Department of Physics and Astronomy
University of Heidelberg

Bachelor Thesis in Physics
submitted by

David Hinrichs

born in Bergisch Gladbach, Germany

2014

Software Development in the context of Dendrite Membrane Simulation

This Bachelor Thesis has been carried out by David Hinrichs at the

Kirchhof Institute for Physics

Ruprecht-Karls-University Heidelberg

under the supervision of

Prof. Dr. Karlheinz Meier

Abstract

The aim of this Bachelor Thesis was to extend the functionality of the neuron circuit simulation, the so called *SimDenMem*. The *SimDenMem* is based on a common abstraction layer for hardware access, neural network simulation and neuron behaviour simulation called Hardware Abstraction Layer Backend (HALbe). The core objective was to include synaptic input data into the *SimDenMem*. Furthermore, the software environment was advanced to allow multi-user access. Finally, the software and simulation were tested by a series of consistency checks.

Zusammenfassung

Ziel dieser Bachelorarbeit war es die Verwendungsmoeglichkeiten der Neuronenschaltkreissimulation, der sogenannten *SimDenMem*, zu erweitern. Die *SimDenMem* basiert auf einer gemeinsamen Abstraktionsschicht, *HALbe*, die sowohl fuer Hardwarezugriffe, fuer neuronale Netzwerksimulationen als auch fuer praezise Neuronverhaltenssimulationen verwendet wird. Das Erweitern dieser Simulation um den synaptischen Eingang war das Hauptaugenmerk dieser Arbeit. Des Weiteren wurden Veraenderungen an der Softwareumgebung durchgefuehrt um den Mehrbenutzerbetrieb zu ermoeglichen. Abschliessend wurde das Softwarepaket getestet und verschiedene Simulationen durchgefuehrt.

Contents

Chapter 1

Introduction

The Electronic Visions research group in Heidelberg develops neuromorphic hardware and software in accordance to the Brainscales [1] and Human Brain Project (HBP) [2] research projects. As a participant of the HBP's subdivision SP9 the Electronic Visions groups foremost objective is the development of a Neuromorphic Computing Platform, the Neuromorphic Physical Model (NM-PM) [3]. The NM-PM, or PM, as a platform enables the emulation of neural networks shaped after the image of the human brain and aims to supply researches with the possibility to explore brain models in great depth and detail. To reach this objective the PM employs a VLSI hardware system, at the core of which are circuit implementations of the AdEx neuron model and conductance-based synapses. These neurons and synapses can be configured to implement arbitrary neural networks. These array blocks, called ANNCORE (Analog Neural Network Core) [3] are the actual heart of the system. The neurons inside the ANNCORE, which consist of 512 neuron and 114k synapse circuits, are designed to implement the 'Adaptive Exponential Integrate and Fire' model [4]. This model provides the system with the capacity to emulate biological neurons to a very precise extent. The design of the circuitry that implements the AdEx on the ANNCORE is described in the PhD Thesis of [5] Sebastian Millner.

This thesis presents the further development, integration and testing of the SimDenMem[1], a simulation of the transistor level implementation of the DenMem circuit[2]. In this context the SimDenMem is a part of the supporting framework for testing and calibration,

[1] Simulation of the DenMem circuit

[2] Dendrite Membrane, the representation of a neuron cell body

the Hardware Abstraction Layer Backend (HALbe), which in turn is a low level software interface that provides configuration access to the PM.

The presented thesis details the further development of SimDenMem's functionality, as well as the extension of the software supporting its usage.

The first chapter will give an introduction to the SimDenMem and the adjacent software. Following, the motivation for the work presented in this thesis, as well as the general scope of the attempted goals will be explained. In the third and final chapter, in the discussion the results are evaluated and suggestions for further development are given.

1.1 Motivation and Scope

1.1.1 Simulation

The DenmemSim provides its users with a virtual implemention of two Denmem circuits which in turn are each connected to a synapse circuit delivering synaptic input.

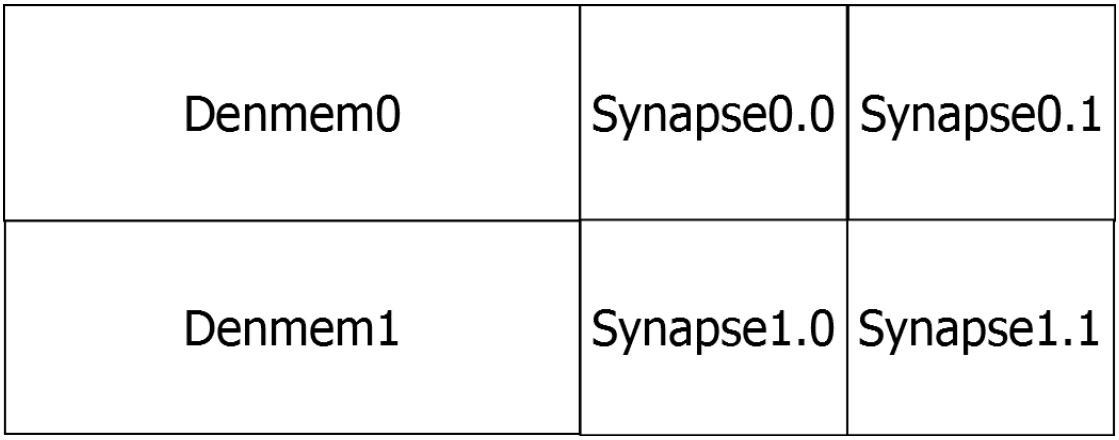

Figure 1.1: The simulation contains two denmem circuits, as well as two synapses

This way, the DenmemSim provides a simple-access method to test the circuit implementation of theAdaptive Exponential Integrate and Fire Model [4], but more importantly it is a tool for understanding the most basic neural networks the PM is able to implement. Since it is anticipated, that the emulation of more complex networks on the PM is going to yield new and previously unknown results in neuronal behaviour, it is important to

have a detailed understanding of the functioning of simple neuron circuits, from which an attempt to understand quantitavely larger phenomena can be made. Also, the DenmenSim enables to read out every node for current and voltage for each timestep, a feature that the actual silicon-cast ANNCORE circuits on the PM machine cannot perform, as there is no way to obtain the data. Therefore, the SimDenMem is an important tool regarding the calibration and analysis of the PM system. Moreover, since the capacity[3] of the silicon-cast PM is currently limited, and software-based simulation can be implemented with much less ado and complication, the usage of the SimDenMem frees up more time for large-scale tasks being performed on the wafer, whereas basic testing and calibration can be looked into via the simulation.

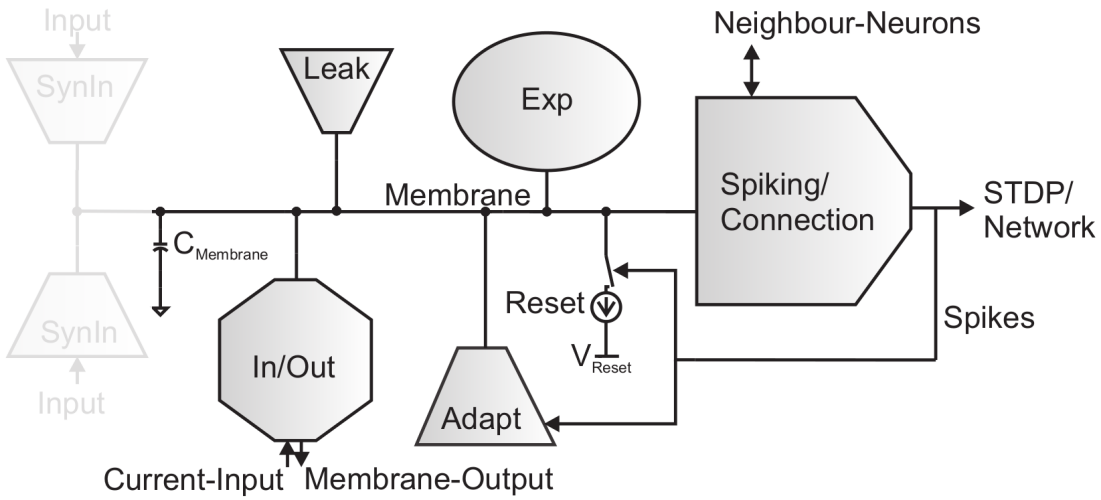

Figure 1.2: The simulated circuit. The work presented in this thesis revolves around the integration of the synaptic input into the input data

Therefore the first step in further developing the SimDenMem lay in expanding its functionality towards synaptic input, which had not been included yet, as shown in figure 1.2.

Secondly, the ability to succesfully emulate the AdEx-model was explored shortly in a series of consistency tests.

[3] As of runtime, since at this point only one wafer is accessible whilst the rest of the system is being set up

1.1.2 Software

Since the SimDenMem at the time of this Bachelor Thesis was only used by a sly handful of people directly involved with it's development, there was no need to account for access of a notable number of users working with it at the same time. Consequentially, the functionality of the SimDenMem's implementation left a couple of things to desire when aiming for a broader access by a larger number of users not as involved with the simulation's software framework.

Foremost, the objective was to enable multi-user access.

Additionally, a functionality to verify the input data against a user-defined standard was included into the simulation's toolset.

Chapter 2

Results

This chapter presents the results achieved in the course of this thesis.

The first section will present the improvement and consistency checks in respect to the SimDenMem, while the second section describes the advances made in scaling the simulation's software framework up to multi-access while also adding some functionality.

2.1 Simulation

2.1.1 Verifying the simulation

To verify the correct implementation of all functionalities, new and old, a series of consistency checks was run. The goal was to reproduce well-known results through variation of single parameters, thus providing a test case for the SimDenMem.

Neuron Membrane

The behaviour of the neuron membrane was examined under the variation of some basic parameters such as the threshold-potential V_t and the leakage potential E_l.

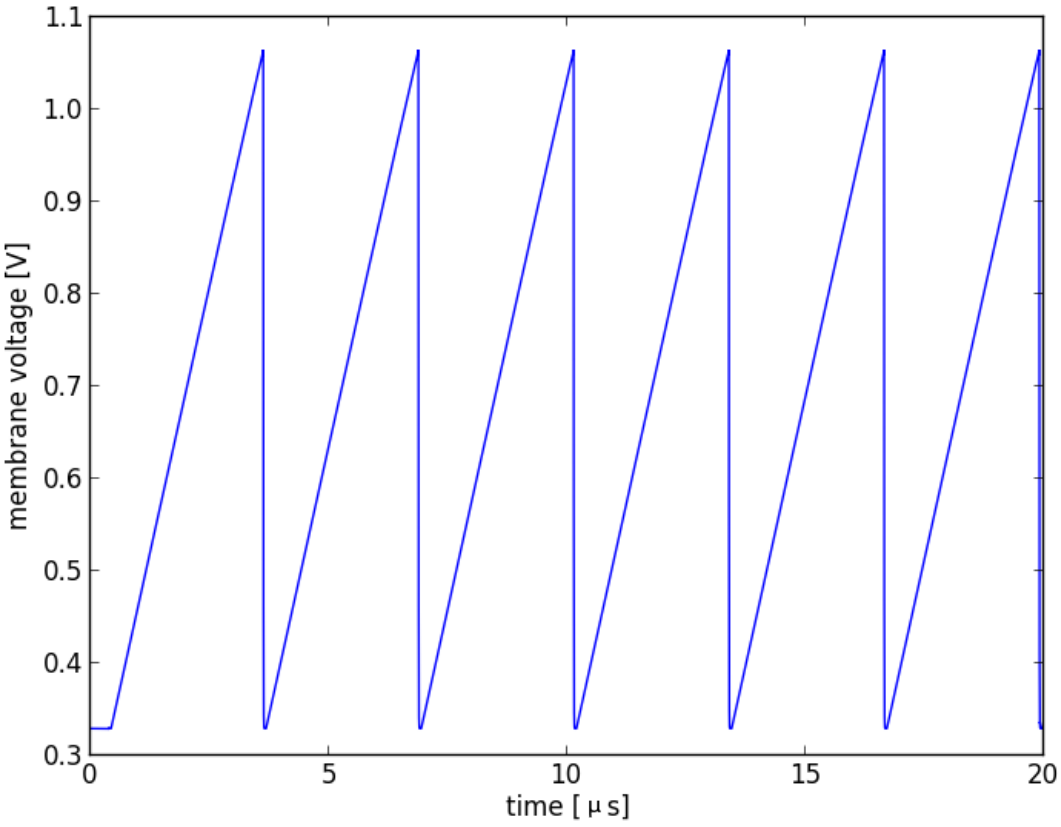

Figure 2.1: Swap of E_l and V_t. As expected, the membrane reacts to this with constant spiking

Current Input

The current input feature was tested via a continuous signal. The plot 2.2 also shows a problem with the result data: Random spikes appear in the current line. Although these spikes do not seem to influence the membrane potential, they may appear due to mis-configuration of the simulator (especially since they are only one timestep wide).

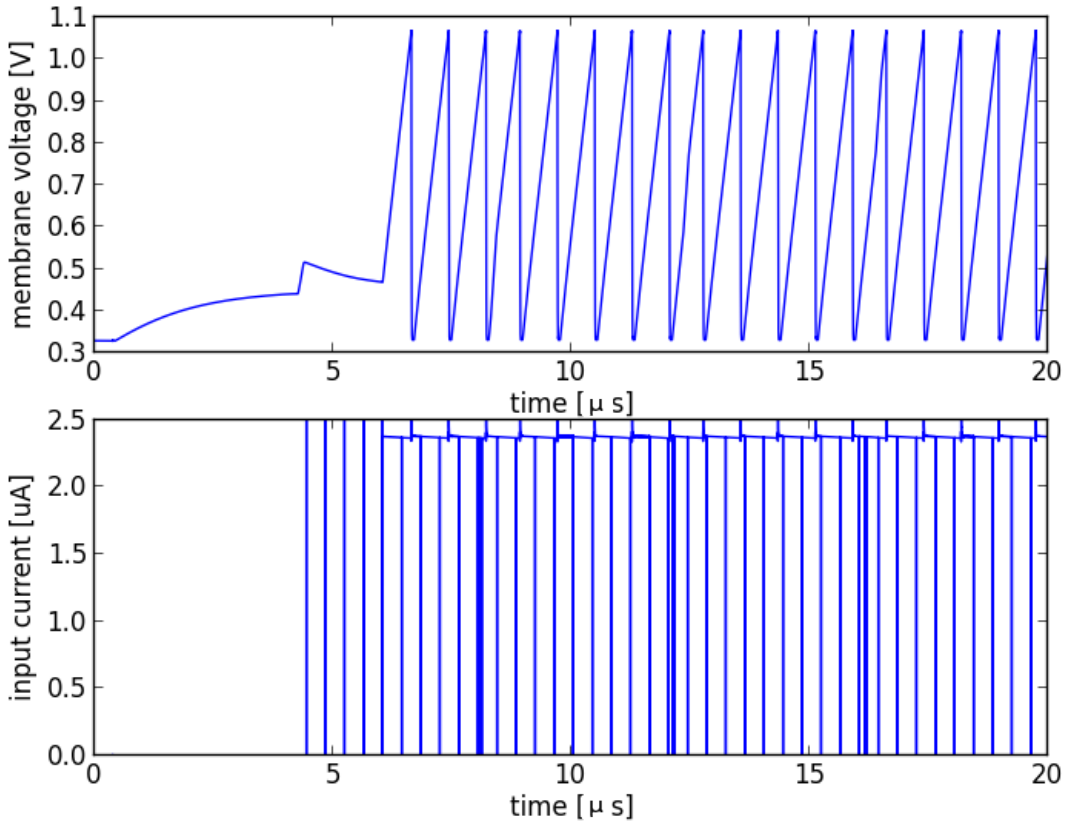

Figure 2.2: The membrane is excited with a continuous current input. Additionally to the actual current input (the flatline starting at about 7 micro seconds), random artifacts from the simulation appear.

Synaptic Input and Synapse Weights

The biggest advance regarding the functionality of the simulation was the integration of synaptic input. This enabled the use of synaptic pulses from either of the two synapses.

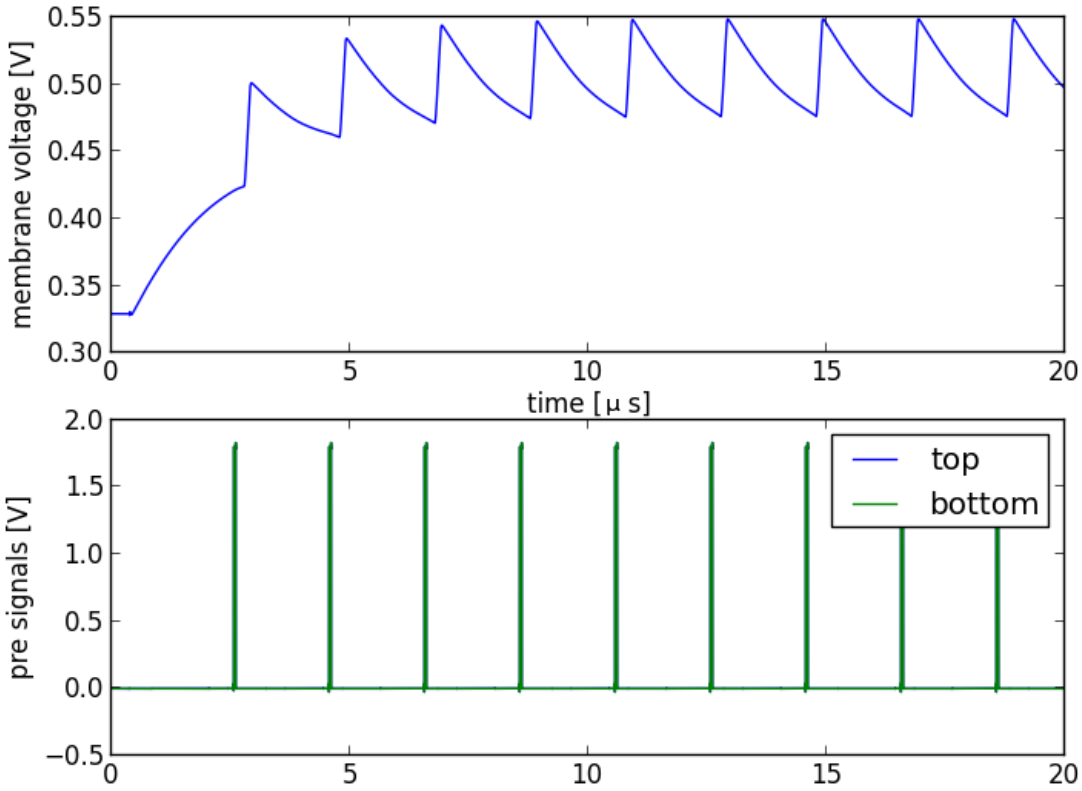

Figure 2.3: Synaptic Input on the left DenMem. The spike trace from both synapse lines - top and bottom - are symmetrical and thus overlap here

2.2 Software development

The software framework (figure 2.4) of the SimDenMem evolves around two major parts.

Firstly, the input file realised as a JSON[1] file container is written by a series of scripts.

[1] JavaScript Object Notation, a data interchange format

Secondly, the JSON file is uploaded to the server via a C++ client-server connection using the Remote Control Framework (RCF)[2]. The server then submits the JSON file to the simulation and returns the results to the client.

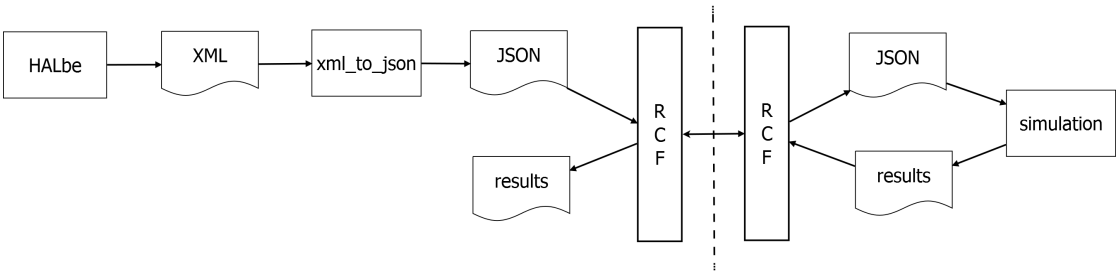

Figure 2.4: Software framework

2.2.1 Towards synaptic input

The first step in further developing the SimDenMem revolved around the addition of previously unavailable parameters (figure 2.5) concerning the synaptic input. This was accomplished by the modification of several scripts from the HALbe environment controlling the writing of the JSON-based input file that is passed to the simulation. This

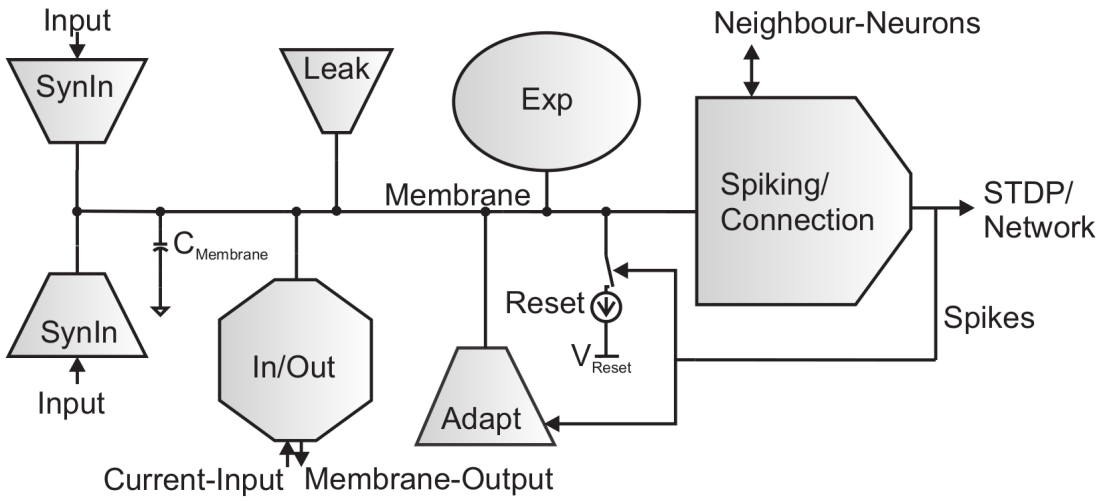

Figure 2.5: This circuit represents the extend of input data written passed to the sim. As opposed to the previous mention in figure 1.2, synaptic input is now included

[2]A C++ framework used to establish remote connections bewteen a client and a server

process also lead to several modifications and additions of new functionalities and scripts to the HALbe environment, a detailed manual of which can be found in the appendix of this thesis.

2.2.2 C++ Client-Server-Connection

Scaling the client-server application up to a multi-user access level presented another task in the attempt to further develop the software framework of the SimDenMem. Previous to the work presented in this section (as illustrated by figure 2.6, the server would refuse a new client connection if there was already another one currently established, causing an abort on the client side. This presented an immediate opportunity to enhance the utility of the entire simulation drastically.

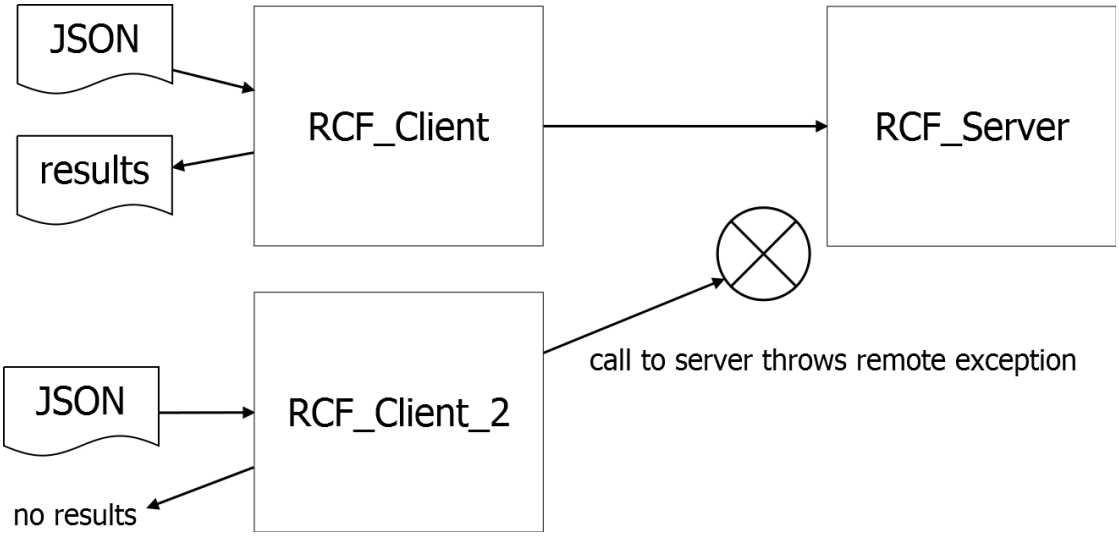

Figure 2.6: The client-server-application before the work presented here

To enable multi-user access to the simulation, the server employs a thread pool that scales with the load of incoming calls up to a fixed maximum. When a client connects, the specific JSON file is uploaded and stored in a file-handling system, as shown in figure 2.7.

This is realised with a call-specific ID generated on the server side. The ID is used as a name for the uploaded input data "upl-ID" and the result data passed back to the client "res-ID". For the client to be able to request the correct result file in the download process, the server hands it the ID as a return value of the upload function.

As a happy accident, this also allows for asynchronous data retrieval, as the client can now request a download anytime by giving the call-spefic ID that connects input data and results.

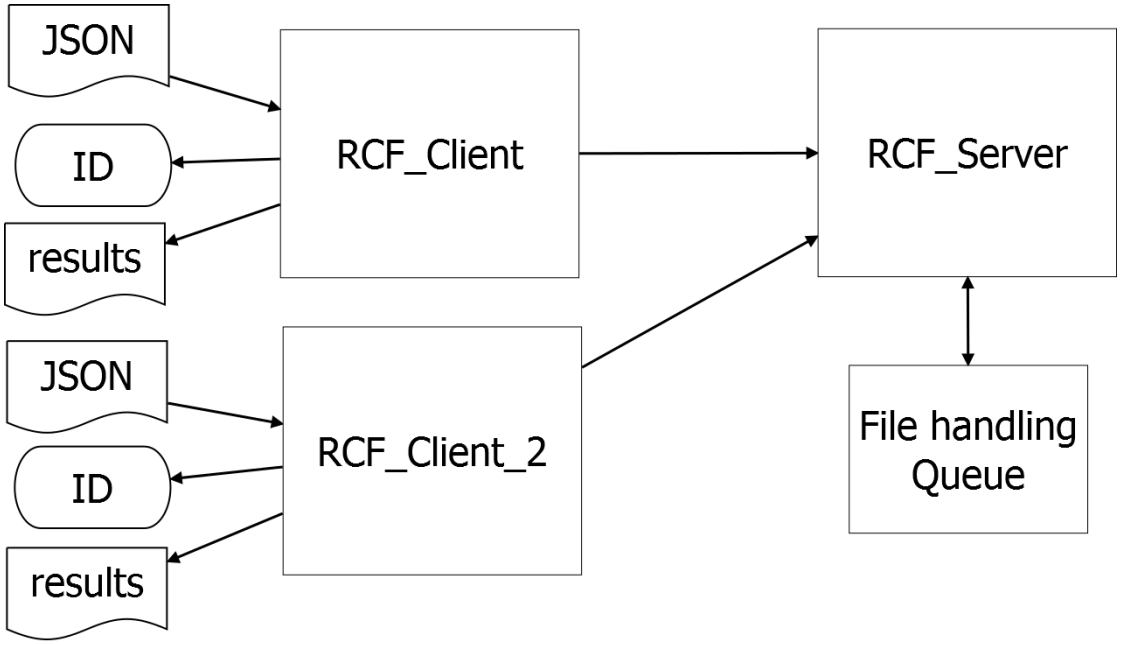

Figure 2.7: The client-server-application after the work presented here

After the uploaded JSON file is stored on the server in the described way, it is submitted for simulation. To prevent a race condition on the server-side, the execution of the simulation is protected by a semaphore [3]. Thus, the server still uploads the input data of other clients but only simulates one at a time. During the time of connection, all clients are passed a "heartbeat signal" to push the remote connection timeout and serve as a "still running"feedback for the user.

In case another client connects while the simulation is still running, the server again uploads and stores the JSON file.

[3] variable to control access to common ressources in a multi-user environment

Chapter 3

Conclusion

In this thesis several improvements to the SimDenMem and it's software framework have been described.

3.1 SimDenMem

The simulation of the Dendrite Membrane Circuit has been executed to a lesser extent as previously intended, as the work on the software environment took up a larger part of the thesis work than anticipated. Yet, the primary goal of including and simulating the synaptic input has been achieved, albeit to a rather qualitative extent on the simulation part.

As for the executed consistency checks for the Neuron Membrane2.1.1, the Current Input2.1.1 and Synaptic Input2.1.1 all simulation results appear in their expected range (save the artifacts in the current input data).

Also, a verification of the findings concerning the saturation effect of the synaptic input circuit as presented in the work of Kiene [6] has been attempted. This particular point requires further inquiry, since not all the data necessary to reproduce the effect could be obtained yet.

3.2 Software

The software environment is now able to handle multiple clients on the user side, which was one of the core objectives set out to be accomplished. Also, the HALbe script[1] now writes the necessary data to generate the synaptic input for the simulation. As a byproduct of the verification process for the input data, a python script to compare two different JSON files (for example "is" and "should") has been created and included in the appendix.

[1] Hardware Abstraction Layer Backend

Chapter 4

Outlook

This chapter will detail several possibilities to further advance the functionality of the SimDenMem and the software suite built around it.

4.1 SimDenMem

Since the work presented here has concerned software for the larger part, a lot of testing and simulating is still to be done. Especially the synaptic input has only been looked into qualitatively. This could as well be the endeavour of another Bachelor Thesis.

4.2 Software

Certain aspects and ideas have not been persued for lack of time. The first further enhancement that comes to mind is the enabling of real parallelism on the server-side of the client-server-application. This would allow to run four different simulations at a time instead of one, but would also require additional licenses and a new workspace for each to prevent possible interferances. This effort seems only warranted if the simulation is used by a signficantly larger number of users in the future than now. Also, as mentioned in section 2.2.2, asynchronous data retrieval is possible now, though it has not been included yet. Since this would mainly be a comfort feature, again an implementation seems only necessary for a larger number of users.

Also, the readout artifacts in the current trace 2.1.1 should be looked into. This introduces the server-side of the simulation as a field of possible advance, referring to the scripts and processes involved after the input data is uploaded via the client-server-connection.

References

[1] BrainScaleS. Research. http://brainscales.kip.uni-heidelberg.de/public/ index.html, 2012.

[2] The Human Brain Project. URL: https://www.humanbrainproject.eu/. [online; accessed: 2014-08-15]

[3] Neuromorphic Platform Specification, Human Brain Project Internal File

[4] Brette R. and Gerstner W. (2005), *Adaptive Exponential Integrate-and-Fire Model as an Effective Description of Neuronal Activity*, J. Neurophysiol. 94: 3637 - 3642

[5] Sebastian Millner (2012), *Development of a Multi-Compartment Neuron Model Emulation*

[6] Gerd Kiene (2014), *Evaluating the Synaptic Input of a Neuromorphic Circuit*

Appendix A

Simulation manual

A.1 HALbe Environment

On the user side, several scripts are involved in the process of writing the JSON input file for the simulation and analysing the retrieved data. The name, respository and location within the repository are given below, including a call example and some additional notes. For a graphic overview of how these scripts interact, see fig Figure A.1.

A.1.1 Building and simulating the JSON file

The following three scripts allow you to create a new JSON file and submit it to the simulation.

1. The XML dumpscript

task: This script from HALbe creates a new data container in accordance with the C++ namespace of HALbe and dumps it into a XML file.

repository: halbe/davhin_enhance_simdenmem

call example: "HALBE_DUMP_FILE=dump.xml python synaptic_input.py"
Dumps the parameters set in the script "synaptic_input.py" into an XML file called "dump.xml"

notes: Be sure to match the following address convention when setting synpatic pulses and snyaptic weights:

neuron_address = 0: syn0.0 (left top)
neuron_address = 1: syn0.1 (left bottom)
neuron_address = 2: syn1.0 (right top)
neuron_address = 3: syn1.1 (right bottom)
in
pulse_address_d = p.FPGA.PulseAddress(
p.Coordinate.DNCOnFPGA(**0**),
p.Coordinate.HICANNOnDNC(p.Coordinate.Enum(**0**)),
p.Coordinate.GbitLinkOnHICANN(**0**),
p.HICANN.Neuron.address_t(**neuron_address**))

This is an arbitrary convention, that simply hardcodes certain adresses to be associated with the places for the pre-pulses in the synapses for easier conversion. Check lines 592-595 of xmldump_to_simdenmem.py for further inquiry.

2. xmldump_to_simdenmem.py

task: converts the HALbe containers into simulation-readable values and stores them in a dictionary. This dictionary then is exported in the form of a JSON file container.

repository: halbe/davhin_enhance_simdenmem

call example: "python halbe/halbe/pyhalbe/tools/xmldump_to_simdenmem.py -i dump.xml -o dump.json -N 0 -H 280"
Takes the previously dumped xml-file and converts it into a json-file called "dump.json", -N and -H give the adressed Neuron and HICANN and need to remain fixed

notes: If you expand the data range by writing new functions into the script1.py, you also need to define new conversion functions in the dumpscript in order to retrieve the data from the xml-dump and add them to the json.

3. run_denmem_simulation

task: The C++ binary that uploads the json and returns the results

repository: halbe/davhin_enhance_simdenmem

call example: "halbe/halbe/tools/run_denmem_simulation –in=dump.json –out=results –timeout=60000 –host=vmimas"
uploads dump.json to the server and returns results to the directory "results". Host needs to be set accordingly, timeout should be chosen long enough (seconds)

notes: Make sure to have enough diskspace on server

A.1.2 Data Analysis

These two scripts allow you to analyse your data, both before and after the json file is submitted to the simulation.

1. compare_json.py

task: Compares two json input files (preferably at least one of which you know to be correct) in regard to keys and values

repository: halbe/davhin_enhance_simdenmem

call example: "python halbe/halbe/tools/compare_json.py -i1 halbe/halbe/pyhalbe/test/simdenmem/reference_ -i2 dump.json"
Compares the two files, prints out missing keys and values that are off. The first file is taken as the "should be" state. notes: When sweeping, comparing the actual values will not be interesting, so do not be alarmed in case of value errors. Watch out for "key error" and "apples and oranges" though, since this indicates that value in file a is a list, while being a float in file b.

2. plot_simdenmem.py

task: The plotscript for the results returned from the simulation

repository: halbe/davhin_enhance_simdenmem

call example: "python halbe/halbe/tools/plot_simdenmem.py –input-directory results –output-file=results.png"
Plots the data stored inside your directory "results" written by run_denmem_simulation and plots it into the current working directory als "results.png"

notes: The input file needs to be the only file inside the input directory.

As a summation and a general piece of overview, this schematic illustrates the functions and connections of the five scripts previously described.

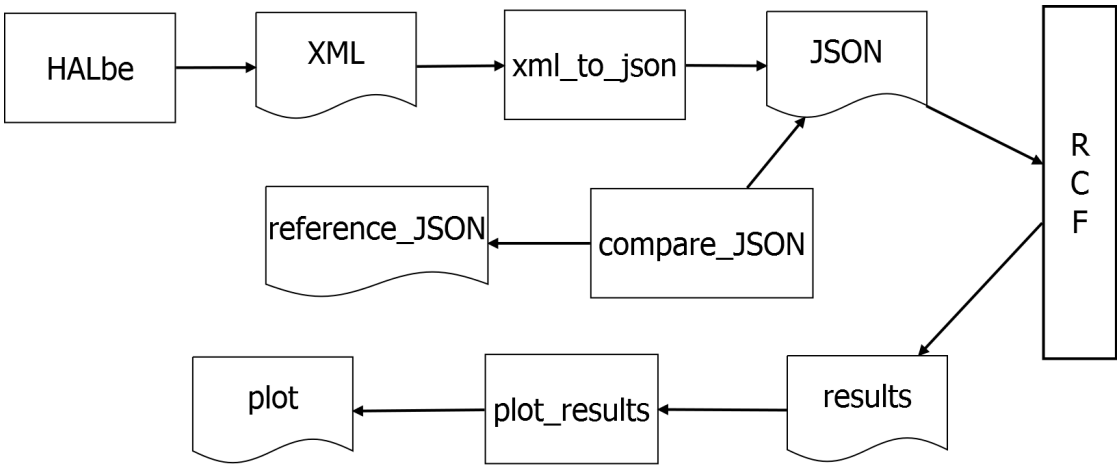

Figure A.1: HALBE Environment

Erklärung

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

Heidelberg, den 18.08.2014,