Department of Physics and Astronomy

University of Heidelberg

Bachelor Thesis

in Physics

submitted by

Jan Debus

born in Frankfurt a. Main, Germany

August 2016

Commissioning of an FPGA-based prototyping environment for neuromorphic hardware

This Bachelor Thesis has been carried out by Jan Debus at the

Kirchhoff Institute for Physics

Ruprecht-Karls-Universität Heidelberg

under the supervision of

Prof. Dr. Karlheinz Meier

Commissioning of an FPGA-based prototyping environment for neuromorphic hardware

To ease the development of neuromorphic hardware a prototyping environment is highly useful. A straightforward approach is the use of an FPGA as a testing platform. This thesis presents the first version of such a system which consists of a Xilinx Spartan-6 FPGA that is connected to the BrainScaleS communication infrastructure. The main focus lies on the communication interface which required a partial re-design of the existing implementation to facilitate compatibility to the existing system. Extensive verification using software simulation demonstrates the successful implementation. Experimental testing was also attempted, but was strongly encumbered by technical constraints on the physical communication link. Further investigations identify possible explanations and suggest further improvement.

Inbetriebnahme einer FPGA-basierten Prototypumgebung für neuromorphe Systeme

Das Aufsetzen einer Prototypumgebung erleichtert die Entwicklung neuromorher Hardware erheblich. Eine FPGA-basierte Testplattform stellt hierfür einen naheliegenden Kandidaten dar. Diese Arbeit stellt die erste Version eines solchen Systems vor, das aus einem Xilinx Spartan-6 FPGA besteht welcher mit der Kommunikationsinfrastruktur des BrainScaleS Projektes verbunden ist. Im Fokus liegt die Kommunikationsschnittstelle, die teilweise angepasst werden musste um Kompatibilität zum bisherigen System zu gewährleisten. Ausführliche Verifikation mit Softwaresimulationen belegt eine erfolgreiche Implementierung. Experimentelle Tests wurden durch technische Einschränkungen an der physikalischen Verbindung beeinträchtig. Untersuchungen weisen auf mögliche Erklärungen hin und diskutieren Verbesserungspotential.

Contents

1 Introduction

When taking a look at the mammalian brain as a computational machine, something fascinating can be observed. While nowadays vastly outperformed in terms of raw computational speed by man-made semiconductor technology, the brain's ability to learn and process sensory data remains unparalleled by state of the art computers. In an attempt to bridge this gap, the field of computational neuroscience has recently witnessed the development of a radically new type of technology: in addition to simulating neural networks on conventional computers, researchers are now starting to use so-called neuromorphic hardware. These devices aim to replicate various architectural and dynamical aspects of spiking neuronal networks using analog electronic circuits.

As some of these platforms are aimed at studying developmental processes in the human brain (in contrast to, e.g., robotics applications), they are supposed to emulate scenarios that take hours, days, or even years of biological real time. Such experiments would therefore have to run faster than their biological counterparts. In the particular case of the hardware developed in the Electronic Vision(s) group in Heidelberg, this acceleration factor lies between 10^3 and 10^5, thereby imposing high demands on the communication infrastructure. Over the years, several different chip designs have been developed and produced, with the most recent being the High Input Count Analog Neural Network chip (HICANN). Tests for these chips are controlled via a host PC, which is linked to the HICANN over a Field Programmable Gate Array (FPGA) (see Figure 1.1). The functionality of an FPGA is freely programmable by a user, and is used here in order to serve as a bridge between host and hardware. It serves many functions including protocol conversions, sorting and general management of the data that is exchanged between host and neuromorphic hardware.

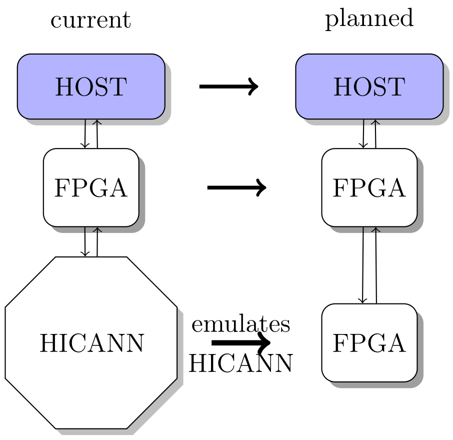

Figure 1.1: Current setup used to test the HICANN Chip, and planned setup for prototyping.

On the host side, an extensive software environment is used which provides access to neuromorphic hardware on several levels of abstraction. Ensuring correct functionality of the HICANN is crucial for the project since bug-fixing of Application-Specific Integrated Circuits (ASICs) is a costly and time-consuming endeavour. Adding to the currently employed extensive software verification scheme, the

possibility of testing HICANN-components in a realistic setup with a low investment in time and effort seems invaluable. In order to achieve this, a testing platform is currently being developed, in which the HICANN is replaced by a second FPGA. This FPGA can be loaded with any design, for example with a prototype for a new HICANN chip. Such a prototyping environment will certainly prove useful for the further development of neuromorphic chip designs which contain digital components.
The first stage of commissioning the prototyping platform is establishing high speed communication between the two FPGAs. As the communication between the HICANN chip and the bridge FPGA has already been established, one would desire the emulating FPGA to possess an interface as similar to the HICANN as possible. The goal of this thesis was to design, implement and test the units necessary for such communication. These units are required to provide high bandwidth, while using as little FPGA resources as possible in order to leave space for prototyping modules.

1.1 Thesis outline

We begin by describing the units that are involved in the high-speed communication used to connect the HICANN. Of special interest are those submodules that needed adaptation to a new FPGA device. The new L2-ARQ unit, which facilitates communication in the emulating FPGA is then introduced. Next, we list the several simulation setups used for verifying the functionality of the L2-ARQ. Following a report on hardware testing, we then discuss the measurements and take an outlook towards improving functionality and proposing future development.

2 Design Implementation

The first step in establishing an FPGA-to-FPGA communication is the design and implementation of a module written in a Hardware Description Language (HDL). FPGAs are versatile tools that are capable of modeling virtually any kind of digital Integrated Circuit (IC). The new design will be implemented into a Spartan-6 FPGA, which can serve many purposes. Yet the main intention is the prototyping of ICs specific to the HICANN-DLS[1]. In order to be compatible with previous designs, the desired interface and modules should resemble the current implementation in the already utilized FPGA Communication PCB (FCP). The FCP consists of a Kintex-7 FPGA incorporated into a custom Printed Circuit Board (PCB).
Figure 2.1 depicts a schematic of the planned system. The Reticle-ARQ serves as a wrapper for a HICANN Automatic Repeat Request protocol (HICANN-ARQ) module, whose counterpart is located in the Spartan 6. ARQ, a communication protocol capable of transmitting data over a lossy medium by resending lost words, is explained further in Section 2.3. Similarly, a Serializer-Deserializer module (SerDes) is located in both FPGA's as part of the FPGA interface module. This module is capable of converting incoming parallel bitstrings into a serial stream of data to transmit it over the physical link as well as parallelizing incoming serial data. In the current setup, data sent by the FCP arriving at the Spartan-6 FPGA interface is handed to the HICANN-ARQ. The HICANN-ARQ receives it, and loops it back through the FPGA interface to be transmitted to the FCP. In the future the HICANN-ARQ's interface will be used to handle data communication to whatever logic is residing in the Spartan-6.

2.1 The FPGA Interface

In computer science, a commonly reoccurring problem is the communication between two separate ICs. Since the number of pins on a chip is usually limited, parallel bitstrings are often converted into high-speed serial streams using a SerDes module. This module then transmits the serialized data while a receiver converts it back into parallel bitstrings. The FPGA Interface (FPGA-IF) currently used in the FCP is capable of both transmitting and receiving serialized data. In addition to serving as a SerDes, it also applies a 16-bit header to the data containing additional information such as a cyclic redundancy checksum that is used to detect data corruption over the physical link.

After deserializing the data that it receives, the FPGA-IF transmits said data to a

[1]The HICANN neuromorphic network chip has seen multiple versions over time. The most recent is called HICANN-DLS, with DLS standing for the ending of the german Autobahn 65, "Dreieck Ludwigshafen Süd". See *Schemmel and Hartel* (2015) for more information.

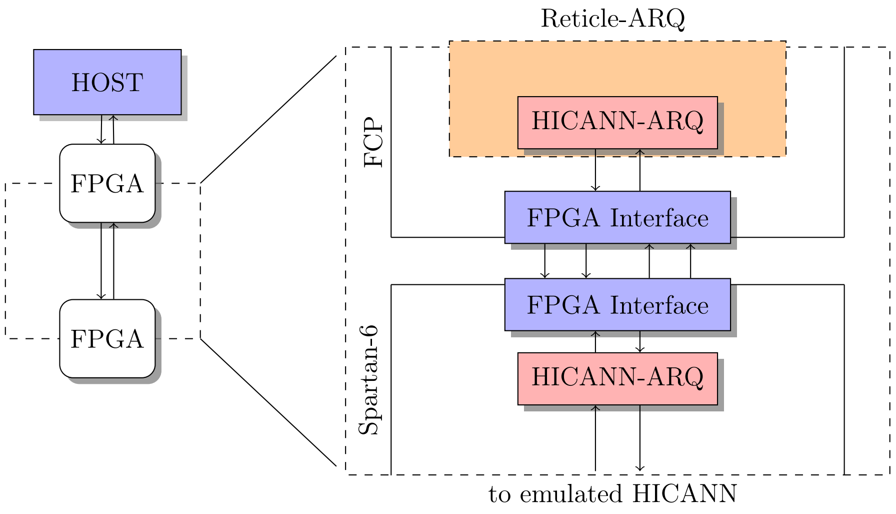

Figure 2.1: Planned setup for the FCP-Spartan communication. The HICANN-ARQ and FPGA Interface modules are already present in the FCP, and shall be implemented in the Spartan-6 as depicted here.

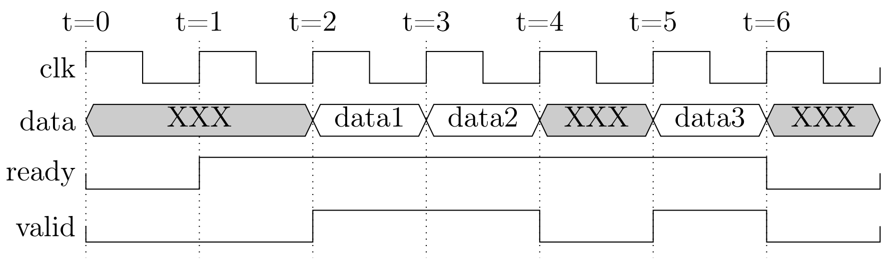

Figure 2.2: Waveform of the communication between FPGA-IF and HICANN-ARQ. If the reciever raises a 'ready' flag, the sender is free to transmit data by applying it and raising the 'valid' signal.

HICANN-ARQ module. On the other hand, it serializes and sends any data given to it by the HICANN-ARQ. The communication between the FPGA-IF and HICANN-ARQ is based on a ready-valid handshake depicted in Figure 2.2. Data is transmitted between the two in the form of 64-bit strings. The Sender waits until it receives a ready-flag, when it does, it can signal data it wants to transmit by raising a valid-flag, indicating the receiver shall store the applied data. After the data is serialized inside the FPGA-IF, it is transmitted between the two SerDes modules. As previously mentioned the number of inputs and ouputs is limited, thus the communication between the two SerDes modules does not utilize a handshake. Hence, the receiver is forced to accept whatever data arrives at it's inputs, otherwise it is lost. However, the serial data stream coming out of the SerDes has to be accompanied by a clock signal as a reference. Only then can the receiver detect whether a signal corresponds to a logical '1' or a logical '0'.
Before this is possible, the connection has to be initialized. During this time, the FPGA-IF does not accept data from the HICANN-ARQ, and only communicates with its counterpart in the other FPGA. In order to initialize contact both send a "training pattern", which is used to determine the local extrema of the signal intensity for different logical transitions. Using internal delay stages, the receiver can determine the phase shift between these local extrema and the rising edge of the clock signal. This information helps the receiver to obtain the clearest differentiation between the two logical states. Once completed, the FPGA-IF is ready to send and receive data.
After the initialization is complete, both the data and clock signal pass into an IO-Buffer. They leave it as a differential p-n signal pair, with the original signal being transmitted along with an inverted version of it. The now differential signals are transmitted with a small voltage amplitude compared to the 5V voltages normally used in digital systems. This type of signaling is known as low voltage differential signaling (LVDS). As changing the voltage of a cable requires comparatively high power, utilizing lower voltages helps reducing the required energy for communication. It also helps to detect and remove error sources from the signal by comparing the original and the inverted signal.
When transmitting data over a physical link, it can be influenced by electromagnetic radiation. Since the differentiation between a logical '1' and '0' inside a chip is based on certain threshold voltages, external noise sources influencing said voltages can 'flip' a '0' into a '1' and vice-versa. LVDS provides some degree of protection against such errors. The cables for a differential signal pair are usually very close to each other, resulting in similar disturbances on both wires caused by external noise, as depicted in Figure 2.3. Disturbances that equally affect each wire are known as common-mode noise which can be removed by taking the difference between the two inverted signals. While the influences on a cable are not symmetrical all the time, the fact that the receiver decides between '1' and '0' based on the differential voltage helps eliminate most of the occurring errors.

LVDS helps reducing the error count, but can of course not fully eliminate data corruption. In order to detect whether the transmitted data was corrupted, the FPGA-IF furthermore employs a Cyclic Redundancy Check (CRC) to the transmitted data. This error detection code appends a check value based on the remainder of a polynomial division to the data before transmitting it. This polynomial division works by interpreting

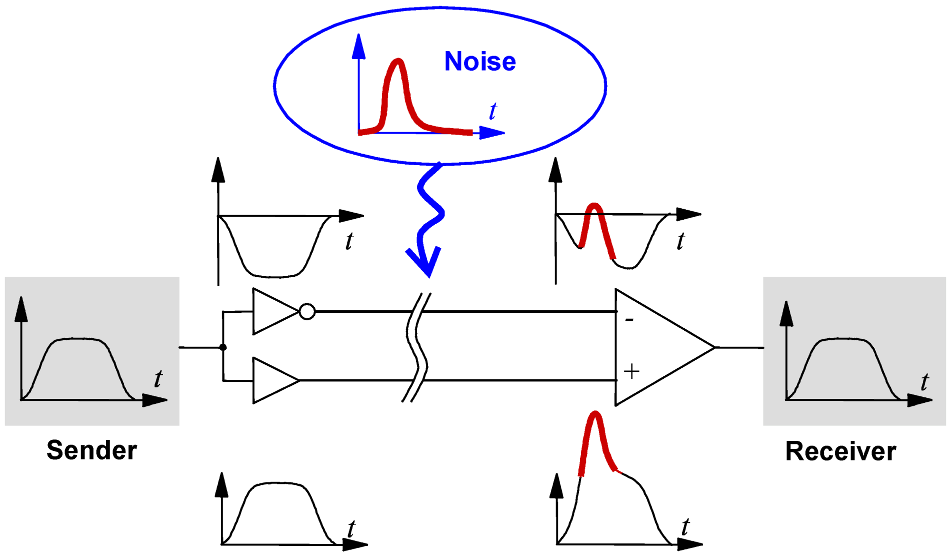

Figure 2.3: Compensation for a noise source on a cable using differential signaling. The received signal is equivalent to the sent signal. As the FPGA interfaces are located on two different FPGA interfaces, they communicate using differential signaling to minimize data corruption. Taken from *Linear77* (2016)

the to-be-transmitted data as coefficients of a polynomial with same degree as the data length. A second 'generator' polynomial is known by both sender and receiver. Before transmitting the data, the sender calculates a remainder by dividing the data polynomial by the generator polynomial. A binary representation of this remainder's coefficients is appended to the original data and then sent to the receiver. The receiver then performs a polynomial division with the data including the appended remainder. If no remainder is left, the data was transmitted correctly and can be passed on. Otherwise the received data was corrupted and is discarded.
A CRC check makes it possible to detect errors to the transmitted data with a very high probability. Using a CRC instead of other error detection schemes has two main advantages. The first is its capability to detect so called 'burst' errors. These types of errors corrupt multiple consecutive bits in a word at the same time. An n-bit long CRC however is capable of detecting all burst errors affecting n bits or less, independent of the size of the transmitted data. As burst errors are common forms of data corruption in most communication channels, a CRC has a very high error detection rate. A second advantage lies in its ease of computation. Applying a CRC check is very efficient in terms of resource consumption compared to its error detection rate. Combining this with the fact that an Automatic Repeat Request protocol (ARQ) protocol is utilized in our communication, a CRC is well-suited for our application.

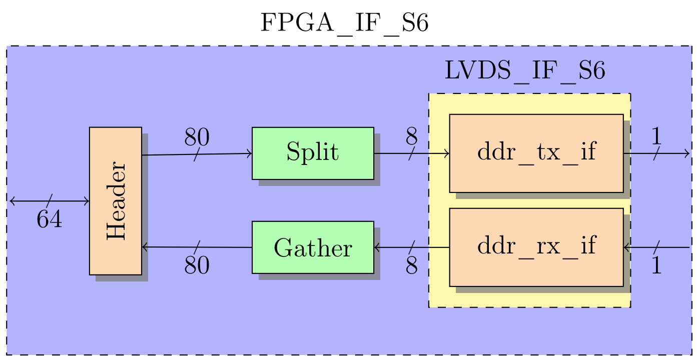

Figure 2.4: Internal structure of the Spartan-6 FPGA Interface. Data arrives as 64 bit words, and a 16 bit header is added. After that, the resulting 80 bit are passed into the LVDS_IF_S6 a byte at a time, where they get serialized and transmitted by the ddr_tx_if. In the other direction, serialized data arriving gets parralelized back into bytes by the ddr_rx_if, and follows the path backwards until 64 bit words arrive at the output.

2.2 Porting the FPGA Interface to the Spartan 6

While already implemented in the FCP, the FPGA-IF design cannot simply be implemented in the Spartan-6. The two FPGAs possess specific internal components, some of which differ between the two. A main aspect of this thesis' work consisted of converting the FPGA-IF into a Spartan-6-compatible form. This was achieved by exchanging low-level modules of the FPGA-IF which utilized these so called 'primitive' components with versions applicable to the Spartan-6. The resulting design was called Spartan 6 FPGA Interface (FPGA-IF-S6) and loaded into the Spartan-6 FPGA. However the original design for the FPGA-IF stayed unchanged in the FCP.

2.2.1 The Serialization/Deserialization Module

The structure of the FPGA-IF-S6 is depicted in Figure 2.4. The main module of the FPGA-IF-S6 which utilizes Spartan-6-incompatible FPGA-primitives is the LVDS interface, or LVDS-IF for short. This module manages the main serialization and deserialization of incoming 8-bit words, also called bytes. Other modules in the FPGA-IF manage the CRC-protocol, and split the resulting 80-bit words into these singular bytes in order to send them to the LVDS-IF. The original module previously used 'channel' modules utilizing the Kintex-7 SerDes primitives to serialize and deserialize these bytes. These channel modules are not usable for the Spartan-6. Instead, the modules DDR-TX-Interface and DDR-RX-Interface, which had been constructed for the Spartan-6, have been embedded into the new LVDS-IF-S6 constructed for this thesis. These modules, utilizing Spartan-6 primitives, perform the same tasks as the channel modules previously used in the FCP. However the interfaces of these modules changed slightly. They require

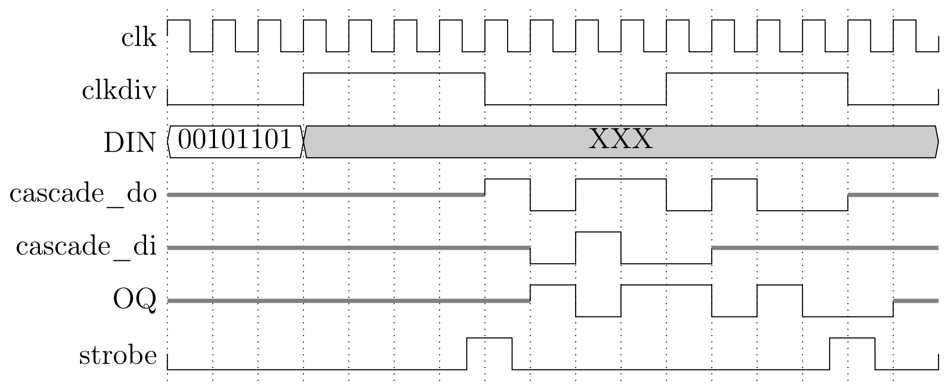

Figure 2.5: Waveform of the ddr_tx_if data serialization process. An 8 bit word applied to the DIN port during the rising edge of clkdiv gets accepted and transmitted following the next rising edge of the strobe signal. cascade_do transmits the first four bits of the data input, and appends the serial data given by cascade_di. OQ outputs the serialized data in the following clockcycle.

further assisting signals, while some signals used for the FCP were unnecessary. The most important change is the addition of the CLK-RST-GEN module, which supplies some of these assisting signals.

2.2.2 Serialization: The DDR-TX-IF

The ddr_tx interface handles the transmission of data from the FPGA-IF to the physical IO-pins. It receives a parallel input with a width of 8 bit, and parallelizes it using 2 output SerDes primitives (OSerDes) connected in cascade. The master OSerDes observes bits 7 to 4, while the slave observes bits 3 to 0. The exact protocol is depicted in Figure 2.5.
The ddr_tx_if is given two assisting signals by the CLK-RST-GEN module. One is a clock signal operating at eight times the frequency of the normally used clock 'clockdiv'. This clocksignal is simply named 'clk' and is accompanied by the second assisting signal, 'strobe'. When data is applied to the input port, it is not transmitted immediately. Instead, when the strobe signal is raised in the following cycle of clockdiv, the slave OSerDes starts to send its serialized data to the master OSerDes via cascade-do. In the next cycle of clk the master sends his serialized data via cascade-di to the slave, which appends this serial bitstream to its output of cascade-do. Finally, the master receives the entire serialized data from the slave, transmitting it to the differential signal output buffer OBUFDS via OQ.
A second pair of OSerDes primitives is connected in cascade to generate the clock output. Similarly in functionality to the data OSerDes primitives, they generate a clock signal with half the frequency of clk, sending it to an output buffer. This clock signal serves as the mentioned reference clock that accompanies the serialized data to the receiver.

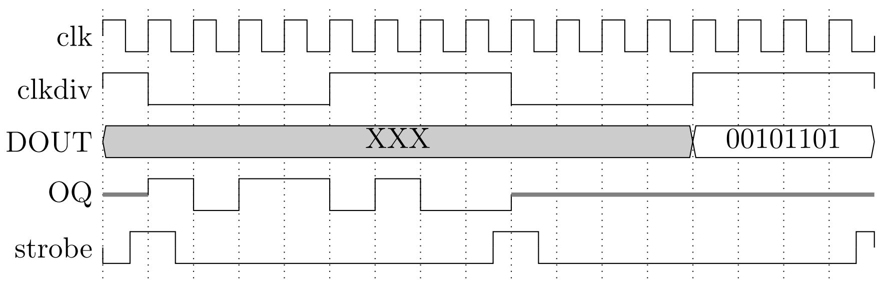

Figure 2.6: Waveform of the ddr_rx_if deserialization process. Inverse in behaviour to the ddr_tx_if, the serialized 8 bits arriving at OQ after the rising edge of strobe are parralelized and applied to the data output DOUT on the next rising clock edge after the entire signal has been recieved.

2.2.3 Deserialization: The DDR-RX-IF

Serialized data arriving at the SerDes module gets processed by the ddr_rx interface. Inverse in behaviour to the tx_if, it receives serial input data, parallelizing it into singular bytes. The arriving differential serialized data gets transformed into single bites, which are passed higher instances in the FPGA-IF. Figure 2.6 shows the temporal behaviour of the connections inside the rx_if. One can see how an incoming data stream is only transmitted to the outputs as parallel data after the second rise of the strobe signal.

2.2.4 Additional Modules

The clock and reset generator module DNC-CLK-RST-GEN provides multiple signals necessary for the operation of the tx and rx interfaces. It resides within the LVDS_IF_-S6 and generates a number of clock signals necessary for the operation of the input SerDes and output SerDes primitives in the interfaces, along with a reset flag. The most notable addition are the two strobe signals, which provide a timing orientation for accepting and sending words. These signals are transmitted to the ddr_tx and ddr_rx_if, and used as specified in the previous sections.

2.3 Backward Error Correction

The FPGA-IF is used only for data transmission and corruption checks. It is not capable of restoring corrupted data, and simply tosses it away. In order to ensure all data is transmitted, and arrives at the receiver in the order it was sent by the transmitter, a method for Backward Error Correction (BEC) must be used. When using BEC, the sender retransmits data if the receiver detects it as corrupted (*Peterson and Davie* (2003)). A common protocol used for this purpose is the ARQ. It was implemented into the FCP in the form of the HICANN-ARQ by *Karasenko* (2014).
The HICANN-ARQ uses a sliding-window algorithm, depicted in Figure 2.7, for its trans-

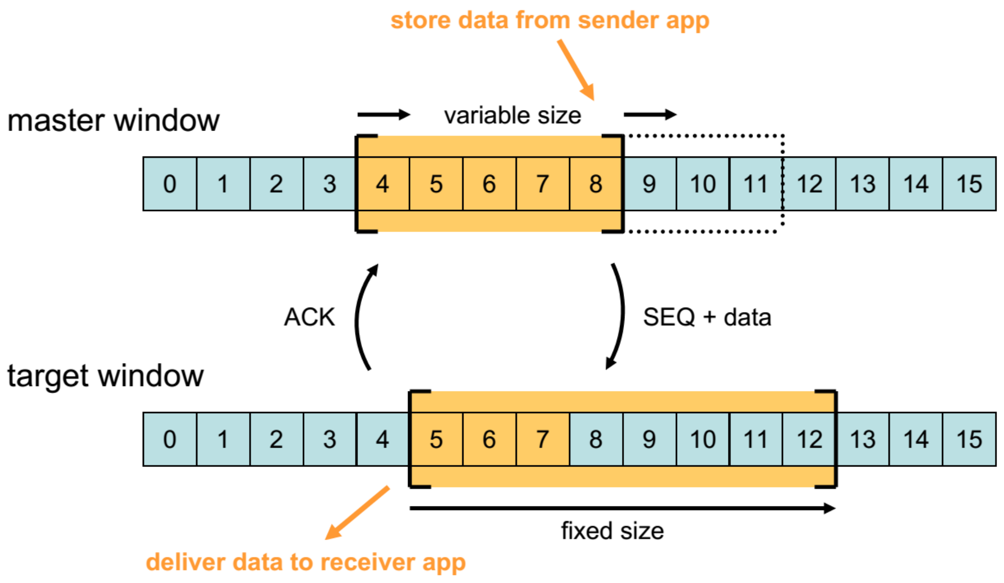

Figure 2.7: HICANN-ARQ sliding window algorithm. The master receives words and assigns a sequence number SEQ in the order they were received. It proceeds to send both to the slave, which periodically informs the master of the last word in a sequence that it received. Image taken from *Karasenko* (2014).

mission. It stores words sent to it from the upper layer inside a buffer window, and assigns a sequence number to each word. When the buffer is filled, it starts sending the data along with the assigned sequence number downwards, where another HICANN-ARQ accepts them. The receiver ARQ starts sending acknowledgement packages (ACKs) back to the transmitter ARQ, to confirm which packets have been successfully transmitted. For this the receiver does not need to ACK every package, if it sends an ack with sequence number n, it confirms all packages with sequence numbers 0 to n have been transmitted. In case of a lost packet, the transmitter ARQ waits for an ACK until a certain timeout value, before it resends all packets with a sequence number higher than the last received packet. The HICANN-ARQ sends a configurable amount of words from the buffer at a time. When an ACK with number n is received, the window of packages to send is moved to $n + 1$. This 'sliding window' algorithm allows for a saturation of the link bandwidth. As a communication towards upper level modules, it uses a simple valid/next handshake transmission depicted in Figure 2.8. When the sender has data ready to send, it raises a valid signal and applies the data, waiting for a response of the receiver. The receiver processes the supplied data, and when it is done, next is raised to confirm that the packet was accepted. This is the communication protocol which will later be used to communicate with modules residing within the Spartan-6. At the moment, the HICANN-ARQ's sending side of this protocol is connected to its receiving side. This way, data arriving at the Spartan is looped back, arriving at the FCP unchanged.

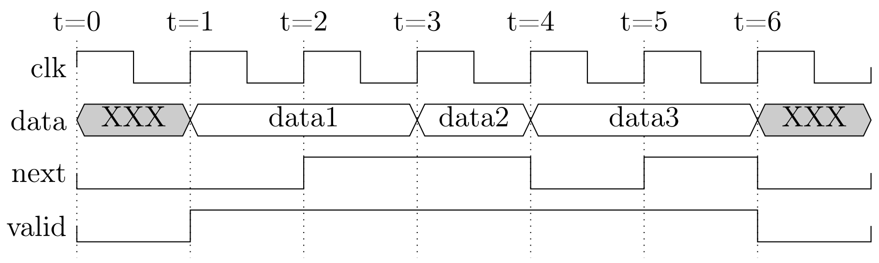

Figure 2.8: Time diagramm of the valid-next handshake utilized by the HICANN-ARQ. When the sender is ready to transmit data, it applies it to its outputs and raises a 'valid'-flag. Once the receiver has processed the data, it raises the 'next'-signal for a single clockcycle. Only then can the sender start to apply new data.

2.4 The L2-ARQ Wrapper Module

The HICANN-ARQ and FPGA-IF have been introduced into the Spartan-6 as components of a wrapping module. This L2-ARQ handles the initialization of the FPGA-IF using an internal state machine. It furthermore utilizes an internal Phase-locked-loop, an FPGA component designed to output clock signals with a customizable phase and frequency relation to a given input clock, to provide custom clock signals for the FPGA-IF. As of this moment, no modules are connected to the L2-ARQ. In later development of the Spartan-6, it will be provided with clients to transmit the given data to. For the time being, it is designed to loop data received by the HICANN-ARQ back into it, in order to transmit it back through the FPGA-IF-S6. Its basic setup, depicted in Figure 2.9, utilizes only one HICANN-ARQ and one FPGA-IF-S6 instance. After the FPGA's powerup, the internal state machine resets the HICANN-ARQ, and prepares the FPGA-IF for its initialization. The PLL provides the necessary clock signals for both modules, and once the initialization is complete, the FPGA-IF is capable of accepting data. Any data received by the FPGA-IF is looped back unchanged to the sender.

2.5 Performance Testing: The ReticleARQ

In order to evaluate the data connection between the two FPGAs, one needs to be able to easily determine the bandwidth of said connection. The ReticleARQ achieves this purpose while simultaneously providing universal access for up to 8 connections, depending on the application. It serves as a wrapping module for 8 HICANN-ARQ's, being able to simultaneously test the connection of all 8 links while in test mode, or manage their communication towards the host while idling. This is achieved using an internal state machine, whose mode can be changed using 2 initiation words being sent from the host towards the ReticleARQ. Figure 2.10 illustrates the internal setup of the ReticleARQ. The finite-state-machine possesses 2 main states, idle and PerfTest, and 2 transition

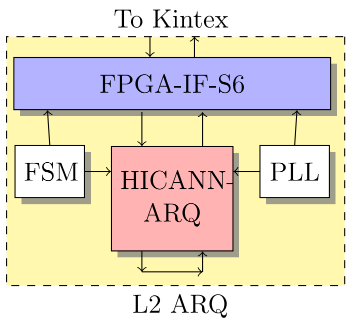

Figure 2.9: Internal Structure of the L2-ARQ wrapper. As of now, the HICANN-ARQ loops received data back to the Kintex-7. Both the FPGA-IF-S6 and the HICANN-ARQ are controlled by a self-initializing Finite State Machine (FSM). In the future, additional modules will be connected to the L2-ARQ and the FSM will be replaced by a JTAG port.

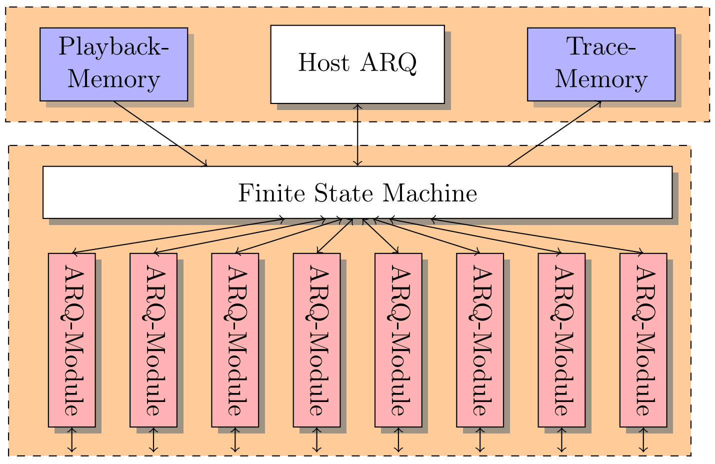

Figure 2.10: Internal structure of the Reticle-ARQ wrapper embedded in the FCP. An internal Finite State Machine manages the communication between the upper level modules and the embedded HICANN-ARQs.

states to navigate between the two. While idling, it organizes communication towards the upper level modules, handling their data towards its 8 embedded HICANN-ARQ modules if necessary. In case the host sends a 64-bit word targeted towards the Reticle-ARQ, the state machine switches to the first transition state. The word is interpreted to contain the HICANN-ARQ's to be utilized during the performance test. A second word containing the total amount of words to be sent during the performance test causes the statemachine to switch into the active PerfTest state. While in this state, words containing empty dummy-data are given to the HICANN-ARQ's specified in the first initiation word. These words are to be transmitted to the receiver, who attempts to send them back immediately. The state machine measures the time required to send and receive the specified amount of words, reporting them back to the Host when transitioning back to the idle state as soon as the Perftest is concluded.
The host is now able to calculate the bandwidth of the connection based on the clocking speed utilized in the FPGA-Design, and the number of words that were sent. This can not only be used to calculate the bandwidth of a single connection, but also for all of them at once. In this case the words get evenly distributed among the clients.

3 Testbench Verification

One central aspect of HDL design is the task of validating the functionality of the written code. This verification process eases the detection of errors and faulty designs. When the design has reached a certain size and communicates with other modules in the chip, the generated outputs are not trivially linked to the inputs anymore. For this reason tracing errors once the code has been implemented in a chip is a very difficult task. Hence one should validate the designs functionality as early and on as many levels as possible. This paradigm is known as 'test early, test often' in digital design.
The main tool used for verification is the so called testbench. This tool, used as a simulation in digital design, stimulates the inputs of the unit under testing, observing whether the outputs behave as expected. Simulation programs provide further methods such as a waveform analysis to help observe the designs inner workings, simplifying the tracing of errors immensely.
Before being implemented in the Spartan-6, the design presented in Chapter 2 was heavily tested and validated using multiple testbenches. First, it was simulated using a variation of the testbench used to verify the Reticle-ARQ. After that, it was implemented in a much larger testbench, which contained the design for the entire FCP communicating with the L2-ARQ. The HICANN-ARQ inside the L2-ARQ was tested previously to this thesis. The modified FPGA-IF itself was tested in another version of the Reticle-ARQ's testbench.

3.1 The Reticle-ARQ Testbench

The purpose of the Reticle-ARQ was to measure the time it took for a desired number of words to travel through a configurable amount of links. In order to emulate this, the original testbench used custom channel modules, which randomly dropped words traveling through it to simulate data corruption detected by the CRC. The setup depicted in Figure 3.1 uses HICANN-ARQ modules and First-element-in-First-element-out buffers (FIFOs) to loop data coming from the channel back to it, representing how the HICANN processes data and gives an answer to the client.

3.1.1 Validating the FPGA Interface

Validation of a design is best done on multiple levels. If one constructs a design comprised of a number of smaller modules, these components are best tested separately as well. This is called 'unit testing' and helps eliminating errors in the individual modules. Larger testbenches in turn ensure that the smaller modules are correctly connected to each other. The HICANN-ARQ is already tested, as it is implemented and in continuous in

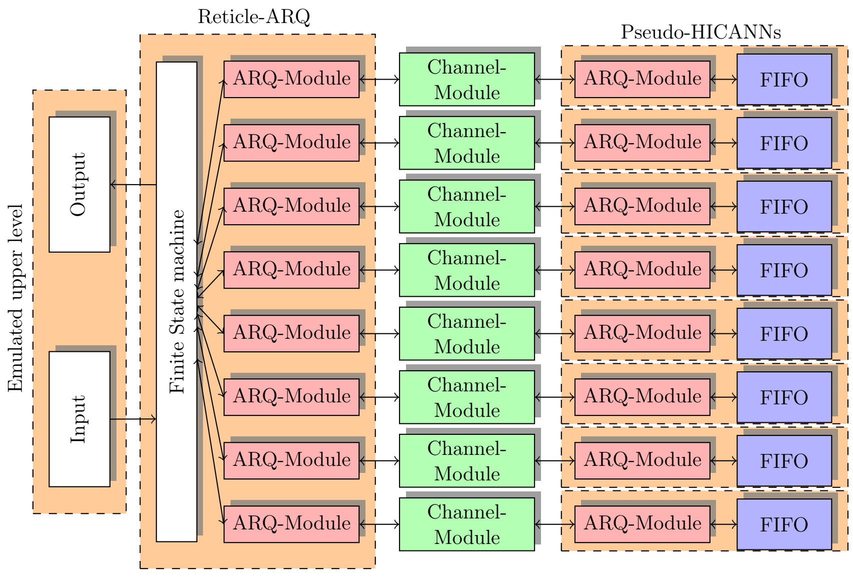

Figure 3.1: Structure of the testbench used to verify the Reticle-ARQ. The emulated upper level instructs the finite state machine inside the Reticle-ARQ to conduct a performance test using a specified amount of data. The data is evenly distributed among the targeted ARQ-modules, which start to transmit it over the Channel modules. The pseudo HICANNs, comprised of an ARQ module and a FIFO-buffer, recieve the data and loop it back.

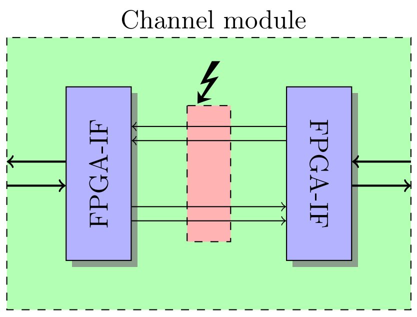

Figure 3.2: Structure of the Channel-Module used for the validation of the FPGA-IF. Both instances can receive data from an ARQ module (see figure 3.1), which they then serialize. The serialized data is tranmitted to the other FPGA-IF, and a random number generator can randomly disrupt the connection (⚡).

the FCP. As part of the aforementioned internship the Reticle-ARQ itself along with the FIFO and Channel modules were tested using the testbench depicted in Figure 3.1. The FPGA-IF, while already tested and implemented for the FCP, had to be validated after being ported to the Spartan-6. For this, the testbench used to originally verify the Reticle-ARQ's functionality was modified. The channel modules previously only emulated the SerDes behaviour by merely dropping words by chance with a given probability. This was replaced by 2 FPGA-IF modules communicating with each other, as depicted in Figure 3.2. A random number generator was placed between the two modules, and corrupted the data transmission between them with a given probabillity. The corruption occured by inverting singular bits of the signals, which disrupts the clock and data signals. In the first iteration, two copies of the FPGA-IF-S6 module were implemented and tested. The second version exchanged one copy with the FPGA-IF module integrated in the FCP. This verified that the FPGA-IF that was ported to the Spartan-6 was indeed capable to communicate with its FCP counterpart.

3.1.2 The DoubleArbiter Module

In later development, the physical connection between Spartan and FCP created by the FPGA-IF will be used to transmit both configuration and pulse data. Pulse data words contain information on neuronal spikes gathered during the experiment. They are also sometimes sent from the Host-PC to the HICANN in order to excite the neuron models residing on it. Previously no module existed to coordinate their transmission, resulting in collisions and packet loss (*Karasenko* (2014)). Alongside the construction of the L2-ARQ a new module called the doubleArbiter was designed to handle this issue. Implemented in the testbench as part of the channel module, it buffers both incoming data types, ensuring only one word is transmitted at any given time. Utilizing an internal counter, it prioritizes incoming pulse data, while granting a minimum bandwidth to the configuration data communication. Whenever a pulse and a configuration word are to be transmitted simultaneously, the configuration word is buffered, while the pulse word is passed on. In case of the counter reaching 0, the configuration word is passed on, while the pulse word is buffered. After this, or in case there

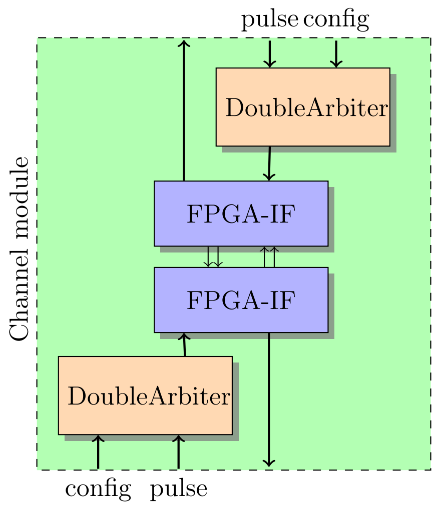

Figure 3.3: Implementation of the DoubleArbiter module inside the channel. Pulse and input packets can be applied to the inputs simultaneously, yet only one is passed on to the FPGA-IF.

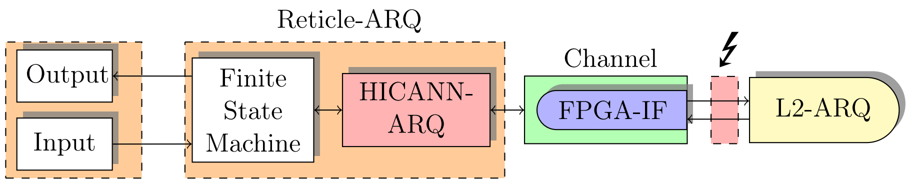

Figure 3.4: Structure of the L2-ARQ-Testbench. The channel module has been halved so that it contains only one FPGA-IF. Its counterpart resides within the L2-ARQ, the signals between the two are again disrupted randomly (↯).

is no pulse word to transmit, the counter is reset to its starting value n. This ensures a minimum bandwidth of 1 configuration word per n pulse words sent.

3.1.3 Validating the L2-ARQ

The testbench of Section 3.1.2 was further modified and used to validate the L2-ARQ's basic functionality. The new setup shown in Figure 3.4 removed the lower HICANN-ARQs and FIFOs to replace them with the L2-ARQ. Furthermore, the channel module was halved, and given a FPGA-IF counterpart to communicate with the L2-ARQ. Data corruption is again emulated by a random number generator, inverting bits of the clock and data signals. The testbench is capable of 2 modes of operation. In the first testmode the already verified Reticle-ARQ is ordered to conduct a performance test. This can be used to determine the theoretically possible speed of the connection to compare it with the real-world experiment later. One can also observe the links behaviour under high load. The second testmode sends ascending data and random data through the Reticle-ARQ. Data is transmitted and looped back by the L2-ARQ. The testbench then checks if the data arrived back in the same order it was sent, printing an error otherwise.

3.2 The HMF-FPGA Testbench

The last verification layer consisted of a testbench utilizing the entire design of the FCP FPGA, called the HMF-FPGA-TOP. Since in later experiments it would only communicate with the Spartan-6 over the L2-ARQ, only the L2-ARQ has been instantiated in this testbench. It can be controlled using C++-programs which are used for the physical system as well. As such, this testbench represents the closest possible emulation of the physical test setup later used for the conducted experiments. Its structure is visualized in Figure 3.5. Again the HICANN-ARQ embedded into the Reticle-ARQ transmits data over the FPGA-IF. The L2-ARQ receives this data and sends it back. The major difference between this testbench and the testbenches presented in Section 3.1 is that the Reticle-ARQ itself is controlled via the modules residing in the FCP. A SystemC interface

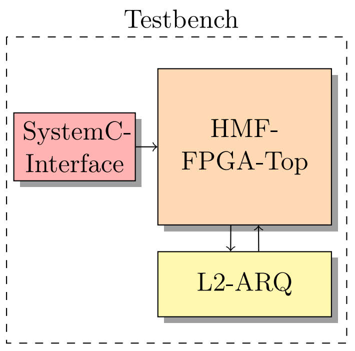

Figure 3.5: Structure of the HMF-FPGA-Testbench. The HMF-FPGA-Top design later implemented in the FCP communicates with a SystemC interface, which can be controlled using user-written C++ programs. It furthermore communicates downwards with the L2-ARQ, which represents the Spartan-6's functionality.

is also part of the testbench. It can be controlled using programs written by the user, and communicates with the FCP using an emulated ethernet connection. This causes an interaction with the Reticle-ARQ closely resembling the real-world-example.
One might question why extensive testbenches like this are not used exclusively, and why the testbenches mentioned in the previous sections are necessary. While it certainly requires a lot of time and effort to construct the amount of testbenches mentioned here, their use has two main advantages. The first is the simplicity of testing. The main reason behind testbench validation in general was already mentioned; tracing errors once a design has been loaded into the FPGA is very difficult. The same problem arises with large testbenches; if a problem is encountered, its cause can be hidden deep in the design. A second advantage lies in the time time required to validate a design. An increased testbench size naturally equates to a higher simulation time. If the error only occurs after a significant amount of simulation time, attempting to remove it takes all the longer. Errors like these can be fixed much faster in smaller testbenches.
After using the testbenches mentioned, we can safely assume that our design has been validated. We deem it error free for the situations we expect it to be utilized in.

3.3 Simulation Results

While mainly used for design verification, the presented testbenches can be used to simulate and measure the behaviour of the unit under testing. As we would like to have referential measurements to compare our experiment data to later on, the testbenches have been used to determine the theoretically possible and expected bandwidths for our design. In its current version the SerDes inside the FPGA-IF is specified to operate at a clock frequency of 500 MHz. It is capable of double data rate (DDR) communication,

sending 2 bits of information during one clock cycle. Thus, it transmits data with a bandwidth of 1 Gbit/s. One has to take into account that the FPGA-IF applies a 16 bit header that includes the CRC to every 64-bit word that is sent. This means that only 80% of the SerDes communication is used to actually transmit data. Therefore the resulting maximum possible bandwidth for the FPGA-IF communication is 800 Mbit/s, or 64 bits sent every 80 ns.
These values only apply if no data corruption occurs. This is not the case in real-world applications, so we set a corruption probability of $p_{\text{corrupt}} = 0.0005/\text{bit}$ for our simulation, which is quite high compared to the drop rate observed during testing with the HICANN. We performed performance tests of 100000 words, and recorded the time it took to send and recieve them. The time required to initialize the link before any data can be sent was observed to be about 1100 μs. We determined the average time for a single word to be sent as 81 ns. Thus, we expect the average bandwidth for our tests to also be in the range of 790 Mbits/s.

4 Operation and Testing

After verifying the design using the testbenches described in chapter 3, the next step was to implement the design into the Spartan-6. In order to do so, a bitfile was created. When loaded into the FPGA, this file changes the internal components to behave the way specified in the HDL design. Furthermore, to enable communication between the FPGA's, a physical connection had to be established. After these steps the communication between the Spartan and a FCP had to be observed.

4.1 Experimental Setup

The Electronic Vision(s) group already utilizes Spartan-6 FPGAs for internal projects. As such, boards connecting the FPGA to a socket already exist and were utilized for this section. A smaller mobile version of the BrainScaleS system exists under the moniker "Cube Setup" incorporating four FCPs and connector boards for single HICANNs. Due to restrictions in the placement of connector pins in the FPGA designs, a custom connector had to be build. Korbinian Schreiber assisted in the soldering of this cable which connects a male and a female plug to serve as a bridge between the two FPGA boards. The experimental setup is depicted in Figure 4.1, a close-up picture of the connector is shown in Figure 4.2. Four FCPs are present in a cube setup, however only one was used for the experiment. When performing a test with a HICANN chip, a circuit board is placed between the FCP and the HICANN. This board supplies the HICANN chip with power, and connects the two components correctly. In our setup, this circuit board was not used since the Spartan FPGA is powered by an external USB port. However, if one desires to connect the Spartan to one of these boards instead, the cable can be separated in the middle. This makes it possible to connect a plug compatible to the circuit board.

The bandwidth was observed using the Reticle-ARQ module embedded into the FCP. It was controlled using a custom script written by Christian Mauch, capable of performing multiple consecutive tests.

4.2 Error Detection and Performance Improvements

In order to create a bitfile to load onto the FPGA, the mapping tool planAhead was used. It allocates resources of the FPGA to structures specified in the design, and configures said resources to behave as required. It does this regarding the constraints imposed by the FPGA's structure, printing an error should the design not be routable.
While attempting to create a bitfile using said tools, multiple problems were encountered. The existing schematic after which the designs were connected to the FPGA's pins had

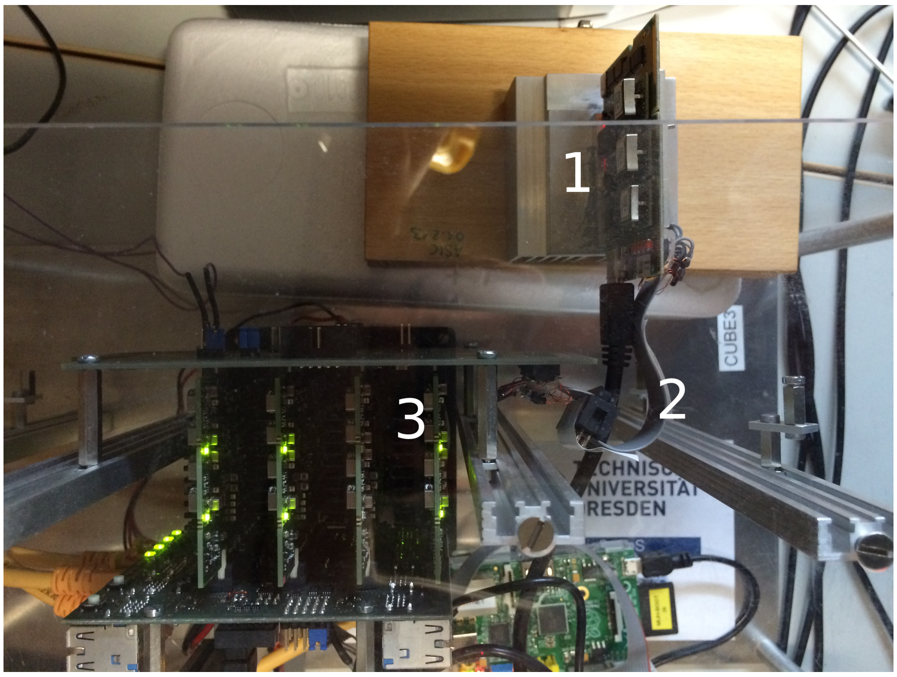

Figure 4.1: Picture of the experiment setup inside a 'Cube'. The Spartan-6 is mounted on a circuit board (1) which is cooled using aluminum plates. It is connected to the FCP inside the Cube (3) via a two part connector cable (2).

to be edited. The L2-ARQ communicates with the FCP over differential pairs of input and output pins, carrying the data or clock signals. These pins are located on several 'banks', sides of the FPGA which connect to the physical cables on the board. The first design intended to have the data input pins on a different bank from the input pins, but this was not possible. Instead, pins originally intended for a second differential data output pair were reconfigured to act as input pins. Another problem had to do with the Phase-locked loops. Originally it was intended to utilize a new PLL in parallel to one already in use, but they had to be connected in series to make the design routable.
After these changes, the bitfile was compilable, and loaded into the Spartan-6. However, no communication between the two FPGAs was observed. The error was traced back to the software testmode. It was based on the initialization of the Spartan-6, which had previously been used as a master during operation, but was required to be a slave in the SerDes initialization protocol.
Now communication between the two FPGA's was observed, yet it was highly unstable, requiring almost a minute to send a single 64 bit word and only working in 10% of attempted performance tests. Another attempt at improving communication was made by implementing a termination resistance for the input pins. This is normally done for all input connections in the LVDS communication, but since output pins intended for other connections had to be used as inputs, this detail had been missed. After terminating the input pins the performance improved a little, as multiple words could be sent before a

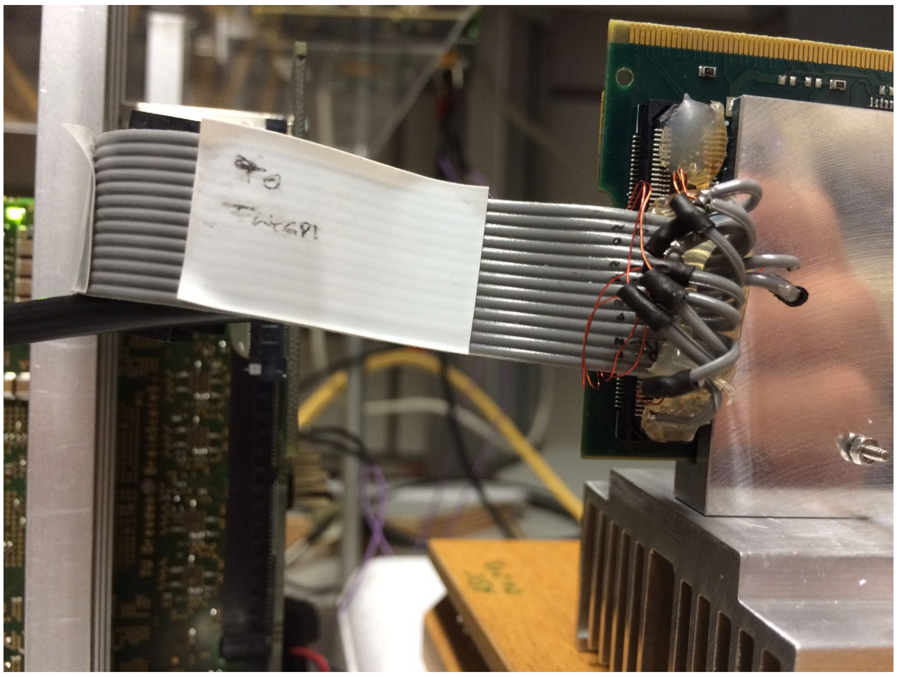

Figure 4.2: Picture of the custom soldered cable connecting the FCP and Spartan-6 FPGA.

timeout occurred.

4.3 Measuring the Bandwidth

In order to evaluate the connection between the two FPGA's, the custom script was used. It begins with the initialization of the FCP, including its FPGA-IF. Once the connection is initialized, the program initiates a performance test and reports the results. Using this program, we observed extremely low bandwidths, sometimes sending a single word of 64 bits length would cause a timeout. This timeout was issued when a response had not been heard for more than 60 seconds. As performance tests with more than 10 words would time out more often than not, only tests with a maximum amount of 10 words were conducted.
The first test was an observation of the bandwidth over time. The program issued a performance test for 1 word every minute, and repeated this 1000 times. The time dependency of the communication speed was plotted in Figure 4.3. The average bandwidth was observed as 23 bits/s, the maximum reached was 1010 bits/s. Another test was conducted, this time using 8 words per performance test. The average bandwidth was observed as 36 bits/s, while the maximum was 246 bits/s. Its time dependency is visible in Figure 4.4.
The very low average together with a very high variance prompted the need to collect more data for further investigation. We executed a performance test for 1 to 10 words

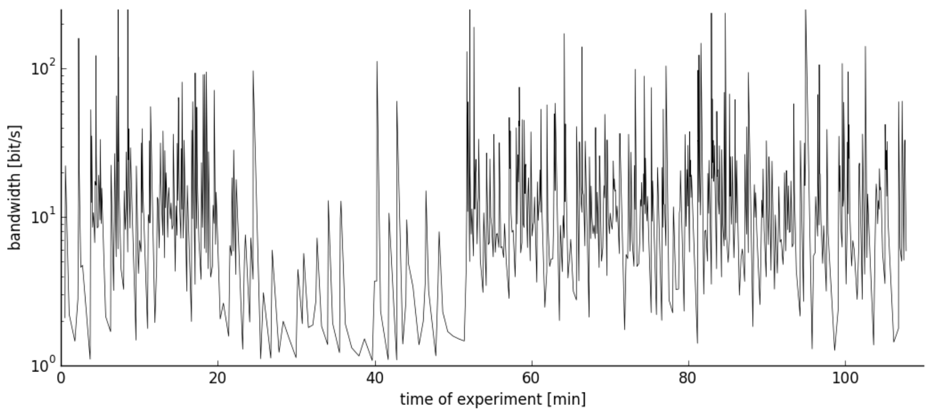

Figure 4.3: Communication bandwidth for single-word communication plotted over time. One can clearly see the large fluctuation in the transmission speed.

with 1000 trials each. The distributions for the reported times are shown in Figure 4.5. Each point represents a single trial with the coordinates denoting the time it took to send that amount of words. The color is an indicator of the trial-to-trial variability. It encodes the relative frequency of occurrence for the particular perftest result. It should be noted however, that there were a lot more timeouts i.e the perftest took more than 60s to complete at larger experiment sizes. This effectively reduces the number of samples by a significant margin, however we calculate the colorplot correctly by normalizing to the number of samples in each bin. The timeout probability is listed in Table 4.1
The plot shows that the average time to execute longer performance test is roughly proportional to the number of transported words. At the same time, the trial-to-trial variability increases with the experiment length.

4.4 Conclusion

As Section 4.3 shows, the established connection between Spartan-6 and FCP did not meet our expectations. The time required to transmit a single word was several orders of magnitude higher than what we observed in simulation. Since the simulation we conducted resembled the test setup as closely as possible, the error has to lie in factors out of reach for the simulation.
We believe that the main cause for this issue must be the physical link. The demands on the cable increase with transmission frequency because dynamic effects start to come into play. Whenever there is a change in impedance in the signal path there is a reflection which superimposes on the original signal. Following

$$r = \frac{Z_L - Z_0}{Z_L + Z_0} \tag{4.1}$$

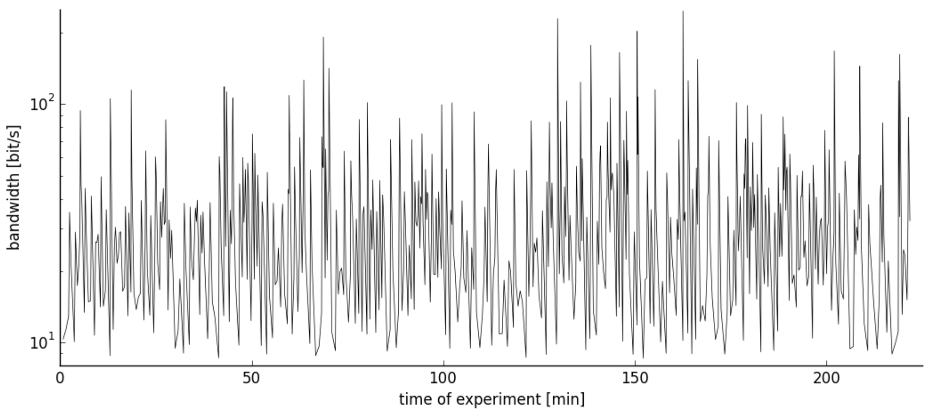

Figure 4.4: Communication Bandwidth for an eight-word communication plotted over time. Similarly to Figure 4.3, the measurements show a high amount of fluctuation.

with Z_L and Z_0 representing the impedances at a connection, an incoming signal is reflected with an amplitude of $U_r = U_0 \cdot r$ *Bogatin* (2004). Considering the fact that multiple impedance jumps are present in the link, for example at the connectors, we assume that the suboptimal bandwidth is a result of a high data corruption rate caused by reflection noise of the physical link. This is further emphasized by the operation frequency for the communication. Operating at $f_{signal} = 500\,\text{MHz}$, the estimated wavelength for a signal in a copper cable is around

$$\lambda = \frac{C_{\text{Copper}}}{f_{\text{signal}}} \tag{4.2}$$

$$= \frac{2 \times 10^8\,\text{m/s}}{5 \times 10^8\,\text{Hz}} \tag{4.3}$$

$$= 0.4\,\text{m} \tag{4.4}$$

The utilized cable in our experiment was about 15 cm long, setting these in the same order of magnitude. This means that stationary waves are likely to form at least temporarily. These kinds of effects are normally mitigated by terminating the receiver endpoints as we have tried during testing. The observed small improvements can be explained by assuming that the effective impedance of the connector was quite far off the $100\,\Omega$ that the internal termination resistors possess in the FPGA.

The high rate of data corruption caused by these distortions can be explained when on recalls the working principles behind a flip flop. These 1 bit storage units are paramount for data storage in an integrated circuit. The LVDS data arriving at the Spartan 6 is stored in flip flops, and once stored is read from them in the next clock cycle. However, for a flip flop to be functioning properly certain constraints have to be met. Flip flops require a clock signal to detect the data that shall be stored in them. The current data

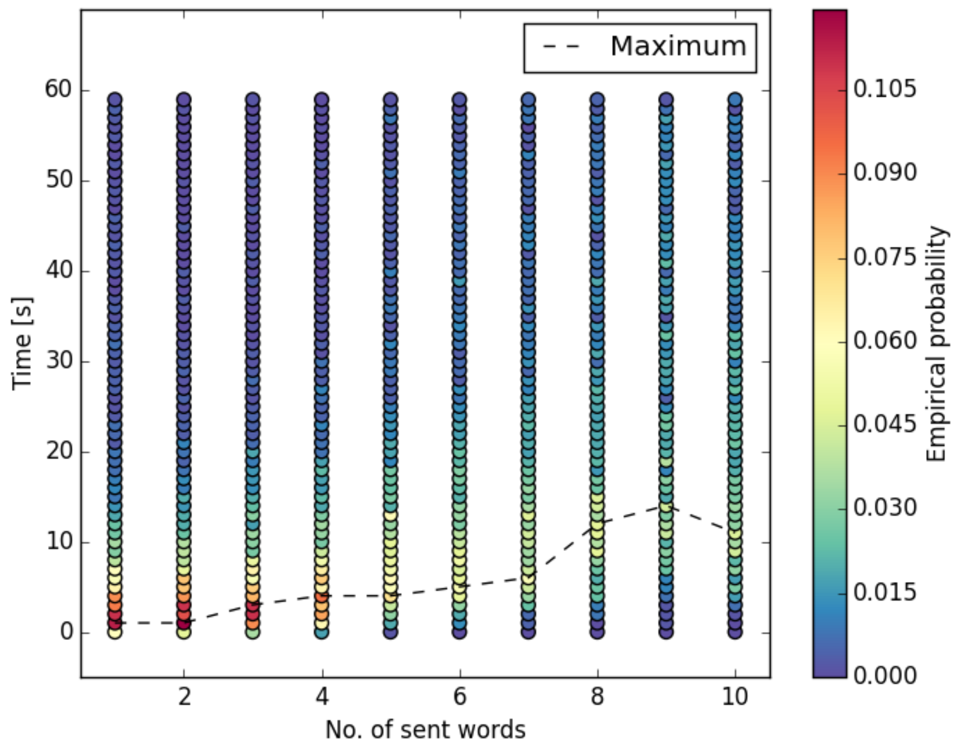

Figure 4.5: Times required to receive a response plotted against the number of words sent in each measurement. The colour denotes the relative amount of data points that were measured in the specified one second time interval. The relative maxima for each amount of words have been indicated using a black dashed line.

value applied to the flip flop is accepted during the rising edge of the clock signal. In order to be properly stored, the input has to be held steadily for a certain interval before and after the rising edge of the clock. Depicted in Figure 4.6, this time interval is known as the aperture and consists of the setup time t_{su} and the hold time t_{hold}.

The reflections existing in the cable could affect this time interval in multiple ways. A superposition of the main signal with its reflections can cause the resulting signal to be unstable during the aperture. The reflections could also influence the clock signal in such a way that a data value is detected to late or too early. It is also possible that the clock edge is disrupted in such a way that the receiving flip flop cannot detect it as such. These are just some possible issues which could cause the flip flop to accept a false or unusable value. The CRC-check of the FPGA interface would detect this false information, resulting in a packet loss.

The second possible reason is that the FPGA-IF takes an immense amount of time to

No. words	Timeout %
1	12.8
2	1.6
3	1.5
4	3.5
5	17.0
6	30.6
7	41.7
8	47.3
9	56.5
10	59.6

Table 4.1: Timeout probability for performance tests depending on the amount of sent words.

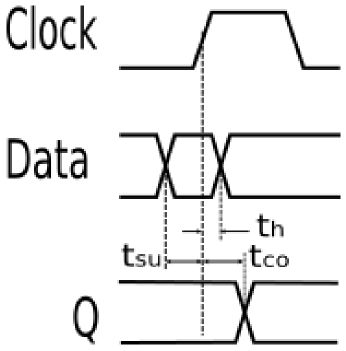

Figure 4.6: Timing diagram of a flop data acquisition cycle. A data value applied to the flip flop needs to remain stable for a setup time t_{su} before and a hold time t_{hold} after a rising edge of the clock signal. Only then can the flip flop correctly output the value. Picture taken from *Michagal* (2007).

initialize before it can be used for communication. As mentioned in Chapter 2, the FPGA-IF initializes in order to configure the time when it choses to detect a data value. The reflections in the physical link can cause the clock and data signals to shift wildly. Thus, the algorithm that determines the optimal point at which data should be detected can take very long, and is sometimes not able to terminate.

4.4.1 Estimating the Drop Probability

The flip flop capturing failures caused by the noisy cable effectively result in a Bernouilli process that randomly corrupts individual bits on the wire. This is exactly the kind of corruption that has been modeled in the simulations. However, the corruption probabilities used during verification were never so high as to significantly reduce the throughput. Using the data in Figure 4.5 we can estimate the drop probability in the current setup. We begin by noting that executing a performance test for one word has a quite sharp peak at about two seconds duration. Since the performance test actually loops the data, we experience the faulty link twice. Assuming symmetry, this means that it will take around one second to sequentially transmit 80 bits through the link without a single corruption. Since the HICANN-ARQ has a resend timeout of around 8 µs, that translates to about 125000 resends until successful transmission. A small calculation yields

$$p_{corrupt} = 1 - \left(\frac{1}{125000}\right)^{\frac{1}{80}} = 0.136/\text{bit} \tag{4.5}$$

This probability is about 300 times larger than what we have used in simulations.

Attempts have been made at slowing the clockspeed for the operation in the hopes that this might reduce the amount of data corruption. But whereas the Spartan-6 is easily configurable to operate at lower frequencies, the design for the FCP cannot be changed that easily. It would require an enormous amount of changes in the software used to operate the FCP, as well as changes to the designs in the FCP themselves, to enable a slower operation. Furthermore, the aim was to establish communication with as little changes to the original setup as possible. Thus, the clockspeed remained unchanged.

5 Discussion and Outlook

5.1 Summary

The main goal of this thesis was the construction of a HDL module capable of establishing a high-speed data connection between two FPGAs. A design similar to the communication interface used by the Electronic Vision(s) group in a FCP has been constructed. Using multiple simulation testbenches, the design's functionality has been validated, indicating that it is working as intended. It has been implemented into a Spartan-6 FPGA, and communication between said Spartan-6 and a FCP has been observed. The speed of the established communication fluctuates heavily, and reaches an average speed of around 23 bits/s. However, the design has been observed performing at speeds of 1 kBit/s, and extensive testing suggests that these slow speeds are the result of a faulty physical connection between the FPGAs due to electrical effects. Simulations suggest that exchanging said link with a cable capable of carrying signal frequencies of around 500 MHz will greatly improve the reached communication speed. Analysis of the constructed bitmap yields a 10 % consumption of the Spartan-6's resources, leaving a lot of space for further designs inside. While not performing in its theoretical limits, the design was proven functional.

5.2 Improving the FPGA-Implementation

As of this point in time, the design inside the Spartan-6 is initializing itself on powerup, given that the FCP-side of the connection is being initialized as well and transmitting a training pattern. For further experiments with the Spartan-6 one will most likely desire a feature to reset the connection using a host-driven command. This would require connecting the FPGA Interface over a JTAG side channel, as it is done already for the HICANN chip. This communication chain is used in multiple scripts to control the configuration of the FPGA-components during the FPGA's startup.
Furthermore, it might prove useful to revisit the design's routing in the Spartan-6. As of this moment, certain FPGA-primitives like a Phase-locked-loop are used exclusively for the L2-ARQ, and posses multiple clock outputs that are unused. More PLLs with unused outputs are used for other components in the FPGA's design, and some could be combined to optimize utilization of the FPGA's resources.

5.3 The Adapter Cable

The main problem of the current setup lies within the physical link connecting the two FPGAs. In the short timespan available for this thesis, it was only possible to construct a mere prototype. The resulting communication observed over this adapter cable was much slower than simulations predicted, and suffered from a very high variation in speed. We suspect that this is a result of a high amount of data corruption, caused by noise in the LVDS-connection due to impedance jumps along the cable. These impedance jumps caused by the variation in conductor material of the custom soldered cable make the connection unfit for communication at the required frequencies. Replacing it with a connection capable of carrying these frequencies with very little noise compared to the LVDS voltage amplitude will most likely result in a much higher bandwidth closely resembling the bandwidth observed in the current FCP-HICANN communication. One possibility for such a connection would be a circuit board that connects the two FPGAs.

5.4 Establishing the Spartan-FPGA Prototyping Environment

While a connection between the two FPGAs has been established, this is only the first step towards building a suitable testing environment inside the Spartan-6. A first approach would be the implementation of an OCP-based communication protocol as designed by *Friedmann* (2013). This would enable the HICANN-ARQ to communicate with all other clients in the Spartan-6. As of this moment, the connection has been implemented via a generic ARQ-protocol using a module created by *Karasenko* (2014). In the event that the physical connection between the two FPGAs is improved, this module can be exchanged for other versions. The most notable example would be the Stream-ARQ designed by Gaetan Deletoille, which achieves higher net communication speeds at lower corruption rates. Since both modules are designed to be easily interchangeable, little effort would be required for this adjustment.

List of Acronyms

ARQ Automatic Repeat Request protocol. 6, 9

ASIC Application-Specific Integrated Circuit. 1

BEC Backward Error Correction. 9

CRC Cyclic Redundancy Check. 5–7, 14

FCP FPGA Communication PCB. 3, 7–10, 12, 14, 16–18, 20–23, 27–29

FPGA Field Programmable Gate Array. 1–3, 5, 7, 11, 13, 17, 18, 20–22, 25, 28, 29

FPGA-IF FPGA Interface. 3, 5, 7–9, 11, 14, 16–19, 22, 25, 27

FPGA-IF-S6 Spartan 6 FPGA Interface. 7, 12, 16

HICANN High Input Count Analog Neural Network chip. 1–3, 14, 16, 19, 20, 28, 29

HICANN-ARQ HICANN Automatic Repeat Request protocol. 3, 5, 9–14, 17, 27, 29

IC Integrated Circuit. 3

PCB Printed Circuit Board. 3

Bibliography

Bogatin, E., *Signal integrity: simplified*, 280–290 pp., Prentice Hall Professional,, 2004.

Friedmann, S., *A New Approach to Learning in Neuromorphic Hardware*, University of Heidelberg, 2013.

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

Linear77, Elimination of noise by using differential signaling, https://commons.wikimedia.org/w/index.php?curid=18321195, 2016.

Michagal, Flip-flop setup, hold and clock-to-output timing parameters, https://en.wikipedia.org/w/index.php?curid=13677277

Peterson, L. L., and B. S. Davie, *Computer Networks: A Systems Approach*, 3rd ed., Elsevier Science (USA), 2003.

Schemmel, J., and A. Hartel, *Specification of the HICANN-DLS ASIC*, Heidelberg University, Heidelberg, Germany, 2015.

Acknowledgments

Prof. Dr. Karlheinz Meier for the chance to contribute to the work of a group such as this, and for his organization and supervision which form the foundation of our work.

Dr. Johannes Schemmel for his technical management of the group's inner workings, and for the motivation to improve at anything that we do.

Vitali Karasenko for his supervision and endless patience when having to listen to my everyday worries and struggles.

Korbinian Schreiber for constructing the cable used to connect my setup. Without him not a single test would have been possible.

Christian Mauch for writing the custom script used to start performance tests, and for helping me with every little problem when attempting to use said script.

Andreas Hartel and Andreas Grübl for explaining the intricacies of FPGAs and the mapping tools used to configure them.

Mihai A. Petrovici, Laura Kriener, Sebastian Billaudelle, Sarah Bartha and Ingmar Middelhoff for proofreading and assistance during my time spent working for this thesis.

The entire Electronic Vision(s) group for an incredibly fun and entertaining workplace during my time here.

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, June 16, 2017

....
(signature)