous dynamic random-access memory. 67

static random access memory. 57, 67

Raspberry Pi
Raspberry Pi. 24, 39, 81

RCF
remote call framework. 81, 84, 89–91, 192

recvfrom
syscall to send a message from a destination on a socket. 120

ReDMan
Database for resource defect management. 78

REST
representational state transfer. 103

RMSE
root-mean-square error. 28, 42, 62, 64, 65, 70, 71, 91, 100, 115

RPC
remote procedure call. 78, 80, 81, 89, 148

RTT
round-trip time. 42, 44, 51, 58, 62, 69, 113, 114, 116, 117, 119, 122, 185, 193

SAS
Serial attached SCSI. 31, 32

SATA 6G
SATA revision 3.0. 31, 32

sched_setaffinity
syscall to set process' CPU affinity mask. 120

sched_setscheduler
syscall to set scheduling policy and parameters. 120

Scheriff
FSM to check correct order of configuration. 83, 84

SCtrlTP
SlowControl transport protocol. 49

sendto
syscall to send a message on a socket to a destination. 120

serialization
Serializing or marshalling means translating memory data structures into a binary format that can be stored or transmitted and can later be used to reconstruct the original data structure. 74, 89, 91, 148, 168, 192

SFP+
Enhanced small form-factor pluggable supporting up to 10 Gbit/s. 34, 177

SimDenMem
HALbe simulation back end for analog circuits. 78, 83, 84

SLURM
simple Linux utility for resource management. 33, 79, 96–104, 148, 151

Spartan-6
Xilinx Spartan-6 FPGA [Xilinx, Inc., 2011]. 23

Spikey
Spikey is a chip-based neuromorphic system developed during FACETS that implements 384 LIF neurons and approximately 100k synapses. 33, 37, 73–76, 110, 170

SpiNNaker
Spiking Neural Network Architecture. 111, 112, 118

SQL
Structured query language. 96

SSD
Solid-state disk. 32

StHAL
stateful hardware abstraction layer. 77–79, 84, 181, 184

SVN
Subversion. 57

syscall
Program request to operation system kernel. 52

TCP
transmission control protocol. 42, 150

TCP/IP
TCP/IP v4. 39, 40

TEPS
traversed edges per second. 29

TUD
Technische Universität Dresden. 111

UDP
user datagram protocol. 40–42, 44, 52, 54, 57, 63–65, 111, 112, 115

UNICORE
uniform interface to computing resources. 99

UNIX
UNIX. 49, 50, 52, 81, 101, 113, 114, 120, 149

UP
HBP unified portal: a general interface providing visualization, simulation and analysis capabilities. 74, 103, 104, 151

USB 2.0
Universal Serial Bus version 2.0 [USB 2.0, 2000]. 23, 33, 36, 37, 75, 76, 81, 94

Valgrind
Versatile Debugging and Profiling Tool. 52, 134, 169

VDD
1.8 V digital power supply voltage for the Wafer (1.8 V). 83

VerCL
virtual environment for closed-loop experiments. 84, 114, 118, 120–122

Virtex-5
Xilinx Virtex-5 FPGA [Xilinx, Inc., 2009]. 24, 56–58, 60, 114

Visions
Electronic Vision(s) group. 92

VLAN
Virtual LAN. 44, 66

VLSI
Combines integrated circuits onto a single chip. 13

VPN
virtual private network. 39

Wafer
Silicon wafer used as the basis of micro-chip production. 17, 24, 152

Wafer module
Assembly of an HICANN wafer, a Main PCB, 48 FCPs and power supply PCBs. 13–19, 21–25, 27, 30, 33–36, 38, 54, 56, 57, 62, 66, 67, 69, 72, 74, 80–83, 94–99, 102, 104, 110, 114, 145, 148, 149, 151, 152, 168

Wafer switch
The wafer switch aggregates the individual 1GbE links of the FPGA communication PCBs (FCPs) to 10GbE. 17, 35, 38, 65, 68, 115

wafer-scale
Wafer-scale integration describes very-large integrated circuit systems that use complete wafers. 18

Wall-clock time
The wall-clock time is the (real) time that passed between starting and finishing a task. 109, 110

WIO
wafer I/O PCB. 19, 22

WIOH
horizontal wafer I/O PCB. 17

WIOV
vertical wafer I/O PCB. 17, 110

Wireshark
Wireshark packet analyzer. 52, 53, 64

x86
Intel x86 CPU architecture. 120

A. Appendix

A.1. Repositories and Links

Project Server	`https://gitviz.kip.uni-heidelberg.de/`
Issues	`https://gitviz.kip.uni-heidelberg.de/issues/%u`
CI Server	`https://gitviz.kip.uni-heidelberg.de:8443/`
HostARQ	`https://gitviz.kip.uni-heidelberg.de/projects/sctrltp`
HALbe	`https://gitviz.kip.uni-heidelberg.de/projects/halbe`
StHAL	`https://gitviz.kip.uni-heidelberg.de/projects/sthal`
Marocco	`https://gitviz.kip.uni-heidelberg.de/projects/marocco`
PyHMF	`https://gitviz.kip.uni-heidelberg.de/projects/pyhmf`
cl-tests	`https://gitviz.kip.uni-heidelberg.de/projects/cl-tests`

Table A.1.: List of repositories and project links.

A.2. Cluster & Network Architecture

Lists of Components

	#	Component
CPU	1	Intel i7-2600
		Sandy Bridge, 32 nm
RAM	32 GiB	DDR3-1333
Main Board	1	Intel Q67 chipset-based
		Remote management via intel AMT
	1	PCIe ×16 Gen3 (16 lanes) slot
	4	Memory slots, DDR3-1333
NIC	1	Intel E10G81G2P
	1	10GbE port
		MPI with RDMA-support
Case/PSU		Desktop case, 300W

Table A.2.: Components of a HMF compute node. The quad-core desktop CPU runs at a base clock speed of 3.4 GHz (turbo speed 3.8 GHz).

Storage Node

- 16×4 TB Hard disk drives (HDDs) as mid-term storage
- based on RAID6 using Linux Software-RAID default settings:
  - chunk size 512 KiB, left-symmetric
- Filesystem: Ext4, using default settings:
  - block size 4 KiB
  - Stride is the number of file-system blocks fitting into the RAID chunk size, the so-called *stripe*. It is the atomic data unit of the RAID. The metadata layout is optimized to not overlap.
  - stride = chunk/block = 512 KiB/4 KiB = 128
  - The formatting is aligned to RAID stripe size in order to prevent a read-modify-write cycle of the parity when data is written.
  - Due to the large disk caches, we use write barriers and an ordered journal on the file system to minimize potentiall data loss after a power cut.
- Some further optimization was performed:
- `/sys/block/mdx/md/stripe_cache_size`

A.3. References

To ensure reproduction of results it is essential to specify software revisions and configuration options. Compared to a purely software-based environment, the BrainScaleS (BSS) and Human Brain Project (HBP) systems have more degrees of freedom. For instance, the FPGA communication PCB (FCP) firmware revisions get updates, the external voltages for analog circuits are adjusted and other changes occur that are not easily traceable by software.

Throughput Measurement of Mapping Input Stage

Host	`AMTHost22`
Memory	32 GiB
CPU	Intel i7-2600 (3.4 GHz base)
Environment	same as HMF-CP cluster
Source	`euter:test/euter-client.cpp` (`0xe8aefe`)

Table A.3.: References for the throughput measurement of the mapping input stage (see figure 2.28).

Critical Timings

All micro benchmarks were performed on Hybrid Multiscale Facility (HMF) compute nodes using Debian wheezy GNU Compiler Collection (GCC) (version 4.7.1-7) and the following parameter set: `-std=gnu++11 -g -O3 -ffast-math`. Additional tests using other parameters (e.g., `-funroll-loops`) performed worse and were not used. The length of the measure `loop()` was tuned to yield a total runtime of approx. 1 s. Time measurements were performed using `clock_gettime(CLOCK_MONOTONIC, &t)`. The first tests showed strong dependency on test input data which was related to handling of *subnormal* floating point numbers [Lawlor et al., 2005]. A deeper investigation [Fog, 2014a,b] shows deficiencies for the HMF compute nodes when handling subnormal floating point numbers. This is why all tests results are given with flush-to-zero and denormals-are-zero bits active. Further investigation is needed for the new NM-PM1 compute nodes.

Closed-Loop Experiments

N	25
w_{proj}	0.04
r	1
F	-0.35
Hosts	`AMTHost1`, `AMTHost2`
Repository	cl-tests `0xfcecae`
Executable	`tests/VisualFeedbackTest`
Network configuration	direct connection via 10GbE (cf. table A.2)

Table A.4.: Reference data for closed-loop experiment using custom cLIF neuron implementation.

N	32
w_{proj}	0.05
r	1.0
F	-0.35
HICANN	276
FPGA firmware revision	r1085, 2014-01-27
Wafer	#0
Host	`AMTHost22`
Host Configuration Revision	`0x3fdd80`
StHAL	`0x54a2c8`
HALbe	`0x1f8175`
hicann-system	`0xc98d08`
Repository	HALbe
Executable	`tools/RealtimeVisualFeedbackTest`
powered reticles	10, 18, 21, 25, 31, 33, 44, 45
analog voltages	not used
Network configuration	10GbE via HP 2910al to FCP

Table A.5.: Reference data for closed-loop experiment using chip-based loopback.

N	9
`wafer_0/hicann-Wafer(0)-Enum(276).xml`	`0xc0597c1b`
`wafer_0/w0-h276.xml`	`0x3075d9b3`

Table A.6.: Reference data for closed-loop experiment using hardware neurons. Only the modified parameters of table A.5 are specified, all other parameters are unmodified. The calibration data files can be identified by their SHA-1 checksums.

A.4. Code Listings

Ping – a `socket()`-based Implementation

The custom ping implementation uses a server to *reply* to incoming requests and a client which sends the request and performs the time measurement. In particular, the server creates a socket and listens to incoming packet on a port. Within the while-loop packets are received and sent back to the original sender. Code for error handling, include statements and helper functions have been removed for brevity.

```
int main(int argc, char * argv[]) {
    int sockfd, n;
    struct sockaddr_in saddr, caddr;
    socklen_t len;
    char mesg[MTU];

    sockfd=socket(AF_INET, SOCK_DGRAM, 0);

    bzero(&saddr,sizeof(saddr));
    saddr.sin_family = AF_INET;
    saddr.sin_addr.s_addr = htonl(INADDR_ANY);
    saddr.sin_port = htons(PORT);
    bind(sockfd,(struct sockaddr *)&saddr, sizeof(saddr));

    while(1) {
        len = sizeof(caddr);
        n = recvfrom(sockfd, mesg, 1000, 0, (struct sockaddr *)&caddr, &len);
        sendto(sockfd, mesg, n, 0, (struct sockaddr *)&caddr, sizeof(caddr));
    }
}
```

Listing 1: Source code of UDP socket-based ping server

The client sends the current time (measured by `clock_gettime()` and converted to ns and stored in a 64-bit variable) to the server and waits for an answer. The answer is used (according to section 3.3.3) to calculate the round-trip time (RTT). Individual RTTs are stored into an array.

```
int main(int argc, char * argv[]) {
    int sockfd, n, i;
    struct sockaddr_in saddr;
    uint64_t localtime = 0, remotetime = 0, *rtt, *rtt_copy;
    double avg = 0.0, rms = 0.0;

    if (argc != 2) {
        printf("usage: %s <IP address>\n", argv[0]);
        exit(EXIT_FAILURE);
    }

    sockfd = socket(AF_INET, SOCK_DGRAM, 0);

    bzero(&saddr, sizeof(saddr));
    saddr.sin_family = AF_INET;
    saddr.sin_addr.s_addr = inet_addr(argv[1]);
    saddr.sin_port = htons(PORT);
    rtt = malloc(sizeof(uint64_t) * ITER);

    for (i = 0; i < ITER; i++) {
        localtime = gettime();
        sendto(sockfd, &localtime, sizeof(localtime), 0,
            (struct sockaddr *)&saddr, sizeof(saddr));
        recvfrom(sockfd, &remotetime, sizeof(remotetime), 0, NULL, NULL);
        rtt[i] = gettime() - remotetime;
    }
```

Listing 2: Source code of UDP socket-based ping server

PyNN Default Parameters of the `IF_cond_exp` and `IF_curr_exp` Models

The default PyNN parameters for the conductance-based and current-based synapses in the leaky integrate-and-fire model neuron models can be found in the PyNN documentation: `http://neuralensemble.org/docs/PyNN/standardmodels.html`

Custom cLIF Implementation

Compiled using GCC 4.7.2 on Debian Wheezy on a HMF compute node. The compiler flags `-ffast-math` and `-O2` were active. In the following code listing, the C code is interleaved with the corresponding assembler code; the AT&T syntax is used, i.e. source before destination.

```
void update() {
  if (refrac > 0)
  400680:   8b 05 f6 14 00 00       mov    0x14f6(%rip),%eax        # 401b7c <refrac>
  400686:   85 c0                   test   %eax,%eax
  400688:   75 66                   jne    4006f0 <_Z6updatev+0x70>
    --refrac;
  else {
    W = w_decay_constant * W + J;
  40068a:   f2 0f 10 05 f6 14 00    movsd  0x14f6(%rip),%xmm0       # 401b88 <W>
  400691:   00
  400692:   f2 0f 59 05 d6 14 00    mulsd  0x14d6(%rip),%xmm0       # 401b70 <w_decay_constant>
  400699:   00
  40069a:   f2 0f 58 05 de 14 00    addsd  0x14de(%rip),%xmm0       # 401b80 <J>
  4006a1:   00
    if (W > w_thresh) {
  4006a2:   66 0f 2f 05 b6 14 00    comisd 0x14b6(%rip),%xmm0       # 401b60 <w_thresh>
  4006a9:   00
    W = w_decay_constant * W + J;
  4006aa:   f2 0f 11 05 d6 14 00    movsd  %xmm0,0x14d6(%rip)       # 401b88 <W>
  4006b1:   00
    if (W > w_thresh) {
  4006b2:   76 1c                   jbe    4006d0 <_Z6updatev+0x50>
      refrac = refrac_periods;
  4006b4:   8b 05 be 14 00 00       mov    0x14be(%rip),%eax        # 401b78 <refrac_periods>
      W = w_reset;
  4006ba:   f2 0f 10 05 96 14 00   movsd  0x1496(%rip),%xmm0        # 401b58 <w_reset>
  4006c1:   00
  4006c2:   f2 0f 11 05 be 14 00   movsd  %xmm0,0x14be(%rip)        # 401b88 <W>
  4006c9:   00
      refrac = refrac_periods;
  4006ca:   89 05 ac 14 00 00      mov    %eax,0x14ac(%rip)         # 401b7c <refrac>
      W = w_reset;
    }
  }
  J = j_decay_constant * J;
  4006d0:   f2 0f 10 05 a8 14 00   movsd  0x14a8(%rip),%xmm0        # 401b80 <J>
  4006d7:   00
  4006d8:   f2 0f 59 05 88 14 00   mulsd  0x1488(%rip),%xmm0        # 401b68 <j_decay_constant>
  4006df:   00
  4006e0:   f2 0f 11 05 98 14 00   movsd  %xmm0,0x1498(%rip)        # 401b80 <J>
  4006e7:   00
  4006e8:   c3                     retq
  4006e9:   0f 1f 80 00 00 00 00   nopl   0x0(%rax)
  4006f0:   83 e8 01               sub    $0x1,%eax
  4006f3:   89 05 83 14 00 00      mov    %eax,0x1483(%rip)         # 401b7c <refrac>
  4006f9:   eb d5                  jmp    4006d0 <_Z6updatev+0x50>
  4006fb:   0f 1f 44 00 00         nopl   0x0(%rax,%rax,1)
}
```

Listing 3: Assembler dump of the custom cLIF implementation. Only the `update()` procedure is plotted; incoming spikes are handled by another function.

The dump shows the `update()` of the custom current-based leaky integrate-and-fire model implementation. As described in section 3.5.5, the implementation uses two floating point multiplications, one floating point addition and two conditional jumps for a typical update (when the neuron is not in the refractory phase). Received spikes are updated in another function (not shown) that modifies `J` by adding a constant (i.e. the weight).

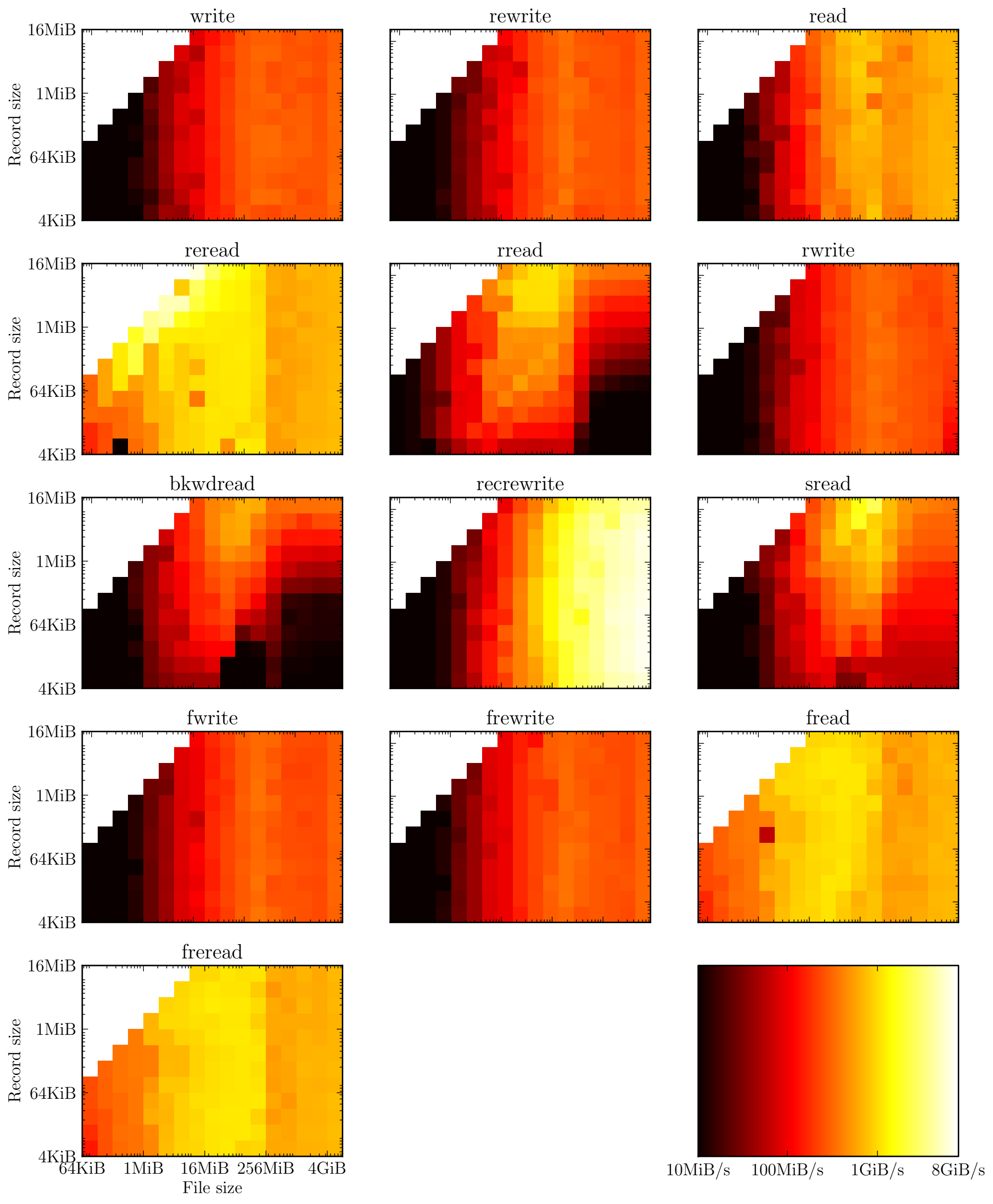


Figure A.1.: IOzone3 results for the storage node, *sto*. The plots show different file access patterns: `rread` and `rwrite` Transaction, or record, size is plotted over file size. For large record sizes the performance numbers show the sequential read/write performance. Different read and write strategies evaluate effects of disk caching, disk seek latencies, filesystem overhead, and kernel buffering. See IOzone3 [2014] for details. The test used the options `-u` to automatically remount between tests, `-e` to include sync times, `-c` to include `close()` times, and `-g 64g -az` to sweep up to 64 GiB.

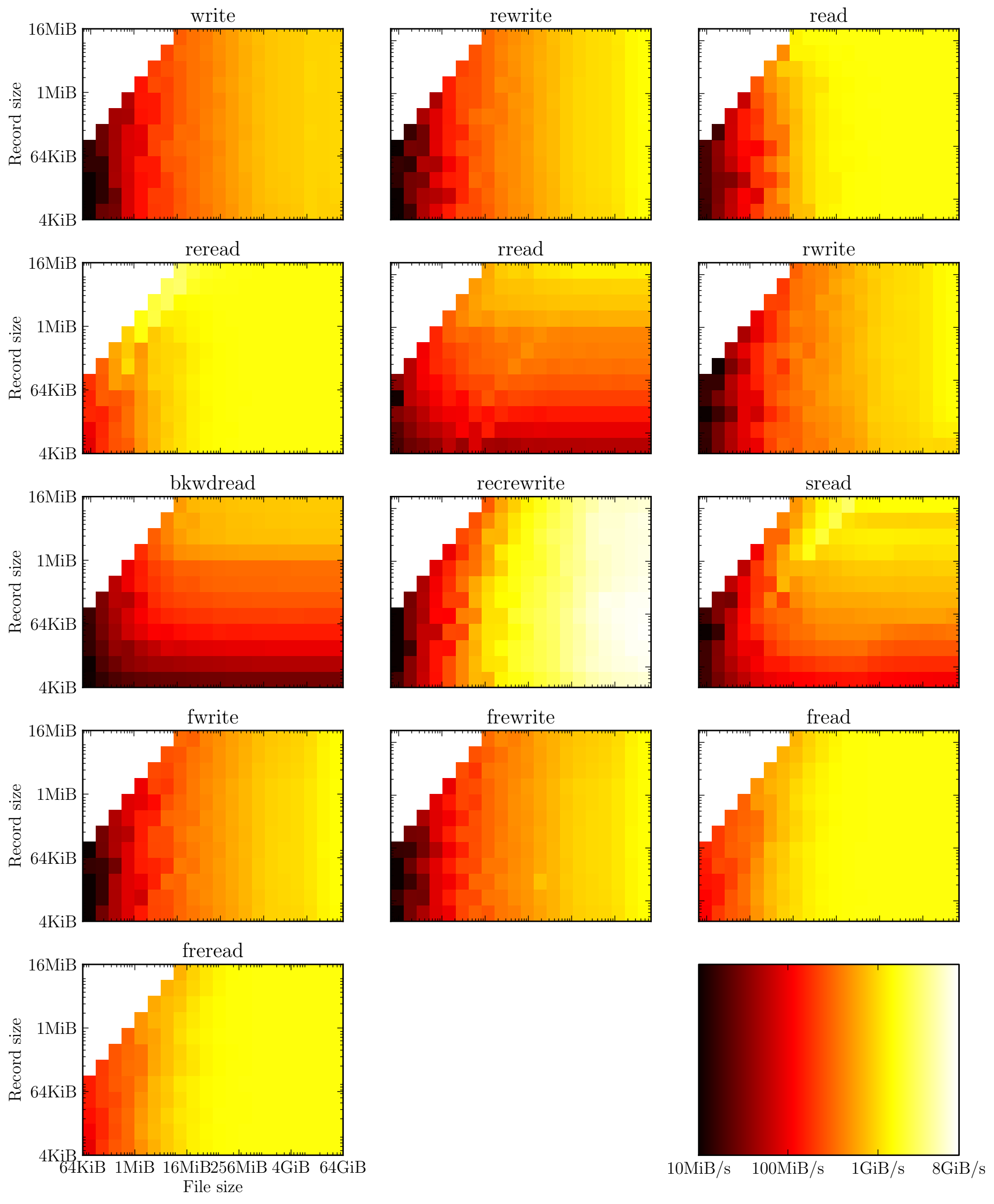


Figure A.2.: IOzone3 results for one NM-PM1 frontend node, *hel*. The same set of options as in figure A.1 was used.

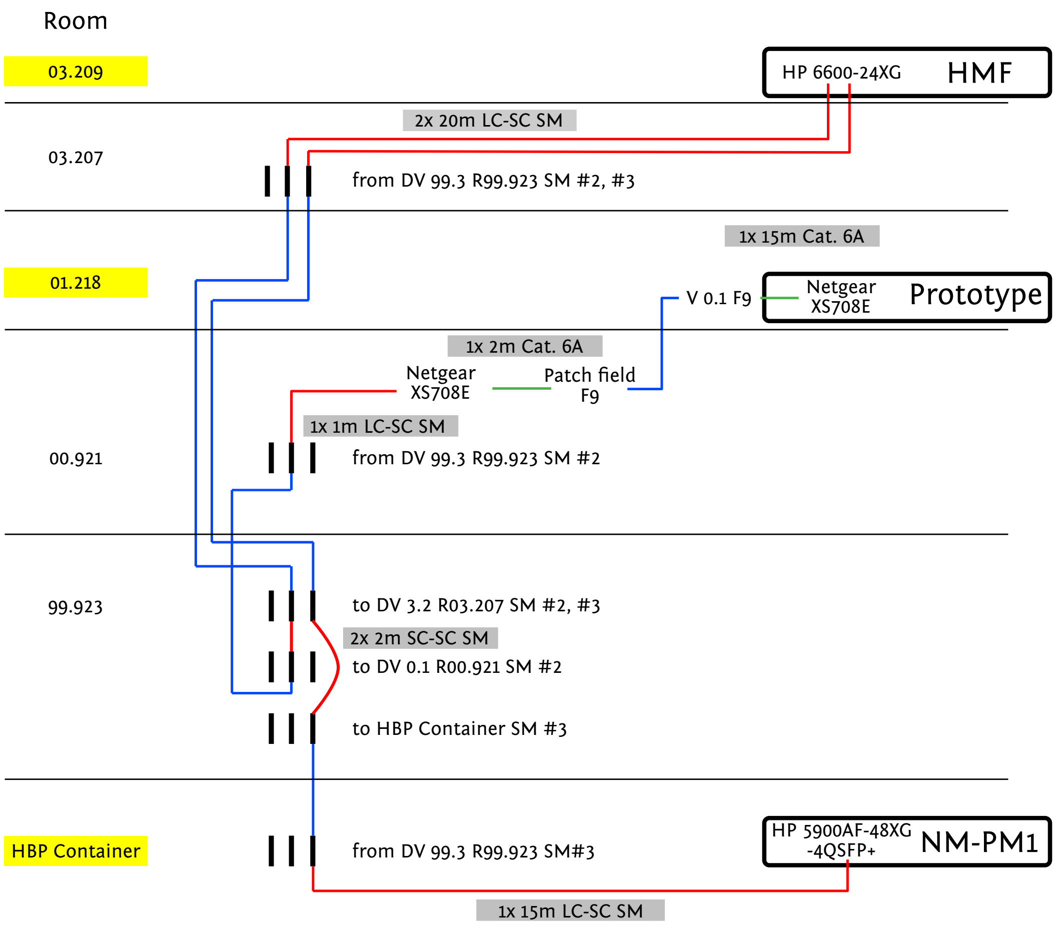


Figure A.3.: Schematic showing the KIP and ENI buildings with the interconnection details.

A.5. Miscellaneous Measurements

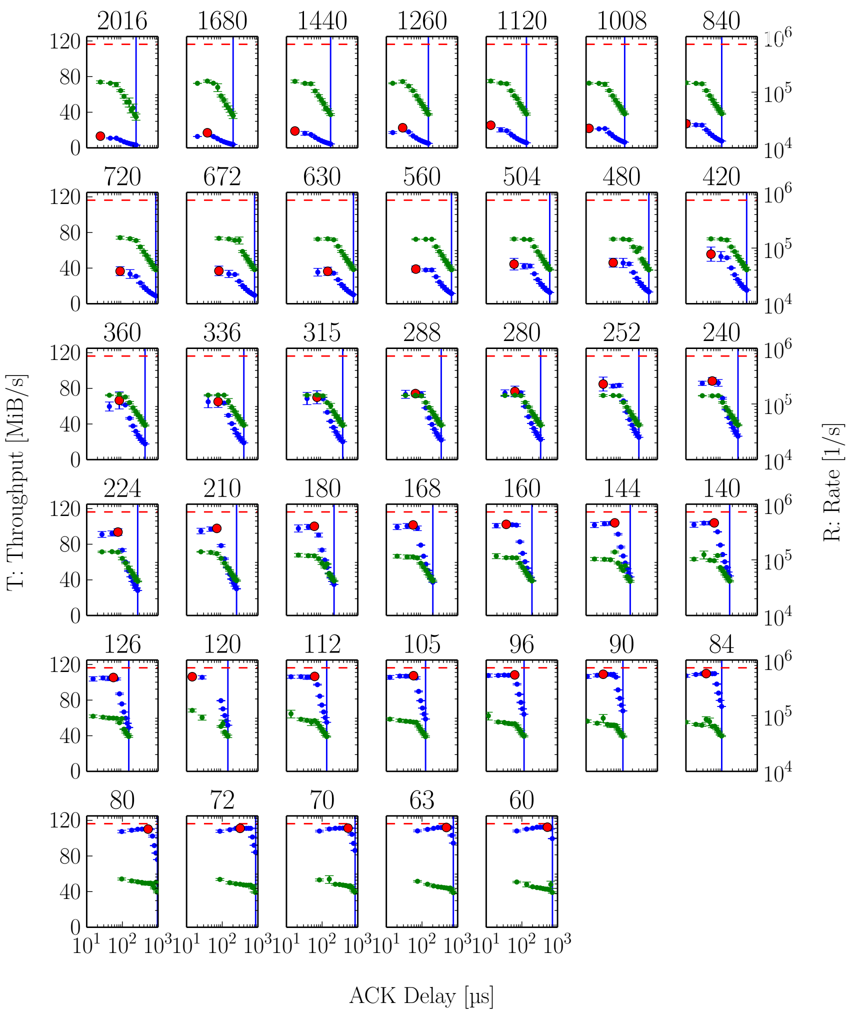


Figure A.4.: Host to FPGA throughput measurement utilizing only the down connection (half-duplex). The payload size is varied while sizeof(packet payload) × sizeof(window) = const using the standard *MTU* = 1500 B. Throughput is plotted against ACK timings. The marked points are used for figure 2.17.

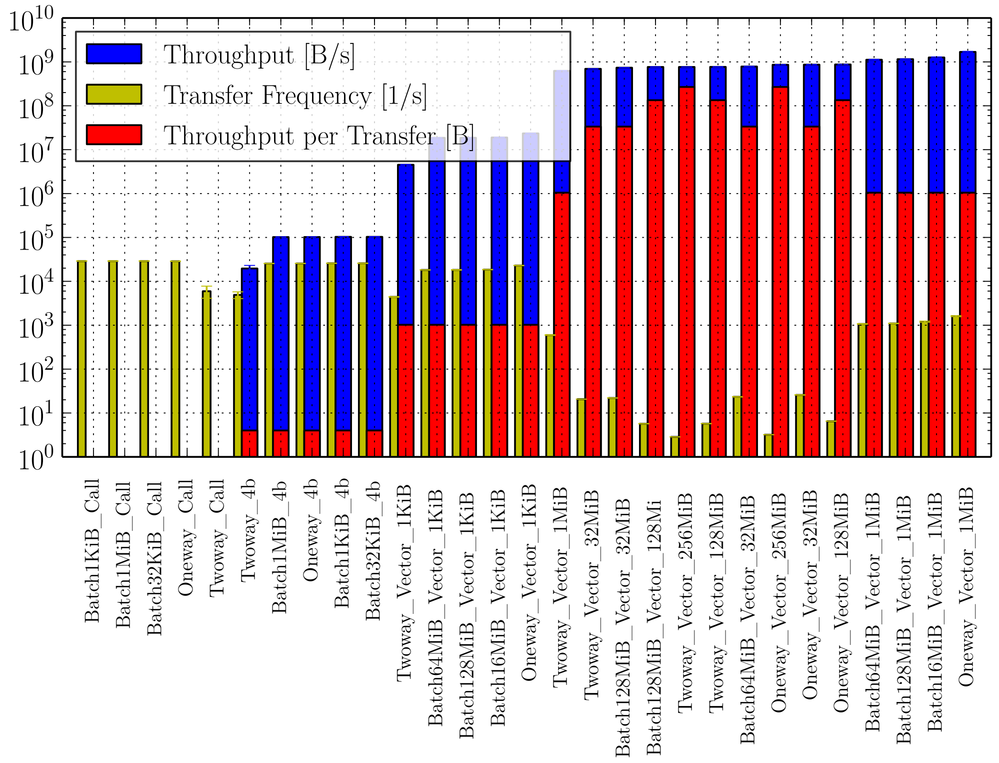


Figure A.5.: See figure 2.27 for a description of the plot; in contrast to the mentioned plot, this plot uses the RCF-internal serialization framework (`SF`). The plot is based on measurements performed by Kai Husmann under the supervision of the author; for details see Husmann [2012].

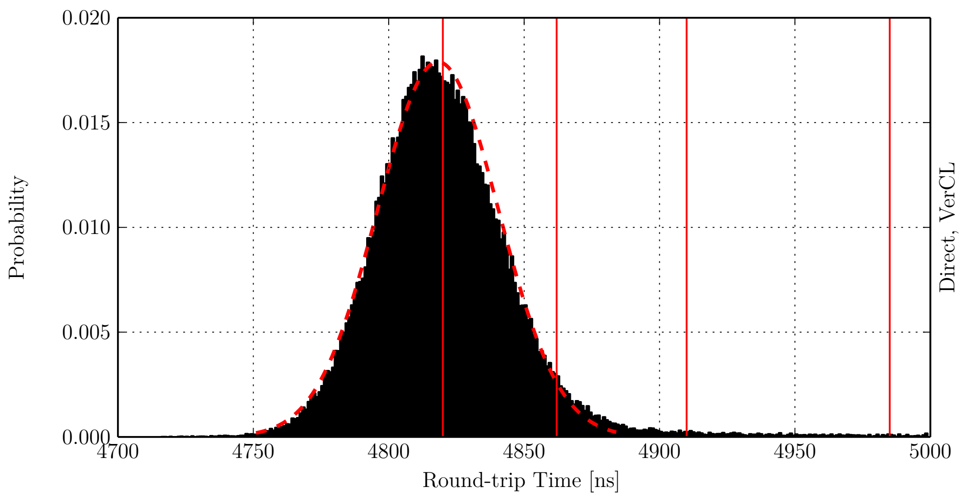


Figure A.6.: RTT measured between two directly connected NM-PM1 compute nodes. In contrast to the plots shown in figure 3.7, the ordinate is linear. An attempt to fit a Gaussian or normal function is shown as red dashed line (fit constrained to 4750 ns to 4885 ns). Compared to the other plots which used a logarithmic scale, the skewness is more difficult to recognize.

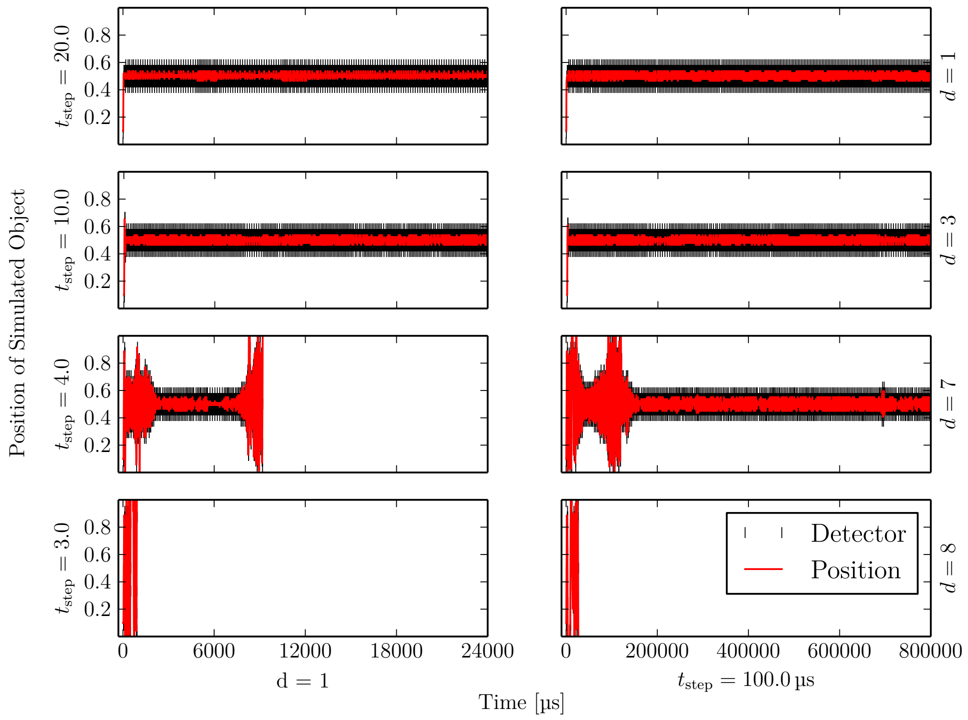


Figure A.7.: Software-based closed-loop experiment. Same data as figure 3.16 but time-axis zoomed out.

Acknowledgments

Ich danke:

Prof. Ulrich Brüning für die Übernahme der Zweitkorrektur.

Prof. Karlheinz Meier und Johannes Schemmel für die Herausforderungen, die ich bei den Visionären angehen durfte.

Allen Softies und allen anderen Kollaborateuren für die schöne und erfolgreiche Zusammenarbeit. Das waren und sind insbesondere Bernhard Kaplan, Daniel Brüderle, Olivier Jolly, Andreas Grübl, Dan Husmann, Johannes Bill, Mihai Petrovici, Matthias Hock, Moritz Schilling, Andreas Hartel, Sebastian Jeltsch, Maurice Güttler, Paul Müller, Thomas Pfeil, Vitali Karasenko, Oliver Breitwieser, Kai Husmann, Sven Schrader, Johannes Partzsch und Christoph Koke.

Meiner Familie,

allen Freunden und

Katharina.

It works! gggqGZZ

Statement of Originality (Erklärung):

I certify that this thesis, and the research to which it refers, are the product of my own work. Any ideas or quotations from the work of other people, published or otherwise, are fully acknowledged in accordance with the standard referencing practices of the discipline.

Ich versichere, dass ich diese Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Heidelberg, October 5, 2014

....
(signature)