Internship Report

Multicompartment Neuron Description and Placement Algorithm on BrainScaleS-2

David Baumeister

May 2024

Supervised by
Philipp Spilger, Jakob Kaiser

Abstract

Multicompartment neuron models have, unlike point neuron models, a spacial extent. Therefore they consist out of multiple compartments, each containing multiple neuron circuits. BrainScaleS-2 is a mixed-signal neuromorphic hardware which is capable of emulating these multicompartment-neurons. For the placement of the neuron models two rows of neuron circuits are available on BBS-2. Therefore placing compartments on the BSS-2 hardware manually becomes challenging, unintuitive and time consuming for larger neurons. Therefore an abstraction model of the multicompartment neuron was created and a placement algorithm which places the neuron onto the hardware was developed. The user task is therefore reduced to creating the abstract model of the neuron, rather than assigning each neuron circuit to a compartment and setting hardware-switches by hand.

Contents

1 Introduction

The spacial extent of the dendrites in neurons allows for pre-processing synaptic input. It for example allows improved coincidence detection of neurons, direction selection at single neuron level and applying different learning rules to different locations in a single neuron. Researchers try to model the spacial extent of biological neurons with multicompartment neuron models [2].

BBS-2 offers a set of switches for each neuron circuit which allows to connect multiple neuron circuits to a compartment and multiple compartments to a neuron.

To solve the problem of placing a multicompartment neuron onto the BSS-2 chip several intermediate steps are taken. First an abstraction of the underlying hardware is done. Therefore a abstract neuron structure is created that allows intuitive construction and manipulation of the neuron by the user. In the next step the hardware resource requirements are determined to calculate how many neuron circuits of the BSS-2 chip are required for each compartment. Then the placement itself takes place, where an algorithm is run stepwise until the neuron is placed correctly.

2 Methods

2.1 Hardware

The BSS-2 chip has 512 neuron circuits placed in two rows of 256 to allow the synaptic arrays coming from the top and the bottom to connect to one of the rows. The chip is partitioned in two halves, see Figure 1. When placing multicompartment neurons onto the chip we can therefore focus on a 2×128 grid [4].

The BSS-2 chip has five switches for each neuron circuit which allows it to be connected to another neuron circuit in two ways, see Figure 1.

The first type of connection, called inner connection in the following, short circuits two neuron circuits located next to each other, either left and right or top and bottom. Therefore there is a switch connecting a neuron circuit to its right neighbour and a switch connecting neuron circuits placed in the same column in different rows.

The second type of connection, called external connection in the following, uses the built-in shared line. There are two shared lines on the chip, one for the top row and one for the bottom row of neuron circuits. A neuron circuit can either connect itself directly to the shared line or via a resistor. The shared line can be interrupted by switches. To create an external connection between two neuron circuits, each compartment can connect to the shared line either directly or via the resistor, depending on the use of the connection.

The inner connection allows us to create compartments out of multiple neuron circuits and the external connection allows to connect multiple compartments via a resistor which creates a spatially extended neuron structure.

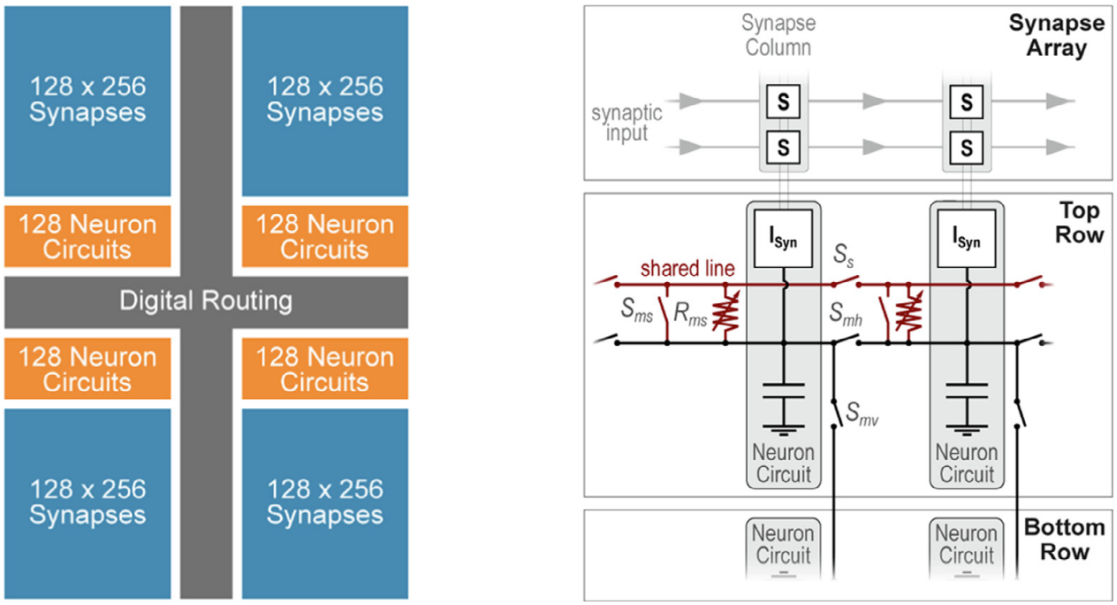


Figure 1: Left Figure shows the structure of a BSS-2 chip. Right Figure shows the switches of a neuron circuit on BSS-2 that can be used for multicompartment placement, taken from [2].

2.2 Software

Without the placement algorithm the placement needs to be done by hand. Therefore each neuron circuit on the BSS-2 chip has an assigned compartment and the five switches of the neuron circuit need to be set manually. For neurons with a more complex structure this is insufficient.

The implementation of the neuron abstraction and the placement algorithm shown in the following is done in the software layer `grenade`. The relevant structures for the user for creating and placing a neuron were wrapped with `genpybind` [3] and can therefore be accessed in python. A code example for the use in Python is provided in the following [Listing 1].

Connecting to the lower hardware layers, a conversion of the placement information used in the algorithm towards the logical description in `halco` has been done.

3 Results

3.1 Neuron Abstraction

As a first step an API with an abstract representation of the multicompartment neuron is created. The neuron is built as a graph and contains its compartments as vertices and the connection between compartments as edges. Each compartment contains its mechanisms. A mechanism abstracts hardware parameters such as the membrane capacitance or the synaptic input on a compartment. Implemented mechanisms are the membrane-capacitance and the synaptic input, both current or conductance based. There is a sufficient API to create more mechanisms analogous to the existing ones.

This abstraction allows the user to create a neuron model by constructing compartments and mechanisms, adding the mechanisms to the compartment and adding the compartments as well as the connection between the compartments to the neuron. There is no need for the user to know where to place the single compartments on hardware as before.

3.2 Hardware Resources

Based on the mechanisms placed on each compartment the resource manager determines how many neuron circuits are required for each compartment.

The Resource-Manager calls each mechanism on each compartment of the neuron to get their hardware-requirements and converts the model-parameters (e.g. capacitance of membrane) to hardware-parameters (e.g number of capacitors needed for required capacitance).

The information stored in the Resource-Manager is later accessed during the placement to get the minimal number of neuron circuits required for a compartment.

3.3 Placement

Placing a multicompartment neuron onto the neuron circuits of a BSS-2 chip requires information about the topology of the neuron as well as the individual resource requirements of the compartments.

The topological information is contained in the abstract neuron model and the resource requirements in the Resource-Manager described above.

3.3.1 Coordinate-System

For placing the neuron we have an intermediate step of placing it into an coordinate system of size 2×256. Using double the size of the space available on the chip allows us to start placement in the middle of the coordinate system and not having to shift the placed structure left or right, since reaching one end of the coordinate system would implicate the neuron does not fit onto the chip.

Each cell of the Coordinate-System contains the ID of the Compartment placed onto this cell as well as the configuration of the five switches of the corresponding neuron circuit.

3.3.2 Placement-Frame

The placement algorithm is run in single steps in which one compartment is placed. The larger structure handling multiple runs of the placement algorithm is the Placement-Frame. It records the result of each run, performs validation checks and converts the coordinate system into the logical structure required for the final placement on the chip.

The Placement-Frame saves the result of each run as a coordinate system with the compartments placed until the current step.

The Placement-Frame has the option to check, after each finished run of an algorithm or at the end of the placement, depending on the type of algorithm used, whether the placed neuron in the coordinate system matches the abstract neuron given by the user. Therefore a neuron as described above is created from the coordinate system with the placed compartments.

The validation checks for multiple errors:

1. Required hardware resources: If the number of allocated neuron circuits of a compartment is smaller than the number of neuron circuits calculated by the resource-manager the validation fails
2. Inner connections of compartments: If the neuron circuits belonging to one compartment are not fully connected by their inner switches the validation fails.
3. Number of compartments/compartment-connection: Checks if neurons are equal by number of compartments and connections between compartments.
4. Isomorphism: If the neuron constructed from the coordinate system is not isomorphic to the neuron of the user input the validation fails.

Since for some types of algorithms this validation needs to be done after each step of placement, a fast validation is required. Therefore the order of checks is important. The isomorphism check is performed last because the comparison of the two neurons has a time complexity of $O(|C|!)$ where $|C|$ is the number of compartments in the neuron [1].

3.3.3 Placement-Algorithm

Three different Types of algorithms are discussed in the following. One of them is chosen and implemented.

3.3.4 Brute Force

The first considered algorithm for the placement was a brute force algorithm which tries out every single possible hardware-switch configuration. Since every neuron circuit has five switches that can either be set or not set and there are 256 circuits on one half of the chip we have $\left(2^5\right)^{256} = 2 \cdot 10^{385}$ options to try.

Because of the high number of possible combinations the time needed for placing would be too long.

3.3.5 Evolutionary Algorithm

Another approach taken in consideration is an probabilistic learning algorithm which checks the fitness of placements after each run and therefore changes the location of neuron circuits and the shapes of compartments. However determining the fitness of the placed neuron is not trivial since a locally optimal solution for one compartment can block the placement of other compartments and is therefore suboptimal on a global scope.

3.3.6 Ruleset Algorithm

The third approach is an algorithm which follows a ruleset for placement. This algorithm was developed by starting with simple placement tasks, such as placing a simple compartment, and than creating new rules for more complex tasks.

For placing a neuron the algorithm follows specific steps as shown in [Figure 2]

1. The first compartment is the compartment with the highest number of connections
2. Place the first compartment in the middle of the coordinate system
3. Find neighbours to this compartment
4. Place the neighbours next to the previous compartment [Figure 3]
5. Connect the placed compartment internally [Figure 4]
6. Connect the placed compartment to its neighbours [Figure 5]
7. Repeat from step 3 until all compartments are placed or no more space for placement is left

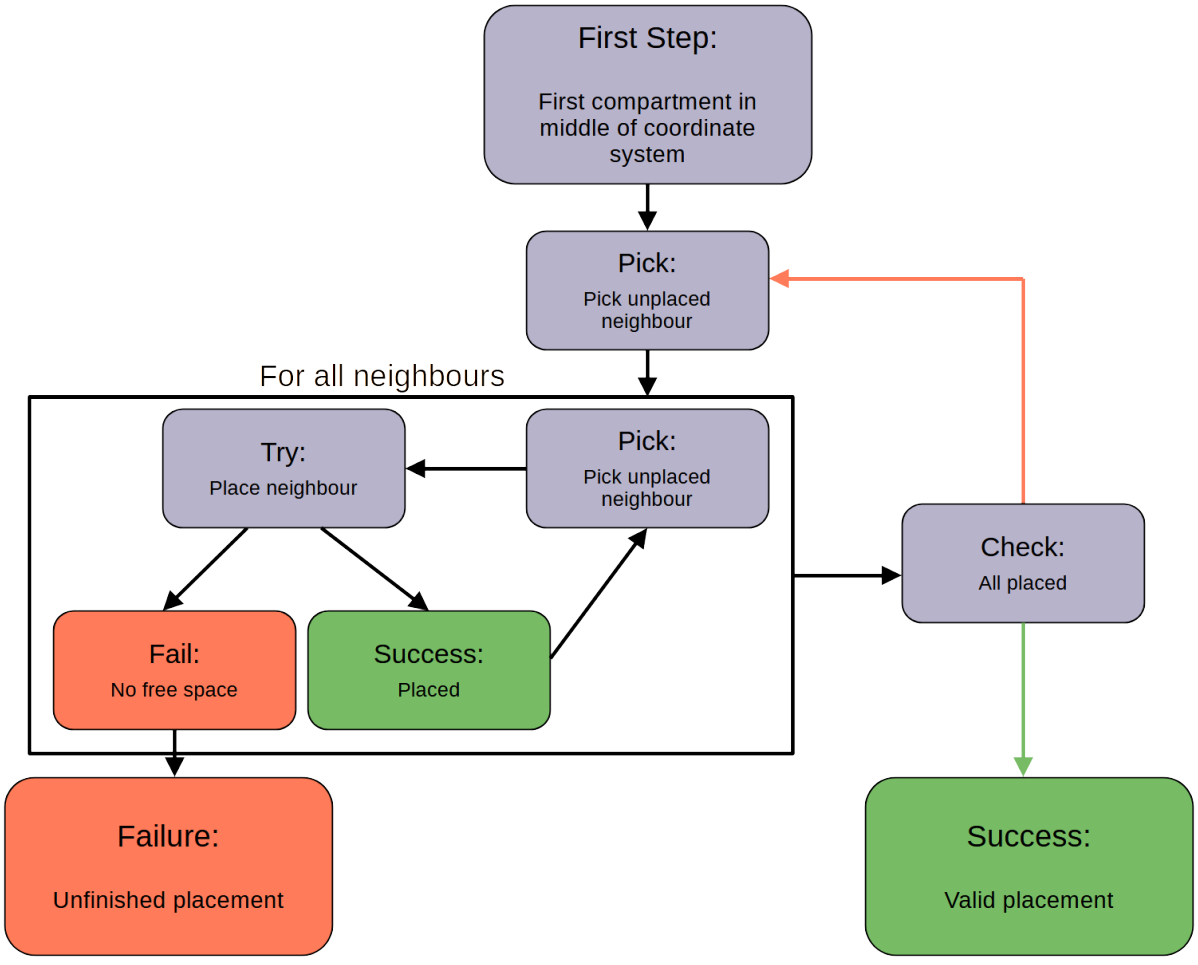


Figure 2: Steps in the ruleset-algorithm.

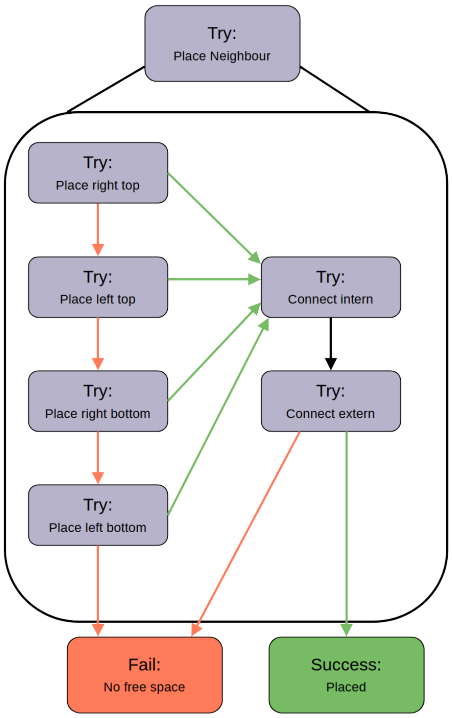


Figure 3: Placing a neighbour compartment.

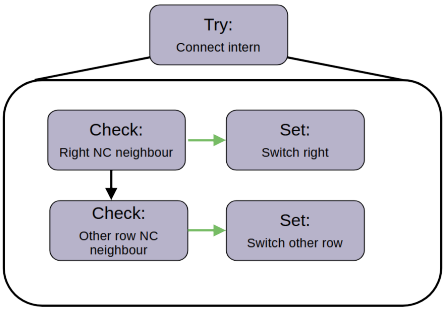


Figure 4: Connect compartment intern.

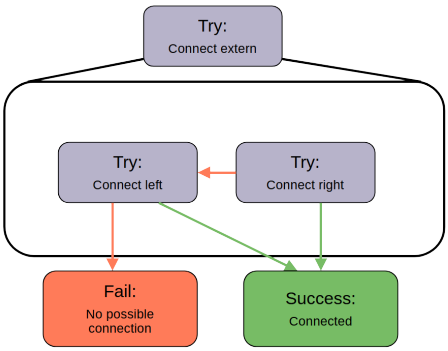


Figure 5: Connect compartment to neighbour compartment.

The algorithm has the advantage of a fast solution and is of linear time complexity $O(n)$, where n is the number of compartments. A disadvantage is that several rules need to be implemented explicitly, so the algorithm can solve complexer neurons. Therefore the amount of neurons placable by this algorithm is limited due to missing rules. If the placement fails the algorithm requires backtracking to solve the placement, which increases the placement time.

The following neurons were placed during development

- single compartment neuron with one required neuron circuit in arbitrary position
- single compartment neuron with multiple required neuron circuits in arbitrary position
- single compartment neuron with multiple required neuron circuits with specification about top and bottom row (to allow synaptic input from both sides if required)
- multicompartment neuron with chain structure (each compartment has ≤ 2 connections)
- multicompartment neuron with multiple branches (≤ 4)

The algorithm can solve these tasks in short time periods. The last tested neuron consisting of nine compartments and two nodes with more than two branches took the algorithm, called in Python, circa 10ms total execution time.

This algorithm was implemented to a point where it can perform placement for simple neurons. Supported neurons must meet the following requirement:

- No compartment must have more than four neighbours
- The compartments must be big enough by resource-requirement if nested structures occur during the placement.

3.4 Demo

A short demonstration is given at this point on how to use the structures described above to create and place a neuron.

The demo-neuron consists of four compartments with a membrane defined on each compartment and one of them with a synaptic input specified [Figure 6]. The result of the placement algorithm is shown below [Figure 7].

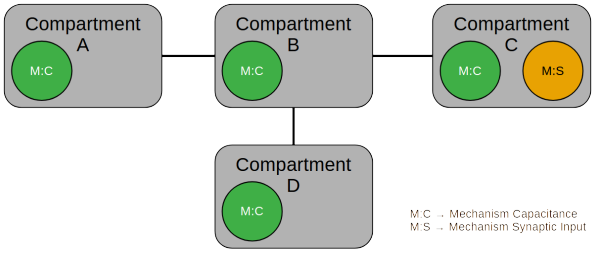


Figure 6: Demo Neuron

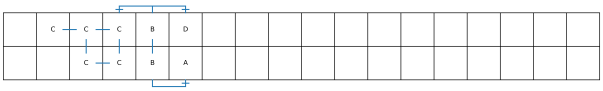


Figure 7: Demo Neuron Placed

```
neuron = grenade.Neuron()

compartment_a = grenade.Compartment()
compartment_b = grenade.Compartment()
compartment_c = grenade.Compartment()
compartment_d = grenade.Compartment()

'...' #Parameterization of the mechanisms

mechanism_capacitance = grenade.Mechanism_Capacitance(parameter_space_c)
mechanism_synaptic_input = grenade.Mechanism_Synaptic_Current(parameter_space_s)

mechanism_c_on_compartment_a = compartment_a.add(mechanism_capacitance)
mechanism_c_on_compartment_b = compartment_b.add(mechanism_capacitance)
mechanism_c_on_compartment_c = compartment_c.add(mechanism_capacitance)
mechanism_c_on_compartment_d = compartment_d.add(mechanism_capacitance)
mechanism_s_on_compartment_c = compartment_c.add(mechanism_synaptic_input)

compartment_a_on_neuron = neuron.add_compartment(compartment_a)
compartment_b_on_neuron = neuron.add_compartment(compartment_b)
compartment_c_on_neuron = neuron.add_compartment(compartment_c)
compartment_d_on_neuron = neuron.add_compartment(compartment_d)

compartment_connection = grenade.CompartmentConnection_Conductance()
neuron.add_compartment_connection(compartment_a_on_neuron, compartment_b_on_neuron,
    compartment_connection)
neuron.add_compartment_connection(compartment_b_on_neuron, compartment_c_on_neuron,
    compartment_connection)
neuron.add_compartment_connection(compartment_b_on_neuron, compartment_d_on_neuron,
    compartment_connection)

environment = grenade.Environment()
synaptic_input = grenade.SynapticInput_Current(exitatory = True,number_inputs.total
    = 1200, number_inputs.top = 0,number_inputs.bottom = 257)
environment.add(compartment_c_on_neuron, synaptic_input)

resource_manager = grenade.ResourceManager()
resource_manager.add_config(neuron, environment)

placement_frame = grenade.PlacementFrame(grenade.CoordinateSystem(), neuron,
    resource_manager)
placement_algorithm = grenade.PlacementAlgorithm_Ruleset()
```

```
placement_frame.run_algorithm_finish(placement_algorithm)
```

Listing 1: Usecase example

The structure `environment` is introduced to define the environment in which the neuron is used. This environment contains the information about the synaptic input of each compartment. Currently this structure is created and filled with the required information for testing purpose manually. In the future the structure will receive its information from the neuron population.

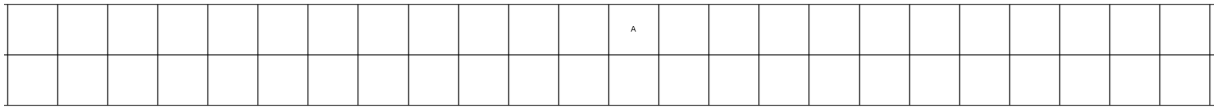
3.5 Testing

For testing and developing the placement algorithm a plot of the coordinate system with the placed compartments and the switch configurations was created. The plotting is done in python and is with an execution time of circa 2s to 5s per plot rather slow. However for placements with countable steps it can be used to plot each step of the placement.

The algorithm following a ruleset was tested on different inputs during development to discover its limits and increase the amount of supported neuron-types. In the following there are plots of some of the neurons used for testing and descriptions of neurons, which can not be placed at the current state because of the limits described above, but are in theory placeable.

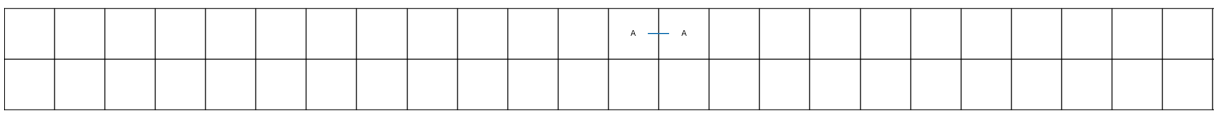
3.5.1 Single Compartment with single circuit

The first placement was done with a single compartment with a single circuit. This was done to ensure placement in general is possible with the implemented data structures. Placing a single compartment on the hardware is no challenge in terms of the algorithmic step, but rather in having all structures working together properly.



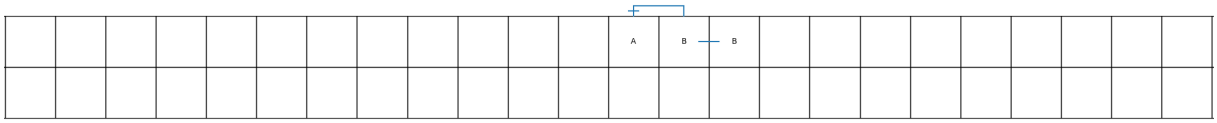
3.5.2 Single Compartment with multiple Circuits

A single compartment with a resource requirement of at least two neuron circuits was placed. The resource requirements have been calculated from the model parameter of the user input. In this step the inner switches are set to connect multiple neuron circuits on hardware to a single compartment. The inner connection is represented by the line between the single neuron circuits of the compartment.



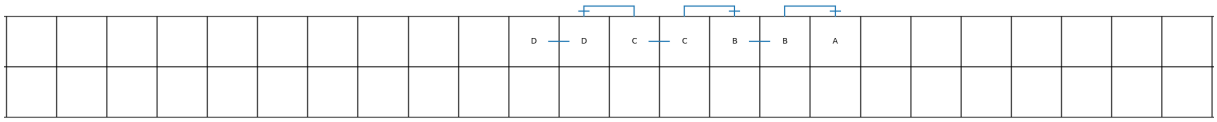
3.5.3 Two Compartments

Two compartments which are connected to each other were placed. Therefore the connection between two compartments, by setting the shared line switches correctly, was implemented. The external connection over the shared line is shown by the horizontal line in the plot. The compartments connect to the shared line directly, indicated by a line, or via a resistor, indicated by a cross.



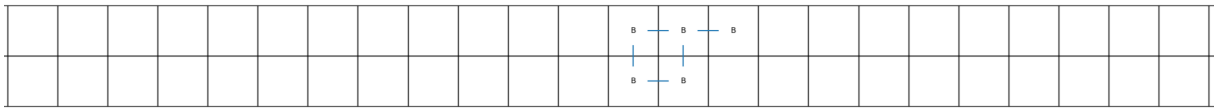
3.5.4 Compartment Chain

Multiple compartments were connected in a chain. Therefore each compartment has a maximum of two neighbours and it is therefore sufficient to only use the top row.



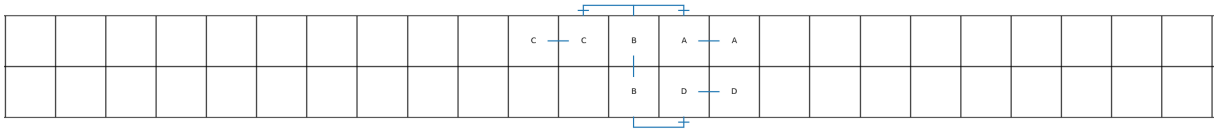
3.5.5 Compartment with Synaptic Input

A single compartment with a synaptic input mechanism is placed. The synaptic input mechanism allows to request a specific amount of synaptic inputs from the top or the bottom and therefore requesting the number of neuron circuits in the top and bottom row. The required resources for this neuron are 5 synaptic inputs in total with 2 in the bottom. Excess neuron circuits are then placed in the top row.



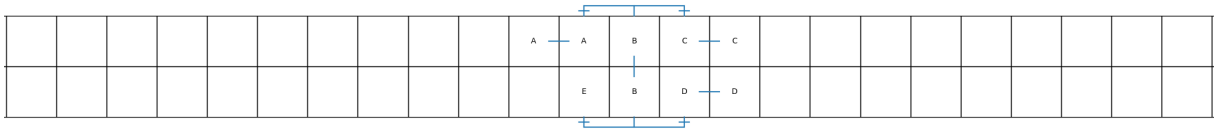
3.5.6 Compartment with more than two neighbours

If a compartment has more than two neighbours it needs to have at least one neuron circuit in the top row as well as in the bottom row to be able to connect to up to four other compartments. If no neuron circuit is explicitly required by synaptic input to be in the top or bottom row the compartment changes its shape to have at least one top row and one bottom row neuron. Therefore it can also increase its size. This change of shape respects the minimal neuron circuits required by the hardware resource requirements.



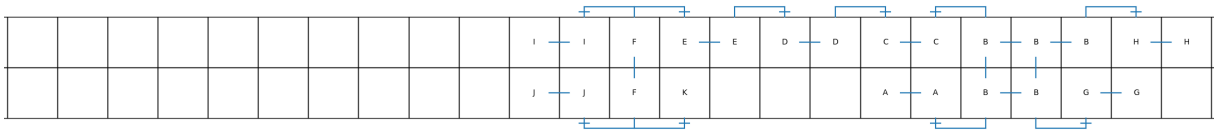
3.5.7 Four Connections

The maximum number of connections one compartment can possibly hold, at the moment, is four. It then has two connections in the top row and two connections in the bottom row. To increase the number the size and shape of the compartment needs to be modified.



3.5.8 Combination Test

The last successfully tested neuron is the one displayed below. It combines structures mentioned above to one larger neuron. The synaptic input which requires resources in top and bottom row is present in compartment B. The change of shape which allows to connect to more than two compartments is present in compartment F. A chain of compartments is present in the compartments C, D, E.



3.5.9 Implacable Neuron Structures

As mentioned before there are still limitations to the complexity of the neuron. Implacable structures are for example

- Compartment with more than four neighbours
- Compartment with multiple leaf compartments connected to a single shared line
- Neuron with a central compartment with more than two connections which each branch into multiple chains
- Neuron with a resource requirement of over 256 neuron circuits

The first two limitations mentioned can be solved easily and will be implemented in the future, whereas the last two points mentioned pose logical problems and will therefore not be possible to be placed at all.

The number of neuron circuits can not be greater 256 since one BSS-2 chip has 512 neuron circuits but only half of the circuits can be used since the chip is partitioned in two halves.

The branches branching into multiple chains are a problem resulting from the fact, that the neurons are placed in two rows on the BSS-2 chip. since a branching compartment needs to be placed on the top and the bottom row to allow more than two connections only one compartment on each side, starting from a central compartment with the most branches, can

be placed. If these compartments branched again the branching of one chain would block the branching of the other. Therefore this combination is logically impossible.

4 Discussion

The placement of complex neurons, consisting of multiple compartments, onto the neuromorphic hardware BrainScaleS-2 was implemented in two main steps. First an abstraction model of the neuron, which is intuitive and fast to use, was implemented. Since the abstraction model is generic it can be used in different cases, for example on different hardware systems or in simulations. In the second step the placement on the coordinate system, representing the BSS-2 neuron circuits, was implemented.

The development of the algorithm happened in multiple steps. The algorithm was sequentially improved. After the implementation of a new feature the possibilities and limitations of the algorithms state were determined and therefore the changes for the next step declared. The verification of each step taken in the development can be seen in subsection 3.5. This allowed the placement of neurons with increasing complexity over time. Placable neurons need to meet the following requirements: no compartment has more than four neighbours and the compartments need to be large enough by resource requirements to allow placement. Therefore chains can be placed as well as neurons with compartments with up to four neighbours.

The use of the implemented features is demonstrated in a example, which shows the abstract neuron, the code needed for the placement as well as the placement result, see Figure 6.

5 Outlook

This work will be continued in a Bachelors thesis with the goal to implement another algorithm which will have an evolutionary approach as well as improving the presented algorithm to work for a larger variety of neurons.

Also added will be the connection between the current Python-Interface and the `PyNN`-library as well as the connection to a deeper software-layer for translation of the model parameter into hardware parameter, calibration and placement on the BSS-2 chip.

References

[1] Boost documentation: Graph isomorphism. https://www.boost.org/doc/libs/1_85_0/libs/graph/doc/isomorphism.html.

[2] Jakob Kaiser, Sebastian Billaudelle, Eric Müller, Christian Tetzlaff, Johannes Schemmel, and Sebastian Schmitt. Emulating dendritic computing paradigms on analog neuromorphic hardware. *Neuroscience*, 489:290–300, 2022.

[3] Johann Klähn. genpybind software v0.2.0. https://github.com/kljohann/genpybind, 2020 doi:10.5281/zenodo.372674.

[4] Christian Pehle, Sebastian Billaudelle, Benjamin Cramer, Jakob Kaiser, Korbinian Schreiber, Yannik Stradmann, Johannes Weis, Aron Leibfried, Eric Müller, and Johannes Schemmel. The BrainScaleS-2 accelerated neuromorphic system with hybrid plasticity. *Front. Neurosci.*, 16, 2022.