Department of Physics and Astronomy

Ruprecht-Karls-Universität Heidelberg

Master's Thesis

in Physics

submitted by

Christian Mauch

born in Stuttgart, Germany

Commissioning of a Neuromorphic Computing Platform

This Master's Thesis has been carried out by Christian Mauch at the
Kirchhoff Institute for Physics
Ruprecht-Karls-Universität Heidelberg
under the supervision of
Prof. Dr. Karlheinz Meier

Commissioning of a Neuromorphic Computing Platform

This thesis presents work performed on the BrainScaleS neuromorphic hardware system enabling its transition from a lab setup to a usable computing platform. A stable communication between the controlling host computer and the system was achieved through the development of a verification test demonstrating uninterrupted transmissions of at least 150 TiB of data. Several new software tools were implemented which improved usability through encapsulation and failure cleanup, increased usage flexibility of single-chip test setups, enabled remote firmware updating and automated the system startup routine. An automated BrainScaleS wafer module test was developed which permitted advancements in the assembly process. It allowed for the first investigation of long-term stability of wafer connections. The monitoring infrastructure was enhanced allowing for automated readout of many components which lead to identification of an overheating issue. The implemented visualization tool assisted in resolving the issue by the system hardware designers. An integrated function call and I/O profiling tool was developed to analyze the performance of communication and software as high performance is vital to utilize the hardware speedup. The improvements of the introduction of a sliding-window transport protocol as well as effects of analog readout compression were evaluated. The digital spike communication and system configuration was parallelized, resulting in a major speedup for multi-reticle experiments. The results of the implemented changes and performance analysis are discussed and further improvements are suggested.

Inbetriebnahme einer Neuromorphen Computing Plattform

Dieses Werk befasst sich mit Arbeiten, die am neuromorphen BrainScales Hardwaresystem durchgeführt wurden um den Wechsel von einem Laborsystem zu einer funktionsfähigen Computing-Plattform zu ermöglichen. Eine stabile Kommunikation zwischen Kontrollrechner und dem System wurde durch Entwicklung eines Verifikationstests erreicht, welcher die fehlerfreie Übertragung von mehr als 150 TiB demonstriert. Mehrere Softwarewerkzeuge wurden implementiert und führten zu einer Verbesserung der Benutzbarkeit, flexiblerer Nutzungsmöglichkeiten der Einzelchiptestsysteme, der Inbetriebnahme der Firmware-Fernaktualisierung sowie einer automatisierten Systemstartroutine. Ein automatisierter Test der BrainScaleS Wafermodule wurde entwickelt, der zu Verbesserungen des Montageprotokolls führte. Dieser ermöglichte die erstmalige Untersuchung der Langzeitstabilität von Waferverbindungen. Die Erweiterung der Systemüberwachung erlaubt das Auslesen zahlreicher Komponenten. Dadurch konnte ein Überhitzungsproblem identifiziert, und durch die Systemhardwareentwickler behoben werden. Es wurde ein Profilingwerkzeug zur integrierten Leistungsanalyse von Kommunikation und Software entwickelt. Die durch Einführung eines Sliding-Window-Transportprotokolls sowie einer Analogdatenauslesekompression erziehlten Verbesserungen wurden untersucht. Die digitale Spike-Pulsübertragung sowie Systemkonfiguration wurden parallelisiert, was zu einer wesentlichen Beschleunigung von Multi-Retikel-Experimenten führte. Die Resultate der durchgeführten Änderungen und deren Leistungsanalyse werden diskutiert und weitere Verbesserungsvorschläge werden dargebracht.

Contents

1. Introduction

Advancements in computation technology and rising interest in applicability for information processing are major factors in the increased effort and progress in neuroscience in the past few decades. Neuroscience aims to understand the brain which is exceedingly well adapted in processing sparse and ambiguous data due to its inherent ability to learn, remember and modify learnt knowledge. This capability emerges from the interaction between neurons in vast dynamically changing networks. These abilities lend themselves well to solve problems in fields like image analysis or robotics.

The desire to scientifically derive descriptions of brain functionality spawned various neuron models and numerical network formulations. These are traditionally simulated on clusters or even supercomputers. It was possible to run large scale spiking single point neuronal network simulations using the common NEST simulator [*Diesmann and Gewaltig*, 2002] on the super computers JUQUEEN in Jülich and K in Kobe with $0.57/1.73 \times 10^9$ neurons and $6.4/1.4 \times 10^{12}$ synapses respectively [*JSC*, 2015; *Riken*, 2013]. For comparison, the human brain consists of about 10^{11} neurons and 10^{14} synapses.

These are remarkable numbers, but they also come at huge costs. The simulation of 1 biological second takes 40 minutes in wall-clock time for the simulation run on K, a slowdown factor of 2400. This is caused by expensive numerical calculations of the differential equations describing the dynamical behaviour of the membrane potential of the single point neurons. The K computer consumes 12.7 MW [*Riken*, 2016], resulting in a power consumption of 21.8 mW for each synapse in biological time domain.

An alternative to software simulation was introduced by *Mead and Mahowald* [1988]. Instead of calculating the numerically expensive differential equations in simulations, the neuronal behaviour is emulated by a physical representations of the model, usually in silicon. *Mead* also established the term *neuromorphic computing* [*Mead*, 1990] which today is associated with analog as well as digital and hybrid hardware that implements neuronal networks. Hardware designers can choose, to a certain degree, the dimensions of the components implementing the neurons and synapses, allowing systems to operate in real-time or at higher speedup factors compared to the biological time domain. Furthermore, the speedup is independent from the size of the network.

The NM-PM1 system, developed by the Electronic Vision(s) group at the Heidelberg University in cooperation with the Technische Universität Dresden within the Human Brain Project, employs this physical modelling technique. It will consist of 20 neuromorphic BrainScaleS wafer modules that emulate the Adaptive Exponential Integrate-and-Fire (AdEx) neuron model with a speedup factor of around 10^4 compared to biology. The 3.9×10^6 neurons and 8.8×10^8 synapses of the entire system consume 24 kW power, including the digital communication layer. The resulting power consumption of 3 nW per synapse yields a reduced power consumption of over 6 orders of magnitude compared

to traditional computing (21.8 mW). The speedup offers an ideal setup for long-term plasticity experiments as the evolution of a neuronal network over a timespan of one year can be emulated in less than an hour, whereas in simulation this would take over two millennia.

To leverage the advantages of physical modelling some challenges have to be faced. The high speedup demands a sophisticated communication network as the system cannot be throttled or paused like software simulators. An average firing rate of 1 Hz yields nearly 2×10^9 spike events per second on one wafer module. The analog nature of the system introduces noise that leads to nondeterministic behaviour. Another constraint is the rigid implementation of the neuron model. Although, some parameters can be varied in a limited range, the fundamental differential equations cannot.

Nonetheless, neuromorphic hardware is promising as a fast alternative to software simulation as it enables the exploration of time-scales which are several orders of magnitude larger than in existing network simulations. This unique property not only allows for faster and interactive modeling but also renders possible long-term plasticity experiments which involve sensor-motor loops that are typically found in robotics. Finally, it could also reveal new computational paradigms.

The next big step is to evolve this system to an emulation framework for the neuroscience community. The system has to run stably and robustly, react reliably to hard- and software faults, provide a user-friendly interface and achieve all that with a limited number of support personnel.

The main goal of this thesis was to drive forth the commissioning of the neuromorphic computing platform by engaging the aforementioned challenges.

Thesis Outline

To give an overview of the neuromorphic computing platform, its hardware components and software framework are introduced in chapter 2. Commissioning work towards a stable and robust operation of the neuromorphic computing platform is illustrated in chapter 3. This includes stability tests of communication modules, implementation of automated wafer module connectivity tests and system monitoring among others. Chapter 4 presents implemented changes to the communication and hardware abstraction software stack and an analysis of the resulting performance improvements. Finally, chapter 5 discusses the results acquired in this thesis discussion and gives an outlook for future goals.

2. The Neuromorphic Computing Platform

This chapter serves as a short introduction of the hard- and software framework. The first section gives an overview of the different hardware components while the second section presents the software to utilize said hardware. A detailed description of the platform is depicted in *HBP SP9 partners* [2014] which is the main source for this chapter.

2.1. NM-PM1

The Neuromorphic Physical Model version 1 (NM-PM1) represents the hardware side of the neuromorphic computing platform. The following section will give a bottom-up introduction to the hardware components relevant in the scope of this thesis.

2.1.1. HICANN Wafer

At the heart of the NM-PM1 lies the High-Input Count Analog Neuronal Network Chip (HICANN) [*Schemmel et al.*, 2008, 2010]. It is the successor of the Spikey chip [*Schemmel et al.*, 2006, 2007] developed in the FACETS (Fast Analog Computing with Emergent Transient States) project [*FACETS*, 2010]. Development of the HICANN started 2008 within FACETS, continued in the BrainScaleS project [*BrainScaleS*, 2012] and is carried on in the Human Brain Project [*Markram*, 2012].

One HICANN chip consists of 512 neuron circuits, 512×224 synapses and the required control and communication circuits. Synapses can be variably assigned to logical neurons. This allows for configuration between 512 *small* neurons with only 224 synapses each and up to 14×10^3 synapses for 8 *large* neurons.

The neurons of the HICANN emulate the Adaptive Exponential Integrate-and-Fire (AdEx) neuron model [*Gerstner and Brette*, 2009]. Neuron parameters like capacitances and conductances are represented by electrical components on a micro-meter scale which leads to a speedup factor of up to 10^4 compared to biological time domain. The various neuron and synapse parameters, e.g. membrane potential, leakage conductance or synaptic weight, are configurable which allows to model different neuron behaviour. The HICANN chip also provides plasticity mechanisms such as Short-term Plasticity (STP) and Spike Timing Dependent Plasticity (STDP).

The HICANN was designed within the focus on wafer-scale integration. Micro-chips are conventionally produced in manifold on a silicon wafer, then cut out, bonded and packaged into a casing. In wafer-scale integration the chips on the wafer are not cut but instead are directly connected. Only 8 HICANNs can simultaneously be fabricated in one photolithography step. Such a group of HICANNs is called a reticle. The post-processing step creates connections between neighboring reticles. One wafer contains 48

reticles which leads to a maximum number of over 196 thousand neurons and 44 million synapses.

HICANNs on a wafer are interconnected by a system of horizontal and vertical bus lanes, named Layer 1 (L1). Spike events between the continuous analog neurons and synapses are transfered as discretized digital events. The routing is done by a network of configurable crossbar switches. External communication to and from the HICANNs is analogously named Layer 2 (L2). The handling of off-wafer communication is covered in the follow-up subsection. The interface between L1 and L2 is the Digital Network Chip (DNC) merger, which is located in the Merger Tree (MTREE). The MTREE allows for a flexible allocation of the digitalized neuron output to the eight sending repeater which represent the L1 output of a HICANN chip.

2.1.2. BrainsScaleS Wafer Module

To access and control the vast amounts of data generated by a HICANN wafer requires a sophisticated readout and communication equipment. The unity of wafer, communication components, controlling devices and power supply constitutes a BrainScaleS wafer module, named after the project in which it was developed [*BrainScaleS*, 2012]. This subsection gives an overview of some components of a BrainScaleS wafer module, primarily focusing on communication and control devices.

Figure 2.1 illustrates the different communication channels between a wafer module and the host machine on which the user operates.

The interface for digital communication between HICANN and host is an Field-Programmable Gate Array (FPGA) which executes the required tasks. The FPGA is incorporated into a Printed Circuit Board (PCB) appropriately named the FPGA Communication PCB (FCP). It was designed by the collaborating team of the Technische Universität Dresden (TUD) [*Hartmann et al.*, 2010; *Scholze et al.*, 2011]. The FCP handles transmission of L2 packets to and from the wafer. There are two types of packets transfered between FCP and HICANN, spike events and HICANN configuration data. Spikes are sent without error-control whereas transmission of configuration packets is controlled by the custom made HICANN ARQ protocol (HICANNARQ). An Automatic Repeat Request (ARQ) protocol ensures transmission of a packet through resending if the receiving side did not acknowledge reception.

To conduct experiments at the accelerated operation speed of the system requires precise synchronisation of spike events and configuration commands. This is realized through buffering of spike trains in two 512 MiB memory modules named Playback and Trace for in- and out-going spikes respectively. This allows for spike release at clock cycle precision.

Hybrid operation of the system, i.e., interaction of the neuromorphic hardware with outside stimulus, requires unbuffered insertion of spikes and configuration commands. The SpiNNaker real time interface and the asynchronous HICANN communication allow for direct communication with the HICANNs.

Direct hardware access to FPGA and HICANN are provided by a Joint Test Action Group (JTAG) side channel. This channel is used for hardware control, e.g., setting

of system counters or clock frequencies, and readout of hardware values for debugging purpose.

The Analog Readout Module (AnaRM) enables readout of analog membrane voltages of HICANNs. It consists of two interconnected circuit boards. One board, named flyspi board, provides most of the functionalities. It contains a FPGA, an Analog-to-Digital Converter (ADC), 512 MiB memory and a Universal Serial Bus version 2.0 (USB 2.0) controller. The other board, named Analog Front End Board, provides a multiplexer for 8 input signals in order to enable a fast consecutive readout of these signals. Every reticle has two analog output signals, hence 12 AnaRM are required to access all analog signals of one wafer module. In the NM-PM1, 4 wafer modules will share 12 AnaRMs.

Control and monitoring of most components of the BrainScaleS wafer module are done by the Main System Control Unit (MaCU) which is implemented by a Raspberry PI. It is connected to the wafer module via a ribbon cable. Communication with module components is done via the Inter-Integrated Circuit Link (I2C) protocol.

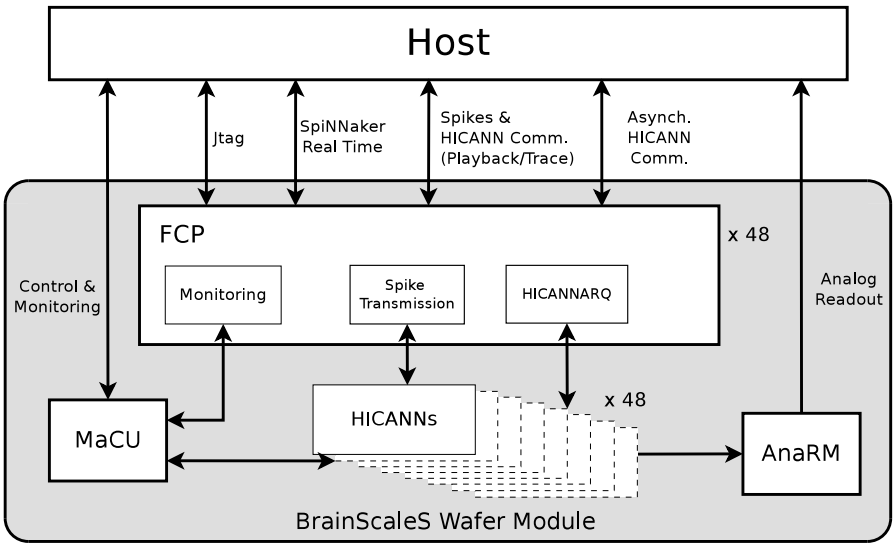


Figure 2.1.: Overview of communication channel between BrainScaleS wafer module and host machines. See text for explanation.

2.1.3. Cluster Architecture

The NM-PM1 will consist of 20 BrainScaleS wafer modules. A cluster of 20 compute server nodes, further on named host machines, with 10 Gbit Ethernet cards supplies the computational power and communication throughput for experiments. The 48 FCPs of one wafer module are connected to a 1 Gbit switch with a 10 Gbit backplane. The switches of wafer modules and the host machines are interconnected via 10 Gbit backbone switches with a backplane of 40 Gbit. A detailed description of cluster architecture is presented in *Müller* [2014]. Figure 2.2 shows an image of the NM-PM1.

Figure 2.2.: Image of the NM-PM1 system. Date: 2016-02-25 by B. Kindler

2.2. Software

A sophisticated software stack is required to operate the NM-PM1. Figure 2.3 gives an overview of the different repositories that constitute the framework to operate the neuromorphic hardware. This highly modular structure makes the software more scalable and maintainable. It can be split into three categories. *Python for the Hybrid Multiscale Facility (PyHMF)*, *Euter*, *Ester* and *Marocco* cover the software that manages the translation from neuron network models to the hardware structure. Hardware calibration and classification of defect HICANNs chips is handled by *calibtic*, *redman* and cake. *StHALbe* and further low level communication repositories handle the direct access to the hardware. The following subsection will give an brief overview of these three software compartments. Resource management of the hardware is handle by Simple Linux Utility for Resource Management (SLURM) [*LLNL et al.*, 2014]. An interface for external usage is provided by the HBP Neuromorphic Platfrom Interface (NMPI) *Davison* [2016].

A comprehensive list of repositories of all tools and tests developed in the course of this thesis ist presented in appendix A.1

2.2.1. User Software Stack

A widespread tool used by neuronal network modelers is PyNN [*Davison et al.*, 2008]. PyNN is a python package that defines and implements a unified Application Program-

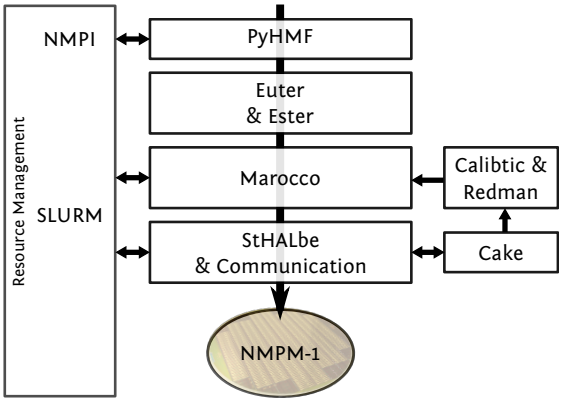


Figure 2.3.: Overview of the software framework for the NM-PM1. By E. Müller.

ming Interface (API) to different neuronal network simulators (NEURON [*Hines and Carnevale*, 2003], NEST [*Diesmann and Gewaltig*, 2002], Brian [*Brette and Goodman*, 2008]) and some neuromophic hardware systems (BrainScaleS, SpiNNaker [*Furber et al.*, 2012]). The software stack of the neuromorphic computing platform is designed to be operated with this interface, to allow modelers an easy usage of the hardware. *PyHMF* is an adapter between the PyNN API and the underlying software layers written in C++. *Euter* implements the biological description of the neuronal network, e.g. neuron parameters, connections and spike sources. This abstract neuronal network needs to be mapped onto the hardware system. This is implemented in *Marocco*. The software provided in *Ester* acquires the abstract network from *Euter* passes it to *Marocco* for translation. *Marocco* returns the translated network to *Ester* which then executes the subsequent software stack.

2.2.2. Calibration Software Stack

As hardware manufacturing entails imperfections and defects, it is necessary to circumvent those imperfections and cope with defects. The hardware was designed to allow calibration of various parameters. The task of the calibration is to create a lookup table for different varying hardware parameters. This is done through direct or indirect measurements of the system characteristics. In case of the neuron leakage potential, it is a simple static measurement while for other parameters it is much more difficult. Further information on calibration is found in *Schmidt* [2014]. *cake* represents the implementation of the calibration algorithms. An API for storage and access of gathered calibration data is provided by *calibtic*. Defect HICANNs are handled by white- and blacklisting implemented in *redman*.

2.2.3. Hardware Abstraction And Communication Software Stack

The *Hardware Abstraction Layer Backend (HALbe)* serves as the main interface between low level hardware formatting and user formatting. Its main feature is an abstract coordinate system to give the user an intuitive structure for various hardware elements. It also provides the implementation of backend functions, e.g. setting of synapse connections, write of spike trains into the Playback memory or readout of ADC data.

The *Stateful Hardware Abstraction Layer (StHAL)* utilizes the structures defined by *HALbe* to implement container that hold the state information of the various hardware components. It defines functionalities like the configuration sequence of HICANNs. *StHAL* can be viewed as an enhancement of *HALbe* hence the notation *stHALbe* used in fig. 2.3. Further information is provided in *Jeltsch* [2014].

The low level communication is again split up in modular repositories. Control and data transmission of the AnaRM are handled in the repository *vmodule* whereas the different communication channels to the FCP are implemented in *HICANN-systems*.

The transport protocol of the main communication channel between host machine and the FCP is the Host ARQ protocol (HostARQ), not to be mistaken with the HICANNARQ. It is a custom made Selective Repeat ARQ protocol. The sender continuously transmits a number of frames defined by a window size and resends them after a timeout. These frames are marked by ascending sequence numbers. The sender tracks the sequence number N of the last successfully transmitted frame. The receiver tracks the last sequence number i of received frames without discontinuance but still handles subsequent frames in the window. It acknowledges the sequence number i to the sender after an timeout. The sender then only retransmits the frame $N + i$ after the resend timeout. Upon reception of the resent frame, the receiver acknowledges a later frame in the window in case of a single drop.

Sending and acknowledge timeout are dynamically changed in the software implementation on the host to accommodate to the network capacity. Payload transmission and acknowledgment are combined in one packet if possible to reduce load and latency of transmission. The identification of the payload type is handled by a packet type field in the header of a HostARQ packet. The payload of a packet constitutes of up to 180 64 bit entries. HostARQ packets are encapsulated into UDP packets for transmission over Ethernet. For more information see *Müller* [2014].

3. Commissioning Of The Neuromorphic Computing Platform

The goal of the Neuromorphic Computer Platform is to provide a framework for neuroscience experiments to different high-level users, i.e., non-hardware-experts, from various fields. The user should not worry about technical details like powering of components. This is a huge transition from the lab setups developed in previous projects like FACETS and the beginning of BrainScaleS [*Schemmel et al.*, 2006, 2007, 2008, 2010; *Brüderle et al.*, 2011]. Such setups could only be handled by a few expert users. Another problem comes with complexity and the sheer number of components in the NM-PM1. Many components mean many possible error sources but the number of managing people is limited. To ensure a smooth operation of the system the following requirements must be met:

- Stable and robust operation
- User friendly software
- System control, i.e., automated powering of hardware components
- Automated testing of hard- and software
- Monitoring and archiving system operation parameters

This chapter presents the work that has been done towards these goals.

3.1. Stable And Robust Communication

For a highly interconnected network like the BrainScaleS system a stable and robust communication is essential, whit stability meaning a long-term operation, i.e., several days, and robustness implying handling of damaged or dropped packets. The implementation of stable communication was an extensive effort of many people over several years. Protocol development started in 2008, when it was designed to serve the Multi-Spikey backplane system [*Philipp*, 2008; *Schilling*, 2010; *Müller*, 2008]. Later modifications targeted the Vertical Setups which provided access to the first HICANN test chips, the HICANN wafer lab systems [*Müller*, 2014; *Karasenko*, 2011, 2014] and finally, the BrainScaleS wafer modules.

3.1.1. HostARQ Test Suite

At the beginning of this master's thesis there were still problems with the main communication link, called HostARQ, between the host computer and the FCP boards. To track down the sources of these problems a test suite for the HostARQ was implemented. To isolate receiving and sending problems three new packet types were introduced. A FLUSH type which is dropped by the FCP after receiving, a LOOPBACK type which gets looped back via shortcutting the HICANNARQ in- and output and a DUMMY type generated by the FCP which holds ascending numbers. FLUSH packets have the goal to test the case of only the host sending packets with data and the FCP only acknowledging, DUMMY packets test the opposite case and LOOPBACK packets allow testing of data corruption and dropping. The code implementation for the FPGA of the FCP was done by V. Karasenko, the test on host side was implemented by the author.

The test program on the host machine can send FLUSH and LOOPBACK packets in an arbitrary ratio and either activate or deactivate sending of DUMMY data from the FCP. FLUSH packets hold a constant number as payload whereas LOOPBACK packets are filled with ascending numbers. The ascending behavior is checked upon receiving the looped back packets. This check also happens for received DUMMY packets.

Another feature of the test are varying packet frame size. It is possible to send packets with a constant, random, ascending/descending and specifically selectable order of frame sizes. The values for constant and the intervals for random and ascending/descending can be chosen between 1 and the maximum frame size of 180.

3.1.2. Stability And Long-Term Tests

Through excessive testing it was possible to determine the sources of bugs that corrupted the transport protocol. There is one known bug in communication remaining at the time of writing this thesis. The FCP stops transmitting its Internet Protocol version 4 (IPv4)-address to the host, i.e., it stops responding to Address Resolution Protocol (ARP) requests, after a varying amount of transfered packets. This can be triggered especially by sending only FLUSH packets so that the FCP only sends acknowledge packets. This bug only occurs after at large amount of transfered packets. It is not trivial to investigate as 1 simulated millisecond corresponds to about 1 minute in real time. The collaborating group at the TUD which developed the FCP is working on this issue.

Long-term stability of the HostARQ was investigated, despite the aforementioned bug. 20 tests were simultaneously run on 20 FCPs on one wafer module. This amount of FCPs was chosen to investigate the behaviour under full network bandwidth load. Each test was set to transmit 10^{12} LOOPBACK packets. This was repeated 3 times. LOOPBACK was chosen as it best emulates real experiment behavior. The number of received LOOPBACK packets until the first commutation stall in a test was taken as a lower bound for stability. This test resulted in stable communication for at least 5×10^{10} LOOPBACK packets or 373 GiB of data. Further testing was halted for the time being as it was stable enough for all experiments conducted in the group and a fix of the ARP

bug was anticipated.

The next step was to investigate stable communication with the HICANN. This was done analog to the HostARQ stability tests. Analog tests were performed to investigate the stability of HICANN communication. HICANN configuration readout commands were sent instead of LOOPBACK packets. These commands trigger the sending of configuration data from the HICANN to the host. Readout was done in sequence for each of the 8 HICANNs on a reticle. Long-term testing yielded a stable communication for at least 1×10^{10} configuration commands or 74 GiB of data. Tests were again halted with the same reasoning as above.

These tests were repeated as a system was available with even higher cooling capabilities than shown in section 3.5.2. Both test cases resulted in complete transmission of 10^{12} packets for each HICANN. This corresponds to over 150 TiB of transfered data in 1300 ± 10 minutes which yields a transfer rate of (1.91 ± 0.20) GiB/s. The high transfer rate is possible as each side of a wafer module is connected to a separate 10 Gbit switch. This observation suggests that the aforementioned ARP bug could be caused by temperature issues or the instability with insufficient cooling is a separate issue and requires further investigation.

3.2. User Friendly Software

This section presents changes done to the software stack to improve usability and user friendliness.

3.2.1. Automated HostARQ Daemon Startup

A short overview of the implementation of the HostARQ is given to better understand the changes done in this subsection. Figure 3.1 shows a diagram of the HostARQ daemon implementation. It depicts the communication channel between host and FCP and the interface between daemon and higher level software. The previous work flow was a permanently running daemon on the host for each FCP. This was changed to an automated startup and shutdown on experiment start and stop bringing the advantage that the high level user does not need to care whether the right daemon is running on the host machine that was allocated to him. Likewise does the managing personnel not have to worry if all daemons are running on the machines. Additional robustness for the user in chase of errors is achieved as each startup of the daemon triggers a reset of the HostARQ module in the FCP.

The stable and robust implementation of automated startup and shutdown is challenging. Automated startup was implemented by forking, i.e., duplicating, the process when communication is initialized. The child process starts the daemon and then waits for exiting of the parent process. The parent process waits for the child process to finish the initialization of the daemon and then continues with its previous task. Handling of shutdown, especially in case of errors and interrupt signals, is challenging. Multiple threads need to be exited, the shared memory file must be freed, dynamic objects must

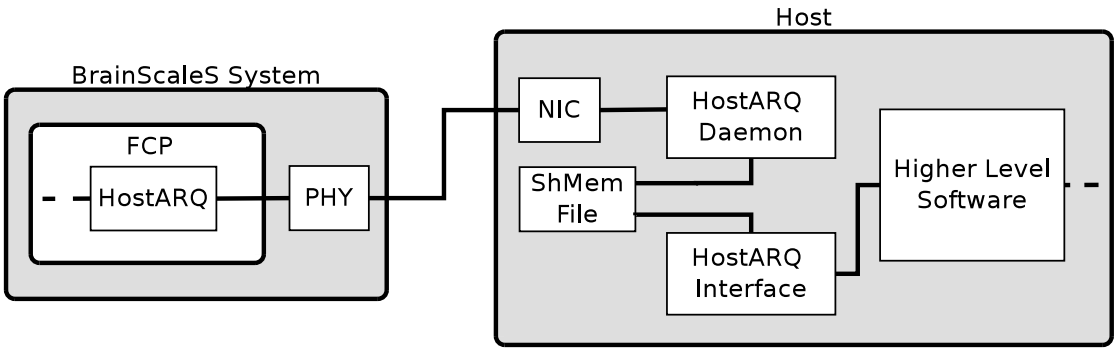


Figure 3.1.: Block diagram of HostARQ daemon implementation. Communication on the side of the BrainScalesS system is handled by a HostARQ module in the FPGA of the FCP. On the host side this is done by the HostARQ daemon. Incoming packets are moved to a receiving buffer in the shared memory file were they can be read out by higher level software via the HostARQ interface. Outgoing packets are pushed into a sending buffer by higher level software. The daemon then sends the packets to the FCP.

be deallocated and everything in the right sequence. This implementation was done in cooperation with E. Müller. An issue of the implementation via `fork()` is that it restricts memory allocation to be executed after communication initialization. A possible solution is to start a detached instance of the daemon after forking via the `execvp()` system call. Another remaining issue where the shared memory object is not cleaned up in case of `SIGKILL` signal requires a proper cleanup by the resource management.

3.2.2. Arbitrary HICANN Slot Arrangement API

An integral part for characterizing and analyzing prototype chips of new hardware revisions are test setups. They allow the testing of single or multiple chips with the same software that is used for the full wafer module. The setups have 4 slots where boards with bonded HICANNs can be inserted. One board can hold either a single or two HICANNs hence one system can have up to 8 HICANNs attached. This corresponds to one reticle on a full wafer.

The APIs between the different layers of the software stack have several parameters to support these test setups. The previous implementation of these APIs had one shortcoming, they only took the number of HICANNs and assumed that these were inserted in succession. This made it necessary to insert boards with two HICANNs if one wanted to use multiple slots and posed a problem with test chips of the new HICANNv4 revision as there were difficulties bonding 2 HICANNs on one board.

The APIs of the several software layers were changed to overcome this deficit. The software now holds the exact position of the HICANNs instead of just the amount of HICANNs.

3.3. System Control

The NM-PM1 has many components that need to be remotely controlled to allow a automated operation and fast maintenance. This section presents improvements to the firmware update routine and the implementation and subsequent extension of a control software for the FCPs.

3.3.1. Remote FCP Firmware Flashing

The fully equipped NM-PM1 will comprise over 960 FCPs. Their firmware is frequently adjusted due to design changes, new features and bug fixes. Therefore, a fast update procedure of the firmware is desirable. Previously, only a slim firmware for rudimentary testing was stored in the Erasable Programmable Read-Only Memory (EPROM) of the FCP. It was necessary to write the firmware directly into the FPGA via a JTAG link after each powerup of each FCP. As there are four JTAG connections on a wafer module, one for each edge, manual replugging of the JTAG cable was required.

The collaborating group from the TUD implemented a boot loader design for the FCP. It loads a firmware from the EPROM into the FPGA which makes it now possible to flash new firmware versions into the EPROM via Ethernet removing the need for manual replugging. Loading a new firmware file into the FCP requires the following steps. After powering the FCP one needs to set the wafer ID to finalize the initialization of the boot loader design. This enables the Ethernet connection of the FCP as the wafer ID defines the third octet of the FCP's IPv4-address. After the initialization is finished a timeout of currently 25 s in which the boot loader waits to receive a new firmware. The flash programming software needs to be started in this time window. The boot loader loads the firmware into the FPGA when the timeout expires. It is possible to skip the timeout by a Remote Procedure Call (RPC) command. Upgrading to the new boot loader design still needs to be done via JTAG.

A tool to update the firmware of the FCP of a wafer module with or without prior update of the boot design was implemented by the author. This enables automated updating of all FCP firmwares without manual replugging. The tool allows to update individual FCPs, entire edges or the complete wafer. Flashing of one firmware, 4.5 MiB, takes about 135 s which is caused by the slow write access to the EPROM. The current implementation of the flashing program from TUD is not yet parallelized. This leads to flashing times of ≈90 minutes for each wafer module. However, parallelization should be possible as the communication streams to the individual FCPs are independent and easily distinguishable by FCP's IP address. As the Transmission of the 48 firmware images, 48×4.5 MiB, in 2 minutes is two orders of magnitude below wire-speed, even a trivial implementation should be able to scale perfectly, i.e., the parallel version should take as much time as a single execution of the serial flashing tool.

3.3.2. FCP Control Software

The central control unit of a BrainScaleS wafer module is a Raspberry Pi, called MaCU (see section 2.1.2) which is able to communicate with most components on the wafer module via I2C. A new daemon to communicate with the FCPs was implemented by S. Hartmann. The author tested it and made additions to its functionality. The daemon accepts commands from remote hosts via an RPC interface. These commands include power up and down of FCPs, setting of wafer ID, powering of high speed links to HICANNs and readout of temperature among others. A characterization of all FCP state values is presented in *HBP SP9 partners* [2014, cap. I-6.3.2]. A control software for the host machines was implemented that utilizes the RPC interface of the daemon. Based on this software a tool to power FCPs and their corresponding reticles on and off was implemented. All three software tools are written in Python. The power up is done in the following sequence:

1. Power up FCP
2. Set wafer ID
3. Skip timeout of boot loader
4. Set wafer ID again
5. Power up corresponding reticle
6. Power up all HICANN high-speed links

The second setting of the wafer ID is required as the loading of the functional firmware into the FPGA resets all states. In case of a shut down it is sufficient to just power off the FCP and its corresponding reticle. As with the firmware updating tool, it is possible to select individual FCPs, edges or the entire wafer. Reticles are controlled by a different daemon implemented by M. Güttler. Issuing commands to this daemon is done by connecting and executing a script on the MaCU via Secure Shell (SSH). The long-term goal is to integrate all functionalities into one daemon.

3.4. Automated Wafer Testing

The most important part of commissioning is the actual assembly of the BrainScaleS wafer modules themselves. After assembly, the correct functionality of the wafer module needs to be verified. This section presents an automated test of digital connectivity to the wafer as well as long term tests to investigate potential changes in the wafer modules. Assembly is carried out by D. Husmann, M. Güttler and student assistants.

3.4.1. Digital Connectivity Test

A test script to verify digital connectivity to the reticles after complete assembly was implemented. The following steps are done for each individual reticle in parallel on the entire wafer:

1. Power up FCP and corresponding reticle
2. Check for correct JTAG ID of HICANNs and FCP
3. High-Speed link initialization
4. 100 high-speed link test transmissions (verified via JTAG)
5. 10^9 HICANN configuration readouts via HostARQ and HICANNARQ over the high-speed links
6. Power down FCP and reticle

If one step fails the test is repeated up to five times. If the errors still occurs the reticle is marked as no-usable with the respective error status. Figure 3.2 shows the visualization of an exemplary test run. Reticles marked in green have functional digital connectivity. Test repetition is marked in color intensity, i.e., the more repetitions were required the less vibrant is the color. This can be seen for reticles 1, 3, 4 and 5. Errors are categorized into five fail states.

JTagID Error The JTAG ID test returned either random IDs or all IDs are 0.

Partial JTagChain The JTAG ID test returned valid IDs for some devices and random or 0 IDs for the rest.

HS Init Fail Either High-Speed link initialization or the subsequent 100 transmission test via JTAG failed for one HICANN. Failure in readout of 10^9 HICANN configuration packets is also marked with this state for legacy reasons.

Ethernet Problems There is no connection possible to the FCP via Ethernet. This mostly occurs due to not properly connected Ethernet cables.

Untested reticle Power up of FCP or reticle failed.

If either of the last two fail states occurs after assembly and the reasons are not external, then the corresponding FCP is exchanged. The teal marking of "No HS Link on HW" is always present for the two reticles in the center of the wafer. These reticles have no high-speed connection by design.

One problem encountered through repeated testing is that the high-speed initialization appears to be unstable. For some HICANNs it fails, seemingly random, and with no visible pattern. Due to this observation, the testing protocol was extended to include several repetitions. The random failing high-speed initializations impede a clear definition of what constitutes a functional reticle. A straight forward solution is to repeat the test several times and then declare a reticle no-usable if it fails more than a certain percentage of test runs. It is planned to extend the test by loopback of digital spike data to verify correct DNC merger behaviour. An approach to identify the cause of

the unstable high speed initializations would be to log the initialization results for all experiments run on the NM-PM1.

Preliminary wafer test statistics are presented concluding this subsection. In total, ten BrainScaleS wafer modules were tested, not including two systems with old assembly protocol. 398 of 460 possible reticles have functional digital communication, corresponding to a success rate of 86.5%. There is one system with all reticles functional while the two systems with the most errors have 34 functional reticles.

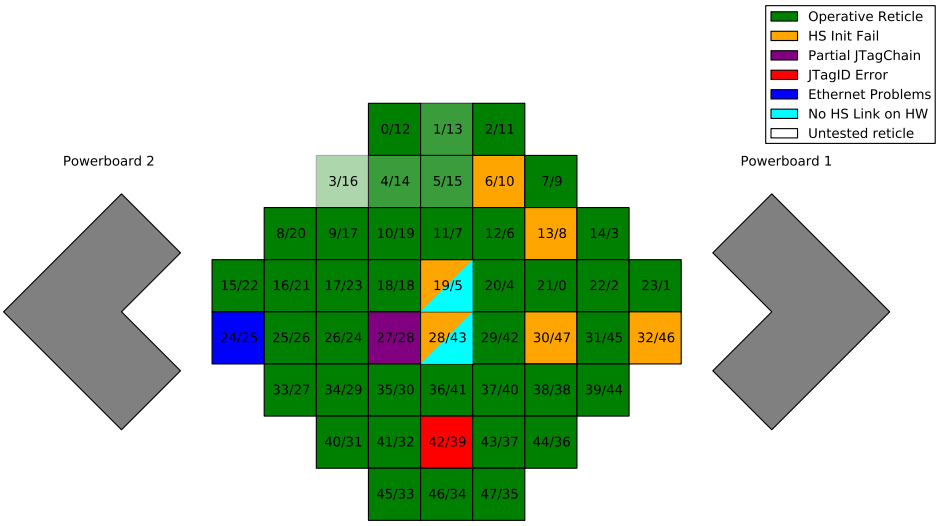


Figure 3.2.: Visualization of an exemplary wafer digital connectivity test. Numbers in reticles represent the reticle number and their respective FCP number. Powerboards 1 and 2 are shown for better orientation.

3.4.2. Investigation Of Long Term Change

It was not yet investigated if the elastomer connections between wafer and Wafer Module Main PCB (MainPCB) undergo deformation or other changes over time. Automated weekly tests were implemented to examine possible long-term changes in connectivity to the wafer. The automation was done with *Jenkins* [*Kawaguchi*, 2016]. Figure 3.3 shows wafer test results for two different BrainScaleS wafer modules at different points in time. The module shown in (a) and (b) was one of the first assembled modules with an older iteration of the assembly protocol. There were difficulties with even pressure distribution on the wafer boards. This was improved in later assemblies by the system hardware designers. (c) and (d) present test results of a later assembled a wafer module. Comparing the test result of (a) and (b) two months later reveals a drastic change. Many reticles that previously had correct connections show JTAG ID errors. Test results changing from failing high-speed initialisation to partial JTAG ID errors is also a worsening as JTAG communication is more stable and less complex than the high-speed links to the

HICANNs. However, looking at the test results (c) and (d) no direct change can be observed after 6 months of operation. The different high-speed initialization errors are due to randomness, as stated before. The wafer module presented in (c) and (d) is, unfortunately, the only system with improved assembly that could be tested over a long period of time therefore long-term changes cannot be completely ruled out in general.

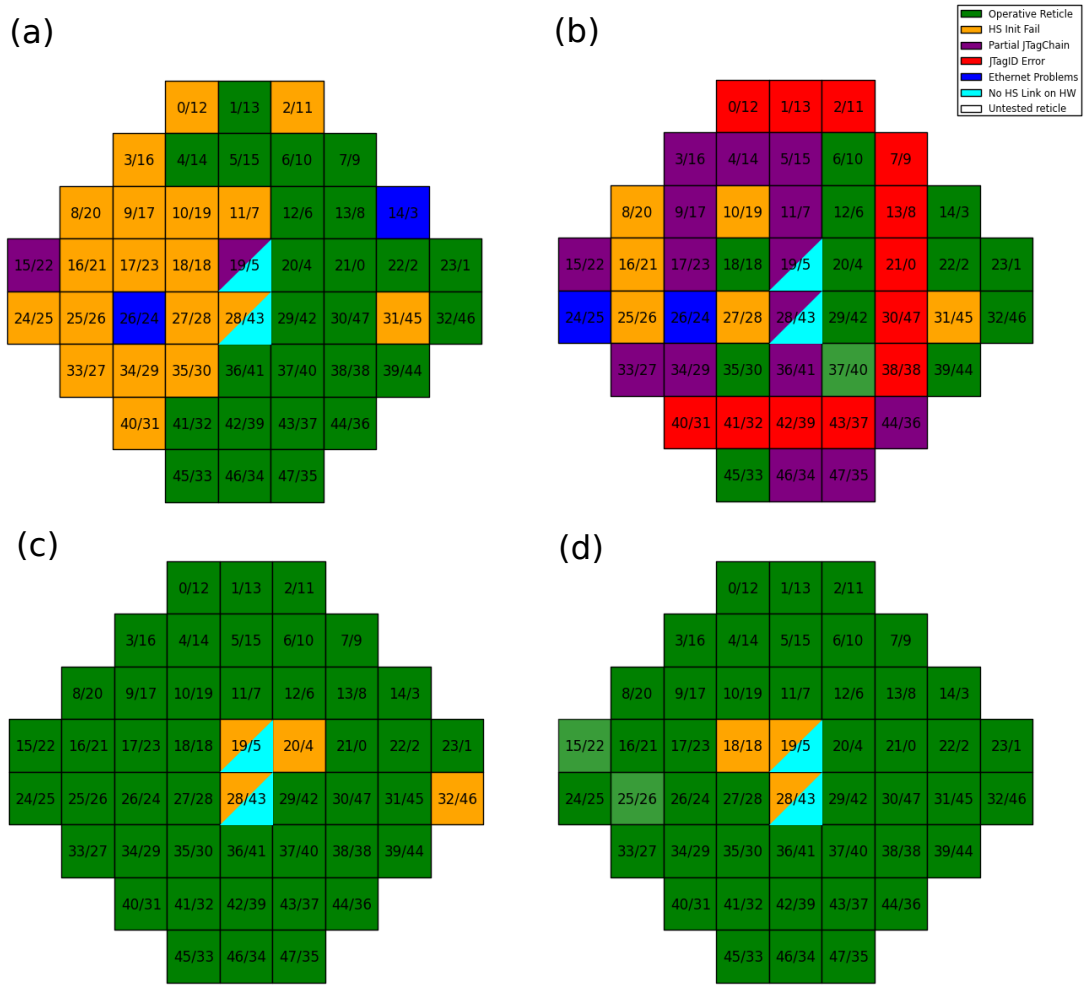


Figure 3.3.: Long-term change in wafer connectivity. (a-b) and (c-d) show test results of wafer modules assembled with old and new protocol respectively. Test dates: (a) 2015-09-02 (b) 2015-11-02 (c) 2015-08-14 (d) 2016-02-16

3.5. Monitoring

For a smooth and robust operation of the NM-PM1, it is important to have direct access to all relevant system state values, i.e., voltages, currents, temperatures or number of

active components. This section presents enhancements performed to the already existing monitoring infrastructure. Additionally, an overheating issue of the first BrainScaleS wafer modules is described and its subsequent solution is presented.

3.5.1. Access And Archiving

Several components of the BrainScaleS wafer module allow for readout of state values. Readout is done by the MaCU. Values are periodically read out by daemons via I2C data bus and stored in files. This reduces the work-load of the MaCUs as they do not have to trigger a readout for each external request. The author added the automated readout of FCP information into the I2C daemon, as explained in section 3.3.2.

Another important point is archiving the measured system information. It is crucial for analysis, e.g. power consumption or usage statistics, but also for determining causes of errors. The system information on the MaCUs is periodically read out by a remote host that stores this information in a central database. RRDtool was chosen as the database type as it stores data in a way that it gets coarser the older it is. This has the advantage that the database does not grow in time.

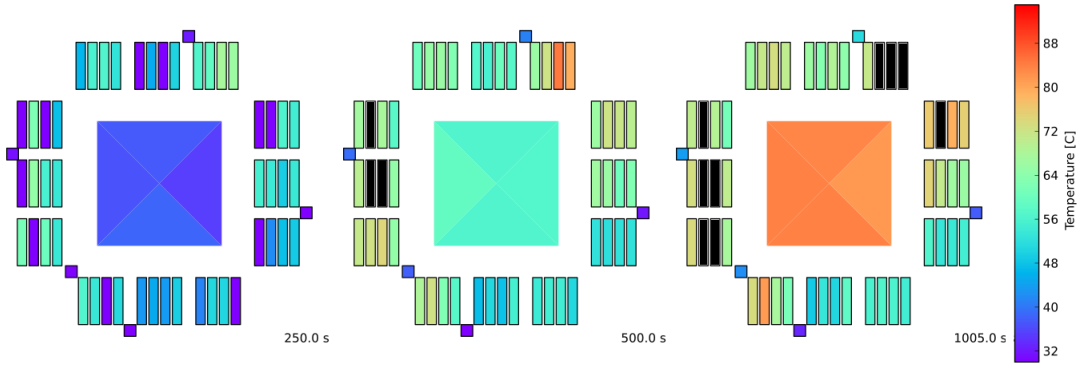
The automated readout and storage of data is done with the software Ganglia. It also provides a graphical web interface for easy access of the archived data of which an example graph can be seen in fig. 3.5.

3.5.2. Overheating Issues

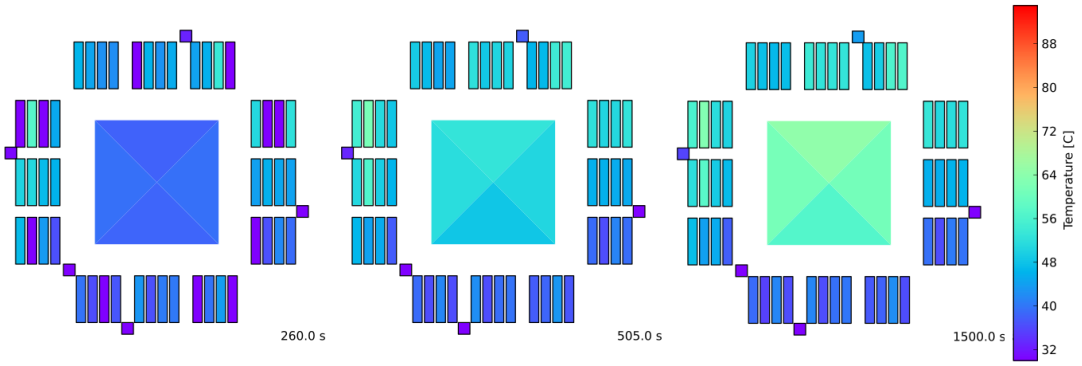
When first wafer wide tests and experiments were conducted, a sudden shutdown of specifically located FCP modules was noticed. One possible explanation for this is an automatic shutdown due to overheating. This was the primary reason for implementing automated readout of FCPs state values as described in the previous subsection. The author additionally implemented a visualisation tool for FCP, wafer and MainPCB temperatures. Figure 3.4a presents a time development of temperatures on a BrainScaleS wafer module with the original cooling setup. It consists of two rack drawers which each contain nine 120 mm fans, below and above the wafer module. The system draws ≈1.2 kW electrical power with all FCPs and reticles powered on. The wafer module was in idle state during measurement, i.e., no further experiments were running. One can see FCPs on the sides already shutting down after 8 minutes of uptime. After another 8 minutes more FCPs shut down but a bigger problem is the high silicone wafer temperature of over 80 °C and after more time the wafer heats up to nearly 100 °C. The desired working temperature of the wafer is at approximately 50 °C [*Schemmel*, 2016]. Especially the FCPs on the sides overheat due to insufficient air flow.

This cooling issue was tackled by D. Husmann and M. Güttler. A mount for a battery of 10 high-pressure axial fans was designed. Figure 3.4b shows the temperature time course of a BrainScaleS wafer module equipped with the new cooling system. One can see the temperature of all FCPs not exceeding 60 °C even after 25 minutes and plateauing at ≈65 °C with the cooling fans only running at half capacity during this measurement. At higher speeds the wafer can be kept at ≈50 °C, as can bee seen in fig. 3.5. The fans

were not ran at these speeds due to resonance problems which are resolved in a later iteration of the cooling system design. The fans draw ≈200 W at maximum speed for each wafer module, However they will probably not need to run at full capacity.



(a) Old insufficient cooling.



(b) New improved cooling.

Figure 3.4.: Temperature time development of BrainScaleS module for old and new cooling. Wafer temperature in center, surrounding rectangles represent FCP temperature and small squares represent main MainPCB temperature. Purple FCPs at 250 s are not yet powered up. Black marked FCPs shut down due to overheating.

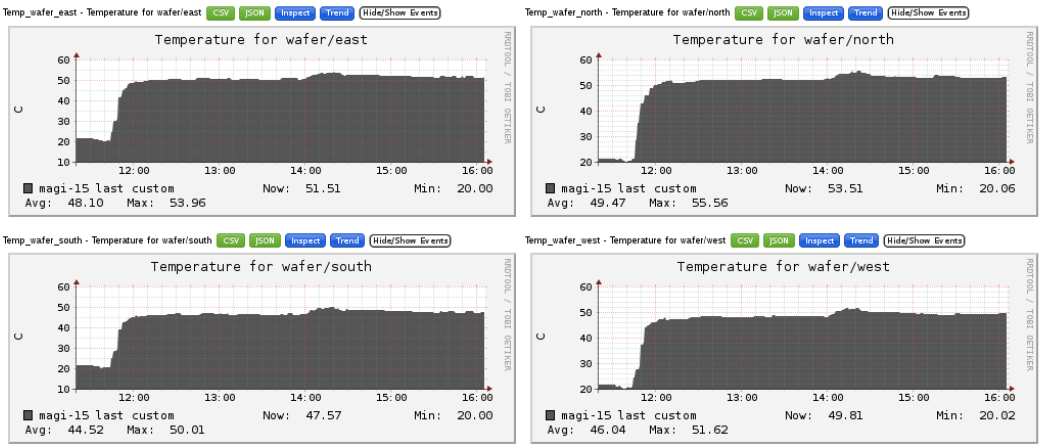


Figure 3.5.: Time development of wafer temperature taken with Ganglia.

4. Performance Improvements

With sufficiently stable and robust BrainScaleS wafer modules in place, improving the connection speed to fully facilitate the high speedup of the system became the next focus. In this chapter the improvements that came with implementation of the ARQ protocol (section 4.2) as well as improvements that were done to ADC Data transmission (section 4.3) and digital spike transmission (section 4.4) are presented.

4.1. Performance Profiling Tool

To analyze performance issues and verify improvements in the low-level software and communication stack (see section 2.2.3) a dedicated profiling tool was developed. The goal of this tool is to analyze which part of the software or hardware is active and thereby find potential bottlenecks that limit the overall speed. Hence, it should have a clean and direct visualization of when backend functions are running and, at the same time, record corresponding I/O traffic. Though, this should not impede the execution speed of the examined program but still yield precise data. It can analyze any test and tool in the *stHALbe* software stack with only little modifications to the main program call.

The tool is split in two parts where the first part is a run-time logger that simultaneously tracks function calls, Ethernet and USB 2.0 traffic as the program executes. Traffic is acquired by summation of payload sizes in the low level send and receive functions of *HICANN-system* (Ethernet) and *vmodule* (USB 2.0) repositories. The second part is a visualization of the acquired data. It is important to note that this is not intended to be a full-fledged code profiler as there already exist many good options like the Linux *perf tools* [*Weaver*, 2013]. Additional insights arise from the visualization of the interaction between the BrainScaleS system and the software stack.

All following sections contain visualizations which are created using this tool. Figure 4.1 shows an profile run of a short calibration step which only serves to illustrate the features of the visualization, hence the data is irrelevant. Panel (1) and (2) give a direct comparison between Ethernet and USB 2.0 traffic and run-time of backend functions. Their names, total run-times and portion of the entire program run-time are displayed in panel (3). The total traffic and average transfer rate are shown in panel (4). Panel (5) presents a histogram of accumulated function run-times. It is possible to compare profiles in the same visualization, e.g. fig. 4.3. The profiles then also share the same time-axis and run-times are directly comparable in the bar graph.

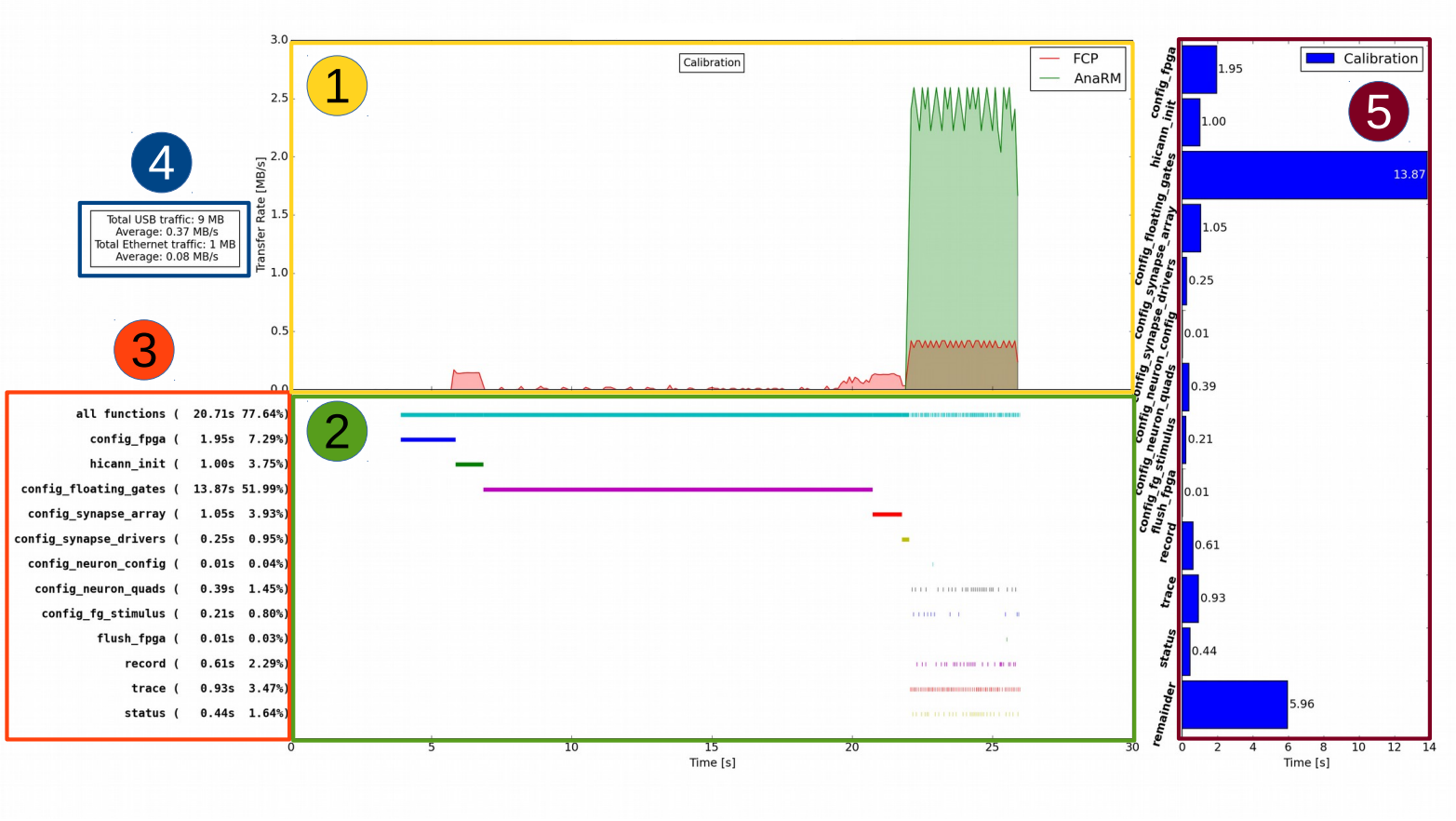


Figure 4.1.: Visualization of the performance profiling tool. (1) shows Ethernet traffic over the FCP and USB 2.0 traffic over the AnaRM respectively. (2) is a time diagram of the called functions displayed in (3) with total run-time of the function and its portion of the entire program run-time. (1) and (2) share the same time-axis. The total traffic and average transfer rate are shown in (4). (5) presents a histogram of accumulated function run-times.

4.2. Improvements Through HostARQ

As mentioned in section 3.1, the implementation of the HostARQ protocol in multiple variants and target FPGA platforms was a major effort of many people in the course of several years. This section presents different measurements that were conducted to analyze performance improvements in the BrainScaleS wafer module due to the implementation of the HostARQ protocol.

4.2.1. HICANNARQ Communication Throughput

One important goal of implementing the HostARQ protocol was communication with HICANNs at wire-speed.

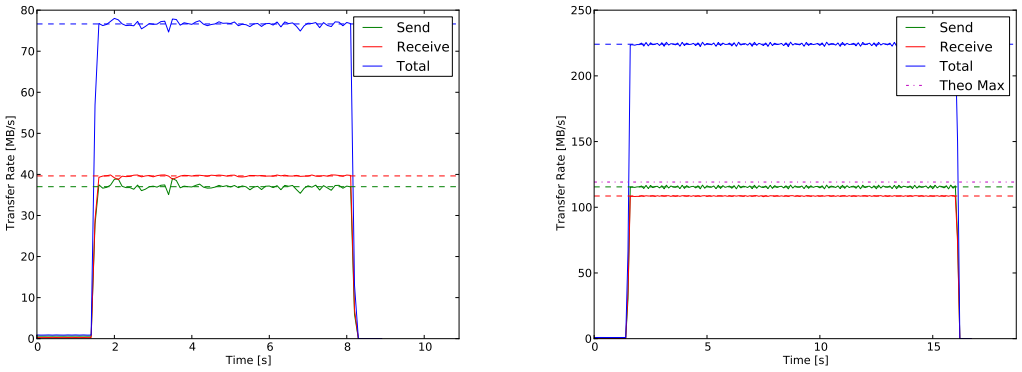
The transfer rate of the HICANNARQ (see section 2.1.2) was estimated in *Karasenko* [2014] where throughput of ≈40 MiB/s is expected for one HICANN with linear scaling for additional HICANNs.

Figure 4.2 shows transfer rate measurements for readout of HICANN configuration information. In fig. 4.2a 2×10^7 HICANN configuration packets were read out from 1 HICANN and in fig. 4.2b 2×10^8 packets from 8 HICANNs on the same reticle. These throughput measurements were performed using the `man` tool at 0.1 s resolution. Table 4.1 presents the averaged transfer rates and standard error of the mean of the measured plateaus. For 8 HICANNs a throughput of (115.5 ± 0.1) MiB/s was measured which is near the theoretical wire-speed of 119.2 MiB/s, whereas the measurement for 1 HICANN yields a transfer rate of (37.01 ± 0.09) MiB/s.

As this only describes the on-wire transfer rate including protocol header and HostARQ resends, the effective payload traffic was determined. The transfer time for the aforementioned experiment setup was measured and repeated ten times for both cases. This yields for 8 HICANNs a throughput of (104.39 ± 0.06) MiB/s but for 1 HICANN it yields (22.71 ± 0.03) MiB/s, nearly half of the transfer rate on-wire. Investigating the transfered packets via the packet analyzer tool Wireshark reveals many resends on host side and mostly packets with only one configuration packet from FCP to host. This explains the large difference between effective and on-wire transfer rate. The estimated and measured transfer rates for single HICANN communication over HICANNARQ differ significantly which requires further investigation. One explanation could be not optimized timings in the HICANNARQ. As 6 HICANNs already saturate the throughput of the HostARQ this does not affect full reticle operation.

	send [MiB/s]	receive [MiB/s]	Total [MiB/s]
1 HICANN	37.01 ± 0.09	39.63 ± 0.03	76.64 ± 0.08
8 HICANNs	115.50 ± 0.10	108.57 ± 0.03	224.07 ± 0.11

Table 4.1.: Measured transfer rates of HICANN configuration readout test in fig. 4.2.



(a) 1 HICANN 2×10^7 packets

(b) 8 HICANNs 2×10^8 packets

Figure 4.2.: Transfer rate measurement of a simple HICANN configuration readout test. Solid lines show transfer rate for send (green), receive (red) and total (blue). Dotted horizontal lines show average values of plateaus. Dotted magenta line in b) represents maximal theoretical one way bandwidth of 119.2 MiB/s.

4.2.2. Calibration Speedup

It is of interest to investigate performance improvements not only for single low-level units but also for higher-level functionality that uses existing layers of the software stack. For this purpose, the calibration tool was chosen because it uses HostARQ for HICANN configuration and performs many analog readout operations with the AnaRM via USB 2.0. An introduction to the calibration software is given in section 2.2.2

Figure 4.3 shows a performance comparison of a calibration run with the old transport protocol (top) and HostARQ (bottom). The calibration run with HostARQ was conducted on a new BrainScaleS system whereas the old implementation was run on an FACETS setup. Each peak in USB 2.0 traffic over the AnaRM corresponds to the readout of one parameter for all 512 neurons. Functions `record`, `trace` and `status` correspond to analog readout, all other functions use HostARQ for communication. Comparing both profiles, the run-time dropped from ≈4000 s to 1500 s which corresponds to a speedup factor of 2.7. The total data volume of both runs remains the same as the payload traffic did not change.

Looking at the comparison of function run-times one can see a speedup for many functions of approximately one order of magnitude. All of these functions communicate via FCP. The only function with a relatively low speedup factor of ≈1.5 is the `config_floating_gates` function, but it still has a huge absolute time reduction. Looking at the portions of total run-time, `config_floating_gates` is now the main bottleneck claiming over 50% of total run-time. Most of this time is caused by busy waiting for the floating gate controller on the HICANN chip. This is a known limitation of the current HICANN revision and cannot be circumvented without either optimizing low-

level controller operation or a chip redesign. The former is currently under investigation by P. Müller. The speedup of ≈2 for `record`, `trace` and `status` is unexpected because these functions do not use the HostARQ. An explanation for this could be that the calibration runs were conducted on different setups (i.e. host machines and neuromorphic setups). This was unavoidable as of how the old and new wafer modules are set up. This probably also explains the difference in remaining runtime. Considering the low average Ethernet transfer rates, the huge speedup cannot result from the higher throughput of the HostARQ, but is due to the reduced round trip times.

4.3. ADC Time-series Data Compression

As described in section 2.1.2, readout of analog values during calibration (see section 4.2.2) is done via ADC boards which use USB 2.0 for communication. Due to the improvements in recently produced HICANNv4 chips it is now required to calibrate time constants which is done by measuring the Postsynaptic potential (PSP). Parameters like membrane potential can be acquired by a short averaging over the parameter as the neuron is in a steady state. The PSP is dynamic which makes it necessary to trace and average over ≈100 PSPs. One trace requires 576×10^3 sample points. This is done for all 512 neurons and repeated 16 times. With each sample having the size of 16 bit, on-wire, one gets 8.79 GiB of readout data.

The maximum throughput of analog readout was determined by taking 10^8 samples (≈190 MiB) of an inactive HICANN, i.e., noise data, resulting in an average maximum transfer rate of (43.8 ± 0.2) MiB/s. The specified maximum bandwidth of USB 2.0 is 53 MiB/s [USB2, cap. 4.7.2]. As the measurement only accounts for payload and neglects additional protocol overhead the actual on-wire transfer rate was estimated. Transfer is done in high-speed bulk mode with a header of 55 B with 512 B payload where 12 B are used for our custom header. One needs to multiply the payload transfer rate with a factor $(500 + 55 + 12)/500 = 1.134$. This yields a total transfer rate on-wire of (49.7 ± 0.3) MiB/s which is near the specified maximum bandwidth of 53 MiB/s.

Taking the measured effective transfer rate of (43.8 ± 0.2) MiB/s and the required trace data of 15.23 GiB leads to readout times of 205 s for calibration of one time constant. Considering that one calibration run calibrates several parameters of one HICANN and a wafer module has 384 HICANNs, makes an improvement in data transfer desirable.

4.3.1. Compression Scheme

To circumvent the bandwidth limitations of USB 2.0 a compression scheme was implemented. A delta compression scheme was chosen as the sampled voltage traces are time correlated. An uncompressed ADC sample has 12 bit data but is stored and transfered in 16 bit due to alignment. The compression packet utilizes the unused 4 bit as header information and stores differences in the 12 bit as payload. The possible compression cases are illustrated in fig. 4.4. The FPGA on the AnaRM calculates the difference for each incoming sample and its precursor and forms packets with the highest possible

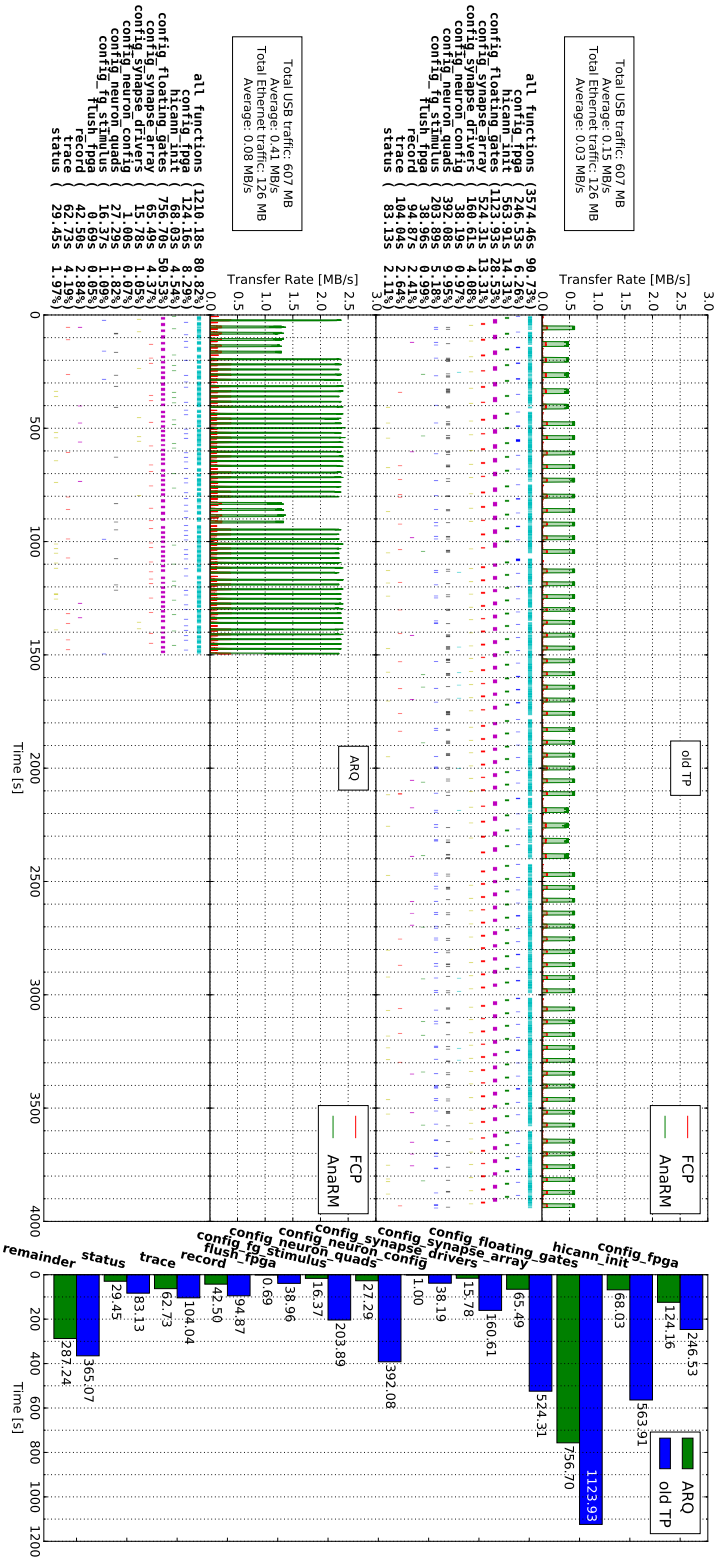


Figure 4.3.: Performance profile of a calibration run with old transport protocol (top) and new HostARQ (bottom). For an explanation of plot structure see section 4.1.

compression. This process is pipelined in the FPGA and only introduces a delay of 5 clock cycles (50 ns) but does not decrease throughput.

0 1 2 3	4 5 6 7	8 9 10 11	12 13 14 15
Header	Payload		
x0	12 bits		
x1	8 bits		4 bits
x2	4 bits	4 bits	4 bits
x3	4 bits	8 bits	
x4	6 bits	6 bits	

Figure 4.4.: Illustration of all possible compression combinations

For decompression, the values are bit-shifted corresponding to the compression combination given in the header and then added to their respective previous value. FPGA code was implemented by V. Karasenko, decompression on host side was developed by E. Müller, testing and performance analysis was conducted by the author.

Compression rate and possible slow down due to decompression was investigated. First, a pure software test of decompression was conducted. It yields for random values a max throughput of (103.1 ± 0.8) MiB/s and for a constant value (418 ± 8) MiB/s. This shows that software decompression should not impede transmission speeds as decompression throughput is significantly higher than the theoretical USB 2.0 throughput of 53 MiB/s. Compression of the FPGA was tested with a modified firmware that takes a fixed value as input instead of real data from the ADC. This allows for reproducible comparison between compression and no compression. Comparison with a fixed value is fair as the FPGA speed is independent of compression and the decompression scheme in software is fast enough.

Figure 4.5 shows transfer rate measurements for readout of 10^8 samples with and without compression. The raw transfer rate without (43.5 ± 0.3) MiB/s and with compression (43.8 ± 0.3) MiB/s agree with each other and with the aforementioned noise measurement of (43.8 ± 0.2) MiB/s. This shows that compression and decompression does not lead to a slowdown. Comparing the total data volume of both cases (189 MiB and 62 MiB) yields a compression factor of 3. The speedup of transmission runtime is also $\approx$3 as both cases have agreeing transfer rates. This is of course the best possible compression factor as there is no difference between values. In comparison, compression of noise yields only a factor of 1.1 as one expects from pure random data. Compression cannot be smaller than a factor of 1 as the header only consumes previously unused bits.

4.3.2. Performance Comparison

Figure 4.6 shows a performance log (see section 4.1) of a time constant calibration step with 16 repeated PSP readouts over all 512 neurons. Uncompressed case is displayed at the top and compression case at the bottom.

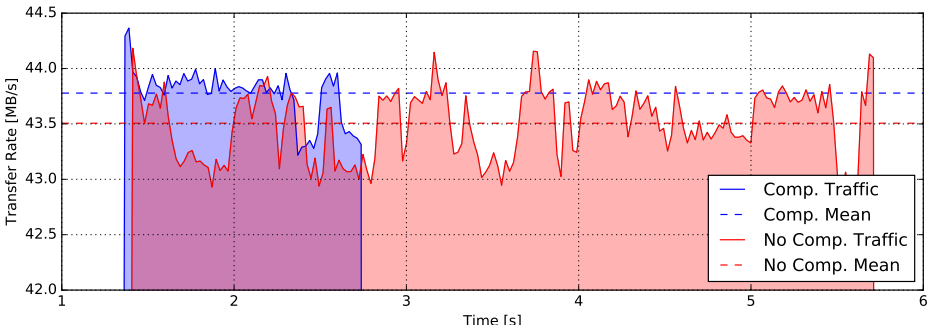


Figure 4.5.: Transfer rate measurement of readout of 10^8 ADC samples with compression (blue) and without (red). Readout was done with modified FPGA code that only sends a constant value. **Transfer rates**: No compression: (43.5 ± 0.3) MiB/s, with compression: (43.8 ± 0.3) MiB/s. **Total traffic**: No compression: 189 MiB, with compression: 62 MiB.

The first two spikes in activity are due to experiment initialization. After that the readout of the PSPs is visible where one can see a drop in activity between the 16 repetitions. Compared to previously shown calibration runs (see fig. 4.3) the massive increase in USB 2.0 data transfer can be seen. Taking the total USB 2.0 traffic of 9011 MiB without compression and 3281 MiB with compression yields a compression factor of ≈2.75. Comparing the total execution time of compression with 2941 s an no compression with 3075 s without compression. This yields a speedup factor of ≈1.05. Looking at the comparison of run-times on the right one can see this difference is mostly due to a shorter run-time of the `trace` function.

Given a compression factor of ≈2.75 one would expect a higher speedup factor than ≈1.05, assuming that the transfer rate is the bottleneck. Looking at the transfer rates one can see that both cases do not nearly reach the maximum bandwidth of 43.5 MiB/s as determined in previous measurements. The speedup of the trace function ≈1.06 similar to the total speedup which is expected as its the only backend function with a noticeable change. Taking the portions of total runtime of backend functions 87.1% for no compression and 86.3% with compression shows that the calibration is mostly impeded by memory readout of the AnaRM. But it also shows that the current implementation of the calibration software is not mainly bound by the maximum transfer rate but rater through transmission latency, i.e., the round trip time. To further reduce the total calibration run-time, larger ADC readouts must be implemented. One possibility is to pipeline the recording of several traces and readout larger blocks of data.

4.4. Parallel Digital Spike Communication

The BrainScaleS system can operate in two different experiment modes. These are the *real-time* and the *batch* mode as defined by *Müller* [2014]. The *real-time* mode utilizes

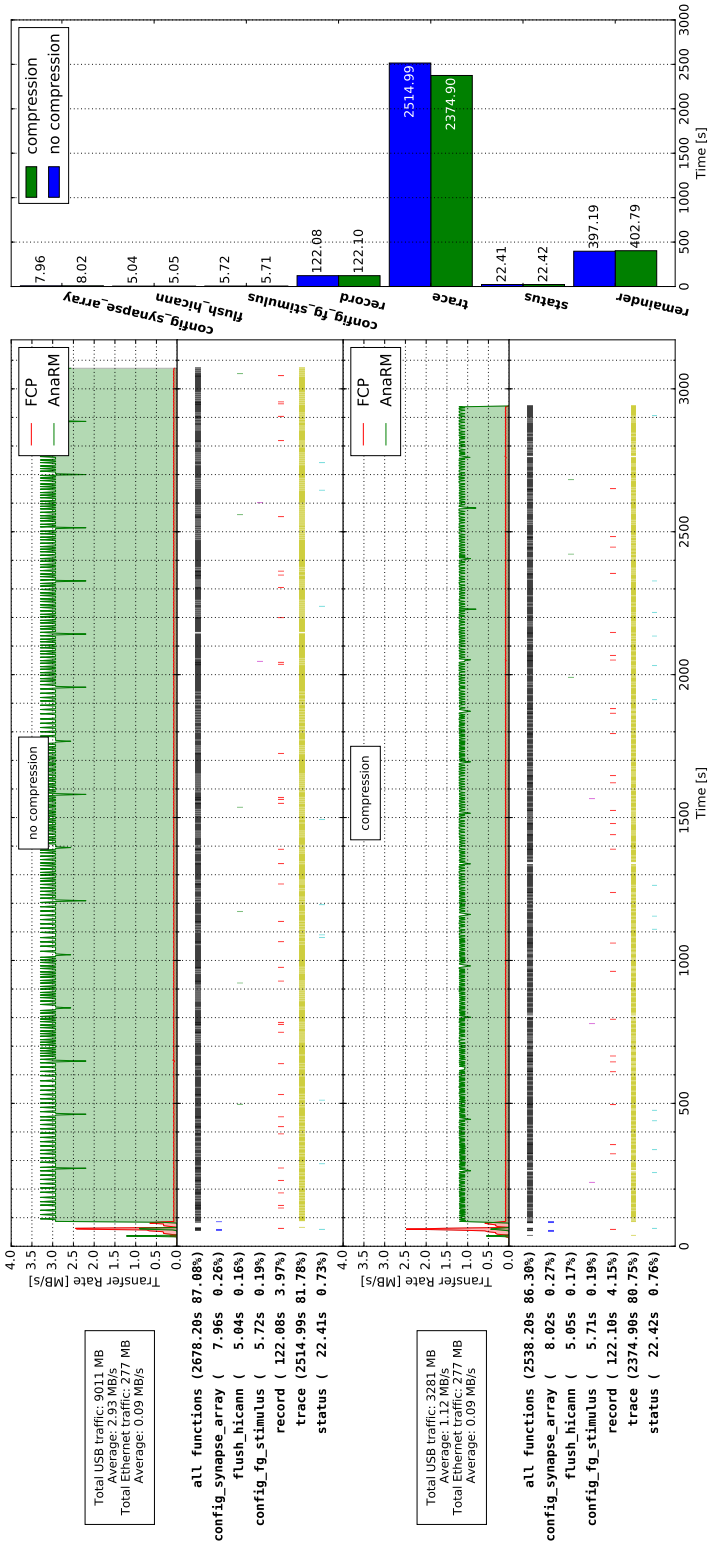


Figure 4.6.: Comparison of performance profiles of PSP calibration steps. At the top without compression of ADC readout and at the bottom with compression. For an explanation of plot structure see section 4.1.

the SpiNNaker real time interface to directly transmit spike and configuration data between host machine and HICANN without prior buffering in the FCP. In the *batch* mode the FCP serves as a buffer which allows for precise timing of spike events and configuration. This section presents improvements to the implementation of the *batch* mode. This includes parallelization of the previously pure serial communication in multi FCP operation to speedup the experiment setup time. Further changes are done to former data structures and functions to reduce memory usage and additional speedup besides parallelization. The general sequence of a *batch* experiment is as follows:

1. Configure FCP and the HICANN chip (i.e., synapses, neurons, the on-wafer network and digital support infrastructure)
2. Send spike train to Playback memory
3. Start Playback and Trace memory of FCP (this denotes the experiment start)
4. Wait for experiment on the wafer to finish
5. Stop FCP Trace memory
6. Read out spikes from Trace memory

Playback and Trace are buffer memories for incoming and outgoing spike trains in the FCP (see section 2.1.2). The separate stop command to the Trace memory is necessary as there is no built-in mechanism yet that denotes the experiment end. A possible implementation would be a *experiment stop* event at the end of the input spike train triggering the trace-stop state. Previously each item was performed sequentially for each FCP. However, items 1, 2 and 6 can be done independently for each FCP. Our goal is to parallelize the experiment run and reach wire-speed for write and readout of Playback and Trace memory. Wire-speed means 10 Gbit/s, because as described in section 2.1.3 each of the 48 FCP boards on a wafer module has a 1 Gbit/s MAC, those 48 links are connected to aggregation switch which aggregates to a single 10 Gbit/s link to the conventional network backbone switch (see section 2.1.3).

4.4.1. Parallelization

The OpenMP API is used for the implementation of parallel execution. It comprises compiler directives and a run-time library. Parallelization was mostly implemented via the `#pragma omp parallel for` directive. Some sections of the low level communication code required locking mechanisms to prevent multiple allocation of system resources like sockets. One issue that needed to be tackled was memory consumption. Each FCP has 512 MiB for Playback as well as Trace memory, this means each wafer module can hold up to 48 GiB of spike data. The host machines communicating with the wafer modules have 32 GiB memory each. Spikes were previously stored in `std::vectors` of spike objects that consist of 16 bit address information and a 64 bit time stamp. The structure of how spikes are stored was changed. The 64 bit time stamp was kept but the 16 bit allocated

for address information are now a `Union` of 64 bit bit fields. The C++ implementation is shown in listing A.1 Either the spike neuron address or a HICANN configuration packet is stored. This was done under the expectation of the not yet implemented feature of mixing HICANN configuration packets and spike events in the Playback and Trace memory. Bit fields were chosen for formatting of the address contrary to a raw 64 bit data entry. This was done as bit fields have the benefit of making the code more readable compared to manual bit shift operations on a raw data entry. The structure of the spike address follows the same format as pulse packets sent to the Playback memory in the FCP [*HBP SP9 partners*, 2014, cap. I-10.2.2]. This optimizes the transmission of spike events as there is no additional shifting needed.

The sending scheme was reworked to reduce memory consumption. When a spike train in the prior implementation was transmitted it was first completely converted into Playback memory format and held as a temporary object doubling the memory consumption in the worst case. Only after the entire spike train was transmitted the temporary object was freed from memory. This was reworked so that HostARQ frames are sent as soon as they are ready. Two buffers with size of a HostARQ packet are filled in alternating order. After both buffers are filled for the first time, every time a buffer is filled the other buffer is forwarded to the HostARQ. This sequence is required as pulse events are formed into pulse groups which consist of a single or multiple 64 bit entries that can contain up to 2 pulse events. The information of group size is held in the first entry and is updated for each additionally included spike. As pulse groups can bridge over two HostARQ packets it is necessary to manipulate the previous buffer as long as spike events are inserted into the current buffer. The scheme for receiving spikes was slightly changed. It was transfered from `HICANN-system` to `HALbe` to reduce unnecessary value copying. The waiting mechanic between inquiries, if new packet were received, was changed from a constant to an exponentially growing wait.

The spike sorting scheme was changed to reduce redundant sorting. Spikes are added individually for each gigabit link of each HICANN but are held for an entire FCP, i.e. 8 HICANNs with 8 gigabit links each. The sorting of the entire spike train of a FCP was previously built into the function which adds spike trains of the individual gigabit links. This leads to unnecessary sorting. The built in sorting was removed and a separate sorting function was implemented. This function is now only called once before sending of the spike trains. In the best case one now saves 63 unnecessary sorting calls. Another advantage is that this allows parallel sorting for multiple FCPs.

4.4.2. Spike Loopback Experiment

Spike loopback experiments were conducted to investigate the performance of digital spike communication. Spike trains with an initial delay and constant interval between spike times, called Inter Spike Interval (ISI), are looped back. For this it is necessary to configure the DNC merger of the HICANN. A diagram of the configuration is shown in fig. 4.7. The even gigabit links are set to loop back the incoming spikes to their neighboring odd links. These experiments were concluded for a variable number of HICANNs, FCPs, ISIs and spikes.

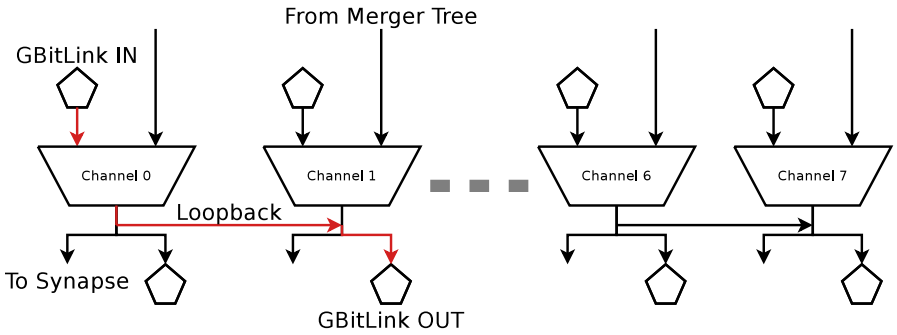


Figure 4.7.: Diagram of loopback configuration of DNC merger. Each channel can be configured to receive external input from its gigabit link and/or from the merger tree. This is forwarded to the synapse and the gigabit link output. As the gigabit link can only be configured for one sending direction it is necessary to connect the output of two channels to achieve loopback functionality. The path of a spike event signal is highlighted in red.

Figure 4.9 shows a comparison of performance profiles (see section 4.1) of spike loopback experiments. The old serial implementation can be seen at the top and the parallel implementation at the bottom. The parallel implementation uses the described parallelization scheme and the new data structure for spikes. Both implementations use the new sorting scheme. 10×10^6 Spikes were sent to 8 FCPs with 8 HICANNs each, resulting in 640×10^6 looped back spikes. This amount of spikes uses most of the memory of the host machine without risking memory overflow in peak usage. The initial delay was set to 2×10^{-6} s and ISI to 1×10^{-6} s in the hardware time domain. With a hardware speedup factor of 10^4 this yields an ISI of 10 ms in biological time.

Table 4.3 presents the run-time of each function, the total run-time and their corresponding speedup factors for five repeated experiments. Comparing the total run-times yields an overall speedup of the experiment of 3.7 ± 0.1. This also includes the waiting time for the experiment run on the wafer, so the effective speedup is even higher ($\approx$4). Because the host machines have 4 cores with 2 hyper-threading cores each, in the best case one can expect a speedup factor of up to 8. The function `sendSpikes`, which is responsible for adding spikes to the spike train, is slower by a factor of 0.28 ± 0.02. This is due to the new structure in which spikes and HICANN configuration commands are stored. The access of single entries in the bit fields is time consuming due to bit shifting. The configuration of the FCPs and HICANNs done by function `configure` has a speedup of 7.3 ± 0.3. This is expected as most of the time is spend in busy waits which should allow for good parallelization. The relatively small speedup of 1.7 ± 0.1 in the case of `sort_spikes` is unexpected. It may be caused by an unfavorable memory alignment. The analysis for the transfer rates was done to better investigate the speedup of sending and receiving. Figure 4.8 presents the traffic analysis of the parallel case of fig. 4.9. Mean and standard deviation were determined in the marked areas. Area boundaries were determined manually. Transfer rate analysis in the serial case was done with a

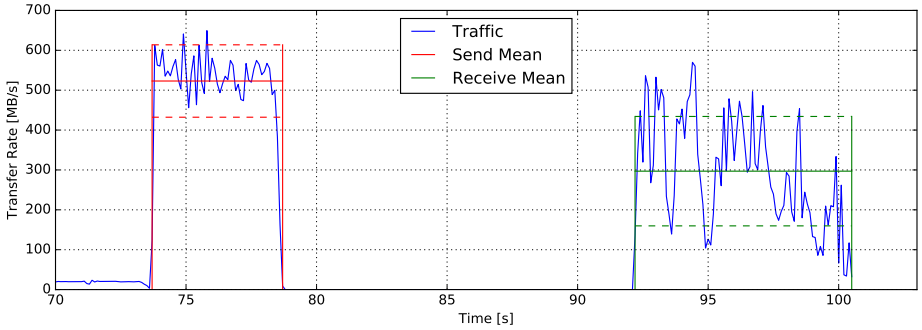


Figure 4.8.: Traffic analysis of sending and receiving of parallel case in spike loopback experiment fig. 4.9. Solid lines show mean and dotted lines show their respective standard deviation. Mean and standard deviation were determined in the marked scopes. Sending traffic rate (red): (523 ± 90) MiB/s. Receiving traffic rate (green): (290 ± 140) MiB/s.

single FCP but with 10^7 spikes to get a better estimate. Both cases were repeated five times. The resulting transfer rates and corresponding speedup factors are displayed in table 4.2. Considering the speedup factor for the transfer rate of 4.5 ± 0.2 during sending makes the speedup factor of 7.9 ± 0.8 for the function `send_spikes` remarkable. One reason for this additional speedup is that the neuron addresses of spikes are now stored in bit fields. In the old implementation every time a neuron address of a spike was read out it needed to be bit-shifted. The high jitter in transfer rate, especially in receiving is not due to network limitations but due to insufficiently fast reception of packets in the higher level software.

	Serial [MiB/s]	Parallel [MiB/s]	Speedup Factor
Sending	105.4 ± 0.6	519 ± 10	4.5 ± 0.2
Receiving	77 ± 6	308 ± 6	4.1 ± 0.4

Table 4.2.: Measured transfer rates of spike loopback experiment for serial and parallel implementation.

The loopback test was repeated to further investigate the performance of the new parallel implementation. Ideally the experiment would be conducted with 46 FCPs and their corresponding reticles. Because of 10 defect reticles on the only available system with sufficient cooling, it was only possible to conduct the experiment with the remaining 36 reticles. The values for initial delay and ISI are the same. The number of spikes was chosen to be near the maximum memory capacity of the host machine.

$$\frac{1}{2} \times \frac{32\,\mathrm{GiB}}{16\,\mathrm{B} \times 48\mathrm{FCPs} \times 8\mathrm{HICANNs}} \approx 3 \times 10^6$$

The factor 1/2 is required because sent and received spike trains are held in memory.

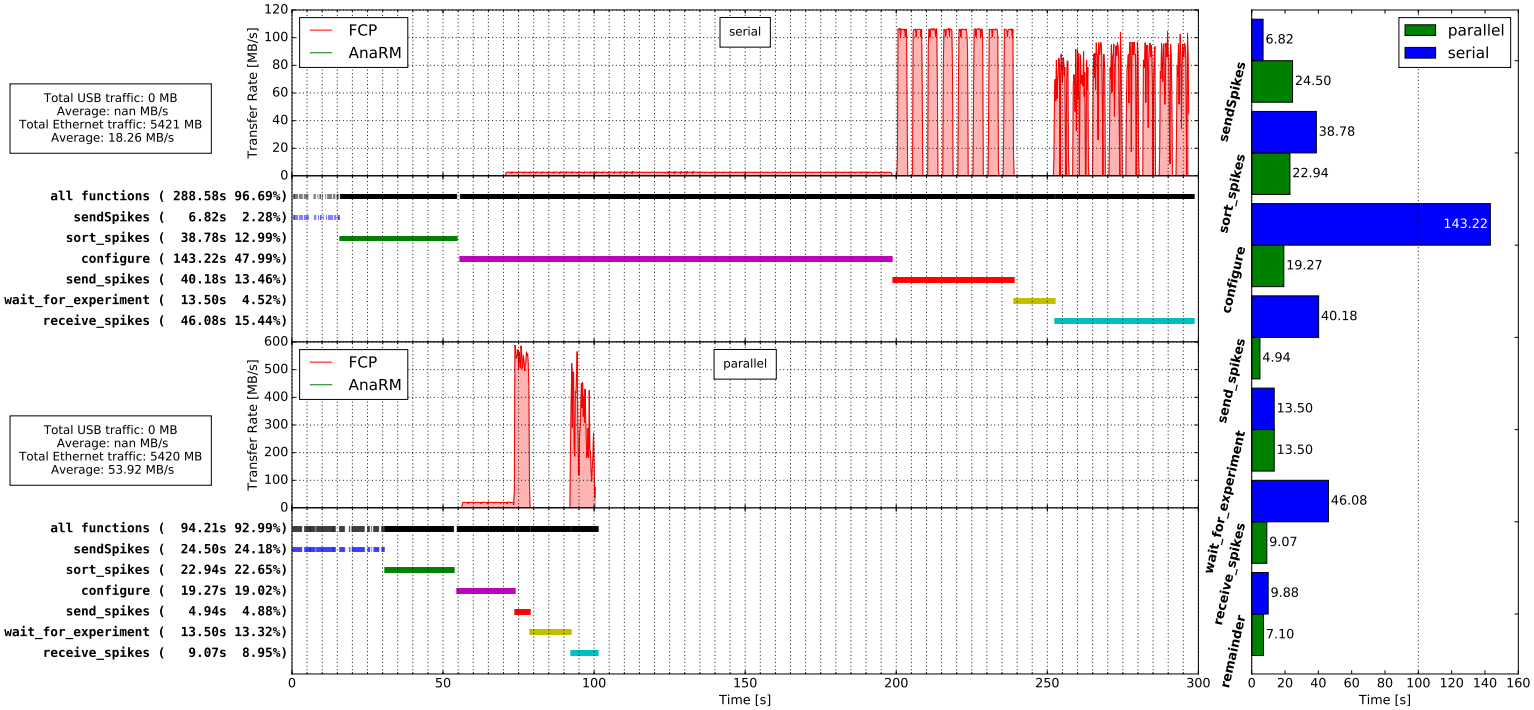


Figure 4.9.: Comparison of performance profiles of digital spike loopback experiments. Old serial implementation at the top and parallel implementation at the bottom. 8 FCPs with 8 HICANNs each. 10×10^6 spikes, initial delay 2×10^{-6} s, constant ISI 1×10^{-6} s. Times are in hardware domain. For an explanation of plot structure see section 4.1.

Function	Serial [s]	Parallel [s]	Speedup Factor
`sendSpikes`	6.8 ± 0.3	24.5 ± 0.5	0.28 ± 0.02
`sort_spikes`	38.4 ± 0.5	22.9 ± 0.4	1.7 ± 0.1
`configure`	143.3 ± 0.8	19.5 ± 0.8	7.3 ± 0.3
`send_spikes`	40.3 ± 0.4	5.0 ± 0.5	8.0 ± 0.8
`receive_spikes`	45.9 ± 0.5	8.7 ± 0.5	5.3 ± 0.3
total runtime	298.0 ± 0.7	101.0 ± 0.9	2.9 ± 0.1

Table 4.3.: Runtime of serial and parallel spike loopback experiment and corresponding speedup factors. Averaged over ten measurements. For exemplary single run see fig. 4.9.

The sending spike train is not freed after sending as one might want to repeat an experiment. Figure A.1 shows the performance profile of such an experiment. The traffic rate was again determined as in fig. 4.8. It yields transfer rates of (474 ± 15) MiB/s and (312 ± 11) MiB/s for sending and receiving respectively. These numbers are still below the aspired goal of wire-speed performance which was shown for HostARQ in *Müller* [2014] but are nonetheless a major improvement compared to the previous implementation.

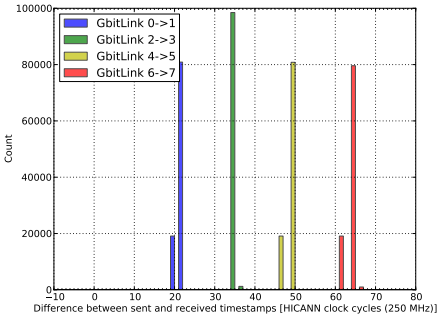
4.4.3. Spike Time Analysis

To ensure proper functionality of the digital spike transmission a time stamp analysis of the returned spikes was done. To verify the correct behavior of the spike release functionality of the FCP, the absolute differences of sent and received spike times was investigated. The same experiment setup as in section 4.4.2 was used with experiment parameters: Initial delay 2×10^{-6} s, ISI 1×10^{-6} s and number of spikes 1×10^6.

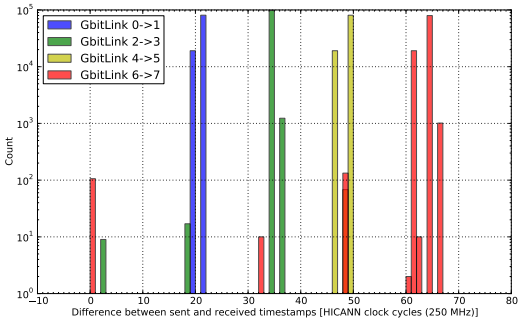
Figure 4.10 shows the result of such an experiment. A histogram of spike time differences is shown in (a), (b) displays the same histogram with logarithmic scale and (c) presents the spike time difference against the spike time.

The division for the different gigabit link configurations is expected and can be explained as follows. It takes ≈70 DNC clock cycles (250 MHz) for spikes to do a loopback [Issue #1820]. This includes the time from release of a spike in the FCP to the point of leaving the DNC merger. The back transmission is not included as the spike time stamp is changed the last time on leaving the merger. The time from release to entering the merger is ≈50 clock cycles and the time spent in the merger are ≈20 clock cycles. The delay of 50 DNC clock cycles is circumvented by releasing the spike events earlier in the FCP. This explains the offset of ≈20 clock cycles for the returned spike times on all gigabit links. The additional offset of ≈14 between the different gigabit links is due to the maximum transmission rate between FCP and DNC of 18 MEvents/s [*Thanasoulis et al.*, 2014]. This corresponds to a minimum transfer time between two packets of 56 ns or ≈14 DNC clock cycles.

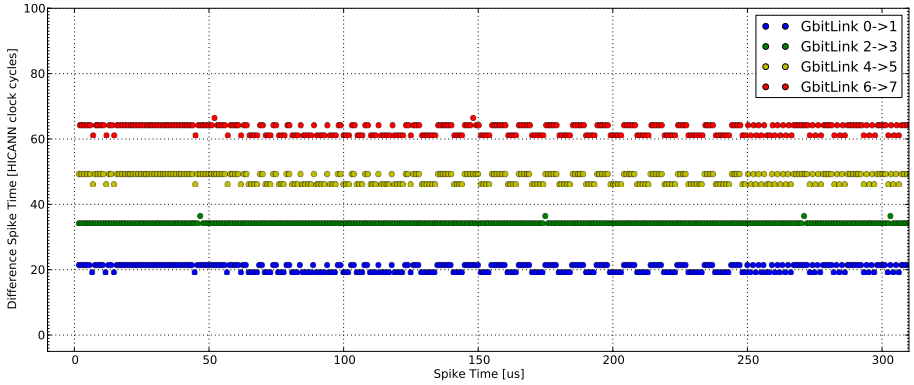
An unexpected behaviour is the jitter of spike times seen in fig. 4.10c. There seem to be two types of jitter. On the one hand there is a frequent ($\approx$ 20%) jitter of -2 or -3 clock cycles seen in fig. 4.10a. The symmetrical behavior for link 0, 4 and 6 is striking. The pattern of the jitter is identical for these 3 links. It is also peculiar that link 2 has almost no jitter. On the other hand there is a less frequent, irregular jitter visible in the logarithmic histogram (fig. 4.10b). Repeating the experiment results in different links having nearly no jitter and in some cases all links have similar jitter patterns (see fig. A.2). Repeating the experiment on other HICANNs results in the same structure for the patterned jitter and varying amount of irregular jitter (see fig. A.3). The time differences of -2 and -3 DNC clock cycles for the patterned jitter could be caused by crossover of the different clock domains of the HICANN (100 MHz) and the DNC (250 MHz). A cause for the regularity of the pattern could be aliasing effects. The exact cause of the regularity of the pattern and why random links do not show this behaviour requires further investigation.



(a) Absolute spike time differences of sent and received spikes.



(b) Same data as (a) in logarithmic scale.



(c) Time development of spike time differences for the first 300 spikes on each link.

Figure 4.10.: Spike time analysis of digital spike loopback experiment. Spike trains are separated for each gigabit link loopback configuration. Experiment parameters: Initial delay 2×10^{-6} s, ISI 1×10^{-6} s and number of spikes 4×10^{5}. HICANN 76 on wafer E_08.

5. Discussion & Outlook

The first part of this thesis presented in chapter 3 engaged stability, usability, automated testing and monitoring to ensure a robust operation of the NM-PM1 systems.

Stability issues of communication between host machines and FCP were tackled. The sources of fatal bugs in communication were located through implementation of a HostARQ test suite enabling the FPGA code-designers to fix these bugs. Stable communication between host machine and multiple HICANNs was achieved, allowing uninterrupted transmission of at least 150 TiB of data. Remaining connection interruption issues, possibly caused by temperature problems, require further investigation.

The HostARQ connection handling was encapsulated which improved experiment work flow and robustness in error cases. Further memory optimizations and robust cleanup in abnormal program termination cases is required.

The extension of APIs between different software layers enabled testing of HICANN prototype chips in arbitrary slot arrangement on test setups. Usability can be further improved, for example, by allowing independent initialization of HICANNs on a reticle.

The previous scheme for FCP firmware control was not feasible due to required manual interaction after each power cycle. The implementation of the automated remote flashing tool allows to update the firmware of the NM-PM1 system in ≈108 min which will be further reduced to ≈2 min by a change which is currently under code review.

The implemented control daemon and remote control tool enable easy powering of FCPs and reticles. Switching from a Python-based implementation to a C++ implementation would be desirable to reduce the computational overhead on the executing Raspberry Pi.

A test of digital connectivity for all reticles on a wafer module was developed resulting in a first estimate of 86.5% for the portion of usable reticles. These tests provide further insight for digital and system hardware designers, it will help in advancing the assembly protocol leading to even a higher success rate. The unstable initialization of HICANN high speed connection requires further attention. A further extension of the test including digital chip functionality verification is under development.

Automated weekly testing of one wafer module over a period of 6 months revealed no changes in digital connectivity over time with the current assembly protocol. This indicates a stable physical connection between the silicon wafer and the MainPCB but requires a larger sample size to allow conclusive results.

The monitoring infrastructure for the NM-PM1 system was enhanced to enable monitoring of the FCPs. The implemented temperature visualization tool revealed a overheating issue of FCPs and wafer. The insight through visualization assisted the system hardware designers to develop an improved cooling system allowing to operate the system with the desired temperature. It is worthwhile to improve the time resolution

of monitoring to support detailed energy consumption analysis for experiments. This is necessary to enable a fair comparison between different simulation backends and platforms like CPUs, GPUs or neuromorphic hardware systems, as it was performed, e.g., in *Diamond et al.* [2015].

The second part of the thesis addressed performance improvements to the communication between host machines and the BrainScaleS wafer modules as is presented in chapter 4.

A performance profiling tool was developed to visualize function run-times and their corresponding I/O traffic to analyze performance improvements.

The improvements by replacing the old transport protocol with the HostARQ were investigated. The data throughput between HICANNs and host machine was determined resulting in $(104.39 \pm 0.06)\,\mathrm{MiB/s}$ for parallel communication with 8 HICANNs which agrees with the estimated throughput in *Müller* [2014]. Throughput of $(22.71 \pm 0.03)\,\mathrm{MiB/s}$ for a single HICANN differs significantly with the estimation in *Karasenko* [2014] which needs further investigation. The calibration tool was used to analyze improvements in higher-level software achieving a total speedup of 2.7 mostly due to reduced round trip time.

Large runtimes of neuron model time constant calibration routines were addressed by implementing a compression scheme for the readout of analog data. This implementation achieves a compression factor of up to 3 without measurable slowdown, yet only yields a speedup of around 5% for calibration. This is most probably caused by small readout chunk sizes and could be drastically improved by larger readout chunks which than will additionally benefit from compression.

The parallelization of the previously serial digital spike communication and the resulting memory issues were engaged to facilitate the full capacity of the NM-PM1 system. Digital spike loopback experiments were conducted to analyze improvements of the implementation. The entire experiment run gained total speedup of 3.7, however the sending of spike data alone reached a speedup of 8. An average throughput of $(519 \pm 10)\,\mathrm{MiB/s}$ during sending and $(308 \pm 10)\,\mathrm{MiB/s}$ during receiving was achieved. The initial goal of reaching wire-speed was not yet accomplished but nonetheless are speedups of 4.5 ± 0.2 and 4.1 ± 0.4 a major improvement compared to the previous implementation. An additional change to the spike sorting scheme achieved further reduction in run time. Further improvements could be achieved, for example, by utilizing zero-copy access, i.e. by directly working on data allocated by lower-level communication layers.

A time stamp analysis of the looped back spikes was conducted to ensure correct implementation of aforementioned changes. It showed that the implementation works correctly but additionally revealed previously unmeasured jitter of spike times with a regular pattern identical for different HICANNs on different wafer, possibly caused by clock domain crossings. The small differences caused by the timing jitter should have little effect on real-world experiments but nevertheless require further investigation.

The investigations, enhancements and optimizations throughout this thesis were vital steps towards a robust operation of the neuromorphic computation platform. The next two big steps for the system are the upgrade of the silicon wafers to the new HICANNv4.1 revision and the opening of the neuromorphic computing platform to the neuroscience community. Early open access is very important as the feedback from modelers is invaluable for improving and enhancing not only the current system but also affects design decisions of future hardware revisions.

There is still plenty room for improvements to facilitate the full capacity of the hardware and ensure a smooth operation of the system. Enabling fully synchronized inter-reticle and inter-wafer experiments, reaching wire-speed communication in the full 20-module system and the decrease of experiment setup time, i.e. the configuration of parameters and preparation of input data, should be the main priorities in the near future. Concerning administration and maintenance of the system, a fully automated system control reacting to monitoring feedback would be optimal. Furthermore, the execution and data management for the calibration of 7680 HICANNs needs do be automated and parallelized.

Closed-loop experiments involve neuronal networks concurrently interacting with a real or simulated environment. For plasticity research in computational neuroscience, such setups could provide deeper insight into mechanisms of learning. However, plasticity happens on a large range of time scales. Large time-scale experiments are of particular interest as the conventional simulation approach limits the absolute simulation time. Large-scale accelerated neuromorphic hardware systems, such as the NM-PM1, would be the obvious solution for this time-scales problem. However, this will only be possible with the interactive real-time operation of the system requiring major improvements to enable robust, long-lasting experiments. In the same regard, a highly modified version of the HICANN, named HICANN-DLS, is under development which will increase the plasticity capabilities of the system. In particular, it will include a plasticity processor [*Friedmann*, 2013] that allows a flexible implementation of various plasticity models. Utilizing the capacities of this chip will require substantial extensions to the current software stack in order to allow more flexible pre-defined plasticity algorithms and eventually user-defined programs. Especially the integration of this freely programmable CPU into the mapping software layer is a major task.

Another huge advancement will be the embedding of the wafer directly into the Main-PCB. This will make the systems substantially smaller and reduce power consumption allowing for higher packing density and eventually more modules. It will also reduce connectivity issues as the error-prone wafer mounting process is skipped [*Güttler*, 2016].

In summary, fast large-scale neuromorphic hardware is the only foreseeable promising solution to conduct long-lasting plasticity experiments, i.e. several years of biological time. It will be essential for the investigation of long-term effects of learning and memory, the key features of brain.

A. Appendix

A.1. Repository Listing

This section will give a comprehensive list all code locations and repository states. All repositories can be found at `https://brainscales-r.kip.uni-heidelberg.de/`. Appendix A.1 shows the git hash IDs of HEAD state of all used repositories. All tests and tools work with these versions if not stated otherwise.

Repository	SHA 1 ID
bitter	`da89a7ece461ad6e97a6d7f9e9c0a5b9fccafbff`
boost_mongo	`b8565e8df102600c919a2a68d3d0d113ac8e6533`
cake	`c6248bc3b2f09c9e768cc2b165836475a4d7d1e7`
calibtic	`5efa90ead80f57507b2036cff49d9c0b56a41e6c`
euter	`4260673d2328b7043ecb1bfa9ce1aa0610f3609b`
halbe	`2ff4a1209ac4ad1d3d52b20cba14ac4ef6bb5790`
hicann-system	`6b4df5b5d6a9c0a60a02f990074e41d8d6e61fc2`
hmf-fgpa	`bcd9fdb3b47b18400eb4845d00923d5f804d9b71`
lib-boost-patches	`b018e7970367f11207358594e670bc1a57114a61`
lib-rcf	`de6fa72d55c186bd89aefd0e1c6b90e4c99323a0`
logger	`826c5ed66f68f17200fadd3863dafd8318cea71d`
pygccxml	`8ae9e19ae00c4152fa5a381eb9e663561c07345f`
pyplusplus	`9b513ec97614d03764f0a0ac5c7b7bd67c27a996`
pythonic	`f2f162e34d7b024e0de79c55133fef00e82fffe0`
pyublas	`9f707f60320e20e5a7714d920ba0530d092e1310`
pywrap	`9d93135270075c745e541d4ea0b7a1977e216665`
rant	`4a8acd076fb9531ce61a990ef0f414935886d85d`
redman	`42d28037a0bcdf1e70c7bbb6734411de63174b3f`
sctrltp	`daa50a17a768f1e433f548f81dd4ad529eeabedf`
sthal	`b6a4f583b318ff639dce5e782ea073f9d3fa52fb`
vmodule	`210885997f0124975cb17cb6710d5fffec585513`
ztl	`068c18233337711e40027aa51dc667f7ed6cdcd8`

Table A.1.: Git hashes of HEAD state for all used repositories. All tests and tools work with these versions if not stated otherwise.

A.1.1. Implemented Tools And Changes

This subsection lists all implemented changes and tools with their respective repository locations. If changes have yet to be reviewed, their Gerrit change ID is given. Visualization tools are located in the repositorie of this thesis `https://brainscales-r.kip.uni-heidelberg.de/projects/masterthesis-mauch`

- HostARQ encapsulation: hicann-system, sctrltp
- Arbitrary HICANN positioning: sthal, halbe, hicann-system
- Control Daemon / Remote Control / System Startup / Remote Flash / Temperature Visualization: hmf-fpga
- Performance Profiling: sthal, vmodule, hicann-system
- Performance Visualization:
  sthal: change ID: `I2478601c5fd93318318f6c214a0df4f03b47327a`
- ADC Compression:
  sthal: change ID: `I484b55ccbc62fe4ecbf3373c0dfc9ae01b05fc1c`,
  halbe: change ID: `I602afacfc01b5d5a271320781e7d9e3fcaceb1f2`,
  vmodule: change ID: `Iea981370a0210119b23bb023f001f4b0a79289bc`
- Parallel Digital Spike Communication:
  sthal: change ID: `Ia45df493616f6e3d17103aeefa747450c10a9679`,
  halbe: change ID: `I29c2fda8afa2b5d57e6bc4738d5f4d9384ceabc2`,
  hicann-system: change ID: `Ie9d0aa06d7b1b268bc0dfb1fe2d985e705004841`

A.1.2. Used And Implemented Tests

This subsection lists all used or implemented tests with their respective repository locations. If tests have yet to be reviewed, their Gerrit change ID is given.

- HostARQ Loopback Test: hicann-system
- HICANN Configuration Readout Test: hicann-system
- Wafer Test:
  hmf-fpga: change ID `Ie9d0aa06d7b1b268bc0dfb1fe2d985e705004841`
- HostARQ Calibration Speedup: see appendix A.1.2
- ADC Compression Throughput: halbe
- ADC Compression Calibration: cake
- Parallel Digital Spike Loopback Test:
  sthal: change ID: `Ia45df493616f6e3d17103aeefa747450c10a9679`,
  halbe: change ID: `I29c2fda8afa2b5d57e6bc4738d5f4d9384ceabc2`,
  hicann-system: change ID: `Ie9d0aa06d7b1b268bc0dfb1fe2d985e705004841`

Repository	SHA 1 ID
bitter	`da89a7ece461ad6e97a6d7f9e9c0a5b9fccafbff`
boost_mongo	`b8565e8df102600c919a2a68d3d0d113ac8e6533`
cake	`a0f99429f6fdde5a4b662bc71800ee42417f4be9`
calibtic	`3fd8c342722fc762a9c6e9ba0a207816c304f86f`
cd-denmem-teststand	`dca94a45c3c3bad338753cb5a6cf172ba82cf9bf`
chip-teststand	`827039a516e718ce4806b46c9a84ddfe8618eed0`
euter	`9a68d336e3b355feeafa1b0a23fd782f093f09d4`
halbe	`96183a9017f67764811f51d379f074a006a2e411`
hicann-system old	`ca751d7f1f2aca6af20ea264e11cbe04c7c8b829`
hicann-system new	`38e55aff26c6c4eb51538691687450eaf25c5253`
lib-boost-patches	`0a64f3c7dbd2b7a11e4190016ce3192367e54a63`
lib-rcf	`b541a38fa1abae3ca22ad4fc66eb0024bddcea4d`
logger	`826c5ed66f68f17200fadd3863dafd8318cea71d`
pygccxml	`8ae9e19ae00c4152fa5a381eb9e663561c07345f`
pyplusplus	`9b513ec97614d03764f0a0ac5c7b7bd67c27a996`
pythonic	`f2f162e34d7b024e0de79c55133fef00e82fffe0`
pywrap	`844777680e463e8a7b8ed14226c763ba653e35bb`
pyublas	`b541a38fa1abae3ca22ad4fc66eb0024bddcea4d`
rant	`4a8acd076fb9531ce61a990ef0f414935886d85d`
redman	`4c0c86022454273ad571d7b378cc068e3712ccca`
sctrltp	`daa50a17a768f1e433f548f81dd4ad529eeabedf`
sthal	`5a9fa0da44e6d0d1174e7bc01f4c9cc6162aeb68`
vmodule	`7b4aba10125d1818c6669ac1aa3cdd7923d1ef24`
ztl	`068c18233337711e40027aa51dc667f7ed6cdcd8`

Table A.2.: Git hashes of HEAD state for calibration speedup analysis without (old) and with (new) HostARQ.

A.2. Code Excerpts

```
1  struct PbTraceEntry {
2  public:
3
4    union Payload {
5      struct {
6        uint64_t               : 63;
7        uint64_t type          : 1 ;
8      } generic;
9
10     struct pulse_data_t{
11       uint64_t l1address     : 6 ;
12       uint64_t gbitlink      : 3 ;
13       uint64_t hicannaddress : 3 ;
14       uint64_t dnc           : 2 ;
15       uint64_t               : 49;
16       uint64_t type          : 1 ;
17       }
18     } pulse_data;
19
20     struct config_data_t {
21       uint64_t hicannaddress : 3 ;
22       uint64_t dnc           : 2 ;
23       uint64_t configdata    : 49;
24       uint64_t               : 9 ;
25       uint64_t type          : 1 ;
26       }
27     } config_data;
28     uint64_t raw;
29   };
30
31     .
32     .
33     .
34
35 private:
36   uint64_t eventtime; // in DNC clock cycles
37   Payload payload; // holds information of Event. e.g. address, config ...
38 }
```

Listing A.1: New data structure for Playback and Trace events. Member functions are omitted.

A.3. Supplementary Figures

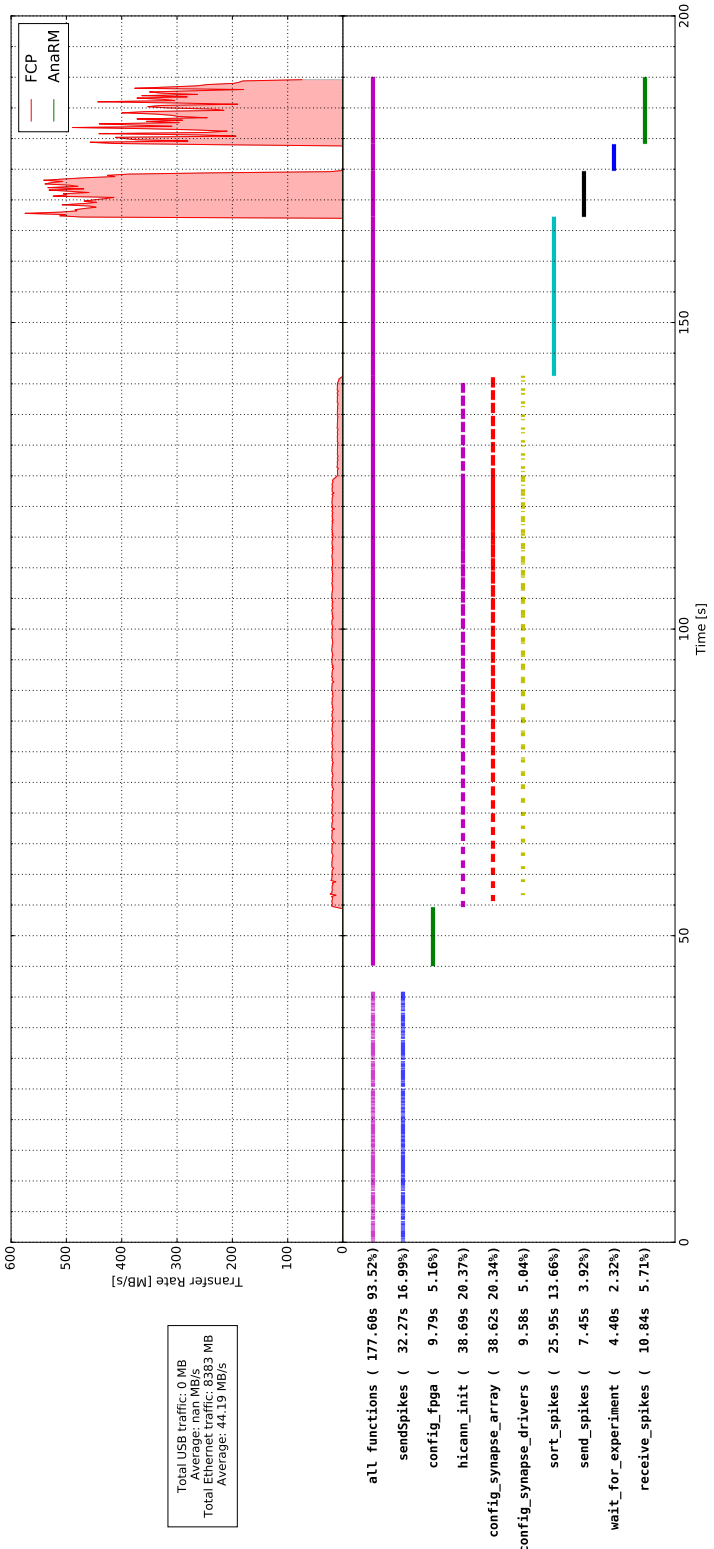
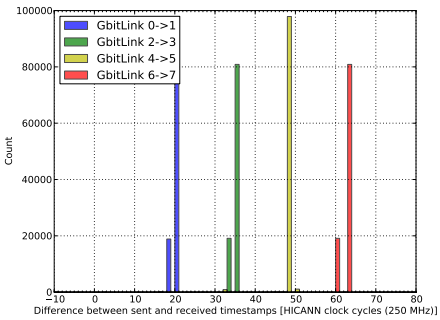
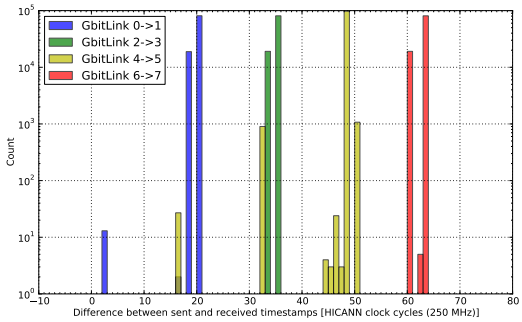


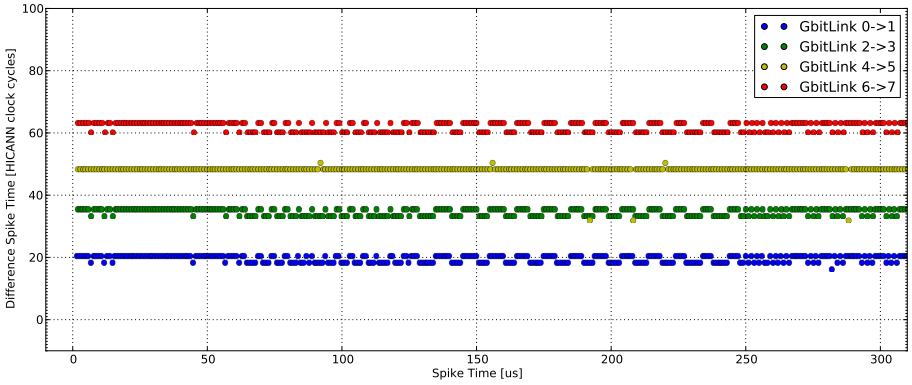
Figure A.1.: Performance profile of digital spike loopback experiment. 36 FCPs with 8 HICANNs each. 3×10^6 spikes, initial offset 2×10^{-6} s, constant ISI 1×10^{-6} s. Times are in hardware domain. For an explanation of plot structure see section 4.1.



(a) Absolute spike time differences of sent and received spikes.

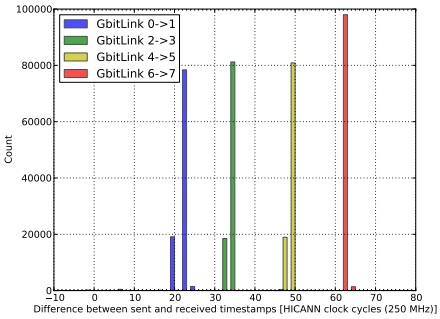


(b) Same data as (a) in logarithmic scale.

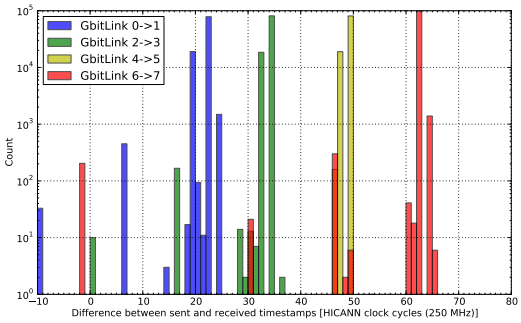


(c) Time development of spike time differences for the first 300 spikes on each link.

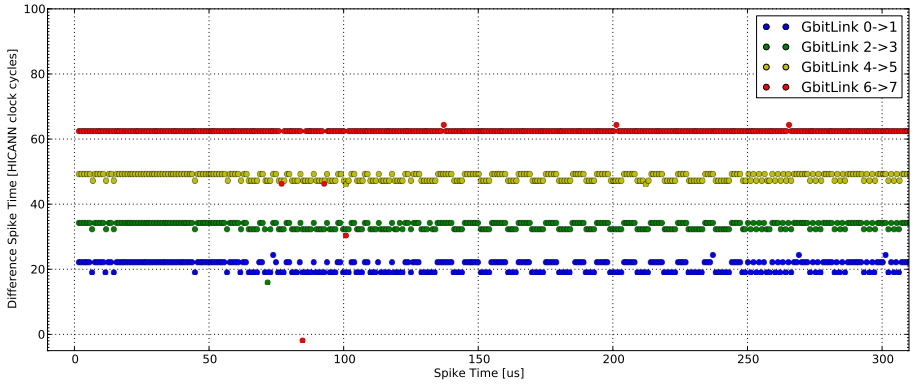
Figure A.2.: Repeated spike time analysis of digital spike loopback experiment as in fig. 4.10. Spike trains are separated for each gigabit link loopback configuration. Experiment parameters: Offset 2×10^{-6} s, ISI 1×10^{-6} s and number of spikes 4×10^5. HICANN 76 on Wafer E_08.



(a) Absolute spike time differences of sent and received spikes.



(b) Same data as (a) in logarithmic scale.



(c) Time development of spike time differences for the first 300 spikes on each link.

Figure A.3.: Spike time analysis of digital spike loopback experiment analog to fig. 4.10 on different HICANN. Spike trains are separated for each gigabit link loopback configuration. Experiment parameters: Offset 2×10^{-6} s, ISI 1×10^{-6} s and number of spikes 4×10^{5}. HICANN 25 on wafer E_08 HICANN.

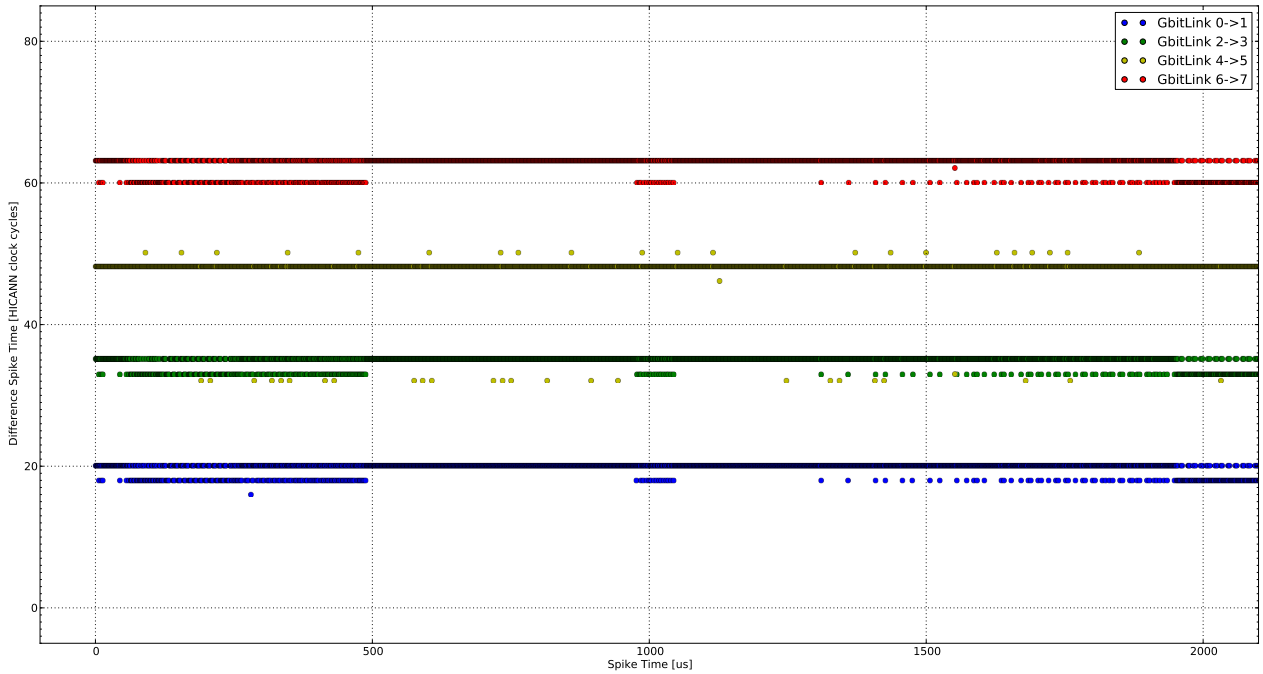


Figure A.4.: Time difference of received and expected spike time stamp of digital spike loopback experiment. Larger segment of fig. 4.10. Spike trains are separated for each gigabit link loopback configuration. Experiment parameters: Offset 2×10^{-6} s, ISI 1×10^{-6} s and number of spikes 4×10^{5}. HICANN 76 on wafer E_08.

List of Acronyms

ADC Analog-to-Digital Converter. 5, 8, 21, 25, 27–29

AdEx Adaptive Exponential Integrate-and-Fire. 1, 3 3.

AnaRM Analog Readout Module. 5, 8, 22, 24, 25, 28

API Application Programming Interface. 6, 7, 12, 39

ARP Address Resolution Protocol. 10, 11

ARQ Automatic Repeat Request. 4, 8

CPU Central Processing Unit. 41

DNC Digital Network Chip. 4, 15, 31, 32, 35, 36

EPROM Erasable Programmable Read-Only Memory. 13

FCP FPGA Communication PCB. 4, 5, 8, 10–16, 18, 19, 22–24, 30–35, 39, 47

FPGA Field-Programmable Gate Array. 4, 5, 10, 12–14, 23, 25, 27, 28, 39

HALbe Hardware Abstraction Layer Backend. 8, 31

HICANN High-Input Count Analog Neuronal Network Chip. 3–9, 11, 12, 14, 15, 17, 23–25, 30–32, 34, 36, 39–41, 47, 49

HICANNARQ HICANN ARQ protocol. 4, 8, 23

HostARQ Host ARQ protocol. 8, 10–12, 15, 23–26, 31, 39, 40

I2C Inter-Integrated Circuit Link. 5, 14, 18

IPv4 Internet Protocol version 4. 10, 13

ISI Inter Spike Interval. 31–35, 37, 47–50

JTAG Joint Test Action Group. 4, 13, 15, 16

L1 Layer 1. 4

L2 Layer 2. 4

MaCU Main System Control Unit. 5, 14, 18

MainPCB Wafer Module Main PCB. 16, 18, 19, 39, 41

MTREE Merger Tree. 4

NM-PM1 Neuromorphic Physical Model version 1. 1, 3, 5–7, 9, 13, 16, 17, 39–41

NMPI HBP Neuromorphic Platfrom Interface. 6

PCB Printed Circuit Board. 4

PSP Postsynaptic potential. 25, 27–29

PyHMF Python for the Hybrid Multiscale Facility. 6, 7

RPC Remote Procedure Call. 13, 14

SLURM Simple Linux Utility for Resource Management. 6

SSH Secure Shell. 14

STDP Spike Timing Dependent Plasticity. 3

StHAL Stateful Hardware Abstraction Layer. 8

STP Short-term Plasticity. 3

TUD Technische Universität Dresden. 4, 10, 13

USB 2.0 Universal Serial Bus version 2.0. 5, 21, 22, 24, 25, 27, 28

Bibliography

BrainScaleS, Research project, `http://brainscales.kip.uni-heidelberg.de/public/index.html`, 2012.

Brette, R., and D. Goodman, Brian, a simulator for spiking neural networks based on Python, 2008.

Brüderle, D., et al., A comprehensive workflow for general-purpose neural modeling with highly configurable neuromorphic hardware systems, *Biological Cybernetics*, *104*, 263–296, 2011.

Davison, A. P., Neuromorphic computing platform interface, 2016.

Davison, A. P., D. Brüderle, J. Eppler, J. Kremkow, E. Muller, D. Pecevski, L. Perrinet, and P. Yger, PyNN: a common interface for neuronal network simulators, *Front. Neuroinform.*, *2*(11), 2008.

Diamond, A., T. Nowotny, and M. Schmuker, Comparing neuromorphic solutions in action: implementing a bio-inspired solution to a benchmark classification task on three parallel-computing platforms, *Frontiers in neuroscience*, *9*, 2015.

Diesmann, M., and M.-O. Gewaltig, NEST: An environment for neural systems simulations, in *Forschung und wisschenschaftliches Rechnen, Beiträge zum Heinz-Billing-Preis 2001*, *GWDG-Bericht*, vol. 58, edited by T. Plesser and V. Macho, pp. 43–70, Ges. für Wiss. Datenverarbeitung, Göttingen, 2002.

FACETS, Research project, `http://http://facets.kip.uni-heidelberg.de/`, 2010.

Friedmann, S., A new approach to learning in neuromorphic hardware, Ph.D. thesis, Ruprecht-Karls-Universität Heidelberg, 2013.

Furber, S. B., 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), doi:http://doi.ieeecomputersociety.org/10.1109/TC.2012.142, 2012.

Ganglia, *Ganglia Monitoring System*, [Online; accessed: 2016-02-16], 2016.

Gerstner, W., and R. Brette, Adaptive exponential integrate-and-fire model, *Scholarpedia*, *4*(6), 8427, doi:10.4249/scholarpedia.8427, 2009.

Güttler, M. G., *PhD thesis*, Ruprecht-Karls-Universität Heidelberg, in preparation, 2016.

Hartmann, S., S. Schiefer, S. Scholze, J. Partzsch, C. Mayr, S. Henker, and R. Schuffny, Highly integrated packet-based aer communication infrastructure with 3gevent/s throughput, in *Electronics, Circuits, and Systems (ICECS), 2010 17th IEEE International Conference on*, pp. 950–953, doi:10.1109/ICECS.2010.5724670, 2010.

HBP SP9 partners, *Neuromorphic Platform Specification*, Human Brain Project, 2014.

Hines, M., and N. Carnevale, *The NEURON simulation environment.*, pp. 769–773, M.A. Arbib, 2003.

Issue #1820, *Measure FPGA HICANN Delay on KINTEX systems*, [Online; `https://brainscales-r.kip.uni-heidelberg.de/issues/1820`; accessed: 2016-02-19], 2015.

Jeltsch, S., A scalable workflow for a configurable neuromorphic platform, Ph.D. thesis, Universität Heidelberg, 2014.

JSC, NEST report, 2015.

Karasenko, V., Design, implementation and testing of a high speed reliable link over an unreliable medium between a host computer and a xilinx virtex5 fpga, Bachelor's thesis (English), University of Heidelberg, 2011.

Karasenko, V., A communication infrastructure for a neuromorphic system, Master's thesis (English), University of Heidelberg, 2014.

Kawaguchi, K., *Jenkins, An extendable open source automation server*, [Online; accessed: 2016-02-16], 2016.

LLNL, SchedMD, et al., *Simple Linux Utility for Resource Management*, [Online; accessed: 2014-04-29], 2014.

man, *ifstat*, Linux man-pages project, [Online; `http://man7.org/linux/man-pages/man8/ifstat.8.html` accessed: 2016-01-23], 2015.

Markram, H., The human brain project, *Scientific American*, *306*(6), 50–55, 2012.

Mead, C. A., Neuromorphic electronic systems, *Proceedings of the IEEE*, *78*, 1629–1636, 1990.

Mead, C. A., and M. A. Mahowald, A silicon model of early visual processing, *Neural Networks*, *1*(1), 91–97, 1988.

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

Müller, E. C., Novel operation modes of accelerated neuromorphic hardware, Ph.D. thesis, Ruprecht-Karls-Universität Heidelberg, HD-KIP 14-98, 2014.

OpenMP, The OpenMP API specification for parallel programming, `http://openmp.org/wp/`, [Online; accessed: 2016-02-16], 2016.

Philipp, S., Design and implementation of a multi-class network architecture for hardware neural networks, Ph.D. thesis, Ruprecht-Karls-Universität Heidelberg, HD-KIP 08-20, 2008.

Riken, Largest neuronal network simulation achieved using K computer, `http://www.riken.jp/en/pr/press/2013/20130802_1`, 2013.

Riken, K computer specifications, `http://www.top500.org/system/177232`, 2016.

RRDtool, *Round-Robin Databank Tool*, OETIKER+PARTNER AG, [Online; accessed: 2015-12-13], 2015.

Schemmel, J., personal communication, 2016.

Schemmel, J., 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.

Schemmel, J., 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)*, pp. 3367–3370, IEEE Press, 2007.

Schemmel, J., 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.

Schemmel, J., 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)*, pp. 1947–1950, 2010.

Schilling, M., A highly efficient transport layer for the connection of neuromorphic hardware systems, Diploma thesis, Ruprecht-Karls-Universität Heidelberg, HD-KIP-10-09, 2010.

Schmidt, D., Automated characterization of a wafer-scale neuromorphic hardware system, Master thesis, Ruprecht-Karls-Universität Heidelberg, 2014.

Scholze, S., et al., A 32 GBit/s communication SoC for a waferscale neuromorphic system, *Integration, the VLSI Journal*, doi:10.1016/j.vlsi.2011.05.003, in press, 2011.

Thanasoulis, V., B. Vogginger, J. Partzsch, and R. Schuffny, A pulse communication flow ready for accelerated neuromorphic experiments, in *Circuits and Systems (ISCAS), 2014 IEEE International Symposium on*, pp. 265–268, doi:10.1109/ISCAS.2014.6865116, 2014.

USB2, *Universal Serial Bus Specification*, USB Implementers Forum, Inc., [Online; accessed: 2016-02-12], 2016.

Weaver, V., Linux perf event features and overhead, in *FastPath Workshop*, 2013.

Wireshark, Wireshark packet analyzer tool, [Online; accessed: 2016-02-16], 2016.

Acknowledgments

Prof. Dr. Karlheinz Meier and Dr. Johannes Schemmel for excellent leadership and scientific guidance which allows the great work done in this group.

Eric Müller for supervision of this thesis and being a near-endless source of all things IT.

Vitali Karasenko for co-supervision and always having time to explain FPGA stuff.

My family for supporting all my life decisions which lead me here.

Andreas, Christoph, Lukas, Mitja, "Onkel" Olli, Paul and Sebastian for proofreading and general support throughout this thesis.

The entire Electronic Vision(s) group and especially my comrades in the container for a unique, joyful and productive work environment.

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, February 29, 2016

.....
(signature)