Department of Physics and Astronomy

University of Heidelberg

A Parametrizable Switch for Neuromorphic Hardware

Bachelor's Thesis

submitted by

Lea Sophie Kanzleiter

born in Nürnberg, Germany

October 2018

This Bachelor's Thesis has been carried out by

LEA SOPHIE KANZLEITER

at the

Kirchhoff Institute for Physics in Heidelberg

under the supervision of

Prof. Dr. Karlheinz Meier

OCTOBER 25, 2018

ABSTRACT

The Electronic Vision(s) group designs neuromorphic chips that model the temporal behaviour of a neuron's membrane potential as an analog circuit. In application, analog neuron models lead to large accelerations in the order of $\sim 10^3$ to $\sim 10^5$ compared to biological time. Therefore the communication infrastructure of the chip needs to handle correspondingly large amounts of data. In the course of this thesis a HDL Switch module was developed to overcome the macroscopic distances between the on-chip event and configuration buses and the read-out links. The parametrizable module accepts data from a set of input connections and transfers it uniformly to all output links. It is suitable for an arbitrary number of input and output connections. HDL simulation results confirm a significant improvement compared to the currently implemented solution.

ZUSAMMENFASSUNG

Die Electronic Vision(s) Gruppe entwickelt neuromorphe Chips, welche das zeitliche Verhalten des Membranpotentials eines Neurons als analoge Schaltung modellieren. In der Anwendung führen analoge Neuronenmodelle zu großen Beschleunigungen in der Größenordnung von $\sim 10^3$ bis $\sim 10^5$ im Vergleich zur biologischen Zeit. Daher muss die Kommunikationsinfrastruktur des Chips entsprechend große Datenmengen verarbeiten können. Im Rahmen dieser Arbeit wurde ein HDL-Switch-Modul entwickelt, um die makroskopischen Abstände zwischen den chipinternen Ereignis- und Konfigurationsbussen und den Ausleselinks zu überwinden. Das parametrisierbare Modul übernimmt Daten von einer Reihe von Eingangsverbindungen und überträgt sie gleichmäßig auf alle Ausgangsverbindungen. Es ist für eine beliebige Anzahl von Ein- und Ausgangsanschlüssen geeignet. Die Ergebnisse der HDL-Simulation bestätigen eine deutliche Verbesserung gegenüber der aktuell implementierten Lösung.

Contents

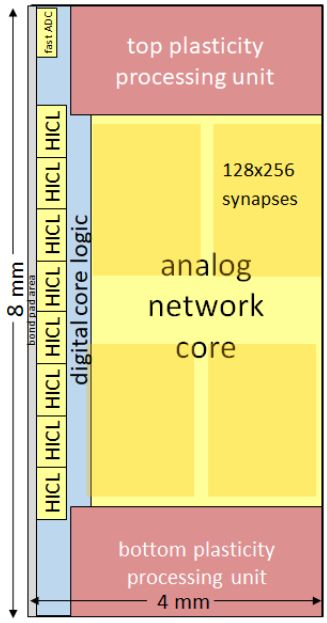
Chapter 1

Introduction

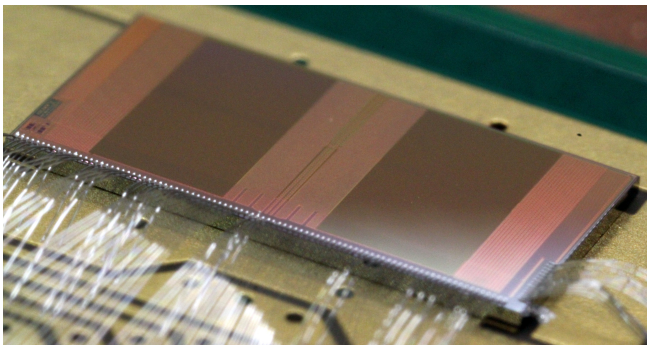
1.1 Neuromorphic Hardware

Neuromorphic computing represents a large interdisciplinary field of research. The possible applications are diverse, ranging from medicine to artificial intelligence. Aspiring to a deeper understanding of the processes within the human cortex, many approaches towards recreating its structure and behaviour have been developed over time. In general, two different strategies emerge when designing artificial neural networks. The first is recreating the evolution of a neuron's membrane potential in simulation on a computer with a digital neuron model. The contrary approach however, is creating a physical neuron model. In the Electronic Vision(s) group (*BrainScaleS Research Project*) as part of the Human Brain Project (*Human Brain Project*) at the University of Heidelberg, the neurons as the brain's constituents are emulated as individual analog electric circuits (SCHEMMEL, FIERES, AND MEIER, 2008). The time continuous evolution of the neural network reaches an acceleration factor between $\sim 10^3$ to $\sim 10^5$ as opposed to the biological real time. Observing processes in the brain that in vivo require large time spans, is made possible within small experiment runtimes. Neuromorphic networks developed within the Electronic Vision(s) group are realized as custom ASIC (Application-Specific Integrated Circuit) neuromorphic chips.

As a successor of previously developed HICANN (High Input Count Analog Neural Network) versions (SCHEMMEL ET AL., 2010), the most recent neuromorphic chip designed in the Electronic Vision(s) group is called HICANN-X. Figure 1.1A shows a schematic of the chip. Implementing an AdEx (AAMIR ET AL., 2017) neuron model, the chip contains $128{\cdot}10^3$ synapses for a total of 512 neuron compartments. The analog neurons evolve autonomously in a continuous time domain as opposed to a digital neuron implementation that works with discrete time steps. The hardware neurons' speed-up allows the generation of large amounts of spike events in a short time span. An essential characteristic of the analog neuron implementation is that spike events are not reproducible and all spike data needs to be processed immediately by the chip's communication infrastructure. The HICANN-X connects to



(A) Schematic of the HICANN-X neuromorphic chip (SCHEMMEL, 2018).



(B) The HICANN-X chip (photo taken by R. Achenbach), corresponding to the schematic in (A).

FIGURE 1.1: The HICANN-X neuromorphic chip uses the AdEx neuron model (AAMIR ET AL., 2017) to emulate neuronal behaviour through analog circuitry. The neuron events are generated in the analog core and read out and further processed in the digital logic core.

an FPGA (Field Programmable Gate Array) via eight links marked as HICL in the schematics in figures 1.1A and 1.2. Data generated within the chip is further distributed via those links.

1.2 Motivation

The events generated on the HICANN-X can be categorized into two types. The neuron blocks produce spike events as a result of neuron activity, which are transferred between the event routing block and the digital logic core in figure 1.2. Furthermore, configuration buses also input their data into the digital logic core. The several configuration buses are further collectively referred to as Cfg. The events are generated on-chip with a certain traffic distribution. The highest rate at which messages can be sent matches a clock frequency of 250 MHz. For further processing, the data needs to be transported to several output channels marked as HICL links in the schematic, that are spread out over macroscopic distances as indicated in figure 1.1A. Overcoming large distances on hardware in a small time span generally demands for lower clock speeds or requires large currents which result in high power consumption. Hence, distances that are covered in one clock cycle should preferably be rather small. The overall goal of this thesis was the development of a

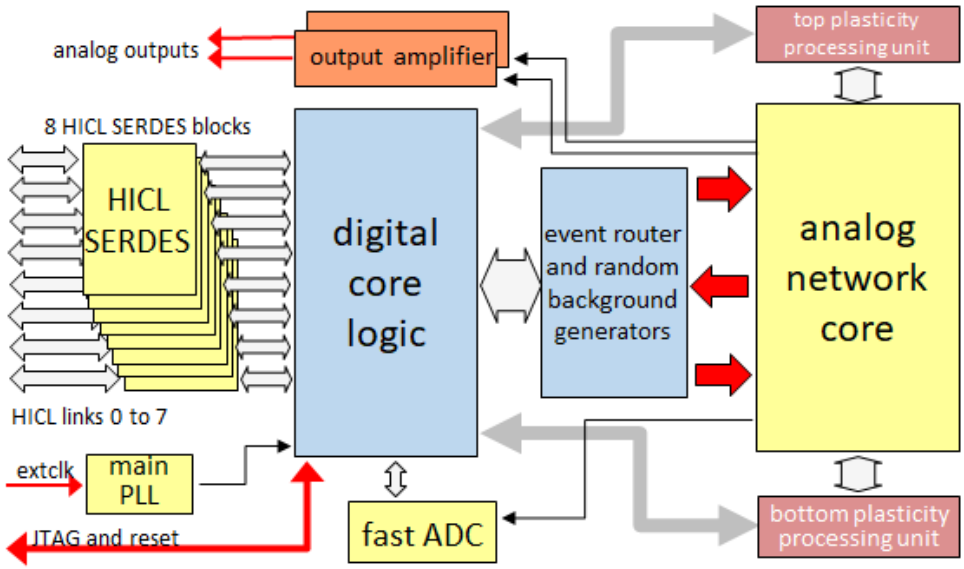


FIGURE 1.2: Block diagram of the HICANN-X neuromorphic chip (SCHEMMEL, 2018). The spike events generated in the analog network core are distributed to the eight HICL links via the event router and the digital logic core. The HICL channels further process the spike event data.

module that evenly distributes the input messages to all available output links while also splitting the covered distance into several localized buffer stages. Those allow the short-time storage of messages and implement data transfer. In the following, the developed module will be referred to as 'Switch'.

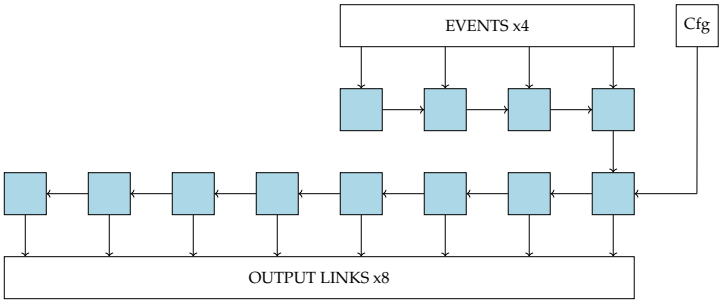


FIGURE 1.3: Schematic of an application of the Switch on the HICANN-X chip. The module is used in the digital logic core. Four event buses transport the spike events from the chip's neuron block and another channel transfers the configuration payload. These five channels serve as input links for a row of buffer stages (marked blue), which transfer data along the chain. The data needs to be evenly distributed to the eight output links for further processing. These links represent the HICL links in figure 1.2. When transferring data from the input to the output links, the outputs are prompted sequentially until a free link can accept and process the data.

Current Solution Consider the setup shown in figure 1.3 which represents the currently used solution for this problem. The input and output links are only connected

via a chain of buffer stages. Any message arriving at the inputs needs to travel sequentially along the entire chain until the next free output link is able to process it. However, a sequential prompting of readout links causes high duty cycles, which means large time spans between the generation of an event at the input links and its processing by an output link. While providing a valid solution, the network as shown in figure 1.3 has significant structural disadvantages leading to an uneven distribution of data and a blocking of input links. These issues will be discussed in chapter 4. To compensate for those, an improved structure is required.

Developing a New Switching Network While the main objective for developing a new switching network still is to overcome the distance induced limitations when processing data, another aspect is the demand for an even use of the hardware resources. Referring to the links that form the Switch's outputs, a fair distribution should allow the handling of a large amount of data. An equal use of all links needs to be guaranteed for using the outputs to their full capacity. Load balancing is one of the essential tasks of the neuromorphic chip's communication infrastructure, so an improvement of the switching network can help towards an overall performance gain. Maintaining flexibility for an arbitrary number of input and output links allows a usage for multiple different applications. Hence, a newly developed network needs to be parametrizable. Another factor that needs to be considered is the space to be occupied on hardware. Therefore, the network's size should be adaptable as well.
As an internship project, a suitable switching network was developed by extending the already existing structure. The network was visualized and simulated using Python in KANZLEITER, 2018. Its analysis showed a remarkable performance gain compared to the currently implemented solution. Since the Python framework only provides a software simulation that is not applicable to the hardware, the newly developed network structure was adapted in this thesis and implemented in RTL.

1.3 Thesis Outline

In the course of this thesis, the implementation of the switching network and its analysis will be covered. Chapter 2 first gives an overview on the submodules required to build the Switch, such as the so-called ut-interface as connector and the Shifter module that implements the buffer stages. The Shifter's internal structure and the temporal behaviour will be focused on as they influence the Switch's performance significantly. The last section in chapter 2 focuses on the Switch module, its parameters, the connection pattern that determines the network's shape and its implementation.
Following the general module specifications, the next chapter contains a description

of the testing environment which is required for simulation. Chapter 4 then explains the experimental setup for the simulation and the analysis plots for different bandwidth conditions. The simulation results can also be found in chapter 4. The last chapter serves as a summary and puts the project results into perspective for future application.

Chapter 2

The Switch Module

The overall intention of this thesis is the development of an efficient switching network for neuromorphic hardware. As mentioned above, the network is meant to form a connection between a set of input and output links, both of which should be parametrizable in size. The components that make up the Switch work as buffer stages that allow the sending and receiving of payload and are further referred to as nodes. For establishing a network structure each node is linked to either its next neighbouring nodes in the network or to an input or output channel. The complete process of connecting the nodes is explained in section 2.4.2. As an example, a visualization for a possible network configuration is shown in figure 2.1.

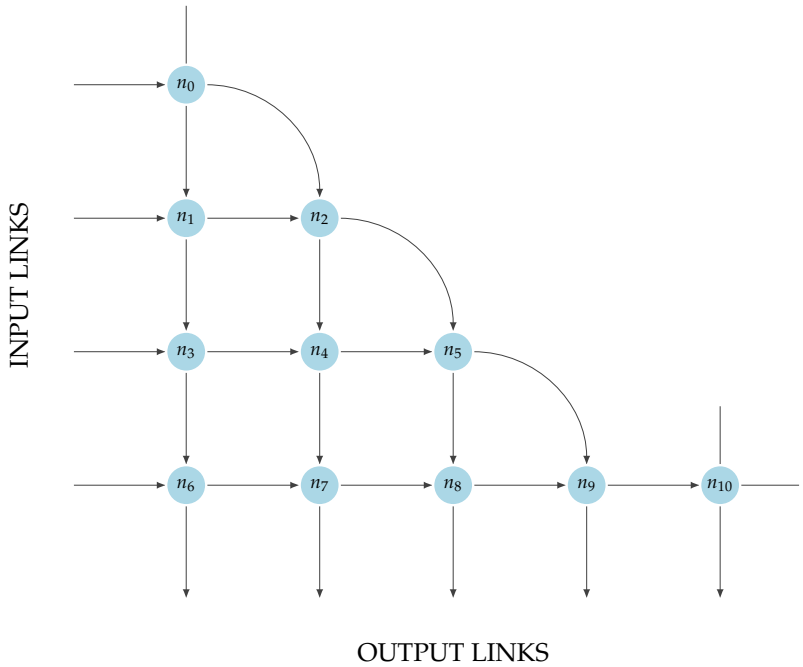


FIGURE 2.1: Visualization of a network for 4 input and 5 output links using 11 nodes. Links without arrows represent dummy connections that don't support data transfer (section 2.4).

This chapter focuses on the constituting elements and their implementation as well as the structural characteristics of the Switch. The last section explains the Switch's implementation.

2.1 Module Overview

The Switch itself is the project's top level module that connects the nodes to form a network. Within the Switch, the nodes are implemented as so-called Shifter module and the connections as ut-interface. The Switch itself is designed for both synthesis on an FPGA that connects to the HICANN-X neuromorphic chip as well as on-chip implementation in future HICANN versions.

2.2 ut-Interface

The ut-interface serves the purpose of interconnecting the Shifter instances to to their next neighbours. At the same time it provides the signals that are necessary for data transfer. In total it contains four signals:

- *valid*: signalizes if the corresponding Shifter holds valid data
- *next*: communicates if the Shifter is able to accept data
- *idx*: provides information about the payload's origin
- *data*: contains a message

In the following, the data signal and the index will collectively be referred to as payload.
Using the *valid* and *next* signal, the payload transfer is realized as a handshake between the involved Shifters as explained in section 2.3.2. As the sending and receiving sides of the Shifter use the same interface, the declaration of port directions must be adaptable. Being written in SystemVerilog, the ut-interface allows the user to bundle several signals of different directions.
The directions are specified in two modports which are referred to as `slave` and `master` which is shown for an arbitrary example module in figure 2.2.

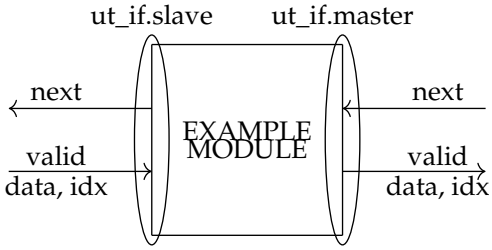
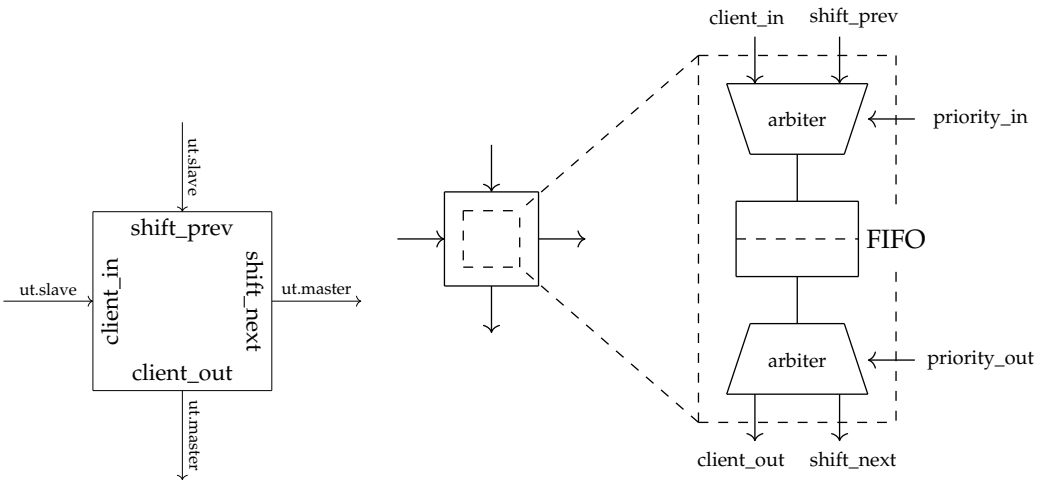


FIGURE 2.2: Usage of the ut-interface modports to specify the signal directions. For the `slave` modport, the *valid* signal and the payload are declared as inputs, the *next* signal as an output. For the `master` modport the directions are reversed.

The declaration of the Shifter's ports as slave or master is shown in the schematic in figure 2.3A which will be explained in the next section.

2.3 Shifter

The Shifter module represents a buffer stage between the input and output channels, which allows the short-term storage of data. It also implements the functionality of payload transfer and steers the data flow within the Switch. Consequently, it defines the network's dynamics. This section gives an overview over the Shifter's internal structure and further explains the process of payload transfer.



(A) Each Shifter is assigned two incoming connections called `client_in` and `shift_prev` to receive payload from. For sending payload it possesses to two outgoing connections called `client_out` and `shift_next`.

(B) The schematic shows the internal structure of the Shifter depicted in (A). It shows the arbiter for the input connections which chooses one link per clock for receiving payload. The payload can then be accepted into the Shifter's two-stage FIFO and sent to one of the output clients. At this point the arbiter selects the corresponding output connection.

FIGURE 2.3: Schematic of the Shifter's top level connections and its internal structure.

2.3.1 Internal Structure

On the top level, each Shifter connects to four ut-interfaces, two of which serve as inputs, the other two as outputs. According to that, input ports are declared as `ut_if.slave` and output ports as `ut_if.master`. As can be seen in figure 2.3A, the incoming connections are called `client_in` and `shift_prev` while the outgoing connections are referred to as `client_out` and `shift_next`. Concerning all following network schematics, the `client_in` and `shift_next` connections will be represented by a node's horizontal connections and `client_out` and `shift_prev` by the vertical connections as shown in figure 2.3A.
Internally, a two-stage FIFO allows the module to store up to two messages (figure 2.3B). Within one clock cycle, each Shifter is allowed two transactions, which depend on its FIFO's state. The first transaction is receiving data from one of the incoming `ut_if.slave` connections, `client_in` or `shift_prev`. To carry out the transfer an input client's request must be accepted. This is only possible with a non-full FIFO.
The second possible transaction is sending valid payload to one of the `ut_if.master` connections, `client_out` or `shift_next`. Given that the FIFO is not empty, a transfer request can be sent to the selected client. The system of requesting and accepting a transfer is called valid-next-handshake and is discussed in the following section. The selection of a client to use as handshake counterpart is marked as arbiter in figure 2.3B and is explained in section 2.3.3.

2.3.2 Handshake

Transferring payload between two Shifters always requires a completed valid-next-handshake. Consider the case of two connected Shifters, one as sender and one as receiver. Given that the sender's FIFO holds valid payload and hence is not empty, it can request a transfer of its payload to the receiving Shifter by raising its *valid*-flag to a value of 1. In case that the receiver's FIFO is not full, it can accept the payload and raises the *next*-flag for the sender. Afterwards the handshake is completed. For further transfer of the message to one of the receiver's output clients, the two-stage FIFO causes a propagation delay of one clock cycle.

2.3.3 Arbitration

One of the Switch's essential characteristics should be an even distribution of payload. To achieve that it is crucial not to disadvantage any of the Shifter's clients, since within one clock cycle each Shifter can send and receive only one message

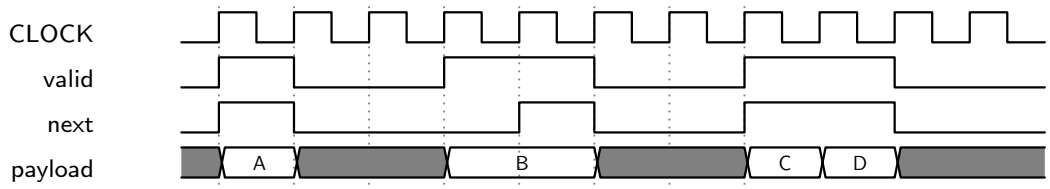


FIGURE 2.4: Examples for three supported types of transactions. The first one shows an immediate transfer of the payload. The second example covers the process if the receiver's FIFO is full until the next clock cycle and the third one shows the back-to-back transfer of two different messages.

each. Consequently, an arbitration is needed for the two input and two output connections as marked in figure 2.3B. For that purpose, a priority for one input and one output client is set at the beginning of each clock cycle. The general concept is remembering the partner for the last payload transaction and setting the priority to the other client respectively. Subsequently, the Shifter will invoke a handshake for the two prioritized connections which yields two cases. In the first case the handshake is completed as described in section 2.3.2. As a result, the priority will be set to the other client for the next transfer. In case of an incomplete handshake, the non-priority port will be considered within the same clock cycle. This situation can be caused by a missing *next*-response in a handshake with an output client or if the prioritized input client stores no valid payload.
In figure 2.5, the arbitration is described as pseudo code for the input and output clients, respectively.

The arbitration process guarantees an alternating pattern for transferring payload. Considering the Switch as a whole, this method not only distributes the payload evenly through the whole network, but also prevents the favouring of any of its input links. This is essential as for some network structures it could result in some of the input links being blocked.

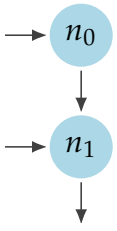


FIGURE 2.6: Schematic of a network for $n_{in} = 2$, $n_{out} = 1$ and $n_{nodes} = 2$. The two incoming connections to the nodes link to the inputs while n_1 is connected to an output link.

Consider the network displayed in figure 2.6 without any arbitration process. We assume that the input link connected to the node n_1 produces new data each clock

```
function priority_out(prio_out,
    client_out, shift_next):

    retval = prio_out
    if(prio_out==0)
        if(client_out.next)
            retval = 1
        else if(shift_next.next)
            retval = 0

    else
        if(shift_next.next)
            retval = 0
        else if(client_out.next)
            retval = 1

    return retval
```

(A) The priority prio_out defines the order of clients for the Shifter's payload transfer request. Its value for the next transfer is dependent on its previous value as well as the client which the Shifter transfers payload to. The next priority value is returned as retval with a value of 0 representing the `client_out` and a value of 1 the `shift_next` port.

```
function priority_in(prio_in,
    client_in, shift_prev, full):

    retval = prio_in
    if(prio_in==0 && !full)
        if(client_in.valid)
            retval = 1
        else if(shift_prev.valid)
            retval = 0

    else if(prio_in==1 && !full)
        if(shift_prev.valid)
            retval = 0
         else if(client_in.valid)
            retval = 1

    return retval
```

(B) The priority prio_in defines the order in which the Shifter responds to the input clients in case they hold valid payload and the Shifter's own FIFO is not full. The FIFO's state is represented by the variable `full`. analog to case (A), the next prio_in value depends on the client which the Shifter receives payload from. The next priority value is returned as retval with a value of 0 representing the `client_in` and a value of 1 representing the `shift_prev` port.

FIGURE 2.5: Pseudo code representation of the arbitration process within the Shifter.

cycle. If there is no specified arbitration process and the node n_1 always prefers its `client_in`-connection, node n_0 gets no opportunity to forward its data. Therefore the input link connected to that node is blocked as long as the other link produces data.

2.4 The Switch Module

The Switch as the project's top level module implements the network of Shifters. Following a certain connection pattern, it adapts the results found during the internship previous to this thesis (KANZLEITER, 2018). The first part of this section explains the required parameters for the Switch, the second part focuses on the connection pattern that leads to an efficient network and lastly, the algorithm for the network generation is discussed in detail.

2.4.1 Parameters

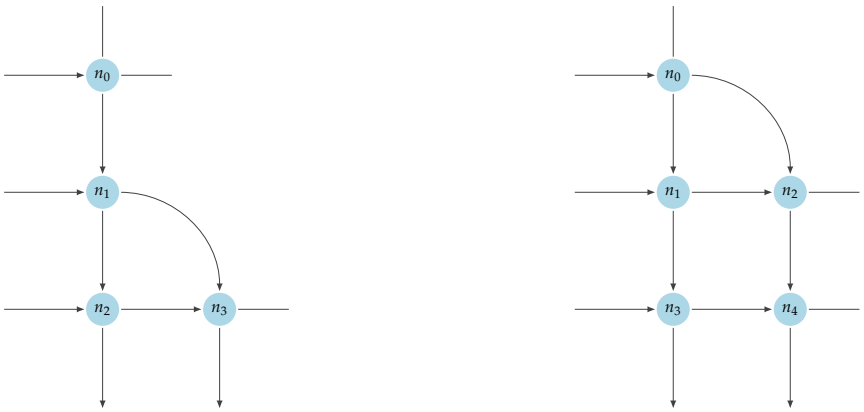
The defining factors for the Switch's size and shape are the parameters that need to be specified before synthesis. The number of input links n_{in} limits the number of rows in the finished network. The number of output links n_{out} defines the number of columns. These two parameters can be adapted to allow the module's use for different applications. The third parameter is the number of nodes n_{nodes}. It determines the network's size and defines its shape and also scales the occupied space on hardware. For n_{nodes} the allowed values range between a minimum n_{min} and a maximum value n_{max}. The minimum node number fulfills the requirement that each input and output need to be connected to a node in the network. Since one node always links to one input and one output at the same time, the minimum node number that is necessary is

$$n_{min} = n_{in} + n_{out} - 1. \tag{2.1}$$

When generating the network for a maximum number of nodes, connecting the last Shifter of the incomplete first row to the last Shifter in the next row replaces a node in the upper right corner.

$$n_{max} = n_{in} \cdot n_{out} - 1 \tag{2.2}$$

An example for the minimum and maximum node configurations are displayed in figure 2.7.



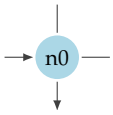
(A) Minimal node configuration with $n_{nodes} = n_{min} = 4$. n_{min} guarantees to link all inputs and outputs to a node in the network.

(B) Schematic for the maximum number of nodes $n_{nodes} = n_{max} = 5$. Connecting the last Shifter of an incomplete row to a Shifter in the next row replaces a node in the upper right corner. This connection can always be made for the maximum node number.

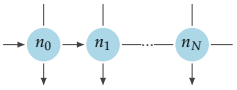
FIGURE 2.7: Schematics of the minimum and maximum node configurations for an example network with $n_{in} = 3$ and $n_{out} = 2$.

2.4.2 Connectivity

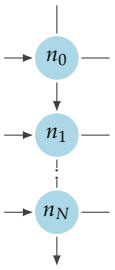
To form the network structure each Shifter connects to up to four ut-interfaces in total according to figure 2.3A. First, the Switch's input and output links are connected. The inputs link to the first element in each row while the outputs link to the Shifters in the last row. The restrictions for n_{nodes} in the above section guarantee those connections, even for the smallest configuration with n_{min}.
Furthermore if possible, a Shifter port should be connected to the Shifter's corresponding next neighbour in the network. In case a Shifter is the last instance in its row, the `shift_next` port can also be linked to the `client_in` port of a Shifter in the next row. This is only allowed if the current Shifter's row is not full. Examples for that can be found in figures 2.8D to 2.8F on the following page.
All Shifter ports that are still unconnected will be linked to dummy ut-interfaces, which do not transfer any payload and for that purpose have clamped signals. Dummies used instead of input clients have their *valid*-signal fixed to zero, while for interfaces used instead of output clients, the *next*-signal is set to zero respectively. This prevents the interaction via a handshake.
Considering these rules, the simplest network possible yields an L-shaped structure. This case is shown in figure 2.8D. The minimum node number is required so that each input and output link can be connected to a Shifter in the network.
When increasing the number of nodes n_{nodes}, the Shifters are added diagonally to the minimal network. Therefore the Switch's shape ranges between an L-shape for n_{min} and a rectangular shape for n_{max}.
Since the tests during the internship demonstrated that the node in the upper right corner is never used, it is omitted and not instantiated in this module. An example of the shape for n_{max} is shown in figure 2.8F.



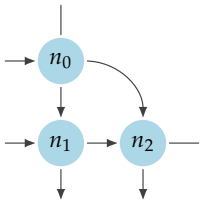
(A) Special case of $n_{in} = n_{out} = n_{nodes} = 1$. The input and output link are both connected to the same Shifter. Since the single Shifter has no neighbouring partners, the other two ports need to be assigned dummy interfaces.



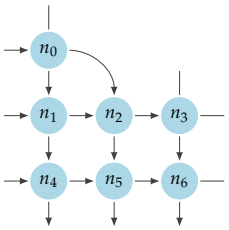
(B) In this case the first node connects to an input and an output link since all Shifters are arranged as a single row. It is linked to the input and the first output channel, the difference to (A) being the existence of at least one neighbouring Shifter in the horizontal. Each Shifter in this network is assigned a dummy interface as `shift_prev` connection.



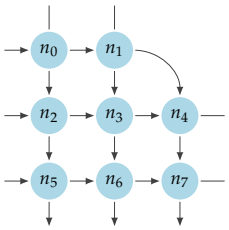
(C) This structure is the opposite arrangement to the network shown in (B). It consists of a single column, where all Shifters are assigned dummy interfaces as their `shift_next` port. The difference for the duplex node compared to (A) are the vertically neighbouring Shifters.



(D) The smallest possible network for more than one input and output is shown here. Since the first row with n_0 is not entirely filled, its `shift_next` port can be assigned as input connection for a Shifter in the next row.



(E) Visualization of a slightly larger network. As the Shifter in the top row can only serve as input to its closest Shifter in the next row, n_3 is assigned a dummy interface instead. As it is the last instance in a full row, the `shift_next` port connects to a dummy as well.



(F) The network represents the maximum number of nodes for $n_{in} = n_{out} = 3$. Each Shifter in the top row is assigned a dummy interface as `shift_prev` connection. The same applies to the Shifters in the last column and their `shift_next` ports respectively.

FIGURE 2.8: Visualization of the distinction between different cases when connecting the Shifters to form the desired structure. Concerning all of the above networks, the input links are always connected to the Shifters in the first column and the output links to the ones in the last row. For all cases a dummy interface is assigned to the last node in the network as `shift_next` connection and to the first node as `shift_prev` connection.

2.4.3 Implementation

Summarizing the above section, the different kinds of ut-interface connections can be divided into five categories:

- input links
- output links
- connections between neighbouring nodes
- input dummy interfaces
- output dummy interfaces

For each of these, an array of ut-interfaces is instantiated. Within the network all connections are numbered as shown in figure 2.9.

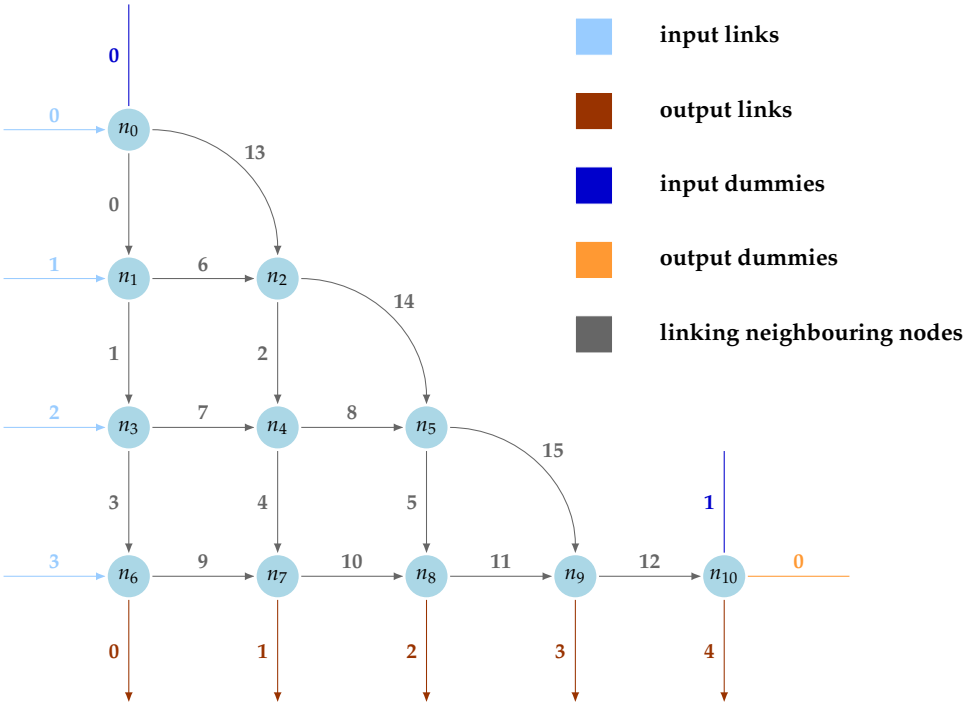


FIGURE 2.9: Indexing system for a network with $n_{in} = 4$, $n_{out} = 5$ and $n_{nodes} = 11$. The colours represent the different ut-interface arrays. The connection numbers are used to assign an element of one of the five arrays to the corresponding ports. Hence, the numbers are used as array indices.

The first step for generating the Switch is determining the individual Shifter's position within the network. This specifies the types of connections needed. The different cases are shown in figure 2.8. In general, the algorithm checks which of the five categories applies for each port. Determining the distinct cases was one of the most

challenging tasks during the development of the module. An example for that is the comparison between figures 2.8A to 2.8D. In these plots, the node on the bottom left needs to connect to an input and an output link. The difference between the plots are the other two connections as in some cases an input dummy and/or an output dummy or no dummy at all is needed. These kinds of distinctions need to be made for all Shifters.

To fully create the network, a SystemVerilog generate loop instantiates a Shifter for each index. By if-statements the Shifter's position is analyzed. Depending on that, four indices are calculated to assign each port to an element of one of the five interface arrays. This process leaves the Switch without unconnected ports.

Chapter 3

Testbench

A fundamental task when designing a RTL module is testing it and verifying the desired behaviour in simulation. In the field of digital design this is done using a testbench. Written in a Hardware Description Language, it is used to create a testing environment that is specific to a certain module and allows for different aspects to be validated. The testbench's main functionality is to emulate the input signals and read out the outputs. Most simulation tools also provide several features for debugging the code, as they facilitate the observation of the individual module constituents such as the ut-interfaces and Shifters for the Switch.
The following sections provide an overview over the features implemented in the testbench.

3.1 Emulation of Input Data

When simulating a RTL module, the goal is to verify its behaviour under different conditions and to comprehend possible malfunctions. To achieve expressive results, the testing environment should not only approximate the real application as close as possible but also allow simulation runs for a variety of parameters. In case of the Switch, the customizable parameters are n_{in}, n_{out} and n_{nodes}, which remain constant after compilation. The testbench then instantiates the module which creates the network structure and connects it to artificial input and output links. A schematic of the testbench setup is shown in figure 3.1. Since the transfer of payload is the Switch's main functionality, a data flow from the inputs to the outputs must be invoked. The generation of test data is implemented in the testbench, as well as the capability of the outputs to receive data. The payload generation process will be explained in section 3.1.2. To control the inserted amount of data, some restrictions for the input bandwidth are introduced in the following section.

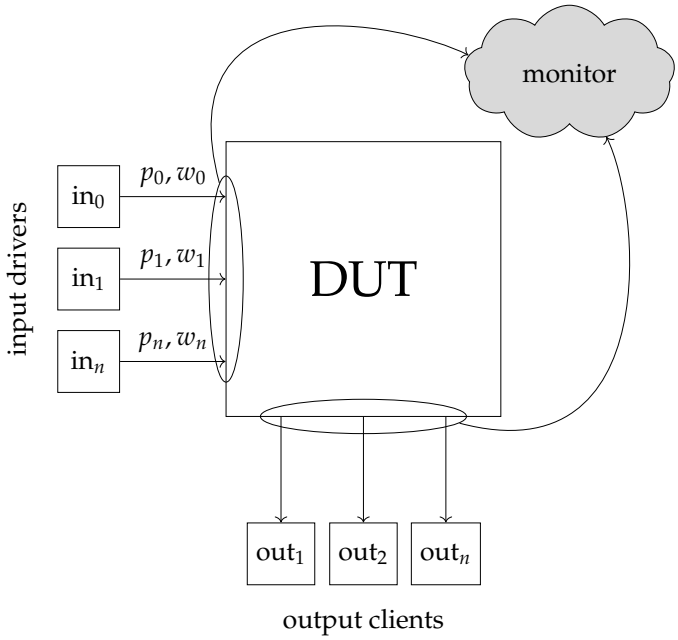


FIGURE 3.1: Testbench setup for the Switch module which is marked as device under test (DUT). The generation of event data is emulated by the input drivers depending on the weights w and the activation probability p. The output clients represent the links that accept and process data in a certain time span depending on the input weights. The number of occupied output links as well as stalling input links are monitored for each clock cycle.

3.1.1 Bandwidth

An important aspect of the Switch is the congestion of output connections that receive messages of a certain width. Until a message is fully processed, the output is not accessible for new data. When reproducing this behaviour in simulation, after each payload transfer to an output, it will be set to a non-responsive state for a certain amount of time. Corresponding to the message's origin, the duration of the output's busy state equals a fixed weight in clock cycles. Each input link is assigned a weight w_i in a randomized process and produces valid data with an activation probability p_i per clock. The average bandwidth inserted into the Switch therefore equals

$$N_{in} = \sum_{i=0}^{num_{in}-1} p_i \cdot w_i. \tag{3.1}$$

Assuming each output processes a weight of one per clock, this yields an output bandwidth N_{out} of

$$N_{out} = n_{out}. \tag{3.2}$$

3.1.2 Input Data Generation

Setting the bandwidth N_{in} to a fixed value before compilation determines the activation probabilities p for the following simulation run. The radomized generation of weights and all calculations are executed at elaboration time. To simulate incoming events on the HICANN-X chip, the inputs' *valid* flags are controlled in the testbench. Each clock cycle, an input i's *valid*-flag is raised with a probability p_i. The corresponding input remains in a valid state until its payload is accepted into the network. In case the valid-state lasts for more than one clock cycle, the input is logged as stalling. After the valid-state ends, another random choice for the *valid*-signal is made based on p_i.
The stalling input occurrences are counted each clock cycle and a stalling input's index is noted to verify that the stalling events are distributed evenly across all inputs.

3.2 Recording Output Data

An important quality of a switching network is the fair use of all output links. This guarantees the best possible use of the available bandwidth. To monitor the output occupancy, the number of outputs in a busy-state is logged for each clock cycle.
After the experiment's runtime, all tracked data is written into text files for further analysis.

Chapter 4

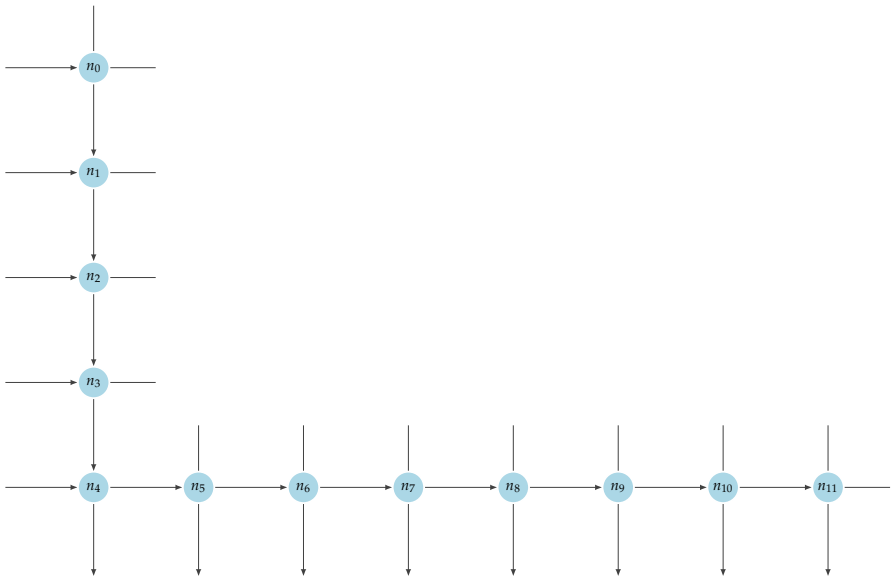
Analysis

In order to devise a new Switch architecture that satisfies the constraints described in 1.2 several node positioning and connection patterns were designed during the internship (KANZLEITER, 2018). Comparisons showed the best results for the layout which was then adapted in this thesis. For the development process during the internship, a software framework was set up in Python to visualize the structure and to simulate the dynamics. This chapter explains the experimental setup for the Python simulation and briefly summarizes the relevant results. Furthermore, the RTL implementation of the Switch is analyzed.

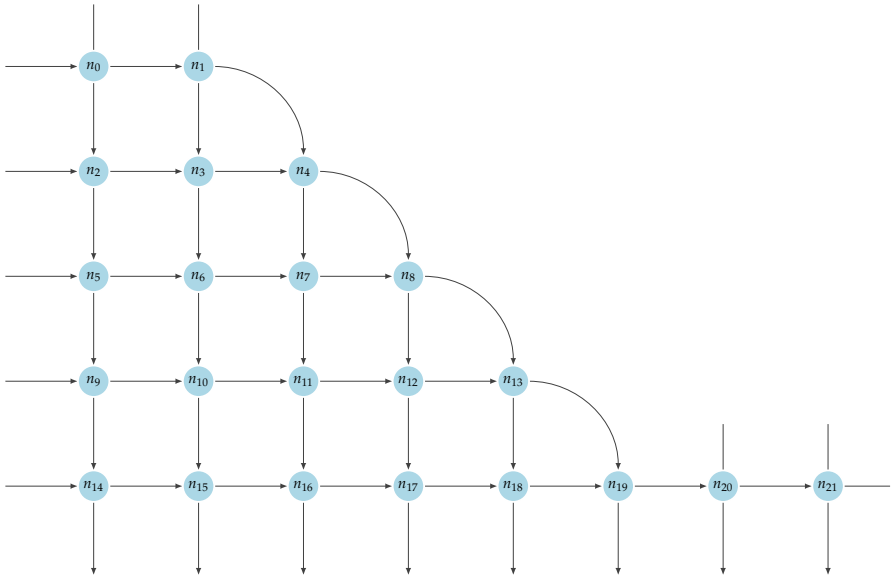
4.1 Python Simulation Analysis

4.1.1 Python Simulation Setup

The analysis in KANZLEITER, 2018 provides the background information for understanding the following discussion. Therefore the main results will be summarized briefly. Additional to the implementation of a Python simulation framework for visualizing the Switch's connection scheme, the network dynamics were analyzed. Observing the temporal evolution of the output links' busy states provided information on the overall usage of the available output bandwidth N_{out}. Assessing the occurrences of a stalling effect at the inputs when generating valid data, helped understanding structural disadvantages. In order to prove the efficiency gain for the newly developed Switch, it was compared to the currently implemented solution.
The testing parameters for both cases were set to $n_{in} = 5$ and $n_{out} = 8$ to resemble a potential application on the HICANN-X chip (figure 1.3). The comparison counterpart to the minimum network with $n_{nodes} = 12$ (figure 4.1A) was determined in a series of tests. Parameter sweeps for n_{nodes} yielded $n_{nodes} = 22$ (figure 4.1B) as most efficient solution for the given n_{in} and n_{out}.



(A) Network structure for the currently implemented solution with $n_{nodes} = 12$. The nodes are connected as a chain which provides only one possible path for data being forwarded to the outputs. As a result, the occupancy of the outputs strongly depends on the inputs' weights.



(B) Newly developed structure with $n_{nodes} = 22$. As most nodes are connected to four active partners(no dummies), more than one path to the outputs is created. This results in a significant efficiency gain.

FIGURE 4.1: Visualization of the currently implemented solution (A) and the newly developed structure (B) with $n_{in} = 5$ and $n_{out} = 8$.

4.1.2 Python Simulation Results

4.1.2.1 Current Solution: $n_{nodes} = 12$

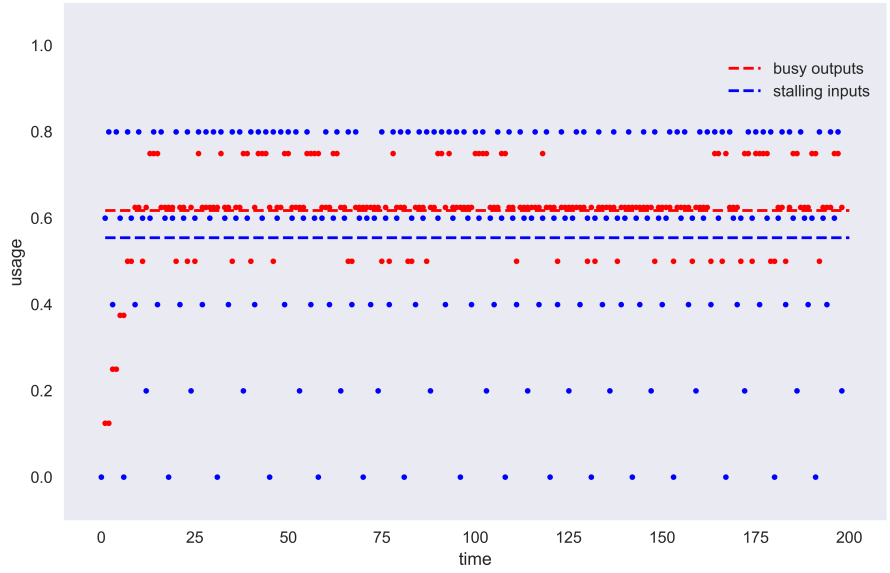
When analyzing the minimal network for three different bandwidth scenarios, it was obvious that with increasing input bandwidth the Switch's performance worsened as a result of the network structure. The analysis plot for a bandwidth of $N_{in} = 2 \cdot N_{out}$ is shown in figure 4.2A. The ut-interface that serves as `shift_prev`-connection for the duplex node $n4$ in the corner (figure 4.1A) causes a bottleneck. As there is only one possible path to forward the input payload to the output links, each message inevitably passes through this connection. As each output is set to a busy state while processing a message, any following payload will be forwarded horizontally towards the next output link. Since the longest possible duration of an output's busy state equals the maximum weight w_{max} (section 3.1.1), the highest possible number of occupied output links also matches the highest weight.
For input bandwidths smaller than w_{max}, the impact of this structural issue on the performance is rather small. For bandwidths larger than w_{max}, this causes more stalling occurrences at the inputs while not using the full output bandwidth. Overall, an output bandwidth of at least

$$N = n_{out} - w_{max} \tag{4.1}$$

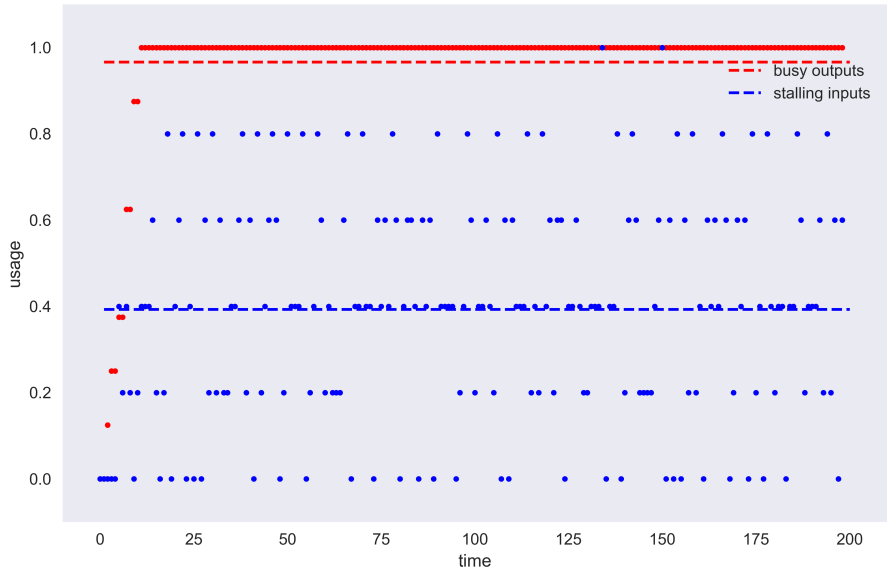
remains idle, independent of the input traffic.

4.1.2.2 New Structure: $n_{nodes} = 22$

In contrast, the performance increase when observing the $n_{nodes} = 22$ network in figure 4.2B is significant. The spatial layout and the connection pattern assure more than one possible path for the payload propagating through the network. In the case of a stalling in the vertical direction, the payload can be distributed horizontally at an earlier time. Input links that would have been blocked in the smaller network configuration can now insert their payload into the Switch. The horizontal distribution also allows the use of all output links without changing the overall highest weight. Replacing the current solution with a bigger, more complex network enables a usage of the output bandwidth very close to 100%. Any gain in the percentage of stalling inputs when increasing the input traffic is justified as the margin of output bandwidth left is already small.
Concluding, the network architecture developed during the internship provides an enormous efficiency gain which was proven through the analysis.



(A) $n_{nodes} = 12$



(B) $n_{nodes} = 22$

FIGURE 4.2: Comparison of the Python simulation results for the current solution and the newly designed Switch for $n_{in} = 5$ and $n_{out} = 8$. Each plot contains the percentage of stalling input links in blue and busy output links in red. The corresponding usage ratio is scatter-plotted for each clock cycle, respectively. Additionally, a mean value for the observed quantity is plotted as a dashed line.

4.2 HDL Simulation Analysis

Having found an improved network structure during the internship, its corresponding RTL implementation needs to be tested. In order to obtain comparability, all further mentioned tests are explained by means of a network with $n_{in} = 5$, $n_{out} = 8$ and $n_{nodes} = 22$. The testbench discussed in section 3 supplies the testing environment for the Switch and produces text files containing all relevant data for later evaluation. In the following, the general testing parameters are listed and the evaluation process is explained. The last two sections in this chapter are dedicated to the analysis and further comparison to the internship results.

4.2.1 HDL Simulation Parameters

For creating expressive test results that are comparable, the runtime was set to the same value of t = 10000 clock cycles for all simulation runs. As the testbench logs the relevant data for each clock cycle, this results in stable mean values, as the data is averaged over the total runtime. The second parameter remaining constant throughout the experiments is the weight array with w = [3, 3, 3, 3, 6], so the dependency of the network's efficiency on the input bandwidth can be quantified. The weights were chosen to resemble a possible case in the HICANN-X as well. The four smaller weights represent the spike events generated in the on-chip neuron block and the weight $w_4 = 6$ resembles the configuration bus which produces data less often. The input traffic for the simulation is controlled by the activation probabilities p as mentioned in section 3.1.1. The customized bandwidths N_{in} determine the activation probabilities p for the following analysis which are calculated to fulfill equation (3.1):

- Low bandwidth:

  $N_{in} = \frac{N_{out}}{2}$

  p = [0.26, 0.26, 0.26, 0.26, 0.13]

- Medium bandwidth:

  $N_{in} = N_{out}$

  p = [0.53, 0.53, 0.53, 0.53, 0.26]

- High bandwidth:

  $N_{in} = 2 \cdot N_{out}$

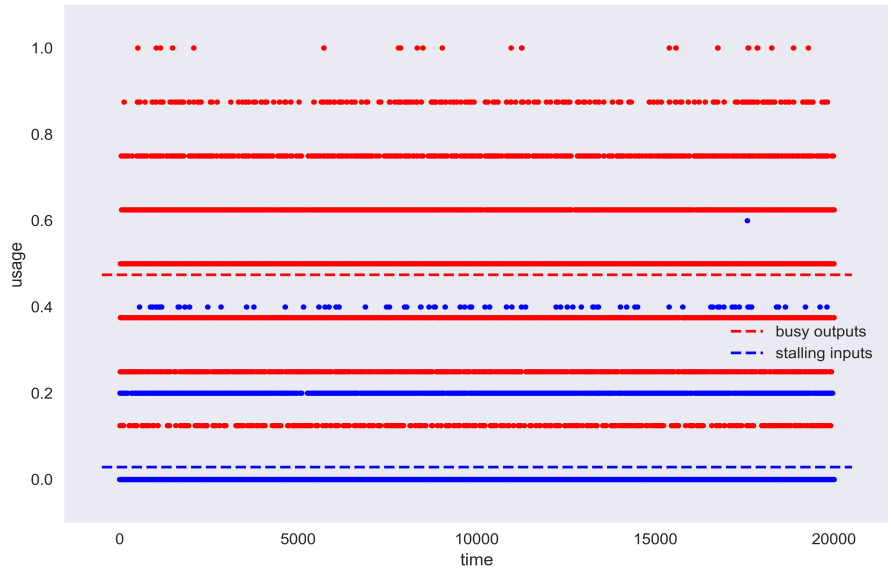
  p = [1, 1, 1, 1, 0.53]

4.2.2 Evaluation Software

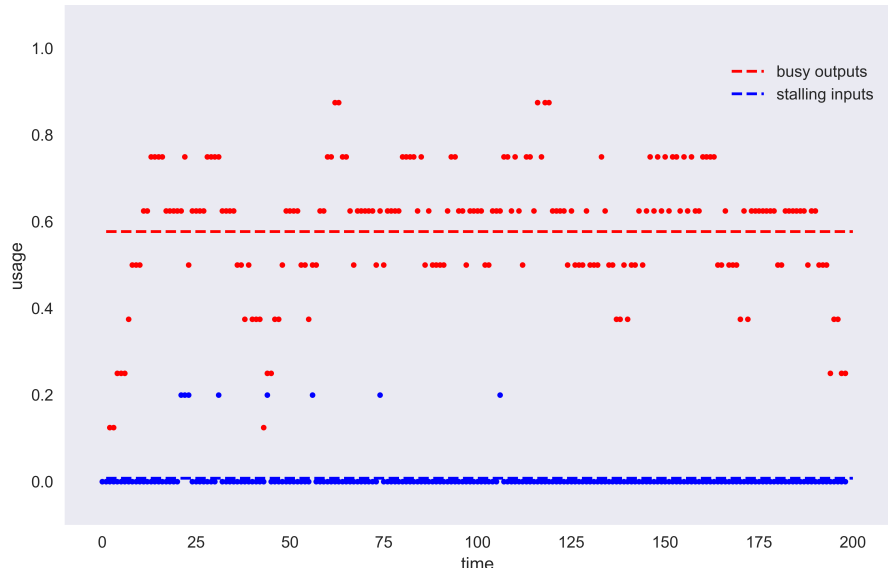
The evaluation of the data and the plot generation are conducted in a Python script which reads the testbench's output files. In each clock cycle, the testbench records the number of occupied output links as well as the number of stalling inputs. For both quantities, the corresponding link's numbers/indices are noted as well.
Analog to the evaluation of the pure Python simulation in figure 4.2, the percentage of stalling inputs and busy outputs is analyzed for the HDL implementation in the following sections.

4.2.3 HDL Simulation Results

A goal of this analysis is to identify how the tendency of a stalling at the input links changes with a rising data traffic, as well as observing the change in output occupancy. Consequently, the evaluation involves three different bandwidth cases as listed in section 4.2.1. The following page contains the HDL simulation results in figures 4.3A, 4.4A and 4.5A compared to the Python simulation results in figures 4.3B, 4.4B and 4.5B.



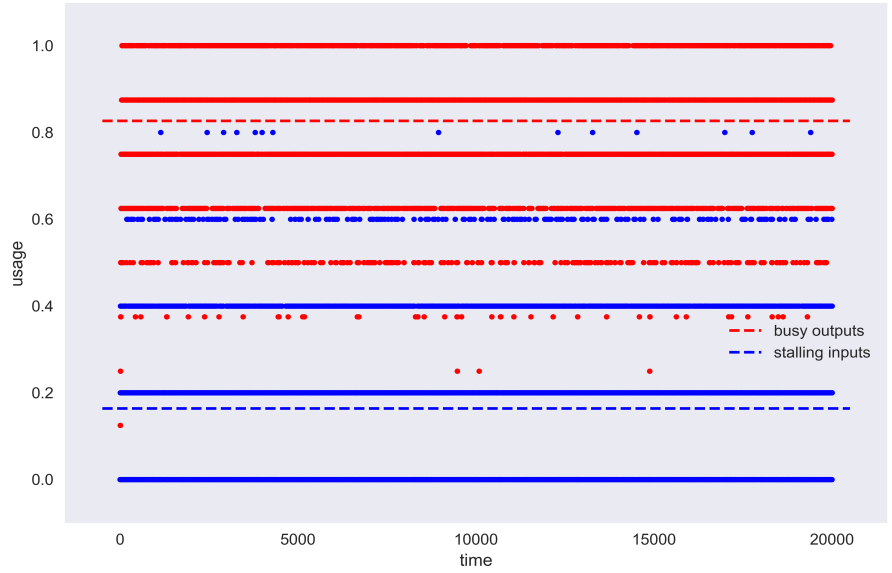
(A) HDL Simulation



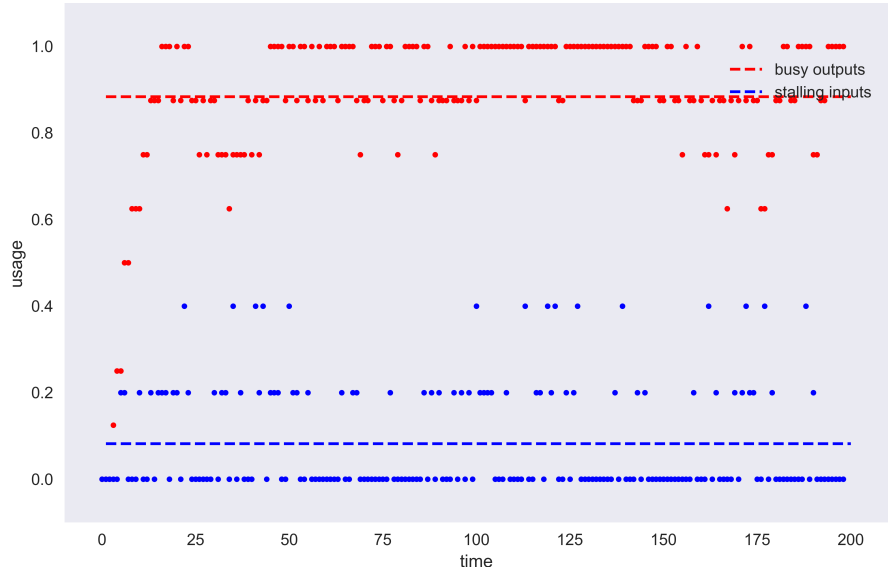
(B) Python Simulation

FIGURE 4.3: Comparing the results for $n_{in} = 5$, $n_{out} = 8$, $n_{nodes} = 22$ for the HDL Simulation and the Python Simulation. The plots show the time course for a low Bandwidth case with $N_{in} = \frac{N_{out}}{2}$

Low Bandwidth: The data traffic applied to the Switch's inputs cannot completely saturate all eight output links. As observed in plot 4.3A, the simulation yields a very low mean number of stalling inputs $n_{stall} = 0.029$. Hence, most data that is generated can be accepted into the Switch instantly. The average occupancy of the output links is at $n_{busy} = 0.475$ which results from the overall low input bandwidth. However, the plot confirms that the new network architecture allows the use of all eight output links even for a low bandwidth. That verifies an even distribution of the payload without any structural issues causing constrictions.



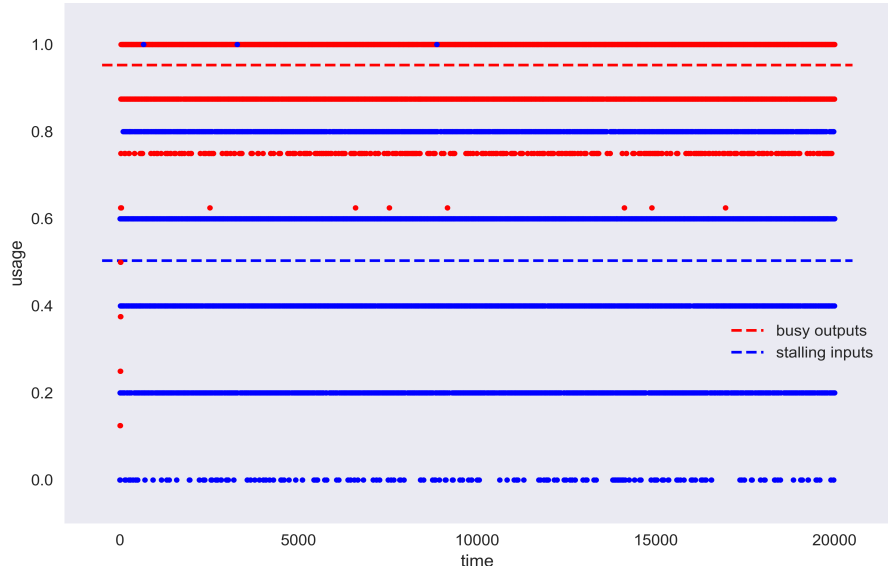
(A) HDL Simulation



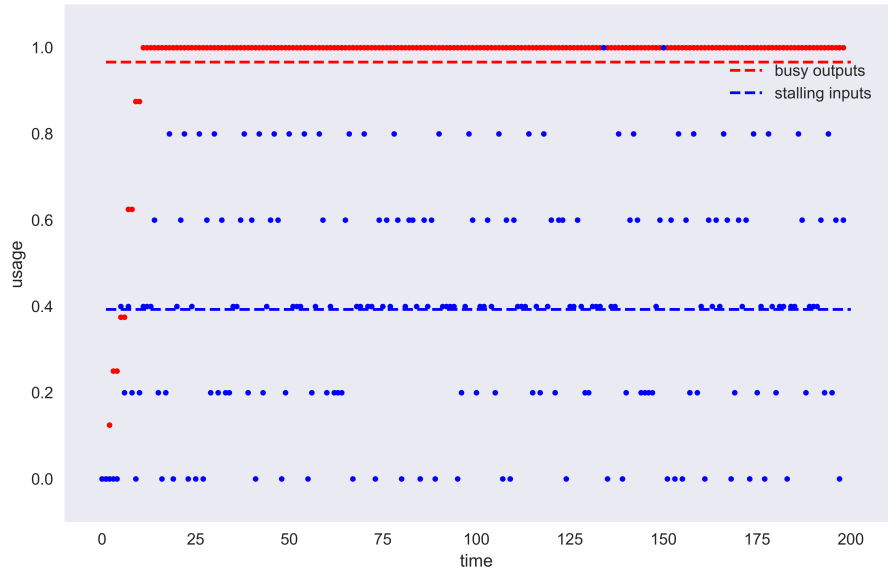
(B) Python Simulation

FIGURE 4.4: Comparing the results for $n_{in} = 5$, $n_{out} = 8$, $n_{nodes} = 22$ for the HDL Simulation and the Python Simulation. The plots show the time course for a saturating Bandwidth case with $N_{in} = N_{out}$.

Saturating Bandwidth: On average, the bandwidth inserted per clock cycle saturates the output links. Plot 4.4A shows a slight increase in n_{stall} for the now higher bandwidth. As the input bandwidth is only an average value, it is possible that in some cases the outputs are oversaturated which can result in a higher number of stalling inputs. However the average is still quite low at $n_{stall} = 0.164$. The output occupancy is at $n_{busy} = 0.827$ which shows a significant increase.



(A) HDL Simulation



(B) Python Simulation

FIGURE 4.5: Comparing the results for $n_{in} = 5$, $n_{out} = 8$, $n_{nodes} = 22$ for the HDL Simulation and the Python Simulation. The plots show the time course for a high Bandwidth case $N_{in} = 2 \cdot N_{out}$.

High Bandwidth: While according to the plot in figure 4.5A the amount of stalling inputs is at $n_{stall} = 0.504$, the mean output usage is at an average of $n_{busy} = 0.953$. Therefore at most times, the outputs are fully saturated and no further payload can be processed. Hence, the high average number of stalling input links is justified and a fair distribution of data within the network is confirmed.

Table 4.1 summarizes the average percentages of the stalling inputs and busy outputs for the different bandwidth experiments.

	low BW	**medium BW**	**high BW**
stalling inputs $\frac{n_{stall}}{n_{in}}$	0.029	0.164	0.504
busy outputs $\frac{n_{busy}}{n_{out}}$	0.475	0.827	0.953

TABLE 4.1: Percentages of stalling inputs and busy outputs for simulation runs of the RTL Switch with three different input bandwidths.

Overall, the Switch guarantees an efficient use of the available output bandwidth and meanwhile provides a fair acceptance of data from all input links without blocking for the majority of the time. The most important difference to the minimum node configuration is the noticeable gain in output occupancy when increasing the amount of data fed into the Switch. This helps to use the hardware to its full capacity.

4.2.4 Comparing Python and HDL Results

When comparing the Python simulation to the HDL simulation, a few minor differences occur. One example are the rather sparse distribution of the data points in the Python simulation plots compared to the HDL simulation as a result of the differing runtimes. In the HDL simulations, more data points were collected and the individual lines for a certain number of stalling inputs or busy outputs appear more distinct. Furthermore, the variations in the data distribution are due to the slightly different implementation of the nodes. During the internship, the arbitration was randomized in each clock cycle for both, input and output clients which differs from the arbitration used in the HDL Shifter module (section 2.3.3).
Overall, the results are similar for the Python and HDL simulations, confirming that the HDL module performs as intended.

Chapter 5

Discussion & Outlook

5.1 Summary

The general aim of this thesis was designing and implementing a highly parametrizable HDL module that provides a structure of buffer stages to evenly distribute input payload to the output channels. Most importantly, it should show improvements in output usage and reduce the stalling effect at the input links compared to the currently implemented network.

Currently Implemented Network The main limitation of the currently implemented network is of structural nature. The node that is linked to an input as well as an output channel creates a bottleneck for the whole network. It restrains the usage of the output links and overall reduces the throughput and the network's efficiency.

Development of the new Switch The individual components providing the temporal dynamic behaviour were implemented in HDL before constructing the Switch with those submodules. The structure for the new Switch was based on the internship results. Conducting controlled tests was made possible by designing a specific testbench for the Switch module. Analyzing test data quantified the efficiency and allowed a comparison to the Python simulations.

5.2 Discussion of the Evaluation Results

So far, the constructed network was tested in HDL simulations, using the testbench with the simulation tool ModelSim. The tests validate a correct functionality that fulfills the requirements stated in section 1.2. In this thesis a network with $n_{in} = 5$, $n_{out} = 8$ and $n_{nodes} = 22$ was analyzed in detail.

A general goal for all of the Switch's applications is to minimize the ratio of stalling

input links per clock. Consider a digital neural network modelling neuronal behaviour through simulation. In some cases the network produces a large amount of spike event data. When distributing those events through a Switch, a stalling effect at the input links is inevitable. In the case of a purely digital neural network, the simulation can simply be paused until all data is handled by the Switch and only then the neurons evolve further. Scaling with the number of stalling input links for this kind of network is the effective runtime since the generation of data is slowed down by interrupting the network's evolution. This is only possible since the simulation would in this case work with discrete time steps.

In case of the HICANN-X chip, this is not applicable because neurons are emulated by analog circuits that are continuously evolving. Analog neural networks are deprived of the opportunity to interrupt their development temporarily, so a stalling at the input links results in dropping the corresponding spike events. This leads to a loss of information, which is why the experiment's precision scales with the number of stalling inputs. Furthermore, one of the HICANN-X's central characteristics is the large acceleration compared to the biological time, which induces high data traffic within the neural network. This speed-up however can only be fully taken advantage of if the communication infrastructure is capable of processing the resulting large amount of data. Within the scope of this thesis an exemplary network was observed for three different input bandwidth cases in particular. The number of stalling inputs was monitored as well as the output occupancy.

Low Bandwidth Results The very low input traffic in this experiment does not require the full output bandwidth and payload can be processed by only using a few output channels in each clock cycle. The average output occupation ratio is at 47.5%. Since the amount of payload generated each clock cycle is rather small, it should be accepted into the network instantly. An average value for the stalling inputs per clock cycle of 2.9% confirms the expected behaviour.
A further observation in the analysis plot in figure 4.3A is a jitter of both quantities that were measured. Since the data generation at the inputs is randomized, it can result in irregular behaviour on a short time scale. This causes temporary stalling effects within the network and leads to fluctuations of the number of stalling inputs or occupied outputs. By averaging over a long runtime, this behaviour is taken into account.

High Bandwidth Results The opposite scenario to the low bandwidth case was oversaturating the output links with a very high input traffic. A large amount of data is fed into the network which on average requires more than the available output bandwidth to be processed within one clock cycle. In theory, the ideal outcome for this case would be an output utilization of 100%, which the Switch approximates

very well in simulation with 95.3%. As mentioned above, the input event generation is randomized and causes fluctuations of the output occupancy as well, which explains the average value not reaching 100%. However, for the majority of the time the outputs are used to full capacity. As a result, the stalling effect will be noticeably larger than for a low input traffic, which is justified by the high output usage. In this case the high number of stalling inputs and the corresponding drop rate for an analog neural network is inevitable.

Saturating Bandwidth Results Assuming that for an application in a chip's communication infrastructure, the available output bandwidth is chosen in respect to the expected input traffic, the saturating bandwidth case will likely be applicable for most of the time. Hence the case with $N_{in} = N_{out}$ is investigated as well. Since that case portrays the border between saturating and oversaturating all output links, the behaviour of stalling input events in a bandwidth margin around $\frac{N_{in}}{N_{out}} = 1$ is observed. The corresponding plot can be found in figure 5.1.

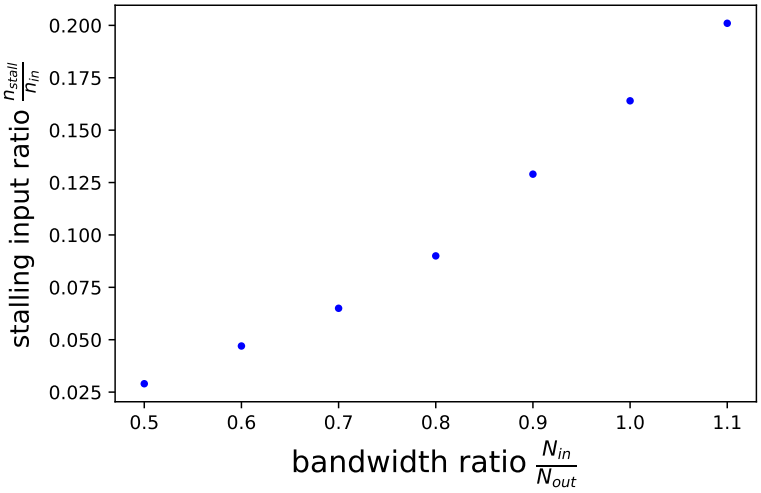


FIGURE 5.1: Dependency of the stalling input ratio on the input bandwidth. The stalling input ratio is plotted for a range of input bandwidths around the optimum saturating bandwidth of $N_{in} = N_{out}$. The data for this plot was accumulated in HDL simulations.

As expected, the number of stalling inputs stays lower than 20% for rising bandwidths up to $N_{in} = N_{out}$. This corresponds to an average smaller than one stalling input link in this particular network configuration with $n_{in} = 5$. Since the generation of input data is randomized for the HDL simulation, the behaviour of the Switch strongly depends on the event distribution. It is possible for the network to be temporarily oversaturated or to not be utilized to capacity. Therefore, the network's utilization is not entirely predictable, but the ratio being lower than 20% signifies no stalling at any inputs for the majority of the experiment's runtime.
Considering the value for $N_{in} = 1.1 \cdot N_{out}$, the results indicate a stalling at some of

the input links for most of the time. This also matches the expectations since on average the input traffic would slightly oversaturate the output links.
The overall desired curve in this plot would resemble a value of zero stalling inputs up until $N_{in} = N_{out}$, followed by a sharp edge indicating a rise of stalling input links as soon as the output links are oversaturated on average. Since the event generation is randomized for the HDL simulation and not periodic on a HICANN-X chip as well, this behaviour can not fully be reproduced.
To compensate for that, the user can however adapt the output bandwidth by taking into account the according plot as found in figure 5.1 for the corresponding network parameters. Considering both the maximum acceptable drop rate and the highest possible input traffic, a value for N_{out} can be chosen accordingly if possible. For a stalling input ratio very close to zero, choosing a setup with $N_{in} = \frac{N_{out}}{2}$ is recommended. For any input bandwidth higher than $N_{in} = N_{out}$ a larger amount of dropped events can be expected.
An approach towards reducing the amount of dropped data would be introducing elastic buffer stages placed at the inputs. Neuronal events could be stored for a short amount of time instead of being discarded immediately if a stalling effect occurs. This is only relevant for spike event data generated in the chip's analog neuron block as it is not reproducible. The configuration buses mentioned in figure 1.3 do not drop any data, they simply stall until the Switch is capable of processing their payload.

Overall the simulation tests confirm significant improvements for the throughput and especially the output occupancy. Furthermore the expectations based on the internship's pure software analysis were met since the resulting plots yield very similar results.

5.3 Outlook

While the project's results confirm a successful development of an efficient network, the new design has only been tested in HDL simulations up until now. Hence, the next step towards commissioning the Switch are thorough hardware tests.
To further broaden the Switch's functionality, a few changes to the module could be implemented in the future. A first approach could be a higher level of connectivity per node. This would be implemented by increasing the number of allowed input and output connections per Shifter to three or more. As a result, diagonal connections would emerge. The arbitration would need to be modified for this case as well. Establishing more possible paths, the payload could be distributed quicker into the horizontal direction.
Another feature for maintaining the usability of the Switch for various use cases is

the option of a routing system. The feature could be enabled for transporting payload from the input clients to a specific output link as destination. Disabling this option would again result in treating all output links equally.
The Switch module will be included in the next generation chip and can already be used on the FPGA connected to the HICANN-X.

Acronyms

ASIC Application-Specific Integrated Circuit. 1

DUT Device Under Test. 20

FPGA Field Programmable Gate Array. 2, 8

HDL Hardware Description Language. 27–33, 36

HICANN High Input Count Analog Neural Network. 1, 8, 21, 23, 27, 34, 36

RTL Register Transfer Level. 4, 19, 23, 27, 32

Bibliography

Aamir, S. A. et al. (2017). "From LIF to AdEx Neuron Models: Accelerated Analog 65 nm CMOS Implementation". In: *IEEE Biomedical Circuits and Systems Conference (BioCAS)*. IEEE, pp. 1–4.

BrainScaleS Research Project. `https://www.kip.uni-heidelberg.de/vision/`.

Human Brain Project. `https://www.humanbrainproject.eu/en/`.

Kanzleiter, Lea S. (2018). *Towards a parametrizable Switch for Neuromorphic Hardware*. Internship Report. University of Heidelberg.

Schemmel, J. (2018). *BrainScaleS Next Generation*. Talk in the ASIC Seminar. University of Heidelberg.

Schemmel, J., J. Fieres, and K. Meier (2008). *Wafer-Scale Integration of Analog Neural Networks*.

Schemmel, J. et al. (2010). "A Wafer-Scale Neuromorphic Hardware System for Large-Scale Neural Modeling". In: *Proceedings of the 2010 IEEE International Symposium on Circuits and Systems (ISCAS"10)*, pp. 1947–1950.

Acknowledgments

I would like to express my gratitude to

- Prof. Dr. Karlheinz Meier and Dr. Johannes Schemmel for giving me the opportunity to be a part of the Electronic Vision(s) group and to carry out my internship and bachelor's thesis on this topic.
- Vitali Karasenko and Mitja Kleider for patiently supervising my work and always being there if I needed help. Thanks for being so supportive and for repeatedly clarifying that the brain is an amazing machine.
- Vitali, Mitja, Oliver, Tamme and Tom for proofreading my thesis.
- The whole group for the welcoming atmosphere and my family and friends for the support throughout my studies.

Statement of Originality

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 others, published or otherwise, are fully acknowledged in accordance with the standard referencing practices of the discipline.

Ich versichere, dass ich diese Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Heidelberg, 25.10.2018 _____